Федеральное государственное автономное образовательное учреждение
высшего образования
«Санкт-Петербургский политехнический университет Петра Великого»

На правах рукописи

Ненашев Олег Вячеславович

РЕИНЖИНИРИНГ ЦИФРОВЫХ УСТРОЙСТВ И ВСТРАИВАНИЕ СРЕДСТВ ТЕСТИРОВАНИЯ НА БАЗЕ МНОГОУРОВНЕВЫХ МОДЕЛЕЙ

Специальность 05.13.05 –
Элементы и устройства вычислительной техники и систем управления

ДИССЕРТАЦИЯ

на соискание учёной степени кандидата технических наук

Научный руководитель:
к.т.н., доцент
Филиппов А.С.

Санкт-Петербург – 2015

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В течение жизненного цикла устройств вычислительной техники и систем управления требуется их доработка для обеспечения соответствия изменяющимся требованиям по надёжности, точности, энергоэффективности и прочим показателям. Изменение характеристик требует внесения в устройства значительных функциональных и архитектурных преобразований. Данный процесс называется реинжинирингом устройств и является одной из наиболее ресурсоёмких задач при разработке современных однокристальных цифровых устройств. Задачи реинжиниринга СнК могут решаться многократно, поэтому требуется их алгоритмизация с использованием специализированных инструментариев: моделей, методов и средств.

Необходимость реинжиниринга устройств обусловлена бурным развитием электронной промышленности и связанных с ней областей в соответствии с законом Мура. Постоянный рост сложности электронных систем, изменение технологий и проектных норм требуют внесения существенных изменений в существующие устройства, для чего требуются специализированные инструментарии, включающие специализированные модели, методы и средства. Данные инструментарии впоследствии могут быть использованы как при построении систем автоматизации проектирования (САПР) устройств, так и при использовании ручных методов.

Существует множество подходов для решения частных задач реинжиниринга систем (оптимизация при синтезе и размещении элементов, внесение средств отказоустойчивости и т.п.). В то же время остаётся открытой задача построения моделей и инструментальных средств, которые позволили бы решать специфические для научных групп и малых предприятий задачи. В современных работах по реинжинирингу систем предлагают создание расширяемых моделей и методик, которые могут быть адаптированы для решения частных задач реинжиниринга цифровых однокристальных устройств.

Существует несколько основных направлений исследований в области построения моделей устройств. Модели на базе низкоуровневых структурных представлений или примитивов языков описания устройств разрабатывались Вилси, Басаргиным, Менсом, Виллисом. Применение высокоуровневых описаний на базе UML (Unified Modeling Language) рассматривалось ранее в работах Видала, Бурместера, Чена, Рамоса и др. Применение абстрактных синтаксических деревьев исходного описания устройств рассматривалось Хатчингсом, Монсоном и Похлом. Кроме академических исследований, современные САПР (например, SAMATE, DMS Software Reengineering Toolkit) содержат внутренние модели представления устройств, но доступ к информации о них ограничен. В большинстве случаев модели направлены на решение частных задач реинжиниринга в узком классе. Универсальные модели на базе UML (Unified Modeling Language) обладают высокой сложностью и недостаточной степенью формализации, что затрудняет их применение в задачах реинжиниринга. В области разработки и реинжиниринга программного обеспечения многоуровневые модели, сочетающие преимущества различных подходов, рассматривались Кларком, Франком, Баторием и другими. При этом практически отсутствуют работы, посвящённые созданию многоуровневых моделей для области разработки цифровых устройств.

Одним из классов задач реинжиниринга является встраивание средств внутрисхемного тестирования (СТ) в системы на кристалле (СнК). Существуют примеры решения частных задач: внедрение тестовых агентов и интерфейсов, генерация тестов и т.п. Автоматизированный синтез подсистем тестирования для цифровых систем с процессором рассматривается в работах Путри, Катлера, Зайцева и Чена. Встраивание независимых от процессора компонентов тестирования и самодиагностики рассматривается в работах Сетхурама, Чена, Мелехина. Также существует множество примеров встраивания специализированных средств тестирования для частных задач. Эмуляция сбоев памяти рассматривается в работах Сигалла, Делонга, Мадейры и других. Существуют аналогичные примеры для других классов задач. При этом

основную проблему представляет универсализация подходов, для чего актуальна задача их адаптации к другим моделям представления устройств, используемых при реинжиниринге систем.

Целью исследования является разработка новых моделей, а также методов анализа, трансформации и контроля элементов и устройств вычислительной техники и систем управления для последующего использования в маршрутах проектирования с целью снижения затрат на разработку и улучшения технико-экономических и эксплуатационных характеристик.

Объектом исследования являются процессы проектирования, анализа, модификации и контроля элементов и устройств автоматики и вычислительной техники. **Предметом исследования** являются расширяемые метамодели устройств, а также методы их обработки и специализации для решения широкого класса прикладных задач реинжиниринга цифровых устройств. Для достижения поставленной цели решены следующие **научные задачи**:

1. Формирование критериев оценки и анализ существующих подходов к построению моделей устройств для задач реинжиниринга устройств.
2. Разработка новой многоуровневой модели устройств, предназначенной для решения задач реинжиниринга цифровых устройств.
3. Разработка методик построения модели из описаний на основе существующих языков описания устройств, и использование модели в задачах анализа, трансформации и верификации устройств.
4. Специализация модели для решения задач реинжиниринга в области встраивания средств внутрисхемного тестирования СнК.

Поставленные научные задачи исследования относятся к специальности 05.13.05 «Элементы и устройства вычислительной техники и систем управления», так как реинжиниринг устройств и встраивание средств внутрисхемного тестирования входят в маршруты проектирования цифровых устройств и позволяют обеспечить новые свойства и характеристики элементов, схем и устройств вычислительной техники и систем управления.

При проведении исследований и апробации разработанных в диссертационной работе моделей и методик использованы методы системного анализа, теории алгоритмов, теории программирования и теории графов. Также применены методы экспертных оценок для сравнительного анализа существующих моделей представления устройств.

Научная новизна работы определяется разработкой новой гибридной метамодели представления устройств, отличной от существующих использованием низкоуровневого структурного и высокоуровневого семантического уровней описания в рамках единой модели, что позволяет расширить класс решаемых задач реинжиниринга. Для данной модели также разработаны новые методы специализации, построения по внешним описаниям, анализа и трансформации модели при решении задач реинжиниринга.

Теоретическая значимость результатов исследования обусловлена тем, что применение разработанных моделей, методов и средств позволяет упростить разработку новых методов реинжиниринга и их прототипирование при проведении научных исследований и конструкторских работ в областях, связанных с элементами и устройствами цифровых систем, а именно:

- исследование общих свойств и принципов функционирования элементов, схем и устройств вычислительной техники и систем управления;
- разработка принципиально новых методов анализа и синтеза элементов и устройств вычислительной техники и систем управления с целью улучшения их характеристик;
- апробация научных подходов, методов, алгоритмов и программ, обеспечивающих надежность, контроль и диагностику однокристальных цифровых устройств.

Практическая значимость работы заключается в возможности расширить область применения реинжиниринга за счёт использования разработанных моделей, методов и инструментальных средств. Это позволяет

улучшить технико-экономические и эксплуатационные характеристики СнК за счёт ряда факторов:

- снижения затрат на проектирование цифровых систем путём повторного использования существующих наработок и архитектур устройств;
- снижения затрат на внутрисхемное тестирование цифровых однокристальных устройств за счёт предложенных в работе методов генерации и встраивания средств внутрисхемного тестирования;
- алгоритмизации и формализации наиболее ресурсоёмких операций из процесса разработки с целью их последующей автоматизации в САПР;
- унификации процессов разработки устройств за счёт использования единого инструментария для различных задач реинжиниринга устройств;
- улучшения качества устройств за счёт минимизации влияния человеческого фактора при внесении изменений в устройство;
- возможности создания инструментариев реинжиниринга для частных задач, встраиваемых в САПР и маршруты проектирования.

Практическая значимость результатов исследования подтверждена внедрением в проектах по разработке сложных СнК в компаниях ООО «Синопсис СПб» (Synopsys, Inc.) и ООО «ЭсДиСи», в которых за счёт внедрения разработок удалось значительно снизить временные затраты на проведение внутрисхемного тестирования. Результаты работы также внедрены в учебный процесс на кафедре компьютерных систем и программных технологий и использованы в проекте №2.1.2/12647 «Исследование фундаментальных свойств асинхронных многопроцессорных вычислительных структур в базисе перепрограммируемых логических кластеров». Имеются соответствующие акты о внедрении.

Достоверность полученных в работе результатов подтверждается актами о внедрении и апробацией. Результаты работы докладывались на 12 научных конференциях и семинарах, в том числе: IX Европейской конференции про разработке программного обеспечения (ESEC, Санкт-Петербург, 2013), I

Всероссийском конгрессе молодых учёных (Санкт-Петербург, 2013), симпозиуме по автоматизированной верификации встраиваемых систем (VES2013, Санкт-Петербург, 2013), II Технической Конференции по производству, разработке и испытанию изделий (Санкт-Петербург, 2013), симпозиуме по автоматизации проектирования СнК (Хайдарабад, 2014), VI Всероссийской научно-технической конференции "Проблемы разработки перспективных микро- и наноэлектронных систем" (MES, Москва 2014), Международной конференции по электрическим цепям, системам и цифровой обработке сигналов (ICCSSP, Санкт-Петербург, 2014) и других научных мероприятиях. Результаты внедрения разработанного прототипа также были представлены на практической конференции, посвящённой системе непрерывной интеграции Jenkins CI (JUC, Лондон, 2015).

По теме диссертации опубликовано 12 работ, из которых 3 - в изданиях, рекомендованных ВАК Минобрнауки Российской Федерации. Пять работ опубликованы на английском языке, две проиндексированы в Scopus.

На защиту выносятся следующие основные положения:

1. Разработана новая гибридная метамодель цифровых устройств, входящая в класс многоуровневых моделей, отличающаяся от других совмещением двух уровней представления и применимая в широком классе задач анализа, модификации и верификации архитектур цифровых устройств.
2. Предложены и обоснованы функционально-полный набор базовых операций над моделью, а также методики построения модели из исходных описаний и специализации модели, позволяющие эффективно описывать алгоритмы реинжиниринга устройства.
3. Разработаны методы встраивания средств внутрисхемного тестирования и самодиагностики, которые снижают затраты на тестирование и риск возникновения ошибок, а также могут быть интегрированы в типовые маршруты проектирования однокристальных цифровых устройств.

4. Разработана методика совместной верификации модели устройств в системе моделирования и аппаратных прототипов на базе единого набора тестов, в отличие от других не требующая процессорных блоков.

Личный вклад автора. Содержание диссертации и основные положения, выносимые на защиту, отражают персональный вклад автора. Подготовка к публикации полученных результатов проводилась автором лично или совместно с соавторами, причем вклад автора диссертации был определяющим. Постановка задач исследования и первичный анализ подходов выполнены автором совместно с к.т.н. А.С. Филипповым и ст. преп. С.Л. Максименко. Прочие представленные в диссертации результаты получены лично автором.

Прототипирование модели, методик и средств выполнено автором лично. Разработка методик внесения неисправностей в блочную память устройств выполнена совместно с О.В. Мамутовой (м.т.т., старший преподаватель кафедры КСПТ СПбПУ). Прототипирование методов анализа структуры нетлистов и поиска элементов выполнена совместно с И.В. Егоровым (м.т.т., аспирант кафедры КСПТ СПбПУ). Внедрение прототипа САПР на базе разработанного инструментария PHRT для задач внутрисхемного тестирования и самодиагностики проводилось совместно с инженерами ООО “ЭсДиСи”. Прочие задачи по прототипированию и внедрению решены лично автором.

Структура и объём диссертации. Диссертация состоит из введения, 5 глав, заключения, списка использованных сокращений и определений, списка использованных источников и 7 приложений. Объём диссертации составляет 195 страниц, в том числе 134 страницы основной части. Работа содержит 42 рисунка, 13 таблиц и список использованных источников из 90 наименований.

В первой главе произведён обзор подходов к построению моделей устройств, сформированы требования к новой модели и поставлены задачи исследования. Во второй главе предложена новая гибридная метамодель однокристальных цифровых устройств. В третьей главе разработаны методы работы с метамоделью: специализации модели для решения частных задач реинжиниринга, импорта модели из внешних описаний, контроля при

выполнении основных операций. В четвёртой главе на базе метамодели разработаны методы встраивания средств внутрисхемного тестирования в устройства. В пятой главе рассмотрены примеры решения частных задач реинжиниринга с использованием разработанных моделей, методов и инструментальных средств. В приложениях приведён обзор существующих средств реинжиниринга цифровых устройств, приведена дополнительная информация о прототипе PHRT и примеры его использования для решения частных задач реинжиниринга. Также приведены акты о внедрении результатов работы в компаниях.

1. ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

В первой главе рассмотрены постановки задач реинжиниринга цифровых однокристальных устройств и СнК на примерах задач встраивания средств внутрисхемного тестирования. Обоснована актуальность выбранной темы исследования, проведён обзор и сравнительный анализ существующих подходов к построению моделей устройств для решения задач реинжиниринга. Подтверждена актуальность разработки новой многоуровневой метамодели устройств, которая могла бы быть специализирована для эффективного описания устройств в частных задачах реинжиниринга., Сформированы требования к подобной метамодели, цели и задачи исследования.

1.1. Задача контроля качества устройств и систем на кристалле

В работе под устройством понимается некоторая аппаратная или программно-аппаратная система, используемая для решения задач управления в областях автоматики и вычислительной техники. Упор сделан на цифровые системы на кристалле, но при необходимости предлагаемые подходы могут быть расширены и на устройства других типов.

При проектировании устройств и СнК выделяют четыре основных стадии проекта: формирование концепции, разработку, трансляцию и завершение. На рис. 1.1 приведены основные этапы проектирования устройства и их составляющие. Данный маршрут проектирования соответствует классической каскадной модели, рекомендованной Институтом Управления Проектами [57].

Параллельно с разработкой устройств идут и другие процессы: управление требованиями и изменениями, документирование и пр. [57]. Также проводится контроль качества, который может своевременно дать сигнал об отклонении проекта от поставленных требований, вследствие чего может потребоваться реинжиниринг существующей архитектуры. В исследовании [35] Гонсалес и соавторы выделяют два процесса контроля качества: анализ качественных характеристик (рис. 1.2) и верификация системы для проверки её соответствия поставленным требованиям.

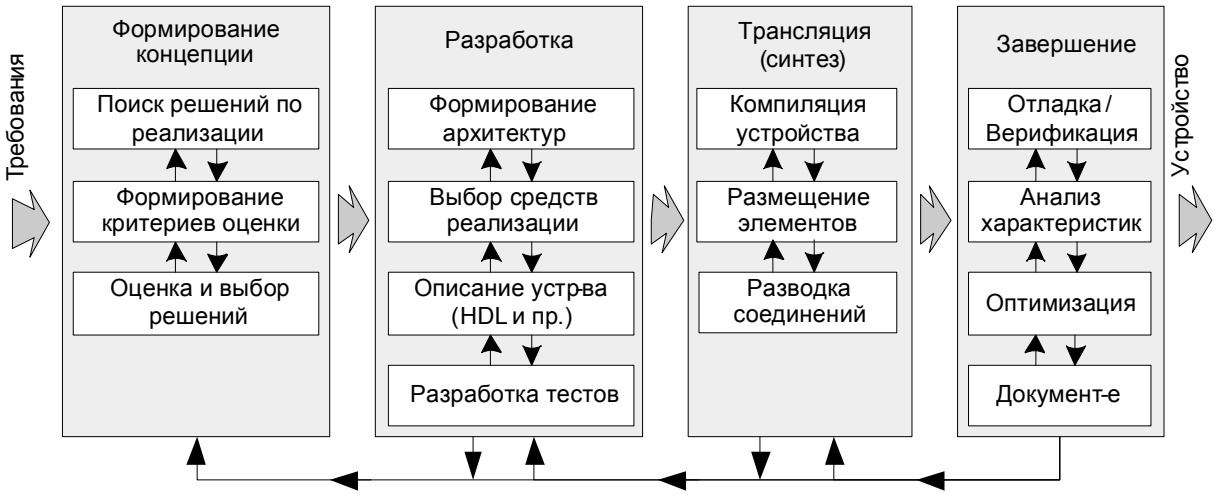


Рис. 1.1. Основные стадии проектов по разработке цифровых устройств [35]

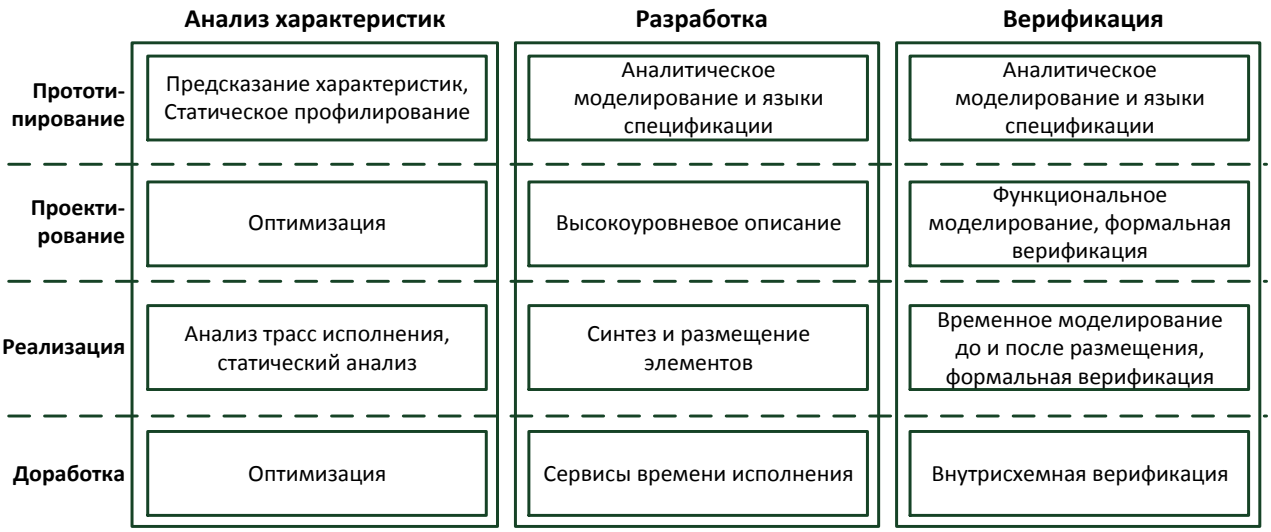


Рис. 1.2. Процессы контроля качества при разработке устройств [35]

Ниже рассмотрены основные этапы проектирования и решаемые на них задачи контроля качества электронных устройств.

Этап формирования концепции соответствует стадии научных исследований и опытно-конструкторской разработки (НИОКР). На нём производятся предварительные исследования, направленные на поиск методологических и архитектурных подходов, реализующих поставленные требования. Также формируются критерии принятия решений при проектировании архитектуры и выборе реализации в устройстве.

На этапе разработки устройства формируется его архитектура, после чего она реализуется на специализированных языках описания аппаратуры. Для реализации устройства осуществляется выбор средств разработки. Средства для решения задачи реинжиниринга могут также выбираться на данном этапе. В конце этапа для разработанного устройства создаются тестовые планы и наборы тестов, которые позволяют подтвердить корректность работы

На этапе трансляции производится преобразование представления устройства в описание его физической реализации. При этом производится оптимизация связей устройства, адаптация архитектуры под аппаратную платформу, физическая разводка соединений и так далее. Данный этап определяет характеристики будущего устройства, а также имеет высокую значимость из-за больших ресурсных и временных затрат. Если для программного обеспечения трансляция идёт автоматически и занимает минуты/часы, то трансляция сложного устройства с учётом частичной ручной оптимизации скоростных характеристик может занимать месяцы [40].

При завершении разработки осуществляется отладка устройства, анализ его характеристик и их доводка под поставленные требования. В случае успешного завершения проекта осуществляются приёмо-сдаточные испытания, подготовка сопроводительной документации и пр. Также на этом этапе производится передача средств диагностики устройств в эксплуатацию и их апробация на промышленных образцах устройства.

1.2. Постановка задачи внутрисхемного тестирования

На рис. 1.3 приведены основные этапы верификации СнК: моделирование при начальной разработке, прототипирование на ПЛИС, а также верификация и диагностика устройств и тестовых интегральных схем. На всех этапах требуется проведение различных тестов для контроля качества изделия и оценки его характеристик. Для тестирования на аппаратных платформах может потребоваться встраивание специализированных средств и подсистем

тестирования, обеспечивающих выполнение тестов, а также необходимый уровень наблюдаемости и управляемости тестируемого устройства.

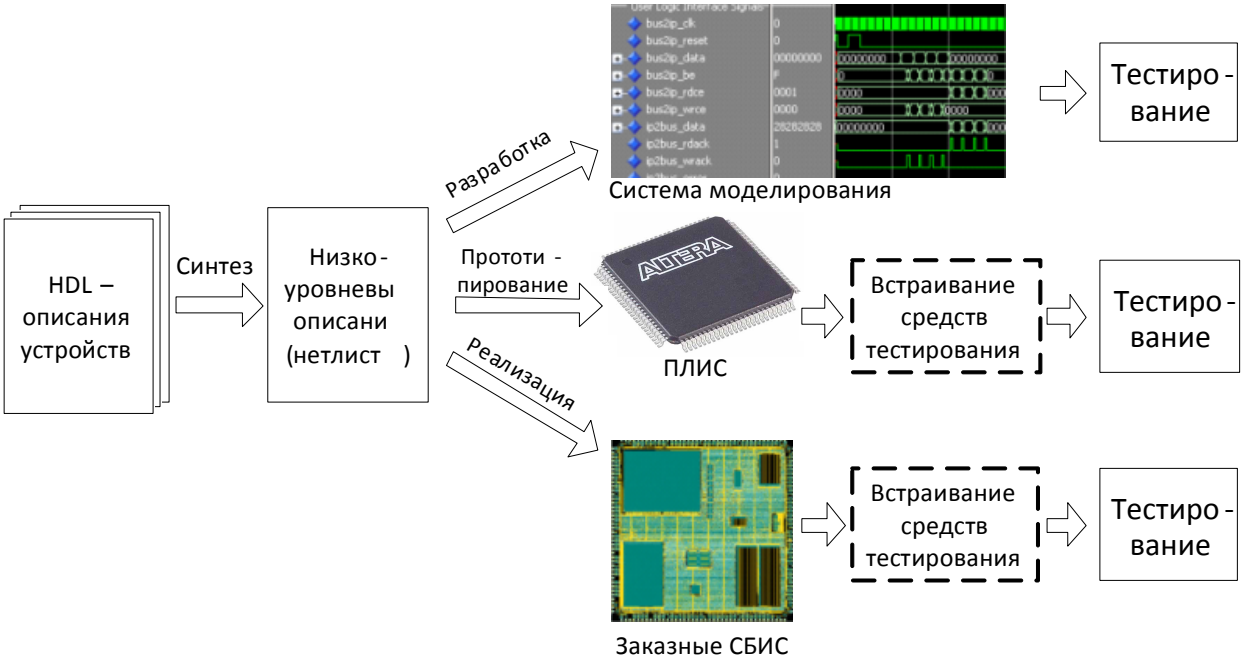


Рис. 1.3. Верификация СнК на различных этапах проектирования.

Поскольку разработка и верификация цифровых устройств начинаются с виртуальных прототипов в системах моделирования, то для снижения затрат на верификацию целесообразно повторно использовать тесты, полученные на этапе разработки [4]. Для тестирования аппаратных реализаций можно выделить следующие случаи:

1. Встраивание СТ не требуется, так как для выполнения тестов используются существующие универсальные компоненты устройства (например, встроенные процессорные блоки).
2. Встраивание СТ производится при проектировании системы.
3. Для верификации СнК требуется встраивание СТ, так как они отсутствуют в проекте устройства.

Применение универсальных компонентов наиболее распространено в сложных системах, оснащённых процессорными блоками, которые управляют остальными компонентами СнК. В этом случае можно использовать специализированные тестовые программы, обеспечивающие значительное

тестовое покрытие СнК. При этом актуальна задача повторного использования тестов на различных этапах проектирования с целью снижения общих затрат на верификацию устройств. Данный подход наиболее прост, так как он не требует разработки специализированной инфраструктуры, но при этом могут возникнуть значительные ограничения: скорость выполнения тестов, плохая наблюдаемость и управляемость отдельных компонентов СнК.

Указанные выше ограничения влияют на самотестирование и самодиагностику процессорного ядра, компоненты которого не могут быть интегрированы в СТ на этапе разработки, в том числе по соображениям информационной безопасности. Примерами труднодоступных компонентов являются блоки защиты памяти, промежуточные буферы данных программ и данных, компоненты предвыборки программ, теневые регистры в системах с переключением контекста. Тестирование перечисленных компонентов возможно лишь по косвенным признакам и без применения специальных методик имеет вероятностный характер [26].

Разработка СнК в современных методологиях проектирования ведётся с учётом требований к тестируемости (методика DFT – Design For Testability), поэтому средства и интерфейсы тестирования обычно встраиваются на этапе проектирования и реализации устройства [76]. Соблюдение правил проектирования позволяет добиться высокой наблюдаемости и управляемости компонентов системы без значительного влияния на характеристики СнК. В случае невозможности построения тестовой архитектуры при проектировании, возможно встраивание тестовых компонентов путём реинжиниринга архитектуры в процессе оптимизации проекта.

Встраивание средств тестирования и самодиагностики

Задачи встраивания средств тестирования и самодиагностики могут ставиться для уже существующих проектов. К этому могут привести следующие факторы:

– повышение требований к надёжности устройства для его повторного использования в рамках новых разработок;
– выявление и устранение неисправностей по итогам эксплуатации;
– изменение функциональности устройства, которое требует внесения новых средств тестирования.

В перечисленных случаях необходима модификация существующей архитектуры, поэтому задача встраивания СТ в готовые проекты устройств является частным случаем задачи реинжиниринга однокристальных цифровых устройств. Понятие реинжиниринга цифровых систем рассмотрено в подпараграфе 1.3.1.

Как и иные задачи реинжиниринга, встраивание средств тестирования требует большое количество временных и человеческих ресурсов проекта, поэтому актуальна задача автоматизации данного процесса. Для ее решения требуются специализированные инструментарии, для описания алгоритмов анализа и трансформации в которых используется некоторая внутренняя модель устройств. В общем случае к этой модели и САР предъявляются следующие требования:

– возможность интеграции в существующие маршруты проектирования электронных устройств и СНК;
– поддержка популярных языков описания устройств (HDL – Hardware Description Language): VHDL, Verilog, SystemC и SystemVerilog;
– возможность повторного использования ранее разработанных алгоритмов реинжиниринга;
– выполнение преобразований в рамках заданных ограничений по системным ресурсам и временным характеристикам.

Встраивание СТ для ускорения выполнения тестов в системах моделирования

Отдельным случаем встраивания СТ является совместное применение методик внутрисхемного тестирования и систем моделирования. Системы моделирования обеспечивают хорошую наблюдаемость и управляемость

тестируемого устройства, но при этом страдает скорость выполнения тестов. Даже при использовании специализированного программного обеспечения и вычислительных комплексов время длительных тестов, таких как загрузка операционных систем и оценка их производительности, может занимать несколько суток [30]. Развёртывание подобных систем является актуальной задачей при проектировании сложных СнК.

Так как встраивание средств тестирования в цифровые устройства и СнК является частным случаем реинжиниринга, к классу задач применимо большинство требований и ограничений. Актуальна задача анализа задачи реинжиниринга устройств в целом с целью учёта данных требований и ограничений при разработке новых моделей устройств.

1.3. Введение в реинжиниринг систем

Реинжиниринг является неотъемлемой частью жизненного цикла систем любого типа: технических, информационных, экономических и прочих (рис. 1.4). Любая система со временем устаревает: разрабатываются более эффективные подходы, меняются требования к системе, появляются новые области применения.

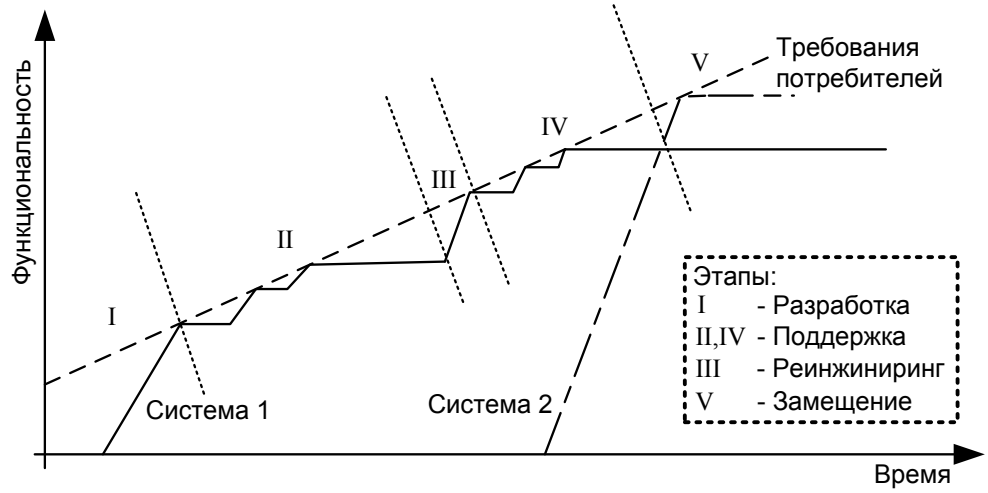


Рис. 1.4. Реинжиниринг в жизненном цикле системы [1]

В течение некоторого времени соответствие системы требованиям обеспечивается за счёт последовательной её доработки, называемой сопровождением системы. Рано или поздно наступает момент, когда малые

изменения при сопровождении системы становятся неэффективными, и требуется кардинальная доработка (или замена) составляющих системы. Данный процесс называется модернизацией или реинжинирингом, формальные определения процесса приведены ниже.

1.3.1. Понятие реинжиниринга систем

В ранних публикациях (до 1989г.) реинжиниринг редко выделялся в отдельный этап и рассматривался как повторная разработка после обратного инжиниринга (реверс-инжиниринга) архитектуры системы и корректировки её под новые требования. Кроме термина “reengineering”, в иностранных работах использовались синонимичные термины “renovation” и “reclamation” [24].

Обратным инжинирингом (реверс-инжинирингом) называется процесс извлечения информации о системе путём анализа её структуры, функций и поведения [29]. Реверс-инжиниринг не предполагает внесения изменений в исходную систему, а служит лишь для получения исходных данных для дальнейшей работы над системой.

Как писали Чикофски и Кросс в 1990-м году, “Обратный инжиниринг развивается как основное связующее звено в жизненном цикле программного обеспечения (ПО), но его росту препятствует путаница в терминологии” [29]. В данной работе сделана попытка систематизировать понятие реинжиниринга и определить его роль в жизненном цикле системы. Авторами введено следующее понятие реинжиниринга:

Реинжиниринг - анализ и модификация целевой системы с целью представления её в новой форме и последующей реализации [29].

В предложенном определении, по мнению автора работы, неточно указаны цели реинжиниринга. В современной трактовке они сформулированы Хаммером (1990г.) при описании методов реинжиниринга бизнес-процессов предприятия с целью повышения их экономической эффективности. Им предложено следующее определение:

Реинжиниринг - это фундаментальное переосмысление и радикальное перепроектирование деловых процессов для достижения резких, скачкообразных улучшений главных современных показателей деятельности компании, таких, как стоимость, качество, сервис и темпы роста [38].

Несмотря на экономическую ориентацию, предложенные Хаммером подходы базируются на техническом анализе и системном подходе. В последующих работах реинжиниринг приобрёл значение самостоятельного процесса в рамках жизненного цикла продукта. Для реинжиниринга систем в диссертации предложено следующее определение:

Реинжиниринг - это систематическая трансформация существующей системы с целью улучшения ее характеристик: расширения поддерживаемой ею функциональности, снижения стоимости ее сопровождения и риска возникновения отклонений от требований, уменьшения сроков работ по сопровождению системы.

В большинстве случаев реинжиниринг работает не с целевым объектом, а с его моделью, которая отражает требуемые характеристики преобразуемого объекта (функциональность, структуру, алгоритмы поведения).

Отличия реинжиниринга от поддержки системы

Принципиальным вопросом является разделение реинжиниринга и поддержки системы. В [77] Баринов В.А. выделяет ряд различий между реинжинирингом и поддержкой, основным отличием которой является эволюционная модернизация системы. В таблице 1.1 приведены наиболее значимые различия.

Положение реинжиниринга в жизненном цикле системы

В соответствии с [77] реинжиниринг системы требуется, когда исчерпаны возможности по поддержке системы путём наращивания функциональности. Во многих случаях его можно начать заранее, чтобы подготовить новую систему до появления серьёзного отставания исходной системы от требований.

Реинжиниринг может потребоваться и при разработке новой системы, если возникает отклонение от поставленных требований. Это может быть обнаружено во время контроля качества или верификации системы, а также при тестовой эксплуатации. Также может потребоваться доработка существующих блоков с целью их повторного использования в новой системе [62]. В этом случае процесс реинжиниринга запускается не для всей системы, а только для её отдельного компонента.

Таблица 1.1
Различия поддержки и реинжиниринга систем [77]

Параметр	Поддержка системы	Реинжиниринг
Подход	Эволюционный	Революционный
Исходные данные	Существующий процесс	«Чистая доска»*
Частота изменений	Непрерывно/единовременно	Единовременно
Длительность изменений	Малая	Большая
Направление изменений	Снизу вверх	Сверху вниз
Охват	Узкий — на уровне функций (функциональный подход)	Широкий — межфункциональный

Процесс реинжиниринга тесно вплетён в жизненный цикл сложных систем. В [1] перечислены следующие частные задачи реинжиниринга, возникающие на различных этапах разработки и эксплуатации системы:

- прямой инжиниринг (Forward engineering);
- редокументирование (Redocumentation);
- рефакторинг (Refactoring);
- реструктуризация (Restructuring);
- обратный инжиниринг (Reverse engineering);
- сопровождение программных продуктов (Software maintenance);
- трансляция исходного кода (Source Code Translation).

На рис. 1.5 показаны задачи реинжиниринга, связанные с разработкой и модернизацией сложных систем.

Стандарт ISO/IEC 12207:1995 определяет множество различных процессов жизненного цикла системы [66]. На рис. 1.6 приведена иерархия процессов в соответствии со стандартом; штриховкой обозначены процессы, связанные с реинжинирингом.

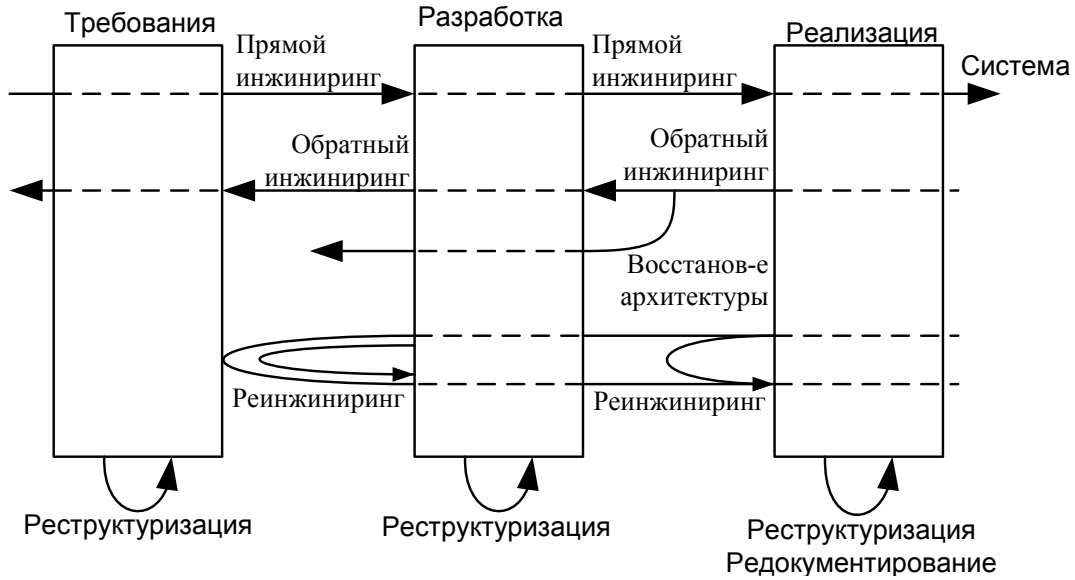


Рис. 1.5. Задачи реинжиниринга на этапах проектирования систем [29]

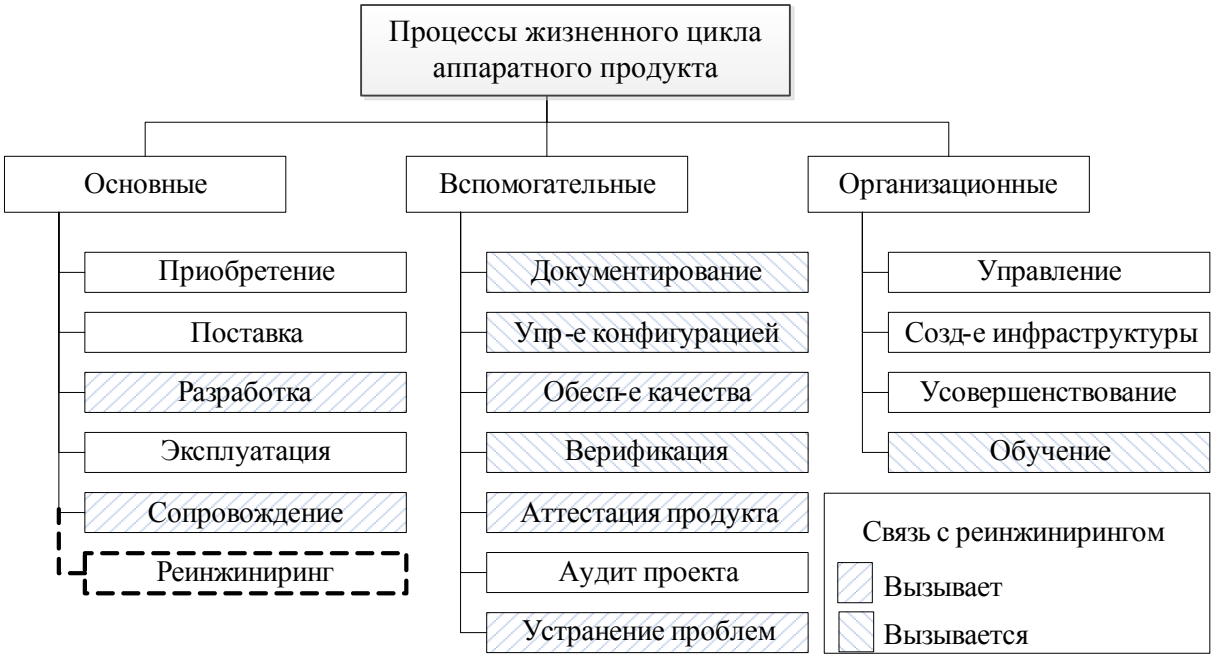


Рис. 1.6. Связи реинжиниринга с процессами жизненного цикла системы по стандарту ISO/IEC 12207:1995 [66]

1.3.2. Процесс реинжиниринга

Некоторые авторы указывают, что процесс реинжиниринга схож с процессом разработки. По их мнению, основное различие состоит в том, что при реинжиниринге исходными данными являются не только требования к системе, но и некоторая исходная система, не соответствующая поставленным требованиям в полной мере [17]. В [1] рассматриваются следующие основные фазы реинжиниринга:

- оценка соответствия характеристик исходной системы требованиям;
- принятие решения о необходимости проведения работ по реинжинирингу или дальнейшего сопровождения системы;
- проведение реинжиниринга;
- внедрение системы, полученной в результате проведения реинжиниринга.

Реинжиниринг систем может производиться различными путями, но при этом чётко выделяются три этапа: восстановление архитектуры, её анализ и преобразование в соответствии с поставленными требованиями, а также последующая реализация новой архитектуры. Данный подход (модель «подковы») предложен Кацманом в 1998г для программных систем [45]. На рис. 1.7 приведена схема реинжиниринга в соответствии с моделью «подковы».

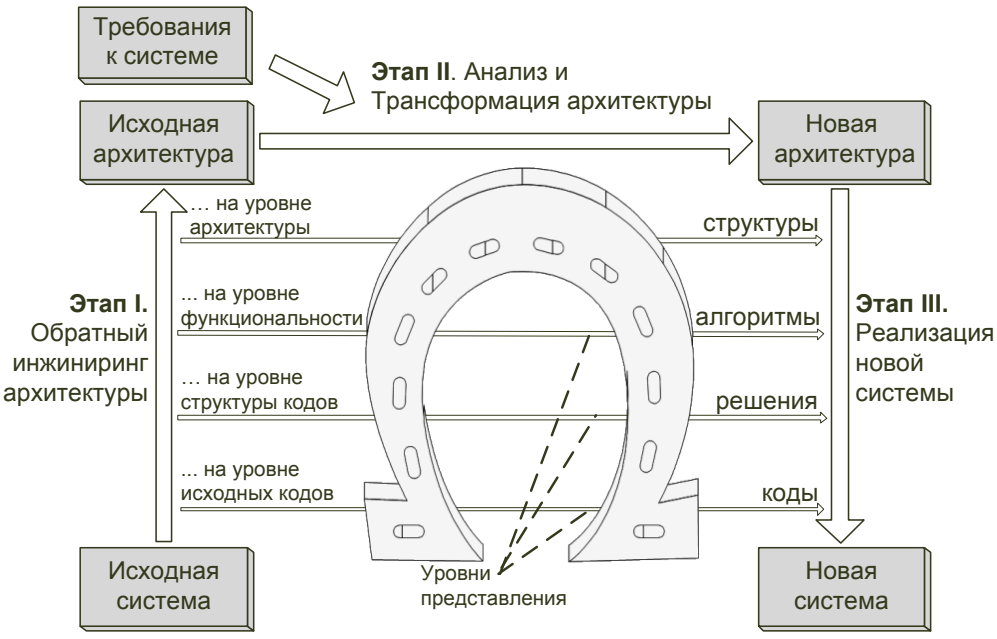


Рис. 1.7. Этапы проведения реинжиниринга системы (модель "подковы") [45]

На первом этапе восстанавливается архитектура существующей системы посредством обратного инжиниринга (реверс-инжиниринга). В случае аппаратных средств исходными описаниями часто выступают описания на уровне HDL, которые фактически являются спецификацией архитектуры устройства и требуют только преобразования архитектуры к представлению, используемому алгоритмами реинжиниринга. Задача может быть осложнена использованием нескольких исходных описаний (например, HDL и нетлист по результатам синтеза) на различных уровнях. В этом случае требуется объединение описаний в единое представление.

На втором этапе полученная архитектура анализируется на предмет соответствия её характеристик поставленным требованиям, после чего принимается решение о трансформации архитектуры. Этапы анализа и трансформации продолжаются до тех пор, пока построенная архитектура не будет соответствовать поставленным требованиям. Таким образом, в рамках модели применяется спиралевидная методика разработки [11].

Третий этап включает деятельность по реализации системы в соответствии с новой архитектурой. Решаются вопросы декомпозиции элементов системы по пакетам, осуществляется выбор стратегий взаимодействия между компонентами системы [1]. На данном этапе производится перенос в новую систему неизменённых частей исходной системы.

1.3.3. Постановка задачи реинжиниринга цифровых устройств

При реинжиниринге цифровых устройств и систем возможны все три класса задач, рассмотренные в предыдущем подпараграфе. Трансформация архитектуры устройства может потребоваться для изменения функциональности устройства или же улучшения его характеристик. Ниже приведены примеры характеристик, которые могут быть изменены:

- функциональность и производительность;
- тестопригодность (управляемость, наблюдаемость элементов);
- надёжность (отказоустойчивость, время наработки на отказ и т.п.);

– временные характеристики (максимальная частота тактирования);
– аппаратные затраты (число ЛЭ, вентилей) и энергопотребление.

В п. 1.3.4 рассмотрены типовые задачи реинжиниринга для каждой из перечисленных групп. В соответствии с [35] параллельно с разработкой решаются задачи контроля характеристик и верификации устройства. Если на каком-либо этапе оказывается, что устройство не соответствует поставленным требованиям, то выполняется его реинжиниринг.

Поскольку устройство описывается на одном из HDL, то восстановление архитектуры не требуется, так как подобные исходные описания уже являются спецификацией архитектуры. Задача реверс-инжиниринга при этом сводится к преобразованию HDL к формату, пригодному для обработки в алгоритме реинжиниринга. Для анализа и синтеза можно использовать стандартные средства разработки, как и в процессе реинжиниринга. В случае наличия необходим средств варианте требуется решить только задачу трансформации архитектуры устройства. На рис. 1.8 приведён пример подобной организации.

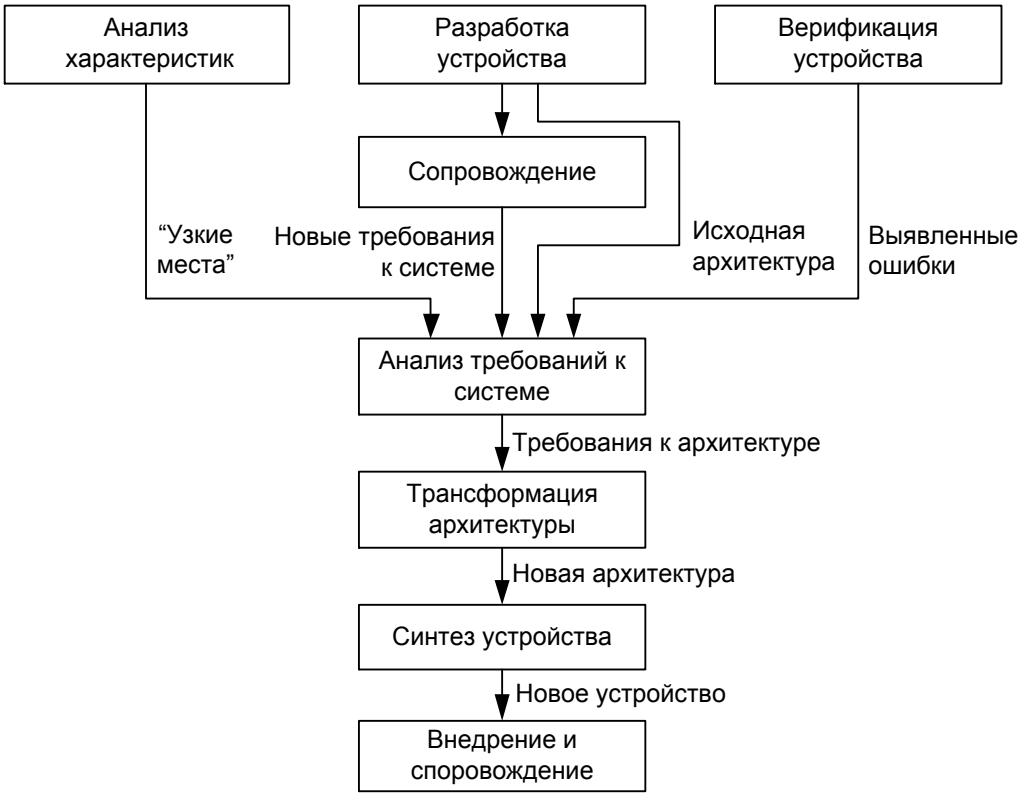


Рис. 1.8. Включение реинжиниринга в процесс разработки устройства

После реинжиниринга требуется провести повторную верификацию и реконфигурация разработанной системы, а также отобразить изменения в проектной документации на систему (редокументирование).

1.3.4. Задачи реинжиниринга устройств и систем

Как было сказано выше, основной задачей реинжиниринга является достижение требуемых характеристик преобразуемой системы. На практике, могут существовать и другие задачи, связанные с преобразованием не архитектуры, а её представления. На рис. 1.5 данное преобразование обозначено малой дугой, не заходящей в фазу разработки. Выделяются следующие классы задач реинжиниринга:

- улучшение характеристик системы путём трансформации архитектуры;
- рефакторинг существующего представления;
- перенос системы из одного представления в другое.

Ниже рассмотрены классы задач реинжиниринга и приведены их примеры в областях реинжиниринга устройств и встраивания СТ.

Задачи изменения характеристик устройства

Изменение характеристик системы предполагает внесение изменений в её архитектуру. Ниже приведены примеры для различных характеристик.

Повышение производительности:

- конвейеризация вычислений;
- аппаратная акселерация программных алгоритмов:
- повышение предельной тактовой частоты устройства.

Повышение надёжности:

- введение структурной избыточности;
- введение временной избыточности;
- замена элементов их отказоустойчивыми аналогами;
- использование помехоустойчивых кодов в памяти и шинах сигналов.

Повышение тестопригодности:

– встраивание средств внутрисхемного тестирования;
– встраивание подсистем самодиагностики;
– встраивание диагностических выводов и отладочных интерфейсов.

Снижение энергопотребления:

– введение регуляторов частоты тактирования;
– введение доменов тактирования;
– добавление средств управления питанием.

Снижение аппаратных затрат:

– удаление избыточных модулей;
– оптимизация архитектуры устройства;
– удаление отладочных и диагностических сигналов.

Рефакторинг исходных описаний устройств на HDL

По определению Мартина Фаулера, рефакторинг – это процесс изменения внутренней структуры программы, не затрагивающий её внешнего поведения и имеющий целью облегчить понимание её работы [86]. Отношение рефакторинга к реинжинирингу является спорным. Некоторые авторы с этим соглашаются (например, [17] и [86]), другие же относят рефакторинг в отдельный процесс [22]. В пользу последнего приводят следующие аргументы:

– рефакторинг не меняет характеристики устройства;
– рефакторинг является непрерывным процессом по улучшению качества представления системы и относится к её поддержке;
– рефакторинг преобразует описание системы, а не её архитектуру.

В соответствии с большинством источников, в данной работе рефакторинг рассматривается как одна из частных задач реинжиниринга. К рефакторингу можно отнести следующие задачи:

– переименование элементов описания (сигналов, модулей и пр.);
– изменение иерархии модулей в описании;
– комментирование исходных кодов представления;
– инкапсуляция функциональности.

При рефакторинге HDL-описаний есть вероятность изменения результатов синтеза в САПР [31]. В отличие от программных средств, при выполнении данного типа реинжиниринга нужен контроль соответствия характеристик системы поставленным требованиям. Функциональная верификация системы необходима для любого класса систем.

При встраивании СТ в большинстве случаев формируются временные HDL-описания для тестовых версий системы, поэтому рефакторинг данных описаний не актуален. В параграфе 5.5 рассмотрено решение задачи рефакторинга для других областей применения реинжиниринга.

Перенос описаний между представлениями

Перенос описаний является промежуточной задачей между рефакторингом и улучшением характеристик. При нём сохраняется функциональность системы, но полностью меняется внешнее представление. Например, можно конвертировать спецификацию на UML (Unified Modeling Language) или MATLAB в исходные коды на некотором из языков программирования [13, 37]. В случае реинжиниринга устройств решение о реинжиниринге для данного класса задач может быть обосновано различными причинами: сменой аппаратной платформы, языка описания устройства, средства и т.д. Ниже приведены примеры задач для каждой из групп.

Задачи смены аппаратной платформы:

- замена специализированных аппаратных модулей их эквивалентами (при переходе с ПЛИС на заказные интегральные микросхемы);
- оптимизация устройства за счёт использования аппаратных средств, таких как умножители или регистры в ПЛИС.

Задачи смены HDL:

- конвертация между поведенческим и структурным описанием СнК;
- генерация HDL из других форм описания (например, генерация тестов из поведенческих описаний на UML-RT [58]);

– объединение в единое устройство аппаратных компонентов, описанных на различных языках HDL;
– конвертация нетлистов в синтезируемые HDL.

Прочие задачи:

– замена программных реализаций модулей (замена блоков с одинаковыми входными и выходными портами);
– реверс-инжиниринг IP-модулей (например, для последующего реинжиниринга);
– реверс-инжиниринг исходных кодов, подвергнутых обфускации (приведение кодов к виду, затрудняющему анализ и модификацию).

1.4. Инструментарии реинжиниринга систем

Под инструментарием в работе понимается совокупность методов, моделей и средств, предназначенных для решения задач проектирования электронных устройств, таких как реинжиниринг или встраивание средств тестирования. Для классификации систем предлагается ввести новое понятие «инструментария реинжиниринга» (ИР), средства которого предлагается называть средствами поддержки реинжиниринга (СПР).

Инструментарий реинжиниринга (ИР) – совокупность моделей, методов и средств, применяемых для решения задач реинжиниринга однокристальных цифровых устройств и систем на кристалле.

Средства поддержки реинжиниринга (СПР) – это инструментальные средства, которые решают частные задачи реинжиниринга системы: восстановления, анализа и трансформации архитектуры, реализации новой архитектуры. СПР могут быть использованы для построения систем автоматизации реинжиниринга (САПР), но в общем случае ими не являются.

Типовая схема средства реинжиниринга предложена Чикофски и Кроссом в 1990-м году (см. рис. 1.9). В этой схеме средство реинжиниринга делится на три модуля, два из которых обеспечивают считывание представления и

генерацию выходных данных. Класс САПР на базе СПР предлагается называть системами автоматизации реинжиниринга (САР).

Предложенная схема может быть применена к любому функциональному блоку, на входе и выходе которого стоят преобразователи данных. К сожалению, широкий спектр задач реинжиниринга исключает возможность построения более детального описания.

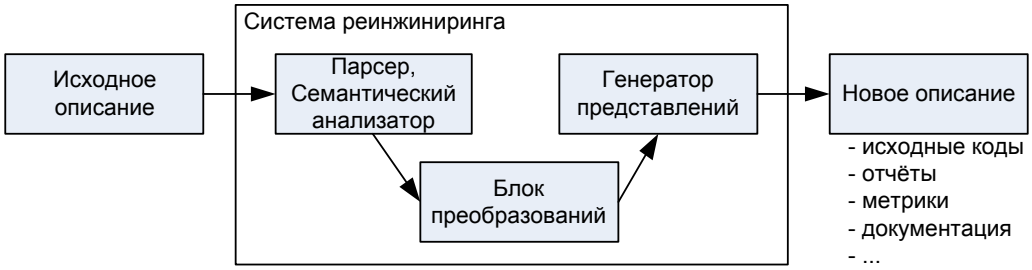


Рис. 1.9. Типовая схема средства реинжиниринга [29]

Проблема программно-аппаратных систем

Большинство современных устройств являются программно-аппаратными системами, при этом их составные части имеют различные представления, что затрудняет их совместный реинжиниринг. Обычно реинжиниринг аппаратуры и программного обеспечения разделяют на две параллельные задачи, связывая их предварительным этапом, на котором формируются требования для обеих составляющих системы (см. рис. 1.10). Подобный подход применяется и в задачах разработки программно-аппаратных устройств [3].

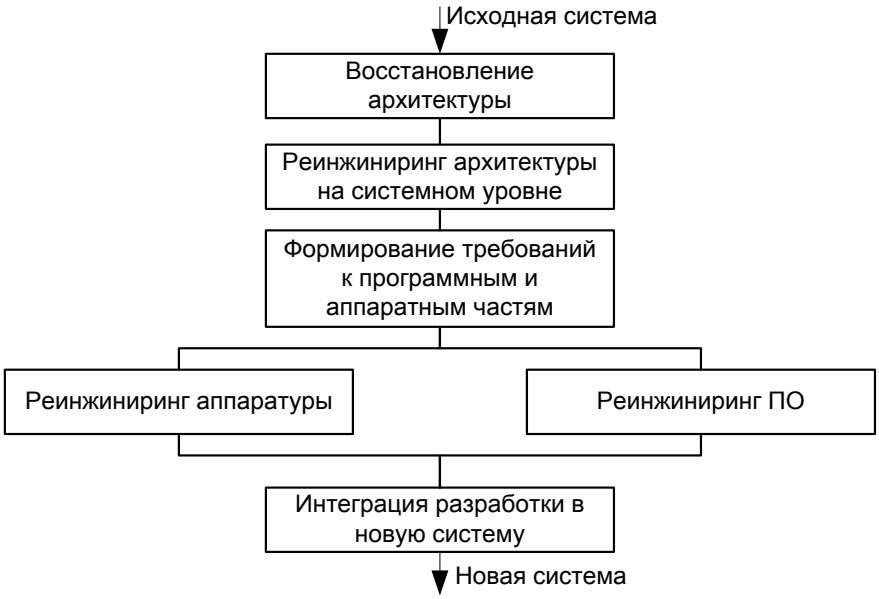


Рис. 1.10. Порядок реинжиниринга программно-аппаратной системы

Реинжиниринг программного обеспечения является отдельной областью знаний, для которой предлагаются свои методики ведения процесса. Существует ряд фундаментальных работ, посвящённых данному вопросу [17]. Поскольку процессы реинжиниринга ПО и аппаратуры можно разделить, то далее будет рассматриваться только реинжиниринг аппаратной части системы.

1.4.1. Особенности описания устройств в инструментариях

В настоящее время существует множество способов описания устройств, в которых используются структурные и принципиальные схемы, блок-схемы, текстовые нотации, модели и пр. Для однокристальных цифровых устройств в настоящее время наиболее распространены описания при помощи специализированных языков программирования, называемых языками описания аппаратуры (Hardware Description Language или HDL).

Языки описания аппаратуры (HDL – Hardware Description Language)

В настоящее время наибольшее распространение получили языки VHDL (HDL for very high speed integrated circuits) и Verilog. Они поддерживают как структурное, так и поведенческое описания [9]. Также существует ряд более современных и высокоуровневых HDL, ориентированных на поведенческое и функциональное представления устройств. Примерами могут быть SystemC, JHDL, UML-спецификации, описания на базе функциональных языков программирования (Lava, Hydra) [23].

Задача синтеза устройства из HDL решается отдельными средствами разработки. Во многом данный процесс схож с компиляцией программного кода. Описания на основе HDL удобны для спецификации устройства разработчиком, но при этом плохо подходят для передачи данных между стадиями маршрута проектирования. Это объясняется следующими причинами:

- существование нескольких редакций для каждого из языков;
- сложность и неоднозначность синтаксиса языков;
- возможность описания одного объекта многими способами.

В большинстве случаев для организации взаимодействия между модулями СПР используются низкоуровневые структурные описания, называемые нетлистами (от англ. «net list» – список соединений).

Низкоуровневые структурные описания (нетлисты)

Так как СПР интегрируются в маршруты проектирования устройства, в качестве внешнего представления целесообразно использовать не только исходные, но и промежуточные описания, передаваемые между стадиями процесса разработки. На рис. 1.11 приведён процесс разработки устройства с указанием типов нетлистов, которые передаются между средствами, реализующими различные стадии процесса реинжиниринга.

Нетлисты можно рассматривать как множество примитивных и чаще всего структурных форматов HDL. Они описывают иерархию модулей и связи между ними, используя минимальный набор функциональных преобразований. На нижнем уровне иерархии находятся функциональные блоки целевой платформы: логические вентили, регистровые элементы, встроенные аппаратные компоненты и т.д. Эти элементы объединяются посредством соединений. Верхние уровни иерархии в нетлистах или полностью скрыты («плоские» нетлисты), или реализуются посредством аннотированных описаний или модульной структуры описания.

Наиболее распространёнными форматами нетлистов является семейство форматов EDIF (Electronic Design Interchange Format), являющийся независимым форматом обмена данными между САПР цифровых устройств различных производителей [82]. Данный открытый формат де-факто является стандартом в области САПР. Кроме него в проектировании аналоговой и аналого-цифровой электроники часто применяется формат SPICE [16].

Средства моделирования также могут включать специализированные форматы нетлистов, которые наиболее приближены к исходным описаниям на HDL. Такие форматы позволяют не только отобразить результаты синтеза, но и представить инженеру описание в формате, удобном для ручного анализа и принятия решений [41]. В приложении D рассмотрены особенности VHDL-нетлистов, которые используются в системах моделирования устройств.

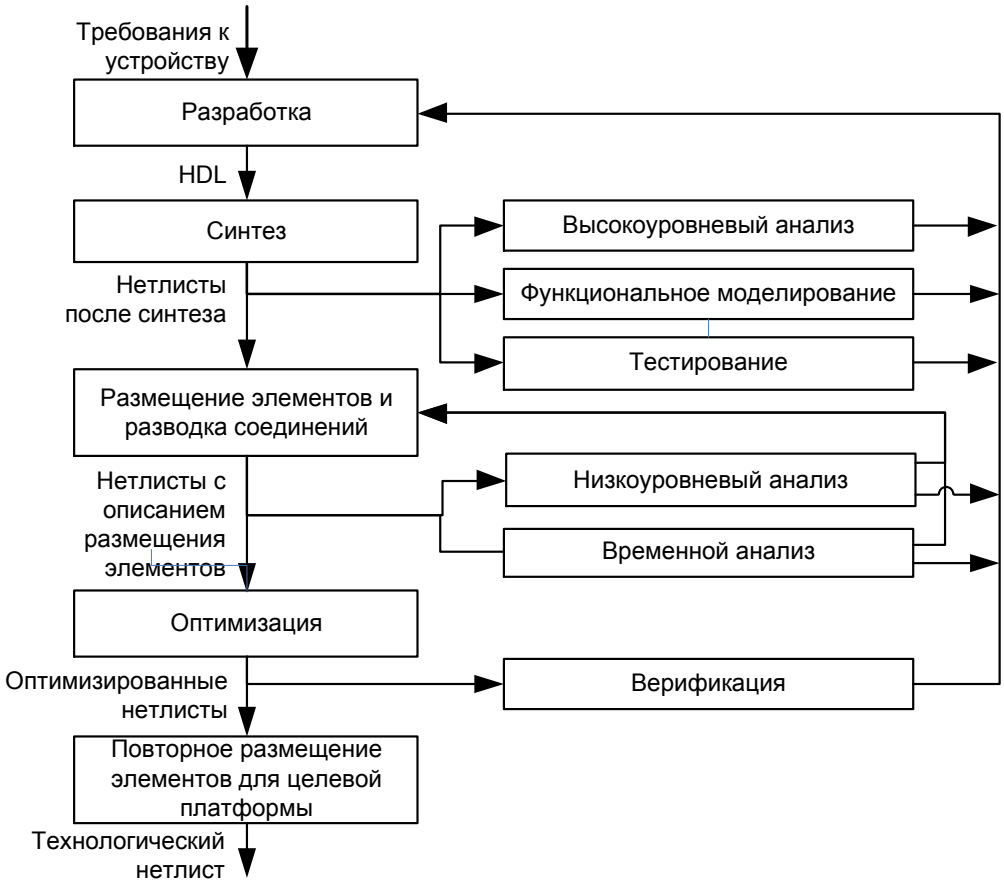


Рис. 1.11. Использование нетлистов для передачи данных между стадиями процесса разработки [72]

1.4.2. Представление устройств в инструментариях реинжиниринга

Способы представления устройств в инструментариях реинжиниринга можно разбить на несколько групп. Ниже приведены описания каждой из них.

Классические представления для ручной обработки

К данной группе представлений устройства относятся текстовые описания, блок-схемы, принципиальные схемы, диаграммы соединений и прочие текстографические описания. Подобные способы представления применяются при низкоуровневом проектировании системы, разработке спецификаций и технических заданий. Они хороши для ручной обработки инженером и используются при документировании устройства. Для машинной обработки их применение затруднено из-за низкой степени формализации, что приводит к

большому числу неточностей в описаниях. Однако, существуют средства, которые позволяют конвертировать описания в синтезируемый код.

Описание на HDL

Если устройство уже специфицировано на каком-либо языке из семейства HDL, то можно использовать данное описание и в задачах реинжиниринга. Данный подход в первом приближении кажется наиболее простым, так как не требует преобразования одного представления устройства в другое.

Использование исходных кодов для машинной обработки затруднено наличием в них незначительных для задачи реинжиниринга составляющих: пользовательских комментариев, форматирования текста, синтаксиса объявления операндов. Поэтому, на практике первым этапом обработки является импорт исходных кодов и формирование некоторого абстрактного представления.

Представление исходных описаний в виде абстрактных графов

Машинная обработка исходных кодов (в том числе и HDL) редко ведётся напрямую. Обычно строится некое дерево, которое в той или иной мере отображает исходное содержимое. Можно выделить следующие виды деревьев:

- абстрактное синтаксическое дерево (AST);
- абстрактный семантический граф (ASG);
- объектный (архитектурный) граф.

Абстрактное синтаксическое дерево (АСД) – это ориентированный древовидный граф, вершины которого сопоставлены с элементами исходного описания (кода) [14]. Представление в виде АСД позволяет абстрагироваться от составляющих исходного описания, не используемых при реинжиниринге. На рис. 1.12 приведён пример АСД-дерева VHDL для объявления сущности (Entity) модуля с переменным числом входных сигналов.

Синтаксическое дерево достаточно громоздко и не содержит связи между элементами. Например, при использовании переменной в коде она объявляется в одном месте, а используется в другом. В случае модификации данного

сигнала в представлении АСД требуется пройти по всему дереву, найти все ссылки на данный сигнал и модифицировать их. Поэтому, представление в АСД не подходит для использования в инструментариях реинжиниринга.

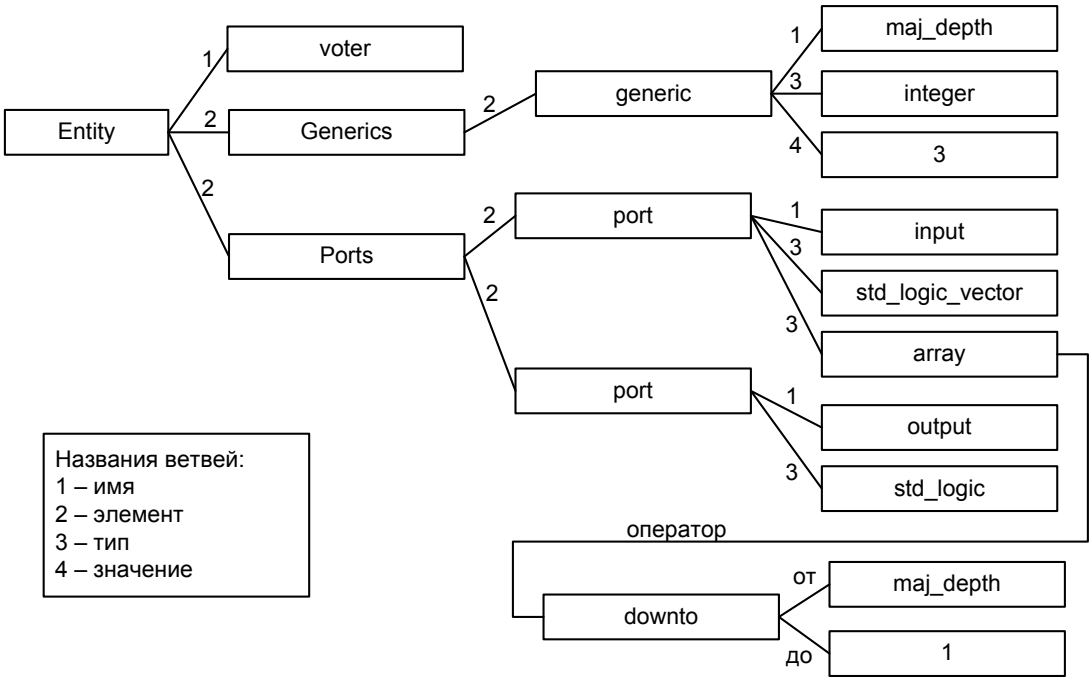


Рис. 1.12. Пример абстрактного синтаксического дерева для VHDL

Абстрактный семантический граф (АСГ) – это расширенное и аннотированное представление синтаксического дерева описания, рассмотренного выше. При формировании такого графа для устройства из вершин могут быть добавлены дополнительная информация о связях между элементами, ссылки между связными элементами или удалены неинформативные вершины (оптимизация).

Объектный (архитектурный) граф

В программной реализации СПР представление АСГ можно расширить, введя вместо узлов графа полноценные объекты со своими свойствами, методами и иерархией наследования. Подобный подход называется объектным [59] или архитектурным [63] графом.

Примером использования объектного графа является методика представления AIRE/CE (Advanced Intermediate Representation with Extensibility – продвинутое промежуточное представление с возможностью расширения),

разработанная компанией FTL Systems и университетом Цинцинатти (см. [73]). Данное представление ориентировано на представление языков VHDL и VHDL-AMS и программную реализацию на языке C++. Расширяемость представления достигается за счёт средств языка реализации: наследования и виртуальных функций. Данная методика представления ориентирована на построение средств анализа исходных кодов, никаких дополнительных решений по трансформации архитектуры авторами не предлагается.

Структурное представление устройства

Предыдущие методы применимы для любого описания. Существуют также специфические подходы, в частности: структурное и поведенческое описание, которые будут рассмотрены далее. Структурное описание является наиболее распространённым подходом в языках описания устройств, а также используется в нетлистах, которые используются для обмена данными в фазах синтеза и оптимизации устройств при проектировании.

В наиболее простых структурных описаниях на уровне логических вентилей обычно присутствуют иерархическое описание модулей и связей между ними. В то же время существующие форматы описания (например, нетлисты EDIF), используемые для передачи между различными средами, могут включать элементы функционального описания – операторы, функции и т.п. Например, формат VHDL-нетлистов поддерживает группированные объекты и присвоения по группам (шинам) [82]. Поэтому, при импорте представления из HDL требуется произвести преобразование всех элементов в некоторое структурное подобие.

Поведенческое описание устройства

Поведенческое описание устройств является более новым направлением, чем структурное. Его поддерживают далеко не все HDL, поэтому при использовании подобных представлений в средстве реинжиниринга потребуется обеспечить конвертацию структурных описаний в поведенческие (и наоборот). Данная задача относится к переносу архитектуры между

представлениями. Для её решения существуют специальные методики, схожие с процессом синтеза устройства из поведенческих описаний [42].

Поведенческие описания наиболее распространены в работах, посвящённых динамическому анализу и верификации, в которых необходимо моделировать поведение устройства. В случае с трансформацией архитектуры поведенческое описание является второстепенным, но существует ряд методик по рефакторингу на основе данного представления.

В современных инструментариях наиболее часто используют поведенческие представления устройств на основе конечных автоматов, что удобно для временного анализа. Такой подход позволяет также абстрагироваться от исходных RTL-описаний устройства, что позволяет повторно использовать методики для широкого спектра проектов [65]. Основная область применения подобных подходов – временной анализ устройств, оптимизация производительности и генерация тестовых векторов и программ.

Одним из наиболее проработанных академических подходов является представление, разработанное в университете Цинциннати [74]. В настоящее время оно используется в нескольких средствах анализа (SAVANT, TyVIS и пр.). Подход заключается в том, что всё описание укладывается в четыре группы: сигналы, иерархии портов, присвоения сигналов и операторы временных зависимостей. Авторами были полностью формализованы методики построения и трансформации архитектуры. Предложенный ими инструментарий не включает методик для структурных преобразований, и, ввиду недостатка элементов структурного описания, применение модели для задач встраивания СТ затруднительно.

Смешанное описание

Использование смешанного структурно-поведенческого описания является наиболее сложным подходом, так как одну и ту же конструкцию можно описать различными способами. В то же время требуется некоторым образом связать

между собой два принципиально разных описания, чтобы обеспечить возможность их совместной обработки.

По итогам обзора существующих СПР не выявлено ни одного инструмента, который одновременно поддерживал бы оба представления. В большинстве случаев средствами низкоуровневого и временного анализа используются не исходные коды на HDL, а более простые структурные описания из нетлистов. Построение такого СПР является актуальной задачей, так как это позволило бы восстанавливать не только структуру устройства, но и его поведенческое описание. Обзор существующих средств поддержки реинжиниринга приведён в приложении В.

1.4.3. Методики преобразований посредством СПР

Методики трансформации устройства тесно связаны с используемыми представлениями и определяют алгоритмы их преобразования. Можно выделить следующие группы подходов:

- использование фиксированного набора алгоритмов;
- трансформация по набору правил;
- программное управление трансформацией;
- управление трансформацией из внешнего инструментария.

Специализированные алгоритмы обработки используются в узкоспециализированных средствах, решающих лишь некоторые, заранее определённые задачи. Данные алгоритмы определяются внутри СПР, и интерфейсы для программирования не требуются.

По аналогии с предыдущим подходом, при трансформации по набору правил в СПР реализуются фиксированные алгоритмы обработки. Однако, данный подход предлагает использовать некоторые входные правила, которые адаптируют поведение алгоритма под конкретную задачу. В качестве примера рассмотрим переименование групп элементов в устройстве. Можно реализовать универсальный алгоритм, который принимает на вход список элементов для преобразования и последовательно осуществляет переименование для каждого

элемента. В этом случае список элементов можно рассматривать как набор правил преобразования. В приложении В рассмотрены примеры СПР, относящиеся к данной группе.

Программное управление трансформацией

При программируемой трансформации разработчику предоставляется возможность самостоятельно реализовать алгоритм обработки данных в виде некоторой программы. Далее посредством компиляции программа превращается в последовательность операций над моделью или же интерпретируется в процессе обработки. Язык реализации алгоритма в работе предлагается называть языком описания алгоритмов реинжиниринга. В общем случае язык является предметно-ориентированным и включает некоторый базовый набор операций над устройством и набор инструкций для объединения операций в единый алгоритм: условные переходы, циклы, точки останова и т.д..

Актуальна задача применения универсальных языков программирования с целью описания алгоритмов преобразований (C/C++, Java, Python, …), которые обеспечивают алгоритмическую полноту языка и повышают его универсальность [33]. Программа управления преобразованиями также может взаимодействовать с внешними средствами, если в САР для этого предусмотрен специальный интерфейс прикладного программирования.

Интеграция средства в существующие инструментарии

Данный подход обратен предыдущему. Вместо разработки нового языка управления преобразованиями можно включить вызов операций преобразования в существующий ИР или САПР (например, в виде дополнительного вызова) или язык программирования.

Таким образом, пользователь инструментария сразу получает полный доступ к внешней функциональности системы через интерфейсы прикладного программирования (API – Application Programming Interface), но при этом затрудняется доступ к внутреннему представлению. Поэтому, данный подход удобно комбинировать с предыдущим: низкоуровневую обработку можно

реализовать внутри специальных программ средства, а высокоуровневую обработку оставить на внешние компоненты системы. На рис. 1.13 приведена структура подобного САР.

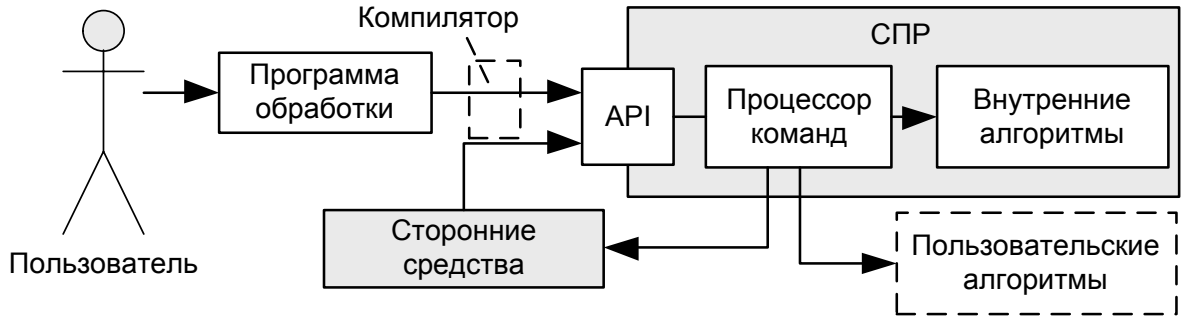


Рис. 1.13. СПР с различными источниками алгоритмов реинжиниринга

1.5. Модели устройств в задачах встраивания средств тестирования

В данном параграфе сформированы требования к моделям устройств в САР и произведен анализ существующих подходов к их построению. Сделан вывод о необходимости разработки новой многоуровневой модели, которая бы совмещала преимущества существующих подходов и была бы пригодна для обработки в алгоритмах реинжиниринга. По итогам исследования во второй главе разработана новая гибридная модель устройств, относящаяся к классу многоуровневых моделей.

1.5.1. Постановка требований к модели

Модель устройств должна поддерживать все стадии реинжиниринга и быть достаточно универсальной, чтобы ее можно было применить к различным задачам и исходным описаниям. На требования к модели также оказывают влияние требования к САР и необходимость интеграции в существующие маршруты проектирования. Сформулированы следующие общие требования к моделям устройств в задачах реинжиниринга:

- поддержка структурного описания устройств;
- использование архитектурного графа;
- ограниченный набор типов базовых элементов;
- параметризация элементов;

– наследование свойств элементов через механизм наследования;
– поддержка одновременной работы с несколькими устройствами;
– близость к одному из языков описания устройств (HDL).

Поддержка структурного описания устройств

С точки зрения реинжиниринга, структурное представление наиболее удобно для преобразования архитектуры устройств, так как оно отражает иерархию компонентов и связей между ними. Структурные описания поддерживают наиболее известные HDL, в том числе и низкоуровневые форматы нетлистов. Последнее может быть очень полезно в рамках оптимизации уже готового устройства. Очевидно, что универсальное СПР должно поддерживать структурное представление.

В HDL не вся информация используется при синтезе устройств. Например, в VHDL к несинтезируемому подмножеству относятся комментарии, временные задержки и пр. Подобная информация может быть использована в САР для задач представления данных в режиме СППР, рефакторинга описаний, автоматической генерации тестов или при анализе временных задержек в устройстве. Тем не менее, для задач встраивания СТ несинтезируемое подмножество не является необходимым, поэтому от его явной поддержки в составе модели можно отказаться.

Использование архитектурного графа

Внутреннее представление, прежде всего, должно быть ориентировано на формализованную обработку в реализациях алгоритмов реинжиниринга. Должен существовать механизм для получения информации об архитектуре устройства, составных элементах и связях между ними. Из рассмотренных ранее подходов наиболее удачным является использование архитектурного графа. Данный подход изначально ориентирован на программную реализацию в виде связанного набора объектов, каждый из которых определяет сущность устройства. В рамках САР данный подход удобен как для анализа, так и для трансформации устройства.

Близость к одному из языков описания устройств

Внутренняя модель устройства должна поддерживать структурное описание устройства, которое может быть использовано в большинстве современных HDL. Для снижения сложности использования СПР целесообразно во внутренней модели использовать семантику дерева, близкую к одному из популярных языков (например, VHDL или Verilog). Это позволит упростить анализ внутреннего представления и его сравнение с исходными кодами на HDL. Также за основу можно взять подмножество некоторого языка, используемое для описания низкоуровневых нетлистов.

Ограниченный набор типов базовых элементов в модели

При использовании представления в виде объектного графа, появляются широкие возможности по заданию узлов этого дерева. Описывая структуру устройства, в модели для каждого класса элементов (сигнал, модуль и пр.) возможно создать специальный тип. Если алгоритмы реинжиниринга будут вводить в модель новые типы, то другие алгоритмы не смогут их корректно обработать. Поэтому, в модели должен быть задан фиксированный набор типов элементов, на базе которых формируется представление устройства. Дополнительная функциональность может быть реализована через расширение отдельных типов, о котором речь пойдёт далее.

Параметризация элементов

Требуется обеспечить получение необходимой информации об элементах модели, а также предусмотреть механизм навигации между ними. Для этого необходимо хранить и предоставлять информацию об имени, типе и специальных параметрах. В качестве решения предлагается для каждого элемента модели хранить набор атрибутов и метаданных, которые бы позволили достичь большей гибкости модели.

Расширяемость модели устройства

Наиболее популярные языки описания устройства VHDL и Verilog появились примерно в то же время, что и языки объектно-ориентированного

программирования (ООП). Однако, данные языки в явном виде не поддерживают наследования свойств, что было бы удобно использовать в некоторых задачах реинжиниринга, связанных с расширением функциональности через добавление новых модулей.

Иногда возникает задача замены одного аппаратного модуля другим с аналогичными интерфейсами. Возможна замена и модификация только части блоков. Например, из внутреннего модуля иерархии выводится некоторый диагностический сигнал A. Во всех уровнях ниже уровня обработчика этого сигнала требуется добавить порты и подключить сигнал, для чего в VHDL потребуется ввести новые сущности («Entity») и архитектуры к ним («Architecture») [56]. В то же время хотелось бы сохранить связь создаваемых элементов с исходными. В этом случае высокоуровневые языки описания устройств (SystemC, Verilog) предлагают механизм наследования некоторых свойств базового объекта и их сохранение при модификации [32]. В работе для данного типа наследования предлагается использовать термин «наследование архитектуры».

В языках ООП существует множество подходов к организации наследования. Наиболее простой формой наследования является расширение архитектуры, когда потомок лишь расширяет архитектуру объекта-прародителя. Как пишет Бадд, “в то время как порождение подкласса для обобщения модифицирует или расширяет существующие функциональные возможности объекта, … расширение просто добавляет новые методы к родительским, и функциональные возможности подкласса менее крепко привязаны к существующим методам родителя” [2]. Большим упрощением является использование лишь одного класса-прародителя. Пример описанного механизма наследования приведён на рис. 1.14.

Если взять структурное подмножество HDL, то можно предложить следующие варианты наследования через расширение:

– расширение интерфейса (добавление новых сигналов);

– расширение структуры (добавление новых модулей);
– добавление новых параметров;
– добавление новых входных и выходных сигналов.

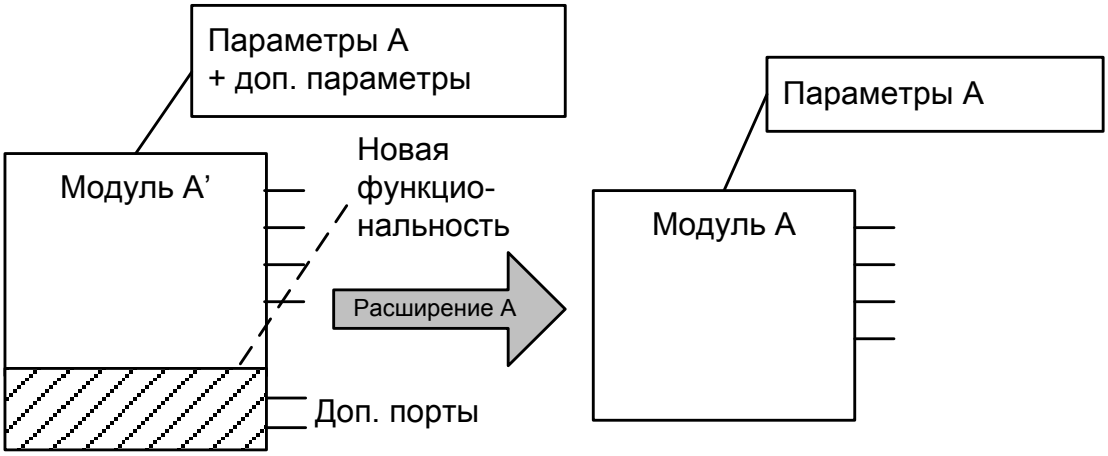


Рис. 1.14. Пример наследования архитектуры элементами через расширение

Поддержка одновременной работы с несколькими устройствами

Возможен случай, когда пользователю требуется вести параллельную обработку сразу нескольких устройств. Например, такой вариант может быть удобен при копировании элементов из одного устройства в другое или их объединении. Корневым элементом внутреннего представления должно быть не устройство, а некоторый синтетический объект. В этом случае на верхний уровень можно поместить пользовательские библиотеки, которые также необходимо отображать в представлении. Удобно было бы предусмотреть случай, когда для одного устройства другие в иерархии рассматриваются как пользовательские библиотеки. Схема иерархии приведена на рис. 1.15.

1.5.2. Подходы к построению моделей устройств

Существует множество примеров решения частных задач реинжиниринга и встраивания СТ. Выделяют следующие основные подходы к построению моделей устройств в САР:

– использование низкоуровневых структурных представлений (нетлисты);
– использование высокоуровневых представлений (языки моделирования);
– привязка к исходным описаниям на HDL.

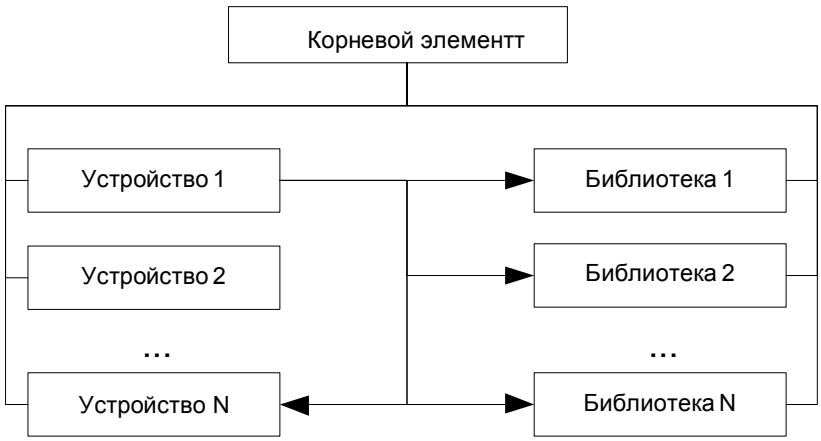


Рис. 1.15. Иерархия элементов верхнего уровня

Низкоуровневые структурные представления

Низкоуровневые структурные представления хранят необходимый минимум информации об устройствах, который позволяет воспроизвести их структуру для средств синтеза. При этом теряется часть информация об исходной архитектуре и связях между элементами, но упрощается анализ используемых системных ресурсов и временных характеристик. Такие подходы также используются для задач оптимизации устройств и простых структурных преобразований.

Примером подхода является модель Вилси для реинжиниринга устройств на VHDL [74]. Модель Вилси содержит минимальный набор структурных элементов (блок, сигнал, соединение), а также включает несколько типов элементов для использования модели с высокоуровневыми средствами представлениями. В конечном итоге модель была развита в “аннотированный” вариант VHDL, названный SUAVE [18]. Одной из причин частичного отказа от исходной модели является сложность её программной обработки из-за отсутствия более сложных типов элементов.

Высокоуровневые представления

Высокоуровневые представления должны хранить максимально полную информацию об устройстве и его архитектуре, для их реализации часто используют HDL с дополнительными аннотациями или подмножества UML (Unified Modeling Language). Преимуществами подхода являются его высокая

переносимость и возможность повторного применения алгоритмов при реинжиниринге систем различных типов. Примерами высокоуровневых моделей являются UML-RT (UML real-time, UML для систем реального времени) и MARTE, применяемые в задачах синтеза и моделирования устройств [71]. Для UML-RT Бурместером рассмотрены возможности применения описания при инкрементальном проектировании и последующей формальной верификации устройств, что соответствует рассматриваемому в работе циклу реинжиниринга [25]. Основным недостатком высокоуровневых описаний на базе UML является сложность их формализации, что затрудняет их обработку в инструментариях. В задачах реинжиниринга и контроля качества программных средств происходит постепенный отказ от подобных описаний [52]. Можно выделить следующие ограничения:

1. Модели устройств и СнК включают большое число компонентов, из-за чего задание всех отношений в универсальной модели приводит к росту её сложности, что влияет на алгоритмы обработки модели.
2. Сложность формализации UML-подобных спецификаций.
3. Отсутствие полнофункционального механизма дискретных событий, что делает невозможным моделирование устройств на UML [46].
4. Для языка не решены проблемы стандартизации, что затрудняет его использование в промышленных проектах с использованием множества форматов описаний и целевых платформ [70].

В таблице 1.2 перечислены основные преимущества и недостатки подходов, основанных на использовании высокоуровневых описаний и структурных нетлистов.

Привязка к исходным описаниям на HDL

Подобный подход применяется в задачах рефакторинга устройств, которые являются частным случаем реинжиниринга. Для рефакторинга HDL-описаний существуют инструментарии, такие как средство DMS SRT, позволяющее описывать правила преобразования описаний устройств произвольных

форматов. Для конвертации внешнего представления в модель средство использует набор правил преобразования, уникальный для каждого HDL [21]. В общем случае подход неприменим, так как объектом реинжиниринга являются исходные коды, а не архитектура устройства.

Таблица 1.2

Преимущества и недостатки при использовании внешних представлений

Формат	Способ внешнего описания устройства	
	HDL	Нетлисты
вход-ной	+ различные описания; + наличие пользовательских комментариев и меток; - сложность конвертации во внутреннее представление; - сложность интерпретации конструкций языка (макросы, условная генерация);	+ простота обработки; + независимость от внешнего представления; - привязка к средствам синтеза; - привязка к аппаратной платформе;
выход-ной	+ произвольная сложность генерации; + универсальность;	+ простота генерации; - жёсткая привязка к средствам синтеза и целевой платформе;

В таблице 1.3 приведено сравнение двух типовых задач реинжиниринга устройств. Для них требуется различная информация об устройствах из различных уровней представления, что затрудняет применение описанных подходов в чистом виде. Актуальна задача разработки новых подходов, совмещающих преимущества уже имеющихся решений.

1.5.3. Обоснование выбора новой модели

Для решения описанной выше проблемы предлагается использовать многоуровневые модели, которые одновременно включали бы информацию как об архитектуре устройства, так и его низкоуровневой реализации. Модели данного класса моделей называется многоуровневыми. Они широко распространены в задачах анализа и модификации программных систем. Франк

указывает, что отличительной особенностью адаптируемых (расширяемых) многоуровневых модели позволяют снизить затраты на проектирование систем за счёт адаптации модели к решаемым разработчиками задачам [34]. Баторий в работе по проектированию программного обеспечения указывает практическую значимость и эффективность применения многоуровневых моделей для задач реинжиниринга программного обеспечения [20].

Таблица 1.3

Особенности построения моделей для частных задач реинжиниринга

Оптимизация структуры устройства	Встраивание средств модульного тестирования
- Нетлисты дают достаточную информацию о структуре устройства - Необходимо учитывать требования к используемым системным ресурсам и временным характеристикам	- Для анализа необходима информация об архитектуре устройства - Встраивание средств тестирования производится на уровне нетлистов
- Один уровень описания - Большое число специальных метрик - Многократное выполнение анализа и трансформации	- Используется два уровня описания - Не требуются дополнительные данные о физической реализации - Однократный анализ, многократная трансформация

Многоуровневые модели позволяют одновременно работать с несколькими уровнями описания, при этом результат преобразования может быть выведен на требуемом уровне представления (рис. 1.16). Преимуществом подхода заключается в том, что все алгоритмы реинжиниринга абстрагируются от исходных форматов описаний и впоследствии могут быть применены для любых входных представлений. В следующей главе предложена новая гибридная модель, относящаяся к классу многоуровневых моделей и удовлетворяющая поставленным в подпараграфе 1.5.1 требованиям.

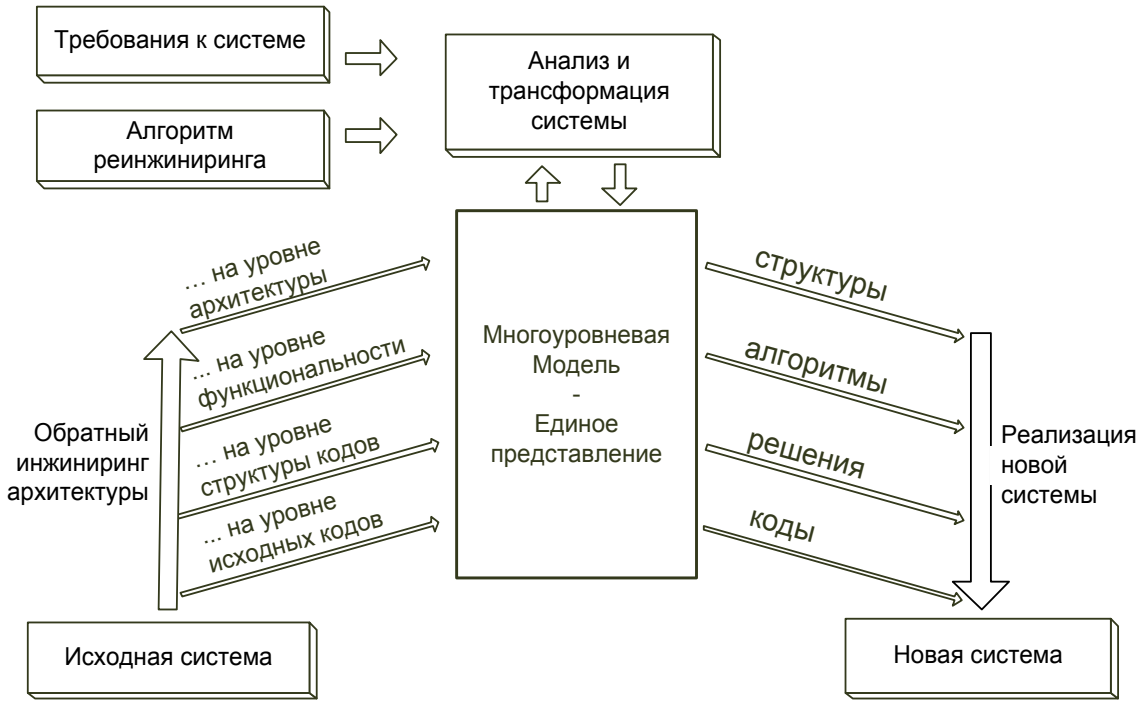


Рис. 1.16. Реинжиниринг систем с использованием многоуровневой модели

Поскольку универсальные модели не могут учесть специфику всех частных задач реинжиниринга, предлагается использовать некоторую базовую метамодель, на основании которых могут быть построены непосредственно модели для отдельных классов задач реинжиниринга (например, встраивание средств тестирования). При этом могут быть разработаны методики для работы с метамоделью, переносимые на производные модели. Процесс построения подобных моделей предлагается называть специализацией. Вводится следующее определение:

Специализация модели – построение производной многоуровневой модели для отдельного класса задач реинжиниринга, которое выполняется посредством детализации структуры исходной метамодели, расширения её типов и введения ограничений.

1.6. Постановка задач на исследование

По итогам обзора и анализа в первой главе сформированы основные задачи для дальнейшего исследования в области встраивания средств тестирования. Решено уделить наибольшее внимание инструментарию: модели

устройства и методикам её обработки. Сформированы следующие задачи исследования:

1. Разработка новой модели устройств, предназначенной для программной обработки в задачах реинжиниринга цифровых устройств.
2. Разработка базового набора операций над моделью, на базе которого будут строиться алгоритмы реинжиниринга.
3. Разработка методик импорта/экспорта модели из имеющихся описаний.
4. Разработка методик проведения внутрисхемного тестирования СнК.
5. Апробация подходов на примерах частных задач реинжиниринга с фокусом на задачи проектирования и тестирования цифровых систем

Поставленные научные задачи решены в следующих главах. Во второй главе предложена новая гибридная метамодель однокристальных цифровых устройств. В третьей главе разработаны методы работы с метамоделью: специализации модели для решения частных задач реинжиниринга, импорта модели из внешних описаний, контроля при выполнении основных операций. В четвёртой главе на базе метамодели разработаны методы встраивания средств внутрисхемного тестирования в устройства. В пятой главе рассмотрены примеры решения частных задач реинжиниринга с использованием разработанных моделей, методов и инструментальных средств.

2. ГИБРИДНАЯ МЕТАМОДЕЛЬ УСТРОЙСТВ

В главе предложена новая гибридная метамодель однокристальных цифровых устройств для задач реинжиниринга. Определена область применения модели в соответствии с результатами обзора, проведённого в первой главе диссертации. Рассмотрены основные элементы гибридной метамодели и их структура. Доказана совместимость метамодели с существующими языками описания устройств. Приведены примеры описания устройств с использованием гибридной метамодели устройств.

2.1. Область применения метамодели и её ограничения

Предлагаемая в данной главе гибридная метамодель ориентирована на использование в задачах реинжиниринга цифровых устройств, рассмотренных в первой главе. Метамодель строится в условиях следующих ограничений:

1. Метамодель предназначена для структурных преобразований в цифровых устройствах, описываемых предлагаемой моделью.
2. Метамодель рассчитана на реинжиниринг однокристальных устройств, специфика распределённых архитектур в работе не рассматривается.
3. Метамодель не включает полную информацию об устройствах для частных задач реинжиниринга. Задача представления информации должна решаться при специализации модели.
4. Представляемые устройства имеют статическую структуру с постоянными связями между элементами.
5. Значимые для задачи реинжиниринга поведенческие и иные высокоуровневые описания могут быть выражены в структурной форме путем частичного синтеза.

Первые четыре ограничения сводят область применения метамодели к классу классических цифровых СнК, к которым относятся большинство проектируемых в настоящее время устройств [60]. Метамодель не рассчитана на реинжиниринг таких составных систем, как описанные в [54] интегрированные электронные и микроэлектромеханические (MEMS) системы.

Вводимые ограничения могут быть частично решены путём специализации модели, подход к которой рассмотрен в параграфе 3.1.

Пятое ограничение предполагает, что вся требуемая для анализа и трансформации архитектура может быть выражена в структурной форме. Для несинтезируемых подмножеств возможна конвертация в структурные описания с использованием информации об архитектуре устройства. Например, операторы задержки WAIT в описаниях VHDL для средств моделирования могут быть сконвертированы в реальные задержки при наличии информации о частоте тактирования целевого устройства [55].

В параграфе 5.5 рассмотрены возможности применения гибридной модели для классов задач реинжиниринга за рамками поставленных ограничений.

2.2. Структура гибридной метамодели

На основании результатов проведённого в предыдущей главе анализа разработана новая “гибридная” метамодель представления устройства, которая одновременно совмещает преимущества низкоуровневых и высокоуровневых описаний. Метамодель основана на низкоуровневом структурном представлении (“нетлисте”), дополненном высокоуровневыми элементами, выбранными путём анализа существующих HDL или требований к модели. Структурная составляющая сходна с моделью Вилси и Бенца, но не содержит элементов для описания поведенческого подмножества языка VHDL [74].

Гибридная метамодель – многоуровневая модель устройств, включающая низкоуровневое структурное и высокоуровневое уровни описания, на основе которой путём специализации может быть построена модель устройства для частной задачи реинжиниринга.

На рис. 2.1 приведена структурная схема предлагаемой модели; серым цветом отмечено структурное подмножество модели, белым – высокоуровневые элементы. Модель берёт за основу древовидную структуру без перекрёстных ссылок, за счёт чего достигается простая навигация по модели в задачах изменения её структуры. Для упрощения задач анализа

вводятся межуровневые связи элементов, которые реализуются через механизмы ссылок и наследования.

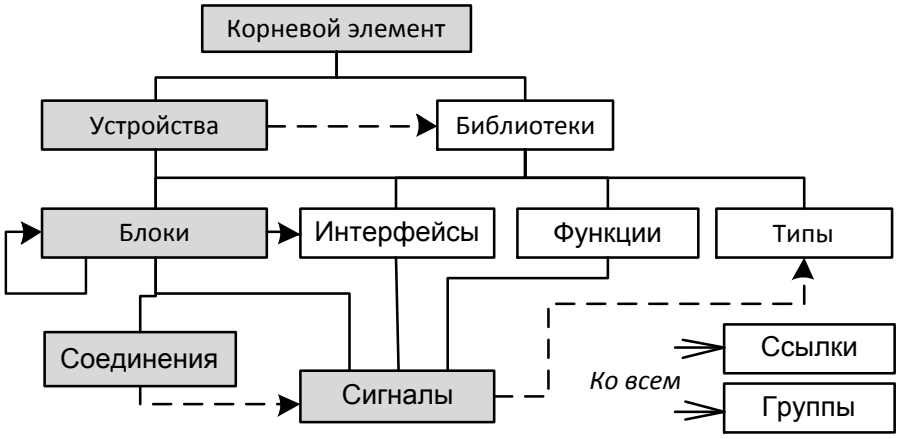


Рис. 2.1. Типы элементов гибридной метамодели

В таблице 2.1 приведены обозначения каждого типа элементов, которые далее работе используются для их обозначения.

Таблица 2.1

Символьные обозначения типов элементов в гибридной метамодели

Символ	Тип	Символ	Тип
R	Корневой элемент	I	Интерфейс
D	Устройство	F	Функция
B	Блок	L	Библиотека
C	Соединение	G	Группа
S	Сигнал	A	Агрегация
T	Тип (элемент модели)	X_L	Ссылка, X – индекс типа

Каждый элемент в модели представляет собой сложный объект, предназначенный для программной обработки. Модель поддерживает реализацию новых типов элементов через наследование свойств одного из базовых типов (см. ниже). Объекты модели описываются следующим множеством:

$$V = (A, I, P, L, C, M), \quad (2.1)$$

где:

- A – множество атрибутов (параметров) объекта, специфичных для каждого типа элементов;
- I – ссылка на наследуемый элемент (см. подпараграф 2.4.5);
- P – ссылка на родительский элемент;
- L – список внешних ссылок на элемент;
- C – массив ссылок на дочерние элементов;
- M – множество дополнительных данных, которые могут быть внесены в элемент пользовательскими алгоритмами.

Множество элементов образует гибридную модель. Для обеспечения навигации по модели также указывается корневой элемент V_{root}, относительно которого вычисляются пути внутри модели.

$$G = (V_1 \dots V_n,\ V_{root}), \text{где } n \in \mathbb{N} \tag{2.2}$$

Введём понятие принадлежности элементов к иерархии для дальнейшей записи отношений в иерархии. Элемент модели считается принадлежащим другому элементу, если ссылка на него присутствует во множестве C.

$$V_C \in V_P <=> \exists i \in \mathbb{N} : V_P.C[i] = V_C \tag{2.3}$$

Адресация к атрибутам и метаданным

В гибридной модели атрибуты A и метаданные M являются ассоциативным массивом элементов без дополнительной внутренней структуры. Они в общем случае могут содержать пустое множество элементов. Аналогичным образом определяется множество метаданных.

$$\begin{cases} a=(key,\ value) \\ A=(a_1, \dots, a_n)\ n \in \mathbb{N}_0 \\ \forall\, i,j\ \exists\ (key(a_i) == key(a_j)) \end{cases} \tag{2.4}$$

Список **атрибутов** элемента зависит от его типа и может быть расширен при наследовании элементов в пользовательских типах. Перечисленные ниже атрибуты A должны присутствовать во всех элементах модели:

$$A = \begin{Bmatrix} \text{name} => N, \text{type} => T, links => R, \\ \text{readonly} => RO, id => ID \end{Bmatrix}, \tag{2.5}$$

где:

- N – имя элемента (узла) модели;
- T – тип элемента;
- R – множество ссылок на внешний элементы;
- RO – флаг запрета модификации элемента (булево значение);
- ID – уникальный идентификатор элемента (подпараграф 2.4.2).

Для последующего анализа введём отношение равенства для множеств в гибридной модели. При этом примем, что порядок следования элементов значения не имеет, а уникальность элементов во множестве обеспечивается внешними средствами.

$$\begin{cases} A = (a_1,\ldots,a_n), n \in \mathbb{N}_0 \\ B = (a_1,\ldots,a_m), m \in \mathbb{N}_0 \\ A = B: \begin{cases} \forall i \in (1..n)\ \exists!\ j: a_i = b_j \\ n = m \end{cases} \end{cases} \quad (2.6)$$

2.3. Элементы модели

В данном параграфе рассмотрены типы элементов, которые входят в гибридную модель. Указаны назначение типов, их свойства и подходы к верификации внутренней структуры объектов данных типов. Им поставлены в соответствие элементы из различных HDL. Подробно вопросы совместимости модели с HDL и форматами нетлистов рассмотрены в параграфе 2.6.

Корневой элемент (Root)

Данный элемент используется для определения списка библиотек и устройств в рамках контекста. Он применяется в задачах навигации по модели, но не трансформируется в выходные представления, и его параметры не могут быть модифицированы командами преобразования.

Устройство (Device)

Устройство является синтетическим компонентом для выделения верхних уровней иерархии, которые могут быть экспортированы в корректные HDL-описания отдельных устройств. Данный объект может быть размещён только в

корневом элементе дерева, что позволяет одновременно работать с множеством устройств. Устройство может включать следующие элементы:

- корневой блок;
- внутреннюю библиотеку элементов модели;
- список ссылок на используемые внешние библиотеки.

Корневой блок является элементом типа Block (см. далее) и включает всю внутреннюю архитектуру устройства, что позволяет работать с ним как с обычным блоком. В корневом блоке определяются все блоки и интерфейсы устройства, загруженные при импорте устройства и не описанные во внешних библиотеках.

Интерфейсы (Interface)

Интерфейсы определяют внешнее представление блоков, устройств и функций. В интерфейсы входят описания портов и параметров (подтипы сигналов). Данный элемент схож с Entity в VHDL, но в разработанном представлении допускается множество синтезируемых реализаций интерфейса (в VHDL - одна), что позволяет использовать понятие интерфейса из объектно-ориентированного программирования. Выбранный подход позволяет заменять одни реализации модулей другими, если у них совпадают внешние интерфейсы. Также возможно наследование интерфейсов, при котором могут быть добавлены новые порты и статические параметры.

$$\begin{cases} I = ((P_1, \dots, P_n), (S_1, \dots, S_m)), n \in \mathbb{N}, m \in \mathbb{N}_0 \\ \quad P_i,\ S_i = (name, L_{S\_value}, L_{S\_default}) \end{cases} \tag{2.7}$$

Интерфейсы I_A и I_B считаются тождественно равными, если все их порты P совпадают. Полное соответствие статических параметров для эквивалентности не требуется, так как они могут быть заменены значениями по умолчанию. При совпадении динамических и статических значений в интерфейсе говорят об их равенстве.

$$I_A \equiv I_B : \begin{cases} I_A.P = I_B.P \\ I_A.S \in I_B.P \end{cases} \tag{2.8}$$

$$I_A = I_B : \begin{cases} I_A.P = I_B.P \\ I_A.S = I_B.S \end{cases} \quad (2.9)$$

Интерфейс I_A считается совместимым с другим интерфейсом I_B, если в нём присутствуют все статические параметры, входные/выходные сигналы I_B и значения по-умолчанию для всех параметров I_B. При этом для остальных входных сигналов должно быть определено значение по-умолчанию.

$$I_A \approx I_B : \begin{cases} I_B.P \in I_A.P \\ I_B.S \in I_A.S \\ \forall i\ , I_A.P_i \in !\, I_B.P : I_A.P_i.default\ != \emptyset \\ \forall i\ , I_A.S_i \in !\, I_B.P : I_A.S_i.default\ != \emptyset \end{cases} \quad (2.10)$$

Блоки (Block)

Блоки формируют иерархию устройства. Концепция блоков близка к структурным описаниям архитектур на языках VHDL и Verilog [9, 68]. Каждый блок ссылается на интерфейс, который определяет его внешние связи. Блок может включать следующие элементы:

- дочерние блоки;
- сигналы, порты, generic-параметры;
- описания соединений между элементами.

Если сравнивать с VHDL, то сам блок реализует архитектуру, а ссылки на него – объявления компонентов.

Библиотеки (Library)

Библиотеки используются для хранения элементов, многократно используемых в модели посредством ссылок или структурного наследования. Возможно построение иерархии библиотек, что может быть использовано для реализации вложенных пространств имён (VHDL поддерживает подобный механизм). Библиотека может включать произвольные элементы, требованием является уникальность имён элементов для каждого из типов.

В метаданные библиотеки рекомендуется добавлять информацию об её источнике и назначении.

Сигналы (Signal)

Сигналы в модели определяют все элементы, которым могут быть присвоены некоторые значения в рамках гибридной метамодели. К данной ним относятся элементы физических сигналов, группы, статические и динамические параметры блоков и функций. Для разделения типов введены подтипы сигналов, которые описаны в таблице 2.2.

Таблица 2.2

Типы сигналов в гибридной модели

Тип	Реализуемые элементы VHDL	Возможность соединения*			
		V	P	G	C
Variable	- сигналы в архитектуре; - внутренние переменные функций.	+	+	-	-
Port	- порты сущностей; - параметры и возвращаемые значения функций;	+	-	-	-
Generic	- generic-параметры Entity;	+	+	+	-
Constant	Константы, значения по-умолчанию	+	+	+	-
*Variable, Port, Generic и Constant соответственно. “+” ставится, если возможно соединение сигнала с типом из горизонтальной строки к типу из вертикальной					

Каждый сигнал обладает следующими параметрами: имя, тип и флаг поддержки двунаправленных соединений (BIDIR), который используется для контроля соединений между сигналами.

Типы сигналов (Type)

В большинстве современных HDL существует типизация сигналов. При этом, типы могут быть описаниями физических сигналов (std_logic, bit, …), объявлениями массивов, специфическими типами (integer, boolean), перечислениями или целыми структурами. Список поддерживаемых типов варьируется в зависимости от HDL, а в некоторых из них пользователь может

вводить свои типы. Для поддержки типизации вводится дополнительный тип элементов, который может быть размещен внутри библиотеки.

Элемент типа может быть составным, т.е. его описание может набираться из других типов и параметров (например, массивы сигналов). Данный механизм во многом приближен к синтаксическому дереву, так как при отсутствии чётко выраженных границ сложности типов невозможно сформировать структуру элемента. Примером сложного типа может быть массив с двойным диапазоном индексов: type 2range_array is array(8 downto 5, 3 downto 0) of std_logic. На рис. 2.2 приведён пример типизации для приведённого примера. В значении свойства ranges формируется иерархия объектов, которая должна контролироваться при проведении реинжиниринга.

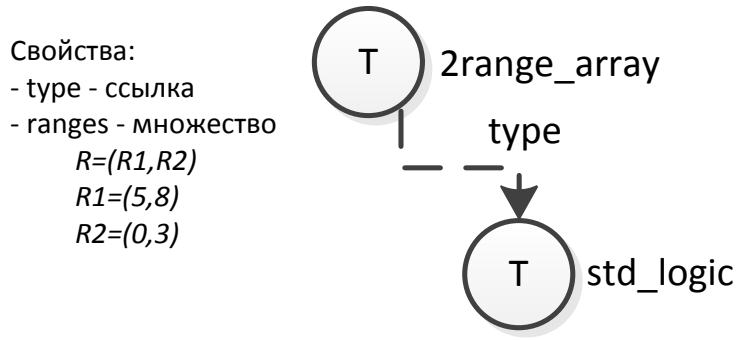


Рис. 2.2. Представление сложного типа сигналов в гибридной модели

Соединения (Connection)

Соединения используются для организации связи между сигналами, т.е. они связывают элементы иерархии устройства. Каждое соединение содержит ссылки на входной и выходной сигналы. Реализация двунаправленных соединений (например, для “BIDIR”сигналов в VHDL) возможна при помощи специального параметра.

При верификации соединений проверяется корректность направлений соединений (см. таблицу 3.2), а также соответствие типов сигналов. Двунаправленные соединения допускаются только между портами и переменными, причём требуется, чтобы в параметрах порта был установлен флаг поддержки BIDIR-сигналов.

Функции (Function)

Функции предназначены для программного описания составляющих модели, которые не могут быть выражены в структурной форме низкоуровневой части модели. С их помощью реализуются:

- несинтезируемые поведенческие описания в языке VHDL;
- логические и арифметические операции над сигналами;
- макросы в SystemC и другие специфичные для HDL возможности.

Функция обладает интерфейсом и может подключаться на место любого блока (рис. 2.3). Описание функции задаётся атрибутом “method”. Элемент функции может быть развёрнут в структурное описание при необходимости использования её внутренних данных. Данная операция называется частичным синтезом модели. Синтезированный таким образом блок с иерархией всегда обладает теми же портами, что и исходная функция. При этом статические параметры интерфейса могут быть изменены.

$$\begin{cases} B_{res}(F) = F.method(F) \\ B_{res} = (V_1 \dots V_n,\ V_{root}), \mathrm{n} \in \mathbb{N} \\ \forall F\ F.interface \equiv B_{res}(F).interface \end{cases} \quad (2.11)$$

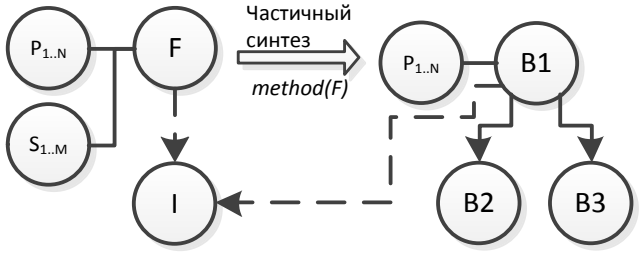


Рис. 2.3. Представление функции в модели и частичный синтез

Ссылки (Reference)

Ссылки предоставляют возможность задания семантических отношений между элементами внутри гибридной модели. Понятие ссылки используется как для типа элементов модели, так и внутри объектов модели. Ссылка *I* в объектах модели используется для задания отношений **наследования**, при которых один элемент неявно включает свойства и структуру другого элемента. Оливьером показано, что подход позволяет поддержать иерархическую семантику высокоуровневых языков (SystemC и др.) [52].

Группы и агрегации (Group, Aggregation)

В ряде случаев может потребоваться группировка элементов. Примером может быть объединение сигналов в шину или же объединение однотипных блоков (например, триггеров памяти в регистр). Поэтому, для удобства работы с деревом элементов было решено ввести тип элементов “группа”. Группа является отдельным элементом модели и хранит ссылки на элементы, в неё входящие. Аналогично типам, группы элементов могут иметь “рваные” диапазоны. Они задаются следующей формулой:

$$\begin{cases} R_i = \left(L_{Ei\ idx_{\min}}, \dots, L_{Ei\ idx_{\max}}\right) \\ G = (R_1, \dots, R_n),\ n \in \mathbb{N} \end{cases}, \quad (2.12)$$

где:

- n – число диапазонов в группе или агрегации;
- *idx_min,idx_max* – индексы элементов.

Требованием к любой группе является наличие хотя бы одного диапазона и отсутствие пересечений между диапазонами. Допускаются диапазоны, состоящие из одного элемента. В отличие от VHDL, для удобства обработки диапазоны следуют в строго возрастающей последовательности.

$$\begin{cases} \forall i \in 2..N: L_{Ei-1\ idx\_max} < L_{Ei\ idx\_min} \\ \forall i \in 1..N: L_{Ei\ \ idx\_min} \leq L_{Ei\ idx\_max} \end{cases} \quad (2.13)$$

В гибридной модели группы реализуют шины сигналов и объединяют однотипные элементы (например, триггеры могут быть объединены в некоторое подобие регистра). Это позволяет включать в группы одноимённые элементы, что не поддерживается для прочих типов. Для задания динамических диапазонов используются функции.

Агрегация является подтипом группы. Она отражает временное объединение элементов для выполнения какой-либо операции. Примером могут быть агрегации сигналов в языке VHDL. В гибридной модели агрегации применяются прежде всего для мультиплексирования сигналов при передаче их в блоки и функции с шинными интерфейсами.

2.4. Механизмы модели

В данном параграфе описан ряд свойств, введённых в модель для поддержки алгоритмов реинжиниринга устройств. Все перечисленные свойства основаны на элементах модели, описанных в предыдущем параграфе.

2.4.1. Представление модели в виде древовидного графа

Предлагаемая модель путём её расширения понятия “ссылка” может быть сведена к древовидному графу (2.14.a) в структурном представлении или к направленному ациклическому графу (орграф или DAG) - в семантическом (2.14.b). В гибридной модели DAG используется для минимизации числа элементов при выполнении алгоритмов анализа и преобразования.

$$\text{a) } G = (V, E) \quad \text{b) } G = (V, A), \tag{2.14}$$

где:

- V – непустое множество элементов модели;
- E – множество пар вершин (рёбер);
- A – множество ориентированных дуг между вершинами.

Представление модели в виде графа позволяет использовать методы теории графов при анализе и трансформации модели. На рис. 2.4 приведены примеры обоих типов графов. На схеме а) показан пример орграфа устройства со ссылкой на определение блока в библиотеке. На схеме б) данная ссылка разворачивается путём копирования элементов библиотеки в иерархию устройства. При этом формируется строго древовидная иерархия, что позволяет использовать математический аппарат для другого типа графов.

При использовании указанных выше представлений в узлах графов находятся сложные объекты модели (в соответствии с формулой 2.14). С использованием методов теории графов возможно выполнять ряд операций при анализе и трансформации устройства. Примерами таких операций являются:

- обход и поиск элементов в дереве;
- добавление, удаление, замена и перенос элементов;

– проверка корректности модели: отсутствие кольцевых ссылок, ссылок на удалённые элементы и пр.;
– оптимизация структуры устройства.

В четвёртой главе диссертации приведены примеры использования модели в виде графов для решения задач внутрисхемного тестирования СнК.

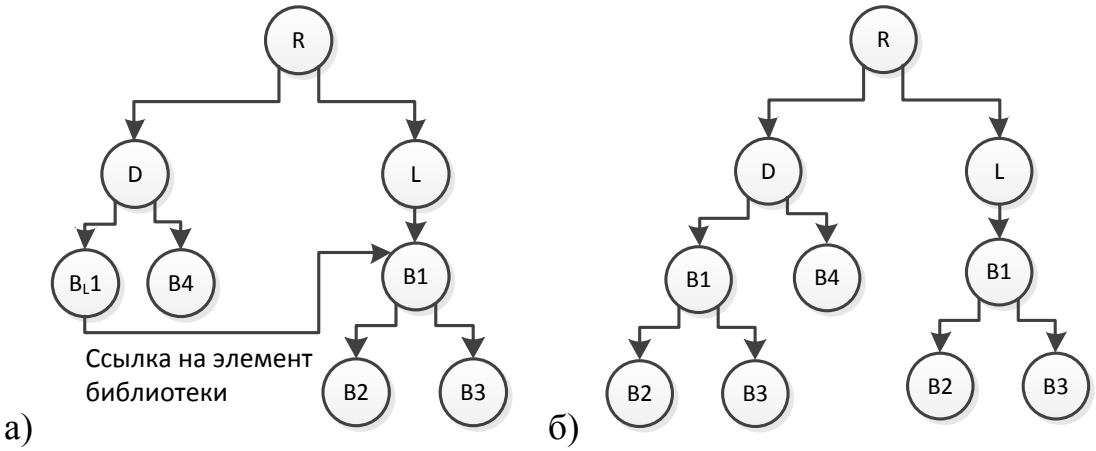


Рис. 2.4. Примеры графов для простого устройства
(а – ациклический орграф; б – развёрнутый древовидный граф)

2.4.2. Навигация по модели

В модели устройства требуется обеспечить доступ к элементам структурного описания и их параметрам. Поскольку атрибуты и метаданные определены только в рамках элемента и не имеют внешних связей, решено выделить доступ к параметру в отдельный механизм адресации, который работает в пределах элемента. В итоге в гибридной модели предусмотрено несколько механизмов адресации, которые рассмотрены ниже.

Именование элементов гибридной модели

В гибридной модели элементы обладают именем, типом и индексом положения в группе (если он включен в группу). Триада данных параметров уникальна в пределах объекты одного уровня иерархии и может быть использована в качестве уникального ключа, который позволяет произвести доступ к элементу.

$$T_i = (ID_{type}, Name, Index) \quad (2.15)$$

В текстовом виде при описании алгоритмов реинжиниринга ключ предлагается записывать следующим образом:

T_i=[идентификатор_типа]имя_элемента[индекс] (2.16)

Для текстовой записи идентификаторов типов элементов выбраны уникальные односимвольные маркеры, перечень которых приведён в таблице 2.3. При записи имени допускается ряд исключений:

- если ячейка не входит в группу, то индекс можно не указывать;
- если идентификатор не указан, то ячейка считается блоком;
- если после имени ячейки следует двоеточие, то ячейка считается устройством (поддержка абсолютной адресации внутри устройства).

В имени ячейки запрещено использование пробелов и специальных символов, которые перечислены в таблице 2.3.

Адресация к элементам гибридной модели

Поддерживается три механизма адресации к элементам модели:

- прямая адресация по абсолютному пути к модели;
- косвенная адресация через путь относительно другого элемента;
- прямая адресация через уникальный идентификатор элемента.

При прямой адресации путь записывается относительно корневого элемента или устройства. В случае косвенной адресации в начале пути ставится метка ‘.’ или ‘..’. Расчёт пути при этом начинается от элемента, который в настоящее время выбран в контексте. Путь к элементу P записывается как кортеж ключей T_i. Путь всегда должен содержать не менее одного элемента.

$$P = (T_1, \dots, T_n),\ n \in \mathbb{N} \quad (2.17)$$

В случае косвенной адресации проверка типов для данных элементов может быть произведена лишь динамически во время вычисления пути, так как маркеры косвенной адресации могут не указывать требуемые типы. При попытке перехода выше корневого элемента метки пути игнорируются.

Таблица 2.3

Специальные метки при навигации по дереву элементов

Модуль	Описание
.	Текущий элемент
..	Родительский элемент
:	Устройство
[x]	Индекс группы, если указан в конце имени. Тип элемента, если указан в начале имени
/ , \	Разделители уровней. Если он указан в начале пути, то адресация считается абсолютной
~	Начало модификаторов имени*
_	Разделитель модификаторов имени*
@	Символ доступа по идентификатору
* - Допускается использование символов в имени устройства	

Адресация по идентификаторам элементов

Каждому элементу в модели присваивается уникальный идентификатор *ID*, по которому к ней можно обратиться. При перемещении элемента внутри модели его индекс сохраняется, а при удалении – не занимается индексами других ячеек. В этом случае индексы можно хранить отдельно в метаданных элементов, например, для отложенной обработки или ускорения доступа. Размерность идентификатора должна быть достаточной для присвоения идентификаторов каждому элементу модели.

Адресация к метаданным и атрибутам

Адресация к метаданным выполняется относительно их родительского элемента, при этом искомая запись метаданных определяется по имени поля, которое должно быть уникальным в пределах элемента. Поскольку метаданные и атрибуты могут быть сложными объектами, то поддерживается адресация к ним с использованием пути, состоящего из нескольких элементов.

Примеры записи пути приведены в таблице 2.4.

Таблица 2.4
Примеры адресации к элементам в модели

Тип адресации	Примеры
Абсолютная адресация	\[I]basic\[I]not\[S]in\ test:\clk~clkctrl\[S]outclk\ [D]test\[B]clk~clkctrl\[S]outclk\
Косвенная адресация	. .. .\ clk~clkctrl\[S]outclk\ ..\..\ [S]outclk\ ..\clk~clkctrl\..\ clk~clkctrl..\clk~clkctrl\[S]outclk
Адресация по идентификатору	@11235813 @42
Адресация к параметрам и атрибутам	Name ports/clk/type

2.4.3. Механизм ссылок

При реинжиниринге в модели может быть несколько блоков, обладающих одинаковой структурой и конфигурацией (регистры, логические вентили, и т.д.). С помощью механизма ссылок можно хранить только одну копию подобных объектов, а для всех остальных указывать, что они являются подобием другого объекта в дереве. При помощи него предлагается реализовать все отношения вне древовидной иерархии устройства, реализуя горизонтальные связи архитектурного графа.

Ссылка полностью повторяет описание элемента, на который ссылается (в том числе принимает его тип). Ссылки становятся прозрачными для внутреннего представления и команд трансформации устройства, что удобно при считывании и анализе, но требует дополнительного внимания. Можно выделить два способа использования ссылок:

– создание связанной копии объекта;

– указание на использование другого элемента в реализации.

В базовом наборе операций должна быть предусмотрена команда замены ссылки на реализацию из внешнего модуля. Это может быть использовано при последующей модификации одного из изначально одинаковых элементов без модификации других. В противном случае, при модификации элемента по ссылке произойдут изменения и в ссылающихся на него элементах.

Ссылки также используются для указания зависимостей между элементами модели. Например, через ссылку указывается тип сигнала для элемента сигнала. В таблице 2.5 приведены некоторые примеры использования ссылок в иерархии устройства.

Таблица 2.5
Использование ссылок в элементах гибридной модели

Тип элементов	Примеры использования
Устройство	ссылки на используемые библиотеки
Блок	ссылка на реализуемый интерфейс; ссылка на блок, от которого наследуется реализация.
Интерфейс	указание на расширяемый интерфейс
Библиотека	ссылки на используемые библиотеки
Сигнал	указание типов сигнала
Соединение	ссылки на входной и выходной сигналы
Функция	ссылка на реализуемый интерфейс
Тип	ссылки на используемые типы нижнего уровня
Группа	ссылки на содержимое элементы

2.4.4. Метаданные

При реализации алгоритмов реинжиниринга встаёт задача передачи дополнительной информации между шагами алгоритма. Одним из типовых подходов является хранение данных в модели путём внесения дополнительных сущностей в элементы модели. В классе адаптивных объектных моделей такие сущности называют метаданными [75].

В гибридной модели к каждому объекту может быть добавлено множество метаданных $M_{1..N}$, где каждый элемент описывается следующим образом:

$$\begin{cases} M_i = (S, Data) \\ M = (M_1, \dots, M_n), n \in \mathbb{N}_0 \end{cases}, \quad (2.18)$$

где

- S – сигнатура записи;
- Data – объект данных.

Сигнатура записи S используется для поиска требуемой записи в метаданных. Обеспечение уникальности сигнатур и операции с хранимыми данными являются задачами алгоритма реинжиниринга.

2.4.5. Структурное наследование

В задачах встраивания средств тестирования часто требуется создать копию блока с рядом модификаций, не влияющих на существующую функциональность. Примерами являются встраивание диагностических модулей и транзит диагностических сигналов через уровни иерархии. В VHDL и Verilog для введения подобных сигналов потребовалось бы создание новых сущностей и переопределение компонентов, что привело бы к проблеме синхронизации архитектур при обновлении одного из блоков в процессе реинжиниринга. SystemC и подобные высокоуровневые HDL включают средства наследования [28].

В ограниченной форме механизм наследования поддержан в гибридной модели устройств. Он реализуется при помощи ссылки на наследуемый элемент V_P. Предлагается следующее реализация:

- элементы всех типов наследуют атрибуты и метаданные родительского элемента, дочерние объекты могут включать дополнительные записи;
- для элементов с интерфейсом *I* наследуется ссылка на интерфейс;
- для блоков наследуется внутренняя реализация, в дочерний блок могут быть подключены новые элементы и соединения;

– интерфейсы при наследовании от другого интерфейса наследуют объявления всех статических и динамических параметров, значения по-умолчанию могут быть переопределены.

2.4.6. Точки расширения гибридной метамодели

Для построения специализированных моделей устройств (см. подпараграф 2.4.6) требуется дополнение структуры гибридной метамодели и хранение в ней дополнительной информации о характеристиках устройства и отдельных элементов. Для этого в метамодель вносятся точки расширения (англ. extension points), которые могут быть использованы при дальнейшей специализации модели (см. параграф 3.1). Точки расширения наряду с формализованной базовой структурой модели обеспечивают возможность описания алгоритмов для широкого класса задач на базе одной базовой метамодели [19].

В гибридной модели к точкам расширения можно отнести следующие структурные компоненты и свойства:

- метаданные элементов модели, которые могут хранить дополнительную информацию, в том числе в виде сложных объектов;
- новые типы элементов, которые могут быть добавлены в модель через структурное наследование от базовых типов, включая универсальные элементы функций, ограниченных только интерфейсом взаимодействия;
- библиотеки, которые позволяют включить в производную модель шаблонные компоненты;
- произвольная типизация сигналов, которая позволяет разделить реализации компонентов для сигналов с различными свойствами.

При использовании описанных выше расширений расширяются требования к верификации модели при её построении и реинжиниринге. Задача верификации при этом должна решаться алгоритмами реинжиниринга с использованием средств, предоставляемых моделью и описанным в следующем параграфе набором базовых операций, который также включает функции обратного вызова.

2.5. Язык описания алгоритмов реинжиниринга

Для предложенной модели устройства необходимо сформировать некоторый набор операций. Совокупность данного набора операций и универсального языка программирования предлагается называть языком описания алгоритмов реинжиниринга. С помощью данного языка предлагается строить все алгоритмы анализа и трансформации модели, поэтому актуальна задача обеспечения его функциональной полноты и удобства применения для решения частных задач реинжиниринга устройств.

Критерии функциональной полноты языка

Поскольку предлагаемая модель устройства предназначена для программной обработки в CAP, необходимо сформировать функционально полный набор операций над моделью, который может быть впоследствии использован для формирования языка преобразований. Сформированы следующие условия функциональной полноты набора операций над моделью:

1. Возможно преобразование произвольной целостной модели в любую другую за конечное число операций.
2. За конечное число операций возможен доступ к любому элементу, входящему в модель устройства.
3. Для заданного объекта модели возможно получение и изменение любой информации за конечное число операций.
4. Функциональная полнота алгоритмической части языка.

Для упрощения доказательства откажемся от требования к наличию специальных операций для изменения информации для заданного объекта. Данную операцию можно реализовать путём создания объекта и последующей замены исходного элемента (или дерева элементов) на новую реализацию.

В ациклических орграфах отсутствуют обратные связи и циклы, поэтому возможно полностью перестроить ветвь графа, не используя операции изменения узлов [12]. Это означает, что модифицированную структуру устройства можно создать заново, добавив в неё модифицированный элемент.

После этого требуется удалить исходное устройство. Таким образом, для сокращения требований достаточно наличия операций создания новых элементов, а также вставки и удаления устройств для корневого блока V_{root}.

Запишем любую операцию над моделью как *f(x)*. Для указания типов будем использовать массив Types, который включает поддерживаемые модели типов. Первые три условия функциональной полноты языка с дополнительными условиями вставок можно записать следующим образом:

$$\begin{cases} \forall V_{inil}, V_{res} \;\; \exists f_{1..N}, n \in N : V_{res} = f_n\left(f_{n-1}\left(\dots f_1(V_{init})\right)\right) \\ \forall V \in G(V_{1..m}, V_{root}) \;\; \exists f_{1..N}, n \in N : V = f_n\left(f_{n-1}\left(\dots f_1(V_{root})\right)\right) \\ \forall V \in G(V_{1..m}, V_{root}), \;\; \exists f, n \in N : (\mathrm{A,I,P,L,C,M}) = f(V) \\ \forall T \in Types \;\; \exists f_{create\_T} : V_{new} = f_{create\_t}(\mathrm{A,I,P,L,C,M}) \\ \forall V_{device} \;\; \exists f_{add} : V_{device} \cap f_{add}(V_{root}, V_{device}) \; != \emptyset \\ \forall V_{device} \;\; \exists f_{delete} : V_{device} \cap f_{delete}(V_{root}, V_{device}) = \emptyset \end{cases} \quad (2.19)$$

Первый критерий в полученном наборе условий может быть также заменён на операции создания и замены коневого блока. Поэтому, конечная запись условий будет следующей:

$$\begin{cases} \forall V \in G(V_{1..m}, V_{root}) \;\; \exists f_{1..N}, n \in N : V = f_n\left(f_{n-1}\left(\dots f_1(V_{root})\right)\right) \\ \forall V \in G(V_{1..m}, V_{root}), \;\; \exists f, n \in N : (\mathrm{A,I,P,L,C,M}) = f(V) \\ \forall T \in Types \;\; \exists f_{create\_T} : V_{new} = f_{create\_T}(\mathrm{A,I,P,L,C,M}) \\ \forall V_{device} \;\; \exists f_{add} : V_{device} \cap f_{add}(V_{root}, V_{device}) \; != \emptyset \\ \forall V_{device} \;\; \exists f_{delete} : V_{device} \cap f_{delete}(V_{root}, V_{device}) = \emptyset \end{cases} \quad (2.20)$$

Язык работы с гибридной метамоделью

За основу специализированного языка работы с гибридными моделями однокристальных цифровых устройств предлагается взять один из универсальных языков программирования, который широко используется в САПР цифровых устройств и гарантирует полноту алгоритмической части (например, TCL). В дополнение в язык вносится ряд операций над моделью, которые позволяют обеспечить выполнение сформированных выше условий полноты набора. В таблице 2.6 приведены категории операций над моделью и краткое их описание. Каждая категория операций использует лишь

подмножество данных из модели. Это позволяет изолировать операции и упростить алгоритмизацию процессов реинжиниринга с использованием базового набора операций.

Таблица 2.6

Категории операций в предлагаемом языке работы с моделью

Категория	Примеры операций	Комментарии
Операции алгоритма	Ветвления, циклы, вызов процедур, …	Реализуются при помощи универсального языка и не взаимодействуют с моделью напрямую
Системные вызовы CAP	Выключение CAP, загрузка расширения	
Навигация по модели	Поиск элементов в иерархии, проход по дереву элементов, получение ссылок	Операции не вносят изменений в модель. Результатом операции является некоторый набор информации *I*. $I = f(G, A);\ I = f(G, P);\ I = f(G, L, C)$
Изменение структуры модели	Добавление и удаление элементов	Меняется только родительский элемент модели, замена дочернего элемента $G = f_{create}(G, V) \quad G = f_{delete}(G, V)$
Изменение свойств элементов	Получение и изменение атрибутов и связей	Меняются только элементы модели, её структура остаётся неизменной. $V = f_m(V, A) \quad A = f_m(V)$ $V = f_m(V, P, L, C)$
Работа с метаданными	Задание и считывание метаданных	Операции используются для расширения модели нестандартными данными $M = f(V, M);\ V = f_m(V, M)$
Обработчики событий	Условная генерация устройств, валидация вносимых изменений	Специальные команды обратного вызова для обработки изменений в модели $G = f_x(G, E)$

Функциональная полнота алгоритмической части языка является следствием функциональной полноты языка программирования, на базе

которого строится модель. Для языка TCL полнота показана в [53]. Таким образом, доказательство данного критерия для предложенного набора операций не требуется. Для гибридной модели предложен набор операций, соответствующий указанным выше критериям полноты. Полный список команд для каждой из групп операций приведёт в приложении D.

2.6. Совместимость модели с языками описания устройств

В предыдущих параграфах показаны расширяемость модели и универсальность языка работы с моделью, что позволяет говорить о её применимости для широкого спектра задач реинжиниринга. С другой стороны, актуален вопрос использования модели для стандартных форматов описания устройств, используемых при проектировании современных устройств и СнК. Ниже доказана совместимость модели с HDL в вырожденном случае. Предложены пути построения модели для различных HDL (VHDL, Verilog, SystemC и System Verilog), а также для формата нетлистов EDIF. Доказательство выполнено посредством анализа элементов языка и предложения эквивалентной реализации в рамках гибридной модели.

2.6.1. Совместимость модели с HDL в вырожденном случае

Наиболее простым способом представления устройств в гибридной модели является представление синтаксических деревьев описаний исходного устройства. Такой подход лишь частично использует возможности модели, что затрудняет операции над ней, но позволяет говорить о её совместимости.

Представим модель в виде одного элемента устройства V_y, все исходные описания на HDL прикреплены к которому в виде метаданных $M_{1..N}$:

$$\begin{cases} G = (V_y) \\ V = (A, I, P, L, C, M_{1..N}) \end{cases} \quad (2.21)$$

В таком вырожденном виде модель может быть использована для представления любых внешних описаний, так как:

1. Язык работы с гибридной моделью включает операции чтения и модификации метаданных в произвольном формате.
2. В соответствии с доказательством из параграфа 2.5 язык работы с моделью основан на универсальном языке программирования, для которого доказана функциональная полнота.
3. В алгоритме реинжиниринга описания на HDL могут быть посредством универсального языка программирования трансформированы и после возвращены в модель в виде нового набора метаданных.

Таким образом, обоснована применимость модели и методик работы с ней для представления и обработки описаний на любом HDL. С практической точки зрения, данный результат мало значим, так как вырожденная модель не удовлетворяет неформальным критериям по удобству её применения при решении частных задач реинжиниринга и встраивания СТ. Ниже рассмотрены подходы к представлению HDL на базе гибридной модели, максимально использующие заложенную в модель функциональность.

2.6.2. Подход к обеспечению совместимости модели с HDL

Предыдущий подпараграф показал возможность представления HDL в рамках гибридной метамодели. Тем не менее, для удобства практического применения модели требуется максимально полно использовать предлагаемый метамоделью набор элементов для отражения структуры исходного устройства в алгоритме реинжиниринга. Предлагается следующий подход:

1. Определение уровней иерархии устройства в семантике языка, перевод иерархии в структуру блоков и соединений между ними.
2. Перевод всех процедур и функций исходного описания в элементы функций модели.
3. Использование функций и метаданных для представления оставшихся элементов языка и правил взаимодействия.
4. Сохранение исходного синтаксиса компонентов в метаданных модели.

Изменение метаданных должно проводиться внешними обработчиками, которые вводятся при специализации модели для обеспечения корректности структуры устройства. Предлагаемый выше подход позволяет отразить синтаксические и семантические деревья исходного описания, но при этом отсутствует полная объектная модель устройства. Для её получения возможно производить частичный синтез, что должно поддерживаться гибридной метамоделью. Таким образом, для поддержки HDL в модели требуются:

- методика построения модели из исходного описания (импорт модели);
- методика экспорта синтезируемого описания из модели.

Вопросы построения импорта и экспорта модели описаны в следующей главе. В таблице 2.7 поставлены в соответствие элементы гибридной метамодели и основные элементы языка VHDL.

Таблица 2.7

Таблица соответствия элементов в VHDL и гибридной метамодели

VHDL	Гибридная метамодель	Комментарий
Entity	Интерфейс	
Architecture	Устройство, Блок	Устройство - компонент верхнего уровня иерархии VHDL, остальные - блоки
Component	Блок	
Package, Library	Библиотека	
Port, Generic, Signal, Const Value	Сигнал	Используется внутренняя типизация
Type, Subtype	Тип	Оператор “Use” реализован через ссылки
Array	Группа	std_logic_vector является группой сигналов
Assigment	Соединение	
Function, Operator	Функция	
Поведенческие описания (Process)	Функция	

2.7. Выводы по главе

Во второй главе разработана новая гибридная метамодель цифровых устройств, относящаяся к классу многоуровневых моделей, отличающаяся от других совмещением двух уровней представления и применимая в широком классе задач анализа, модификации и верификации архитектур цифровых устройств. Модель соответствует требованиям из подпараграфа 1.5.1 и может быть использована при построении специализированных моделей для решения широкого класса задач реинжиниринга. В диссертации кратко описаны основные элементы модели. В параграфе 2.6 подтверждена совместимость метамодели с существующими HDL, поэтому она может быть использована в маршрутах проектирования устройств. Предложенный язык работы с моделью обладает функциональной полнотой для задач анализа и трансформации в маршрутах реинжиниринга на основе гибридной метамодели. Предоставляемый языком набор команд подробно рассмотрен в приложении D.

Применимость модели подтверждена в пятой главе посредством прототипирования. В параграфе 5.5 также рассмотрены возможности применения гибридной модели для классов задач реинжиниринга за рамками поставленных ограничений. В третьей главе предложены методики работы с моделью, позволяющие строить алгоритмы реинжиниринга и при необходимости специализировать гибридную модель для представления необходимой информации.

3. МЕТОДИКИ РАБОТЫ С ГИБРИДНОЙ МОДЕЛЬЮ

В главе описываются методики работы с предложенной моделью представления устройств при решении типовых задач реинжиниринга цифровых систем, на которых основаны описанные в параграфе 3.4 методики встраивания средств внутрисхемного тестирования. Рассмотрены задачи специализации модели для частных задач реинжиниринга, ввод и вывод многоуровневых описаний, выполнение типовых операций над моделью, а также задачи верификации и валидации модели при реинжиниринге.

3.1. Специализация модели для частных задач реинжиниринга

Гибридная модель является универсальной основой для построения специализированных моделей, которые могут быть эффективно использованы в алгоритмах реинжиниринга. Предусмотрены механизмы расширения на трех уровнях, соответствующих компонентам инструментариев: на уровнях модели, методов и средств (таблица 3.1).

3.1.1. Методика специализации гибридной модели

Для специализации гибридной метамодели для класса частных задач реинжиниринга предлагается следующий метод:

1. Определение области применения разрабатываемой модели устройства: класса задач, системных ограничений, методик описания, маршрут проектирования и пр.
2. Определение характеристик устройства, которые требуется отразить в модели для решения задач анализа и принятия решений о реинжиниринге
3. Дополнение структурного представления частной модели: введение специальных типов, специализированных блоков (например, ТА).
4. Введение и спецификация метаданных для хранения дополнительной информации о характеристиках элементов модели.
5. Анализ решаемых задач и формирование дополнительного набора операций для инкапсуляции стандартных операций в алгоритмах.

6. Дополнение методик импорта и экспорта модели для использования специализированной модели устройства в маршруте проектирования.
7. Разработка специализированных методов валидации и контроля корректности модели, интеграция методов в набор операций.
8. Разработка специализированных алгоритмов и средств для решения выбранного класса задач реинжиниринга.

Таблица 3.1

Примеры расширения для различных уровней специализации

Уровень специализации	Примеры расширения модели
Модель устройства	– Добавление метаданных в элементы для передачи дополнительной информации между шагами алгоритма; – Наследование элементов для создания специализированных типов; – Задание зависимостей через ссылки.
Методы работы с моделью	– Создание специализированных наборов операций; – Регистрация команд обратного вызова для обработки событий в модели (изменение элементов модели); – Условная генерация компонентов.
Инструментальные средства	– Использование средств объектно-ориентированных языков программирования: наследование, инкапсуляция и полиморфизм объектов модели; – Интеграция со сторонними САПР.

В таблице 3.2 приведены примеры для двух классов задач реинжиниринга. Рассмотренные примеры показывают, что использования ограниченного числа точек расширения достаточно для специализации модели под требования частной задачи реинжиниринга устройств. В параграфе 4.1 подробно рассмотрена специализация гибридной метамодели для задач встраивания средств внутрисхемного тестирования.

Таблица 3.2

Примеры специализации модели для задач внутрисхемного тестирования

Уровень специализации	Задача внутрисхемного тестирования	
	Встраивание средств самодиагностики	Встраивание тестовых агентов
Модель устройства	– Добавление метаданных о характеристиках элементов в модели	– Создание специализированных компонентов тестовой инфраструктуры
Методы работы с моделью	– Расширение команд верификации модели при преобразовании	– Добавление тестовых точек и интерфейсов – Формирование ТА для заданной контрольной точки
Инструментальные средства	– Использование САПР для синтеза и получения характеристик устройства	– Экспорт описаний тестового окружения для программной части – Синтез параметризуемых компонентов

3.1.2. Ограничения специализации модели

Предложенный подход позволяет использовать ориентированные на базовую модель алгоритмы анализа и трансформации в любых производных моделях. Тем не менее, он связан с рядом ограничений:

1. Специализация модели и введение новых типов элементов в общем случае ухудшают переносимость алгоритмов реинжиниринга.
2. Специализация усложняет модель и увеличивает риск возникновения ошибок при её реализации.

Ограничение переносимости связано с тем, что надстройки над моделью описывают часть семантики устройства и недоступны для стандартных

алгоритмов. Это может приводить к потере информации при преобразованиях и формированию некорректных устройств. Для предотвращения подобной ситуации в модель должны быть введены обработчики обратного вызова, которые обеспечивали бы корректность и целостность модели при выполнении преобразований архитектуры устройств при реинжиниринге.

Рост сложности модели обусловлен введением дополнительных сущностей и правил их обработки. При этом могут использоваться операции из базового набора и расширений CAP, из-за чего необходимо тщательное тестирование алгоритмов. С другой стороны, специализация позволяет ввести новые слои абстракции и инкапсулировать особенности строения элементов, что наоборот снижает общую сложность модели.

3.2. Импорт и экспорт гибридной модели из внешних описаний

В соответствии с маршрутом проведения реинжиниринга первым шагом является восстановление архитектуры устройства из исходных описаний. Для предложенного подхода задача заключается в построении гибридной модели устройств по исходным описаниям. В случае, когда используется одно исходное описание, формирование модели может производиться в соответствии с рекомендациями в параграфе 2.6. В соответствии с концепцией многоуровневых моделей для построения модели также может использоваться несколько входных описаний. На рис. 3.1 приведены маршруты импорта и экспорта модели для обоих случаев.

3.2.1. Импорт гибридной модели из одного описания

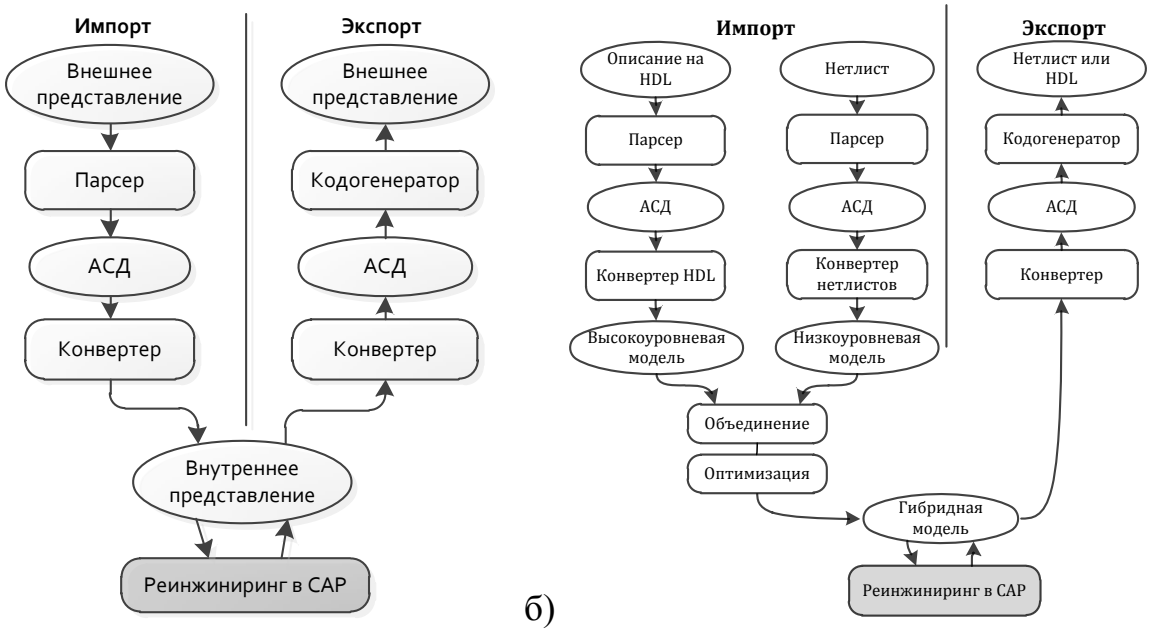
Предположим, что представление строится на основе структурного описания, из которого возможно извлечь иерархию устройства или набор элементов для библиотеки. Предлагаемый порядок построения для данного случая приведён на рис. 3.2.

Предлагаемые алгоритмы предполагают последовательное считывание иерархии с последовательной её детализацией. Это сделано для того, чтобы

исключить возникновение ситуации, когда некоторые исходные данные не готовы к добавлению в иерархию (например, для соединения считан только выходной сигнал, или интерфейс для считываемого компонента ещё не загружен в библиотеку). В предлагаемых алгоритмах не отражены ветви обработки ошибок импорта модели, которые могут возникать на любом этапе алгоритма. В данном случае есть четыре варианта решения, выбор между которыми остаётся за разработчиком модулей импорта:

- переход к следующим этапам импорта описания, чтобы извлечь максимум информации об архитектуре устройства;
- остановка считывания и возвращение частичной архитектуры;
- отмена модификаций дерева элементов, сделанных при выполнении операции считывания;
- попытка исправления дерева элементов путём исправления ошибки (например, создание несуществующего интерфейса).

Выбранная реализация алгоритмов может потребовать нескольких проходов по синтаксическому дереву считываемого представления. Для сложных устройств это может быть длительным процессом, поэтому могут быть разработаны алгоритмы с однократным проходом.



а) б)

Рис. 3.1. Методика импорта и экспорта гибридной модели

(а – одно описание; б – гибридная модель на базе нескольких описаний)

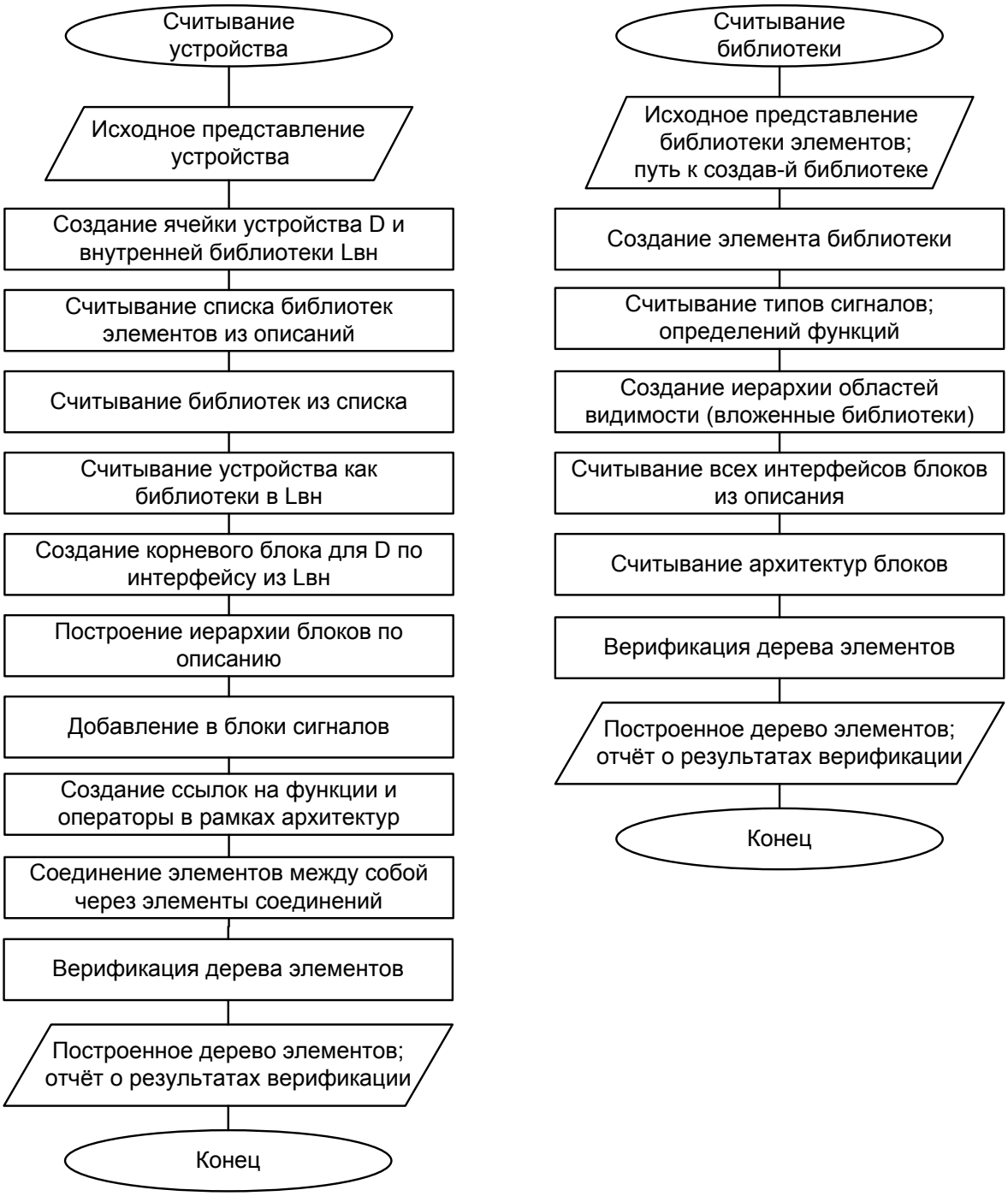


Рис. 3.2. Импорт устройств и библиотек из внешних описаний

3.2.2. Импорт гибридной модели из множества описаний

Гибридная модель разработана для задач реинжиниринга, в которых требуется одновременная работа с высокоуровневым описанием устройства и его низкоуровневой реализацией. Рассмотрим задачу построения модели для устройств, одновременно описанных на уровнях HDL и низкоуровневых

нетлистов, полученных в результате синтеза или разводки исходных описаний. В этом случае требуется объединение двух представлений в единое описание.

На рис. 3.3 приведён предлагаемый подход к импорту моделей из нескольких описаний. Предлагается разбить задачу на следующие шаги:

1. Построение гибридной модели для высокоуровневого описания на HDL.
2. Построение гибридной модели для низкоуровневого описания.
3. Объединение двух независимых гибридных в единую модель устройства.
4. Оптимизация объединённой модели для устранения избыточности.

Первые два шага импорта модели рассмотрены в предыдущих параграфах. Ниже рассмотрены шаги объединения и оптимизации применительно для различных комбинаций исходных описаний.

Объединение и оптимизация моделей

Для решения указанных выше проблем при объединении моделей предлагается произвести ряд преобразований для синхронизации иерархии устройств (рис. 3.3), после чего возможно объединение двух описаний. При восстановлении нетлистов решаются следующие задачи:

- исключение несинтезируемых сигналов из нетлистов;
- восстановление иерархии по объявлениям блоков;
- выявление однотипных блоков и объединение их в библиотечные компоненты (триггеры, блочная память и пр.);
- восстановление интерфейсов по неполным объявлениям компонентов;
- устранение сквозных сигналов (рис. 3.4);
- восстановление иерархии компонентов.

Можно выделить два основных способа, применяемых при удалении сквозных соединений в устройстве:

1. Поиск эквивалентов, которые обеспечивали бы аналогичные сигналы внутри уровня иерархии (в т.ч. применимо для сигналов GND, PWR).
2. Внесение изменений в иерархию устройства, проведение сигнала сквозь уровни иерархии.

Рис. 3.3. Объединение и оптимизация гибридной модели

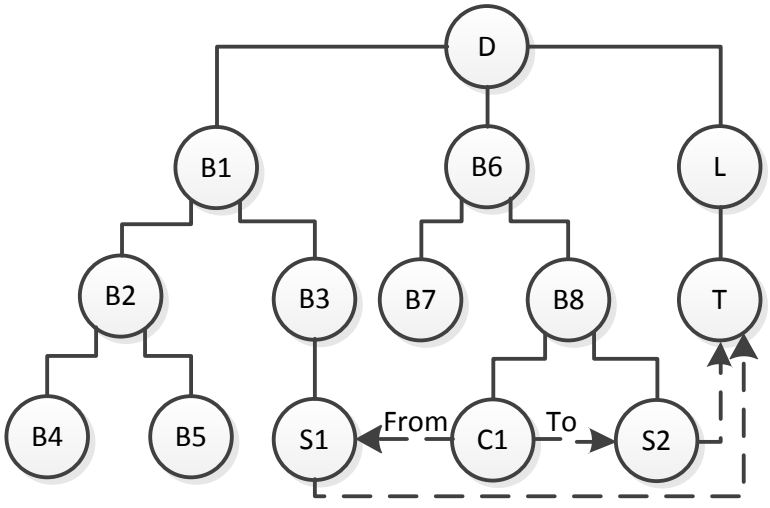


Рис. 3.4. Пример сквозного соединения в устройстве

Рассмотрим пример сквозного соединения сигналов на разных уровнях иерархии устройства в исходном нетлисте, приведённый на рис. 3.4. Подобная

ситуация является типовой для «плоских» нетлистов, где архитектура восстанавливается по косвенным данным об устройстве. Для проведения сигналов сквозь уровни иерархии предлагается следующий алгоритм:

1. Поиск общего родительского блока устройства B_{parent}.
2. Формирование трассы блоков B_X, через которую должен пройти соединяемый сигнал (в примере $B_X = (B3, B1, B6, B8)$).
3. Создание библиотеки *L* внутри устройства для хранения элементов.
4. Замена всех блоков B_X, на блоки B_L, которые наследуются от исходных.
5. Перенос блоков B_X в библиотеку L.
6. Добавление вывода с типом T к каждому элементу B_L. Направление ввода/вывода соответствует направлению следования сигнала.
7. Последовательное соединение всех блоков и сигналов на маршруте $(S1, B3, B1, B6, B8, S2)$.
8. Удаление элемента *C1* из гибридной модели устройства.

При выполнении повторных проведений сигналов через модифицированные блоки, не требуется копирование в библиотеку L. Поскольку все элементы уже сделаны уникальными за счёт структурного наследования, то достаточно расширения интерфейса.

3.2.3. Вывод описаний в синтезируемые форматы

По сравнению с операцией импорта модели из внешних описаний, задача экспорта описаний в синтезируемые описания не требует объединение и последующая оптимизация описаний. Алгоритмы экспорта описания должны обеспечить соответствие выходного описания спецификациям для требуемого формата, но можно говорить о возможности данной операции, если целевой формат поддерживает описание устройств на уровне логических вентилей. При этом, после завершения трансформации, в модели устройства могут присутствовать следующие ситуации:

- часть синтезируемого описания находится в метаданных устройства;

- имеются блоки функций с неразрешённым для целевого формата описанием (не произведена операция частичного синтеза);
- модель является некорректной или неполной из-за ошибочного преобразования на одном из этапов реинжиниринга.

Решение указанных выше проблем должно производиться алгоритмом реинжиниринга. Возможен обход дерева элементов модели и контроль с целью валидации и верификации элементов (см. параграф 3.3). Конечная верификация может производиться как средствами CAP внутри алгоритма реинжиниринга, так и внешними средствами после экспорта модели.

На рис. 3.5 приведён ещё пример ввода-вывода гибридной модели из нескольких входных описаний, когда посредством синтеза получается структурное описание с дополнительной информацией о реализации устройства для целевой платформы.

3.3. Контроль модели при проведении реинжиниринга

Предлагаемая модель должна поддерживать верификацию и валидацию устройств, которые заключаются в специализированном анализе их моделей. Ниже приведены основные требования к модели, при которых возможно проведение подобных операций. Сформированы следующие критерии верифицируемости модели:

- модель содержит конечное число элементов;
- структурные блоки имеют строго древовидную структуру;
- при развёртывании ссылок модель является орграфом;
- возможен доступ к любому элементу модели за конечное время;
- все элементы модели имеют полное внутреннее описание.

Все критерии формализованы во второй главе диссертации как свойства модели или требования к набору операций над моделью. Таки образом, любая корректная гибридная модель является верифицируемой.

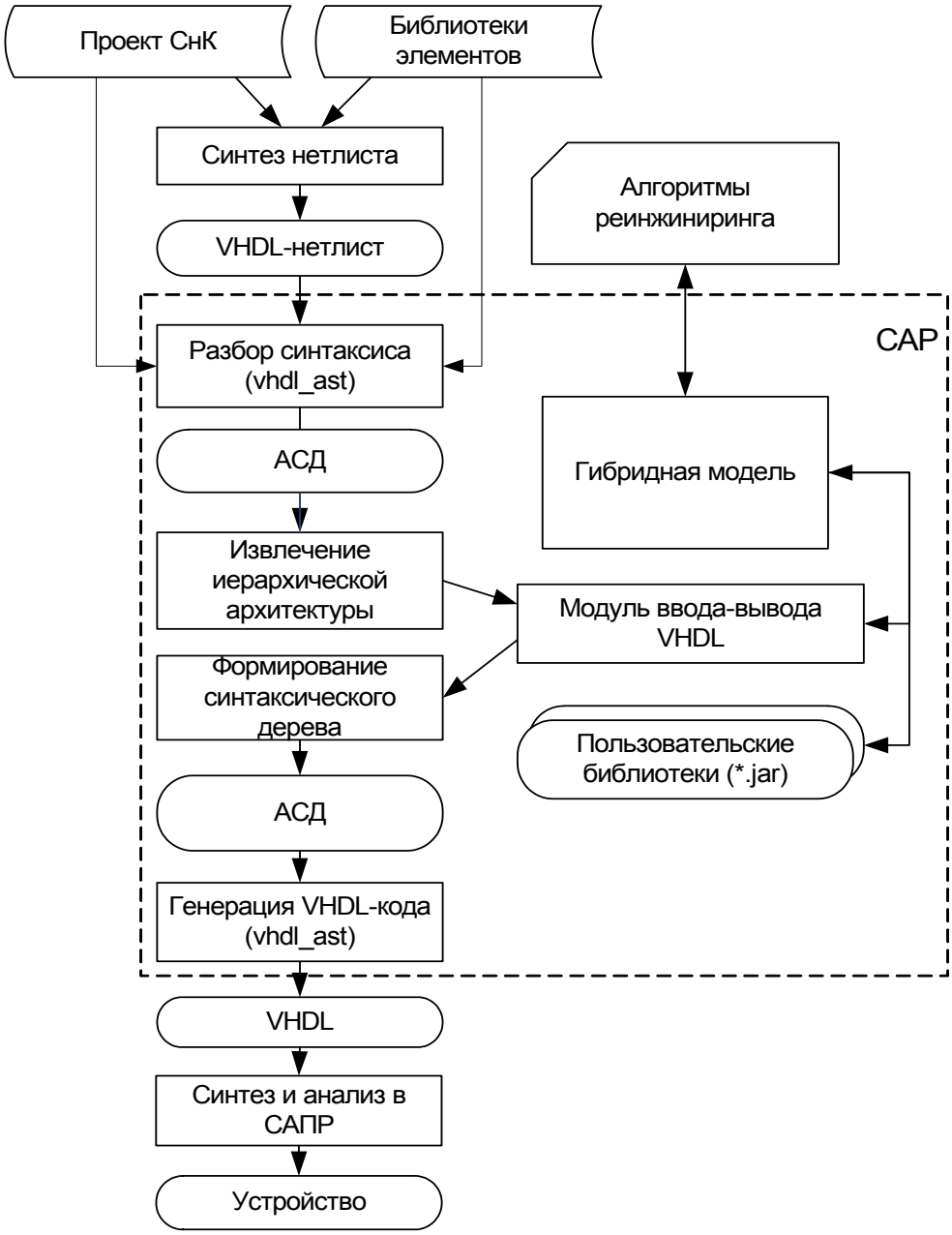


Рис. 3.5. Организация ввода-вывода гибридной модели

Валидация модели при преобразованиях

За счёт функций обратного вызова возможна регистрация обработчиков для изменений в модели, которые позволяют проверить корректность проводимых операций. Поскольку функция обратного вызова передаёт ссылку на модифицируемый блок, то алгоритм анализа получает доступ ко всей гибридной модели устройства. Обработчик обратного вызова имеет возможность вернуть ошибку вызывающему алгоритму. Также, за счёт хранения истории операций, обработчик может отменить ряд предыдущих

операций с целью сохранения целостности модели на уровне логических вентилей.

3.4. Выводы по главе

В третьей лаве рассмотрены методики работы с гибридной метамоделью и её специализации для решения частных задач реинжиниринга. Предложен базовый набор операций над моделью, который позволяет формализовать алгоритмы анализа и трансформации моделей. Таким образом, сформирован базис, на котором могут строиться прикладные алгоритмы и инструментальные средства для практических задач реинжиниринга.

Для маршрутов проектирования на базе ИР предложены подходы к решению типовых задач, решаемых при реинжиниринге: обратного инжиниринга архитектуры на основе исходных описаний устройства на различных уровнях (HDL и структурные нетлисты), валидации и верификации устройств при выполнении структурных изменений.

В следующей главе приведён пример специализации модели для в классе задач встраивания средств внутрисхемного тестирования, которые являются частными задачами реинжиниринга устройств. Также в главе продемонстрировано построение инструментальных средств для задачи совместной верификации аппаратных прототипов устройств и их моделей в системах моделирования.

4. МЕТОДЫ ВНУТРИСХЕМНОГО ТЕСТИРОВАНИЯ УСТРОЙСТВ

В четвёртой главе рассмотрено применение разработанных моделей и методик для решения типовых задач внутрисхемного тестирования и самодиагностики устройств. Для класса задач внутрисхемного тестирования устройств выполнена специализация гибридной метамодели. Рассмотрен вопрос встраивания средств тестирования в условиях системных ограничений. Предложен подход к совместному тестированию устройства и его программной модели в средствах моделирования.

4.1. Специализация гибридной метамодели для задач встраивания СТ

В данном параграфе приведён пример специализации гибридной метамодели для класса задач встраивания средств внутрисхемного тестирования в СнК. Постановки задач реинжиниринга в данном классе задач рассмотрены в параграфе 1.2. В общем случае, в устройства требуется встраивать средства тестовые агенты, интерфейсы и дополнительные блоки контроля. После встраивания компоненты соединяются между собой в единую систему. Специализация модели производится в соответствии с рекомендациями из подпараграфа 3.1.1.

Специализация на уровне модели

В соответствии с постановкой задачи, система тестирования может быть описана следующим образом:

$$FI_M = (B_{MUT}, B_{TA}, I_{Test}, C), \text{ где} \quad (4.1)$$

B_{MUT} $S_M = (S, I_{CPU}, C)$ - множество модулей устройства, к которым подключаются ТА,

B_{TA} – множество тестовых агентов, подключаемых к модулям,

I_{Test} – тестовый интерфейс (ТИ) СнК,

C – множество соединений между ТА и тестовым интерфейсом.

В случае сложных систем тестирования может потребоваться введение дополнительной топологии коммуникационных элементов, которые бы

обеспечивали передачу информации между ТИ и ТК. Интерфейсы могут играть роли коммуникационных концентраторов, мультиплексоров и дешифраторов для совмещения передачи информации по единым линиям соединений. Топология описывается через дополнительные блоки V_{TC} и введение дополнительных сигналов в множество C:

$$\mathrm{FI_M} = \left(\mathrm{B_{MUT}}, \mathrm{B_{TA}}, \mathrm{B_{TC}}, \mathrm{I_{Test}}, \mathrm{C}\right) \quad (4.2)$$

В модель предлагается ввести библиотеку элементов, содержащую следующие элементы:

- параметризуемые блоки тестовых агентов (см. подпараграф 4.2.1);
- набор блоков параметризуемых стандартных ТИ (JTAG и пр.), выбор которого определяется периферийными СТ и требованиями к системе;
- набор базовых коммуникационных блоков для реализации топологии соединений между ТА и ТИ.

Архитектуры и реализации предлагаемых библиотечных компонентов рассмотрены ниже. Для блоков ТА в гибридной модели возможно хранить дополнительную информацию, которая упрощает последующую реализацию алгоритмов реинжиниринга и тестовых программ:

- идентификатор ТА в тестовом интерфейсе ТИ;
- символьное имя управляемого сигнала для внешней СТ;
- ссылки на перечни свойств ТА и поддерживаемых операций;
- статистика по размеру блоков (например, число логических элементов в ПЛИС) для первичных оценок в алгоритме реинжиниринга;
- статистика по временным характеристикам (задержка распространения сигналов, максимальная тактовая частота и т.д.) для первичных оценок.

Специализация на уровне методов

Для встраивания средств тестирования требуются специальные методы, которые позволили бы строить тестовые инфраструктуры по описаниям устройств на HDL и описаниями программ тестирования. При этом методика должна учитывать требования по системным ресурсам и временным

характеристикам, которые сильно влияют на топологию тестовой инфраструктуры в случае жёстких ограничений. Методика встраивания средств тестирования описана в подпараграфе 4.2.3.

Для рассматриваемого класса задач не требуются дополнительные методы контроля структуры устройства, так как расширение на уровне модели использует стандартные точки расширения, а метаданные элементов не хранят значимую для синтеза информацию.

Специализация на уровне инструментальных средств

Инструментальные средства ориентированы на синтез и размещение тестовой инфраструктуры с последующим контролем её соответствия поставленным требованиям по использованию системных ресурсов и временным характеристикам. Синтез, размещение и оценка характеристик (временной анализ, оптимизация размещения и т.п.) являются проработанными областями как с научной, так и с практической точки зрения. В СПР возможно применение внешних инструментальных средств, и задача специализации в большинстве случаев сводится к интеграции с уже существующими инструментальными средствами, для чего должны быть разработаны:

- методики импорта гибридной модели из описаний, сформированных внешними инструментальными средствами;
- методики экспорта описаний в форматы, поддерживаемые внешними инструментальными средствами;
- методики синхронизации изменений в двух представлениях для возможности внесения изменений в модель;

Все перечисленные методики являются специфичными для целевых инструментальных средств, и для них может потребоваться дополнительная специализация модели. В приложениях D и E частично рассмотрены методики импорта/экспорта данных в САПР Quartus II компании Altera, которая содержит необходимые инструментальные средства [15].

4.2. Встраивание средств тестирования

При встраивании средств тестирования в устройство необходимо обеспечить полную управляемость и частичную наблюдаемость тестируемого компонента системы. Это обычно требует внесения в элементы дополнительных функциональных блоков, что является частной задачей реинжиниринга устройств (рис. 4.1). При этом решаются следующие задачи:

1. Изоляция тестируемого компонента от влияния остальной системы.
2. Изоляция системы от влияния компонента при проведении тестирования. В ряде случаев может быть использован останов системы.
3. Встраивание средств генерации тестовых векторов и/или установки требуемого состояния модуля для обеспечения управляемости.
4. Встраивание диагностических интерфейсов и выводов для наблюдения за состоянием системы в процессе выполнения тестов.
5. Соединение ТА с интерфейсами через уровни иерархии устройства.
6. Контроль соответствия инструментированного устройства поставленным ограничениям по системным ресурсам и временным характеристикам.

4.2.1. Тестовые агенты (ТА)

При встраивании средств тестирования с использованием CAP тестовый агент предлагается реализовать как универсальный параметризируемый компонент. Поскольку гибридная модель независима от исходных языков описания, то требуется только одна реализация, которая может быть подключена к описанным на других языках HDL модулям системы.

На рис. 4.1 приведена предлагаемая структура тестового агента, который решает перечисленные выше задачи изоляции устройства. Для изоляции остальной части системы тестовый агент может быть подключен непосредственно к выходам устройства, если у него есть соответствующие интерфейсы управления (например, общий сигнал сброса).

Приведённые на рис. 4.1 компоненты ТА являются опциональными и добавляются алгоритмом реинжиниринга при необходимости управления или

считывания определённых классов сигналов. Количества и размерности входных/выходных сигналов определяются параметрами блока в модели, реализующего тестовый агент. Таким образом, в гибридной модели ТА должен быть представлен или как параметризируемый библиотечный компонент, или как функция с частичным синтезом.

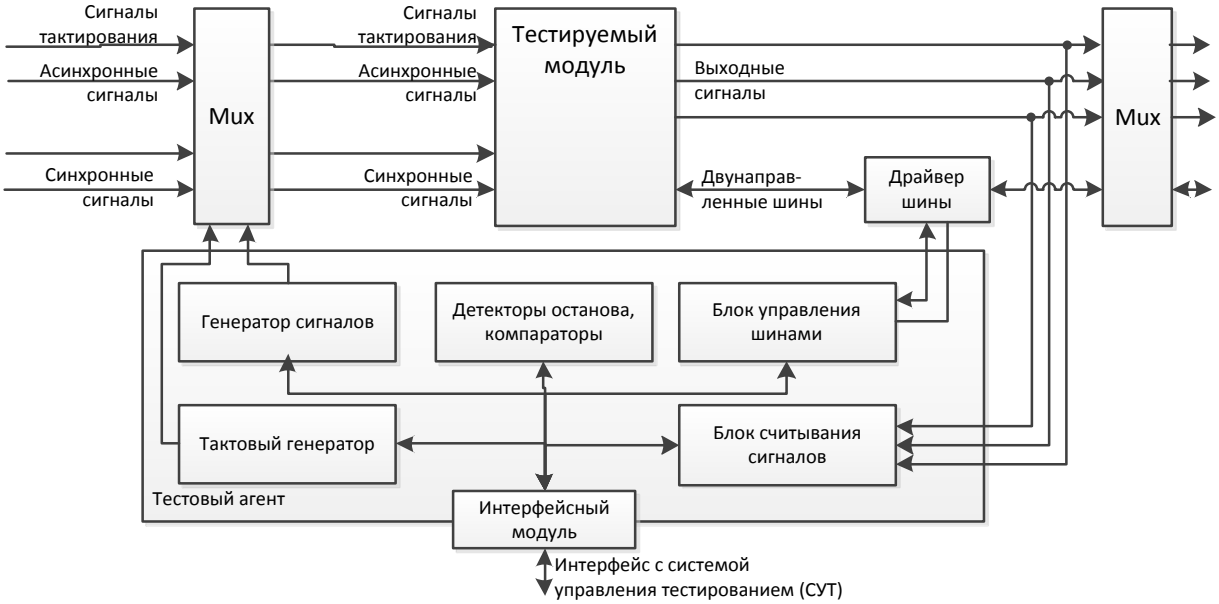


Рис. 4.1. Пример встраивания ТА в тестируемый модуль устройства

4.2.2. Построение системы тестирования

В устройстве может присутствовать большое число тестовых агентов, поэтому возникает проблема их подключения к внешним средствам тестирования. При реинжиниринге устройства предлагается также встраивать систему управления тестированием (СУТ), которая будет подключаться ко всем агентам и при этом взаимодействовать с внешним окружением через стандартный интерфейс (например, JTAG). Могут быть встроены следующие классы подобных СУТ:

- дешифратор тестовых агентов, управляемый внешней системой;
- СУТ с памятью для промежуточного хранения тестовых векторов при проведении тестов на реальных скоростях работы;
- модули диагностики и самодиагностики устройств.

Для подключения СУТ к агентам могут использоваться различные топологии межсоединений и сетей на кристалле, которые кратко рассмотрены в параграфе 1.2. Сложность компонентов зависит от числа ТА, передаваемых данных и требований к производительности системы. Систему внутрисхемного тестирования можно представить как множество тестовых агентов, узлов связи межсоединений и СУТ. Предлагаемая схема тестовой инфраструктуры приведена на рис. 4.2. Как и гибридная модель, она представляется орграфом. У него может быть несколько вершин при использовании внешних интерфейсов.

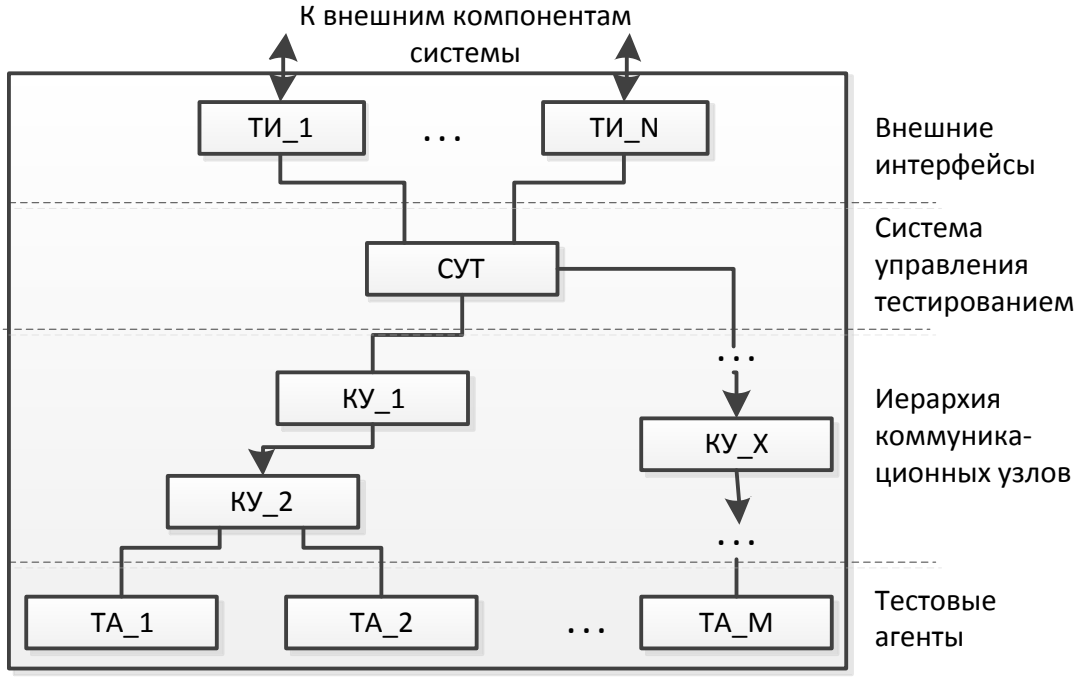


Рис. 4.2. Структура тестовой инфраструктуры

4.2.3. Методика встраивания средств тестирования

В качестве входных описаний одновременно берутся исходный HDL-код и сформированный при синтезе в CAP список соединений. Требования к тестовым агентам задаются при помощи списка портов и сигналов, к которым требуется подключить внешние выводы. По итогам трансформации в CAP генерируется синтезируемое описание, которое используется на последующих стадиях маршрута проектирования.

Предлагается использовать итеративный подход к проведению реинжиниринга, в цикле которого производится модификация архитектуры

устройства в САР и промежуточный синтез внешними средствами для контроля соответствия поставленным требованиям и ограничениям. Решение об окончании реинжиниринга принимается разработчиком или САР при полной автоматизации процесса. За счёт реализации эффективных алгоритмов анализа и трансформации в САР возможно снизить число итераций вплоть до одной. Исходные нетлисты синтезируются внешними средствами для построения низкоуровневой части гибридной модели.

При встраивании средств тестирования в устройства выделяются следующие стадии алгоритма реинжиниринга:

1. Синтез исходного описания в сторонних средствах проектирования.
2. Импорт гибридной модели из исходных описаний.
3. Импорт списка компонентов, которые должны быть инструментированы для проведения тестов.
4. Импорт библиотечных компонентов из исходных описаний.
5. Генерация тестовых агентов из библиотечных компонентов в соответствии со списком тестируемых модулей.
6. Генерация СУТ из библиотечных компонентов на основании ранее сгенерированных агентов.
7. Подключение тестовых агентов к СУТ через уровни иерархии.
8. Генерация выходного описания.
9. Повторный синтез сгенерированного описания для получения тестовой конфигурации устройства.
10. Анализ соответствия устройства требованиям и принятие решения о завершении реинжиниринга.

Импорт модели и генерация синтезируемого выходного описания осуществляется средствами САР. Шаги 5-7 алгоритма оперируют с внутренней моделью устройства, поэтому они независимы от исходных описаний устройства. Единожды реализованный алгоритм может быть применён для произвольных устройств, описанных на любых HDL, поддерживаемых САР.

4.2.4. Метод встраивания СТ в условиях системных ограничений

При проектировании или реинжиниринге цифровых устройств необходимо учитывать требования по системным характеристикам устройства (площадь на кристалле, число логических элементов и т.д.). СнК могут проектироваться без резервирования ресурсов кристалла на дополнительные средства тестирования из-за ограничений по технико-экономическим требованиям. Иногда возможно встраивание СТ без использования дополнительных ресурсов: использование свободных мультиплексоров и соединений в [64] или применение встроенных в СнК процессорных ядер [4], но в общем случае может встать задача встраивания СТ в готовые проекты и прототипы. Предлагается следующий метод внесения изменений:

1. Поиск минимальной конфигурации тестовых агентов, обеспечивающих необходимый уровень наблюдаемости и управляемости.
2. Независимый синтез устройства, СУТ и тестовых агентов для проверки требований по временным характеристикам.
3. Получение суммарных оценок по ресурсам. В случае невыполнения требований пересматриваются целевая платформа или набор тестов.
4. Выбор способа подключения тестовых агентов к СУТ для обеспечения требуемой скорости проведения тестирования.
5. Оптимизация сети подключения тестовых агентов.
6. Итерационная оптимизация реализации устройства посредством внешних средств проектирования.

Поскольку тестовые средства встраиваются в распределённые на кристалле элементы, то высокий вклад в характеристики системы дают межсоединения. В [67] рассмотрены независимые подключения ТА, использование системных шин и сетей на кристалле устройства. Выбор способа подключения определяется требованиями к скорости проведения тестирования и системным ресурсам СнК. Для оптимизации топологий предлагается использовать методы, основанные на направленных ациклических графах (DAG), что позволяет использовать гибридную модель. Прочие операции в

предложенном методе являются задачами реинжиниринга и могут быть решены посредством предложенных ранее методов.

4.3. Встраивание средств самодиагностики

Средства самодиагностики (BIST – Built-In Self-Test) являются важной составляющей высоконадёжных устройств. В дополнение к тестовым агентам и интерфейсам, средства самодиагностики включают систему управления тестированием (СУТ), которая ответственна за проведение тестов и принятие решений об исправности тестируемых компонентов системы.

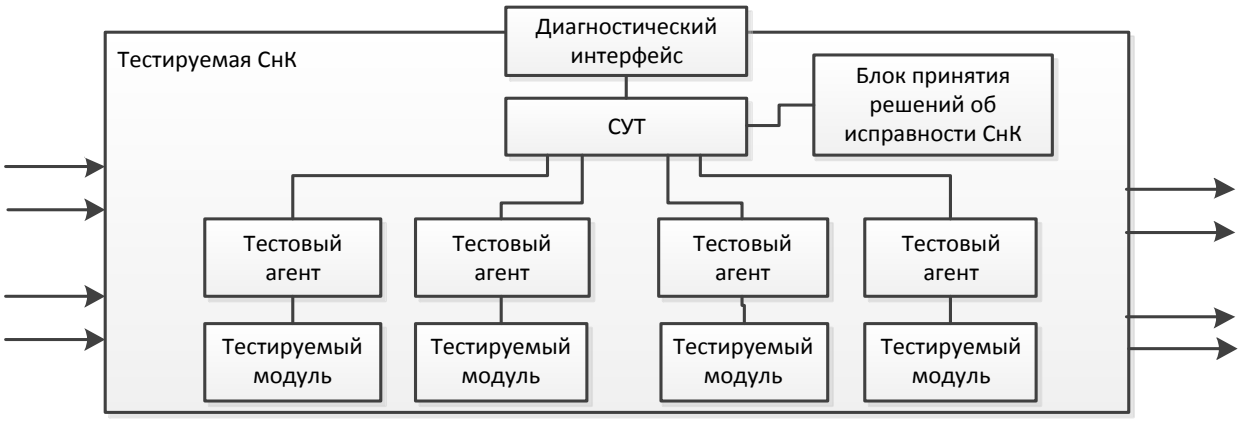


Рис. 4.3. Система самодиагностики устройства

Задача встраивания BIST-компонентов сводится к задаче встраивания средств тестирования (параграф 4.1) и последующему встраиванию блоков принятия решений, что также является задачей реинжиниринга. Задачи выбора архитектур блоков самодиагностики рассмотрены в [50], [43] и других трудах.

Встраивание средств самодиагностики может выполняться параллельно с другими задачами реинжиниринга, связанными с надёжностью устройств. На рис. 4.4 приведён пример подобной операции. В приведённом примере производится троирование модуля с целью диагностики и предотвращения сбоев при выходе из строя одного из модулей [8]. Выходы голосователя подаются на модуль самодиагностики устройства (BIST), который принимает решение о переходе на резервный функциональный блок. На рис. 4.5 приведён алгоритм реинжиниринга устройств для подобного примера. Более подробно задача внесения структурной избыточности рассмотрена в приложении F.

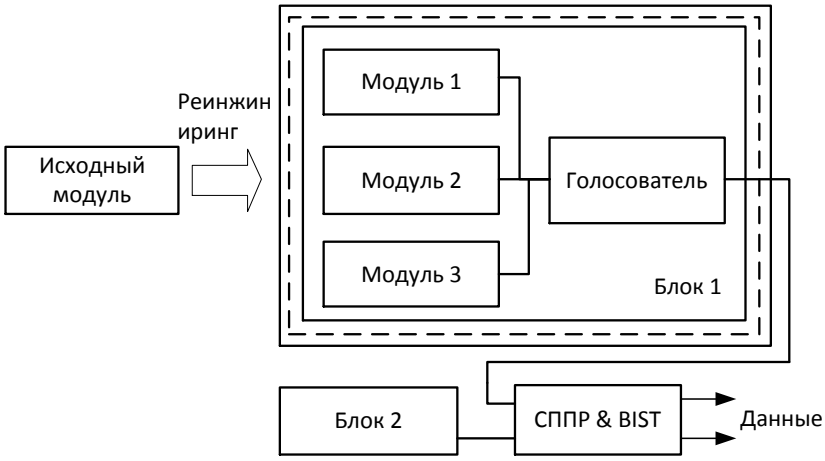


Рис. 4.4. Встраивания средств улучшения надёжности и самодиагностики

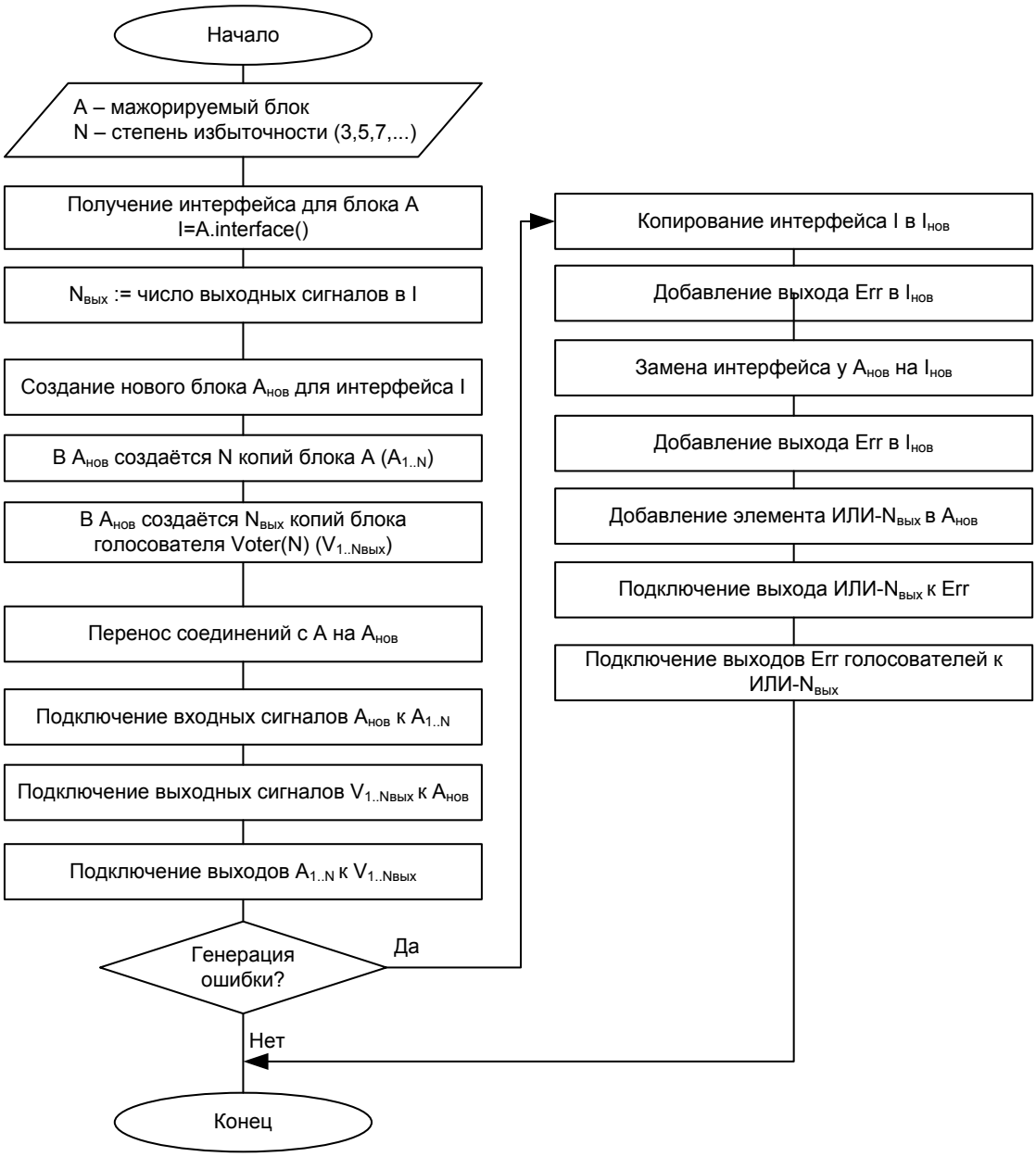


Рис. 4.5. Алгоритм встраивания средств структурной избыточности и самодиагностики в устройство с использованием САР

4.4. Совместное тестирование модели и аппаратного прототипа

В данном параграфе рассмотрена проблема повторного использования тестовых алгоритмов, разработанных в рамках имитационного моделирования устройств на этапе разработки. Предложены подходы по созданию тестовых компонентов, которые позволили бы проводить внутрисхемное тестирование на ПЛИС с использованием средств моделирования.

Встраивание средств внутрисхемного тестирования не требуется в случае, когда тестирование выполняется через внешние интерфейсы системы, универсальные компоненты (программируемые модули и процессорные ядра) или же через СТ, встроенные при проектировании устройства. Данные случаи рассмотрены в работах [4, 5]. В иных случаях может потребоваться встраивание средств тестирования в соответствии с параграфом 4.1, для чего требуется выполнить следующие шаги:

1. Встраивание средств внутрисхемного тестирования в устройство.
2. Встраивание тестовых интерфейсов в устройство.
3. Интеграция внешних СТ с тестовой инфраструктурой внутри СнК.

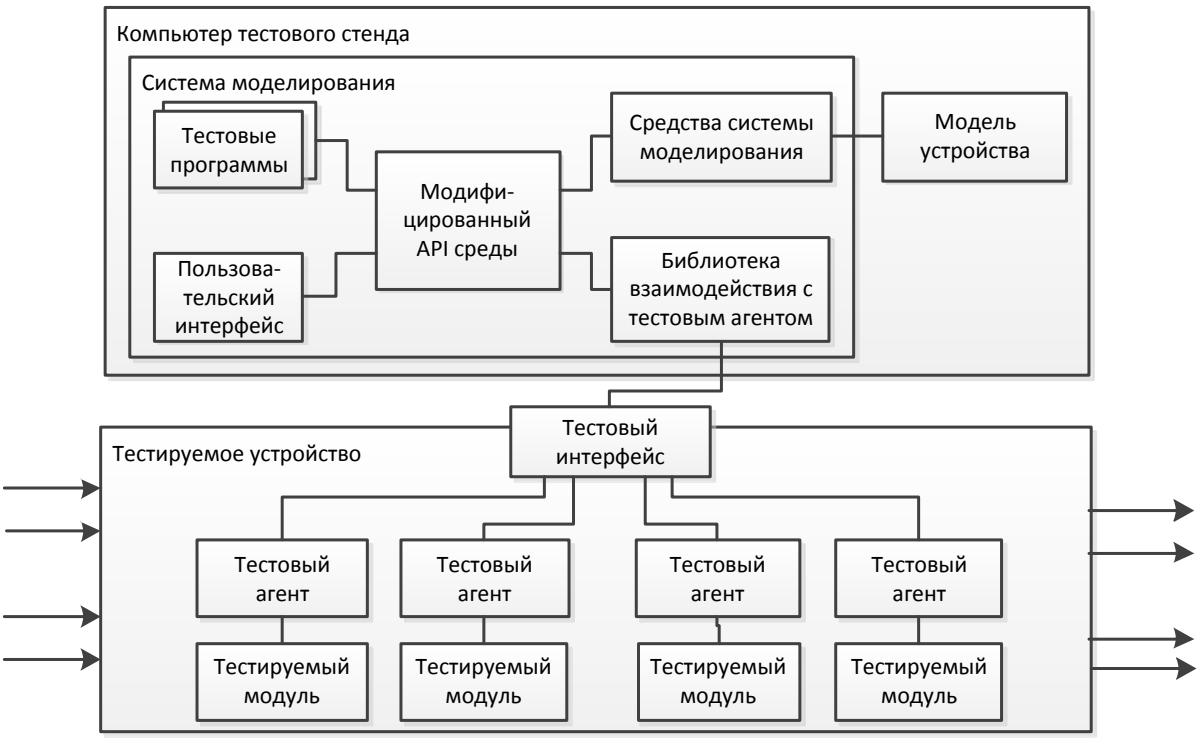


Рис. 4.6. Организация совместного тестирования модели устройства в системе моделирования и физического прототипа устройства

Отдельным случаем является применение виртуализированных окружений (например основанных на UVM - Universal Verification Methodology) в системах моделирования, когда средства языка позволяют обрабатывать события и формировать управляющие сигналы для тестовой инфраструктуры [5]. В этом случае модификация API не требуется.

4.4.1. Методика формирования тестового окружения

Задача сводится к задаче встраивания специализированных тестовых агентов в устройство по следующему алгоритму:

1. Проведение моделирования работы устройства под тестовым воздействием.
2. Извлечение списка сигналов, которые изменяются или считываются при проведении моделирования.
3. Формирование списка блоков *M,* в которые требуется встроить ТА.
4. Встраивание СТ посредством реинжиниринга устройства:
    a. Добавление тестового интерфейса T_I.
    b. Встраивание ТА для каждого проверяемого модуля ($T_{A1..N}$).
    c. Подключение тестовых агентов к T_I.
    d. Формирование выходного файла, который предоставляет данные для доступа к ТА через встроенный тестовый интерфейс.

На рис. 4.6 приведена предлагаемая схема подключения тестовых агентов к системе моделирования устройства. За счёт использования универсального интерфейса прикладного программирования (API) возможна инкапсуляция целевой платформы, что делает тестовые программы независимыми от реализации устройства. Предложенный подход обладает рядом ограничений:

1. Подход плохо применим для тестов, где требуется подключение ТА к большому числу компонентов (например, блочной памяти).
2. Отсутствие поддержки разнотипных тестовых агентов, что ограничивает область применения для гетерогенных СнК.

ТА предпочтительно встраивать на уровне общих интерфейсов. При использовании предлагаемого подхода, наоборот, ТА будут встроены на наиболее низкий уровень с целью снижения затрат на независимый ТА. Выбор конфигурации ТА должен производиться алгоритмом реинжиниринга. Данная задача является одним из рассмотренных ниже факторов оптимизации.

4.4.2. Описание тестовых программ

В [49] выделены следующие подходы к описанию тестовых программ в средах моделирования:

- тестовые векторы или диаграммы – временные последовательности входных сигналов и ожидаемых выходных значений;
- тесты, описанные на HDL как внешние тестовые компоненты, которые могут включать несинтезируемые блоки ([10]);
- тестовые программы, описанные на универсальных языках с использованием средств среды моделирования.

Во всех перечисленных выше подходах возможен перехват внутренних событий среды моделирования, через которые производится взаимодействие с моделью устройства. В [36] авторы выделяют следующие основные операции, которые требуется перехватывать:

- команды управления:
  - считывание значения сигнала;
  - установка значения сигнала (форсирование);
  - однократное изменение сигнала;
- периодическое изменение по заданному закону (clk);
- управление временем (запуск прогона, ожидание);
- отладочные команды:
  - вставка отладочных точек в скрипты;
  - добавление триггеров (вызов обработчика на событие).

Для создания переносимого тестового окружения достаточно строить тесты на основе указанной выше группы примитивов. Для второй группы

может потребоваться доработка тестовой инфраструктуры путём встраивания дополнительных ТА. Для организации тестового окружения в этом случае предлагается следующий подход:

1. Однократный прогон тестов в системах моделирования со сбором информации о наблюдаемых и управляемых сигналах.
2. Формирование списка сигналов в системе, которые должны управляться и наблюдаться в соответствии с тестовым алгоритмом.
3. Генерация тестовой инфраструктуры в соответствии с предложенным в параграфе 4.2 алгоритмом и сформированным списком сигналов.

Такой подход обеспечивает корректный запуск внутрисхемных тестов только в случае фиксированных тестовых наборов и полной управляемости устройства. Иными словами, требуется повторяемость результатов при одинаковых исходных условиях и управляющих сигналах теста.

При встраивании средств тестирования в устройство дополнительно к СТ могут генерироваться наборы тестовых программ для последующей верификации устройства. Данная задача рассматривается в [65], [51], [39] и других работах. Поскольку базовый набор операций включает универсальный язык программирования, то он может быть использован для взаимодействия с внешними инструментариями и формирования необходимых им данных, что позволяет связывать средства проектирования в единый маршрут. Особенно это актуально в случае совместного проектирования аппаратной и программной составляющих тестовой инфраструктуры.

4.5. Выводы по главе

В четвёртой главе предложены подходы к встраиванию средств внутрисхемного тестирования в устройства на основании модели устройств, являющейся производной версии гибридной метамодели устройств и построенной в соответствии с предложенной методикой специализации гибридной метамодели. Построенная модель может быть применена для

широкого класса задач в областях внутрисхемного тестирования и верификации устройств. Для предложенной модели предложены:

- архитектура тестового агента, позволяющего решать выбранные задачи внутрисхемного тестирования однокристальных цифровых устройств;
- метод построения тестовой инфраструктуры в устройства по описаниям наблюдаемых и управляемых сигналов;
- метод проведения совместного тестирования модели устройства в среде моделирования и аппаратного прототипа.

Поскольку предложенные подходы основан на гибридной метамодели с минимальной её специализацией, то он может быть применён для различных форматов исходных описаний, импортируемых в модель устройства. Сложность генерируемой тестовой инфраструктуры определяется количеством наблюдаемых и управляемых сигналов, а также требованиями к скорости выполнения тестов, что влияет на организацию коммуникаций между тестовыми агентами и СУТ.

В следующей главе приведены результаты прототипирования предложенных методик, которые подтверждают их применимость для реальных устройств и систем на кристалле. Для решения задач встраивания средств тестирования разработан прототип САПР, который также решает рассмотренные в данной главе задачи и может быть использован в типовых маршрутах проектирования на предприятиях. Также рассмотрено применение разработанных моделей и методов для других классов задач реинжиниринга.

5. ПРОТОТИПИРОВАНИЕ И ВНЕДРЕНИЕ ПРЕДЛОЖЕННЫХ МОДЕЛЕЙ И МЕТОДИК

В главе описаны результаты апробации гибридной модели и методик её обработки на примерах частных задач внутрисхемного тестирования и реинжиниринга цифровых устройств. Подтверждена практическая применимость результатов исследования, разработан прототип инструментария для автоматизации задач реинжиниринга. Проведён анализ результатов и рассмотрены возможные пути развития исследования.

5.1. Построение прототипа CAP цифровых устройств

Для апробации предложенных подходов в рамке исследования разработан прототип CAP, реализующий гибридную модель представления устройств и описанные в диссертации методики. Прототип представляет собой расширяемый инструментарий PHRT (Programmable Hardware Reengineering Toolkit) и набор расширений для частных задач реинжиниринга (в том числе для встраивания средств тестирования). В данном параграфе сформированы требования к CAP и предложена его архитектура. Подробное описание CAP приведено в приложении С.

5.1.1. Требования к CAP на основе гибридной метамодели

Любая CAP реализует средства для проведения реинжиниринга, которые основаны на предложенных модели и методиках её обработки. В случае гибридной модели к прототипу предъявляются следующие требования:

- возможность интеграции в существующие на предприятиях маршруты проектирования электронных устройств;
- поддержка наиболее распространённых HDL: VHDL, Verilog, SystemC и SystemVerilog;
- возможность повторного использования разработанных алгоритмов реинжиниринга (поддержка пользовательских библиотек);
- наличие интерфейс прикладного программирования (API);

- возможность интеграции расширений с методами обратного вызова, присутствующими в базовом наборе операций с моделью;
- работа в рамках заданных ограничений по системным ресурсам и временным характеристикам.

Ниже рассмотрена реализация требований в разработанной архитектуре прототипа.

Встраиваемость в маршруты проектирования

Актуальна задача интеграции САР с другими инструментариями, используемыми в маршрутах проектирования СнК. Одним из путей является его полное включение (встраивание) в некоторую среду разработки. Это позволяет при реинжиниринге использовать ресурсы самой среды, прежде всего, средства синтеза и анализа. Возможна как жёсткая интеграция средства со средой, так и динамическое подключение (механизмы плагинов и расширений). При разработке требуется лишь обеспечить возможность подобного использования.

Наличие интерфейса прикладного программирования (API)

Разрабатываемое средство реинжиниринга может быть использовано в различных процессах разработки, использующих свои среды проектирования. Нужно предоставить интерфейс прикладного программирования (API), через который САПР смогут взаимодействовать с разрабатываемым САР. При наличии API, возможно будет автоматизировать не только сам реинжиниринг, но и вызов данного процесса из высокоуровневых САПР, пример приведён на рис. 5.6. Выявив узкие места по производительности в анализаторе, возможно из этого же средства вызвать САР с необходимыми параметрами, чтобы оптимизировать архитектуру и выполнить повторный анализ (рис. 5.).

Поддержка механизма событий

Управление преобразованиями производится через API, передача данных между шагами алгоритма реинжиниринга осуществляется через метаданные гибридной модели. Также возникает проблема динамической верификации

модели при преобразованиях. При отправке команды “встроить в устройство отказоустойчивые блоки памяти” САР потребуется вернуть информацию обо всех изменённых элементах, после чего внешнее средство должно будет обновить свои внутренние представления.

Подобный подход кажется не очень удобным, и для решения проблемы предлагается ввести в средство реинжиниринга дополнительный интерфейс, который будет работать в направлении “САР→клиентское средство”. Через данный интерфейс внешнее средство будет извещаться об изменениях в представлении и других аналогичных событиях. Использование подобного механизма позволит разбить обработку результатов выполнения команды на составляющие, что упростит разработку. Также станет возможным возврат информации в клиентское средство до завершения обработки, что удобно при выполнении длительных операций.

На рис. 5.2 приведён пример для случая, когда взаимодействующее с САР средство отображает структуру внутреннего представления. При этом синхронизация двух представлений производится через механизм событий.

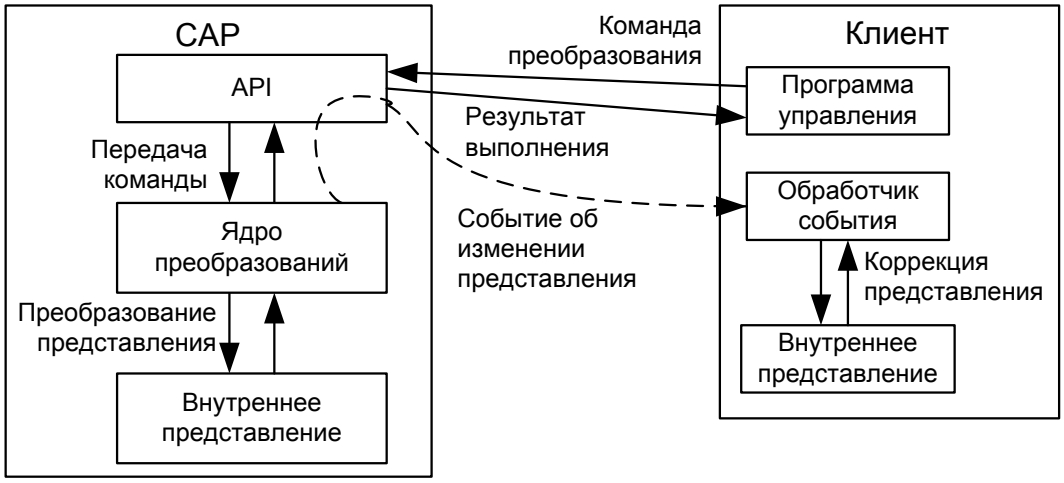


Рис. 5.1. Пример взаимодействия клиентского приложения со средством реинжиниринга через API

Поддержка пользовательских библиотек

При выполнении реинжиниринга зачастую требуется встроить в устройство новые компоненты (блоки в терминах гибридной модели), отсутствующие в описании исходного устройства (например, при переносе

реализации между различными целевыми платформами или встраивании тестовых агентов). Для повторного использования компонентов необходимо, чтобы разработчик имел возможность группировки элементов в некоторые описания, которые можно было бы сохранить отдельно от проекта и в дальнейшем повторно использовать в средстве реинжиниринга. Подобные элементы предлагается называть пользовательскими библиотеками. В рамках гибридной метамодели подобные элементы могут быть реализованы через элементы библиотек, которые могут хранить произвольную информацию.

Поддержка проектов

При формировании требований к функциональности было сказано, что должен быть режим последовательного выполнения команд. Если подобный режим используется пользователем, то для него была бы удобной возможность сохранения текущего контекста средства: внутреннего представления, подгруженных библиотек и прочего динамического окружения. Не все составляющие специализированного внутренней модели могут быть отражены на внешнее. Примером является наследование элементов, поддержка которого отсутствует в языках VHDL и Verilog. Также при проведении автоматизированного реинжиниринге с участием человека может потребоваться сохранение внутреннего контекста в промежуточном незавершённом состоянии, когда внутреннее представление не проходит валидацию.

Исходя из перечисленного, было бы удобно в средстве реинжиниринга организовать экспорт контекста в некий формат, который можно было бы легко экспортировать и импортировать. Данный формат, по аналогии с существующими интегрированными средами разработки (IDE – Integrated Development Environment), предлагается называть “проектами”.

5.1.2. Архитектура CAP устройств

Ключевым требованием к архитектуре CAP цифровых устройств на базе гибридных моделей является её расширяемость и встраиваемость, поэтому

предлагаемая архитектура во многом повторяет принципы построения современных IDE, где реализуются схожие механизмы расширения функциональности.

Для построения САР предлагается использовать модульный подход. При этом формируется некоторое общее “ядро”, которое включает минимально необходимую функциональность средства и программную реализацию гибридной модели устройства. Всё остальное выносится в дополнительные модули-расширения, которые могут загружаться ядром в случае необходимости. Специализация гибридной модели осуществляется путём средств расширения, обеспечиваемых программной реализацией модели.

Ядро и расширения вместе образуют средство реинжиниринга, которое, в свою очередь, может быть использовано в пользовательских целях: интегрировано с САПР, адаптировано для специфических задач и т.п. На рис. 5.2 приведена схема САР, основные составляющие описаны ниже.

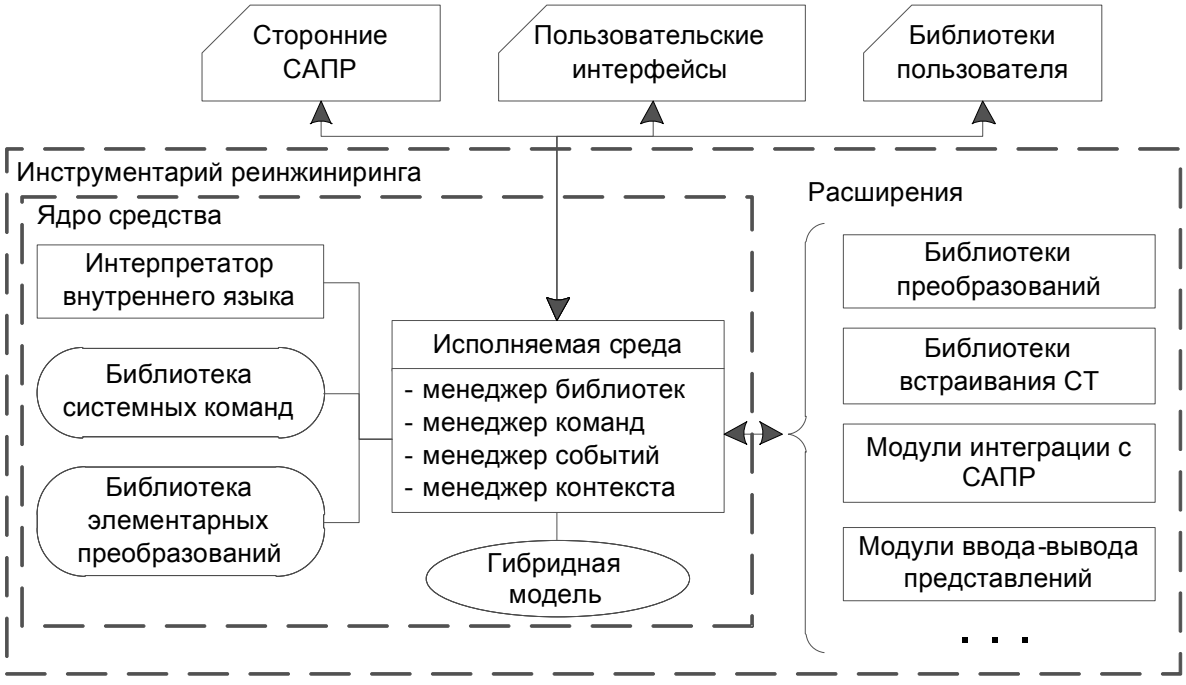


Рис. 5.2. Предлагаемая архитектура САР

Ядро САР включает в себя программную реализацию предложенной гибридной модели устройств. В соответствии с подходами к специализации модели из параграфа 3.1, предлагается реализовать модель с использованием

методологий объектно-ориентированного программирования. Подобная реализация позволяет расширить возможности специализации для частных задач реинжиниринга, тем самым повысив эффективность применения предложенных подходов для решения частных задач реинжиниринга устройств.

Для выполнения программ реинжиниринга в ядро САР включены средства поддержки языка работы с моделью: библиотека базовых операций над моделью и интерпретатор базового языка программирования (см. параграф 2.5). В ядро также включён ряд команд для управления инструментарием (загрузка расширений, останов преобразований и т.д.). Вызов операций и их связь с моделью устройства осуществляется исполняемой средой, которая хранит контекст системы и обеспечивает передачу информации между компонентами системы.

Подробное описание архитектуры САР приведено в приложении С.

5.1.3. Встраивание СПР в маршруты проектирования

Современные маршруты проектирования цифровых устройств предполагают использование систем автоматизации проектирования на всех шагах проектирования. Таким образом, задачи реинжиниринга цифровых устройств также решаются посредством САПР, которые реализуют инструментарий: модели устройств, методы работы с ними и СПР. Для интеграции САПР реинжиниринга с остальными средствами в маршруте проектирования требуются специальные интерфейсы.

В средствах реинжиниринга предусмотрено два типа интерфейсов: интерфейсы прикладного программирования (ИПП, API – Application Programming Interface) и внутренние интерфейсы для расширений. ИПП, в соответствии со сформированными в подпараграфе 5.1.1 требованиями должны обеспечить передачу команд в САР и возвращение результатов. Также должен присутствовать механизм для оповещения внешнего средства о возникновении событий. Внутренние интерфейсы используются для предоставления расширенного доступа к функциональности библиотекам ядра. Основной

задачей интерфейсов ядра является предоставление контролируемого доступа к менеджерам исполняемой среды, которые обеспечивают исполнение передаваемых команд.

Процесс реинжиниринга включает многие операции из стандартных маршрутов проектирования устройств (синтез, анализ, моделирование). Актуальна задача повторного использования существующих САПР при решении задач реинжиниринга. В качестве примера для рассмотрения выбрана САПР Quartus II компании Altera, поддерживающая все стадии разработки устройств для ПЛИС [15].

Интеграция разрабатываемого средства реинжиниринга с САПР позволяет использовать встроенные средства синтеза и анализа, а также использовать САПР как транзитную программу при доступе к средствам моделирования. На рис. 5.3 средство реинжиниринга встраивается в маршрут проектирования, используя результаты первого синтеза в САПР и порождая синтезируемые исходные коды для повторного синтеза с оптимизацией.

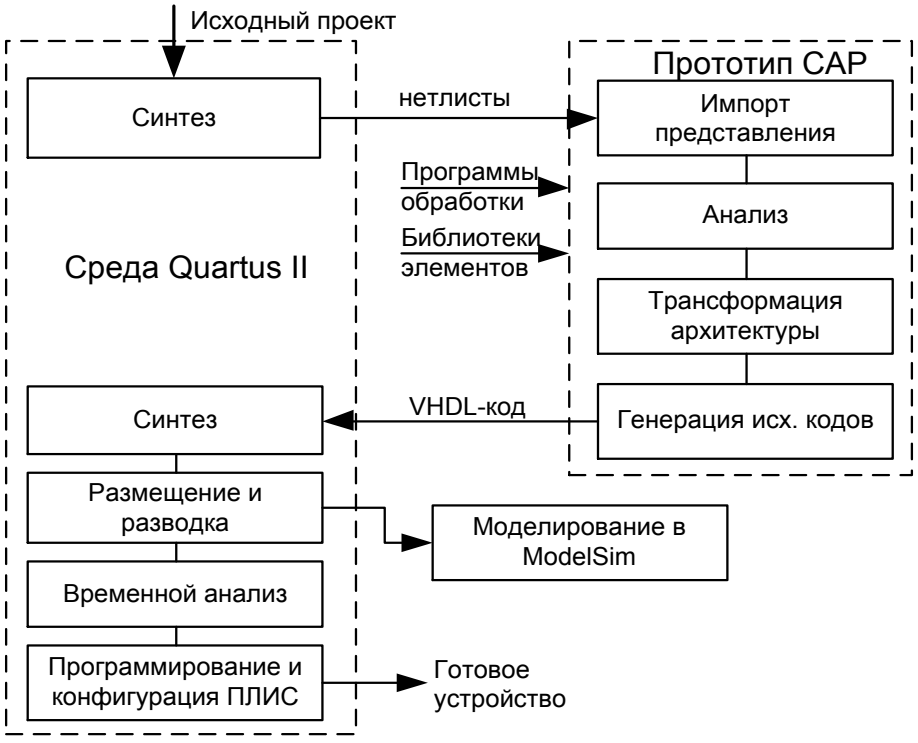


Рис. 5.3. Включение прототипа PHRT в маршрут проектирования цифровых устройств на ПЛИС в САПР Quartus II

На рис. 5.4 приведена итерационная схема реинжиниринга, в которой производится анализ результатов и принимается решение о повторном проведении реинжиниринга для выполнения поставленных требований.

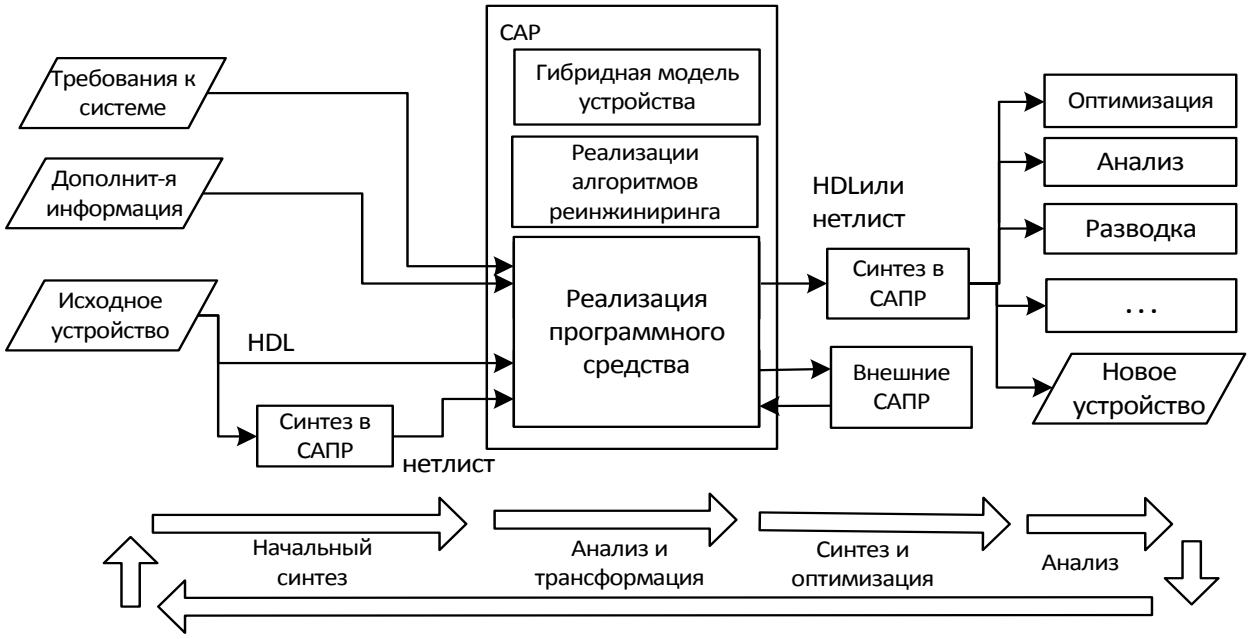


Рис. 5.4. Включение реинжиниринга в маршрут проектирования устройств

5.2. Построение интегрированной среды разработки на базе PHRT

Предложенные подходы внедрены в ряде проектов, посвящённых разработке специализированных СнК. Они успешно опробованы на сложных системах на кристалле, включающих процессорные ядра LEON3 и OpenRISC 1200. Данные процессоры обладают открытыми исходными кодами на VHDL и Verilog, что позволяет использовать их в исследовательских экспериментах, связанных с модификацией процессорного ядра [69].

Реализация подхода в САР позволила автоматизировать встраивание СТ во внутренние компоненты устройства (регистровая память, АЛУ) без модификации исходных описаний. На рис. 5.5 приведена структура системы с тестовой инфраструктурой, генерируемой прототипом САР. В предлагаемой реализации IDE пользователь редактирует исходные описания устройств на HDL и задаёт тесты в редакторе тестовых векторов или VHDL. После

подготовки тестов пользователю достаточно нажать кнопку в графическом интерфейсе, после чего система автоматически выполнит инструментирование устройства, запустит тесты на ПЛИС и отобразит в графическом интерфейсе системы. На рис. 5.6 приведен специализированной САР, построенного на базе интегрированной среды разработки Eclipse и ранее созданного прототипа САР.

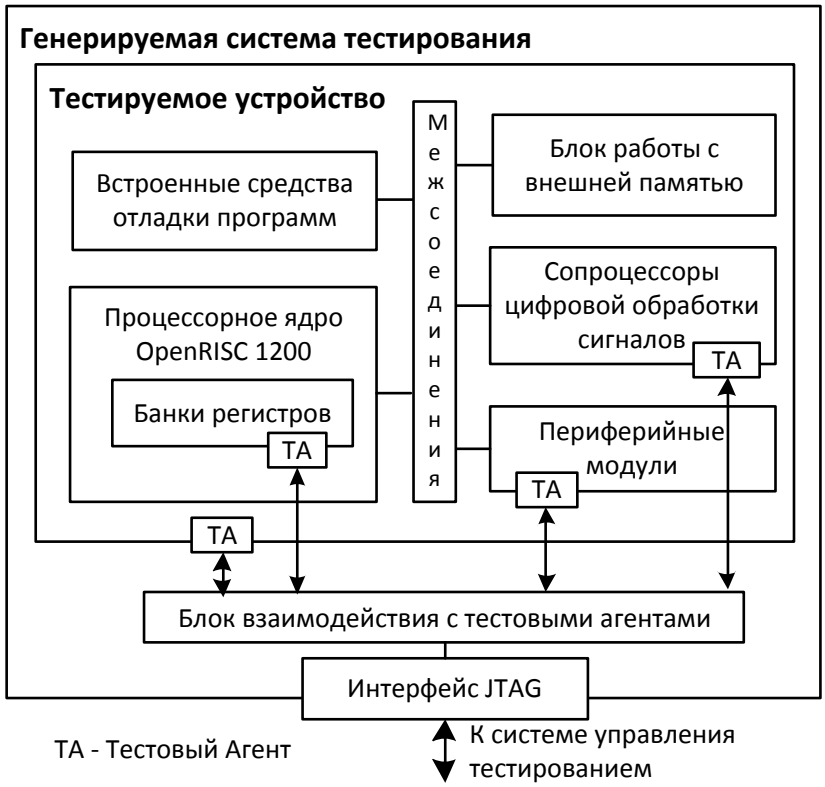


Рис. 5.5. Пример встраивания тестовых агентов в СнК на основе OpenRISC 1200

В таблице 5.1 приведены сравнительные результаты ручного и автоматизированного инструментирования для системы на кристалле, включающей минимальный набор периферии и процессорное ядро OpenRISC 1200 с внешней памятью программ и данных. Восемь тестовых агентов встраиваются в интерфейсы устройства и в несколько компонентов на разных уровнях иерархии (рисунок 5.5), ТА подключают к СУТ 68 управляемых сигналов и 84 - наблюдаемых. В качестве целевой платформы использованы отладочные платы на базе ПЛИС Cyclone IV компании Altera [47].

Рис. 5.6. Процесс проведения тестирования устройства при помощи интегрированной среды на базе САР

Таблица 5.1

Результаты встраивания средств тестирования в СнК на базе OpenRISC 1200

Эксперимент	F_{max}, MHz	LCELL	Число регистров	Блочная память, бит
Исходное устройство	39,31	8055	3105	53888
Тестовые интерфейсы	56,14	516	270	0
Ввод-вывод в САР без преобразований	37,18	8247	3105	53888
Ручная модификация	39,15	8752	3417	53888
Автоматическая модификация	37,02	8916	3417	53888

По приведённым в таблице 5.1 данным видно, что автоматическое встраивание средств тестирования требует больше системных ресурсов и приводит к снижению максимальной частоты устройства. Результат обусловлен восстановлением низкоуровневой архитектуры из нетлиста из-за “сквозных” соединений между уровнями иерархии, что приводит к отличиям в исходных и

выходных нетлистах даже при отсутствии преобразований в PHRT. В свою очередь это приводит к разным результатам оптимизации при синтезе устройства в САПР Quartus 2. Различия в результатах синтеза составляют не более 5% от общих системных ресурсов, что приемлемо для тестовых прототипов устройств. Для улучшения показателей требуется доработка используемых в прототипе PHRT методик импорта и объединения нетлистов.

5.3. Контроль устойчивости СнК к однократным сбоям памяти

В данном параграфе описаны результаты применения разработанной модели и методик для задачи тестирования устойчивости памяти устройств к однократным сбоям, что актуально для устройств, подверженных воздействию заряженных частиц. Совместно с О.В. Мамутовой разработан подход к тестированию памяти с использованием встроенных процессорных ядер. Подробно подход описан в [48].

Предлагаемый подход основан на использовании подключаемых к блокам памяти средств внесения сбоев ("саботажников") под управлением встроенного процессора, позволяющих минимизировать дополнительные аппаратные затраты. По сравнению с подходами ко встраиванию СТ из параграфа 4.1, данный подход позволяет исключить дополнительные затраты на СУТ и тестовые интерфейсы, так как возможно использовать имеющиеся в СнК шины данных процессора (рис. 5.7).

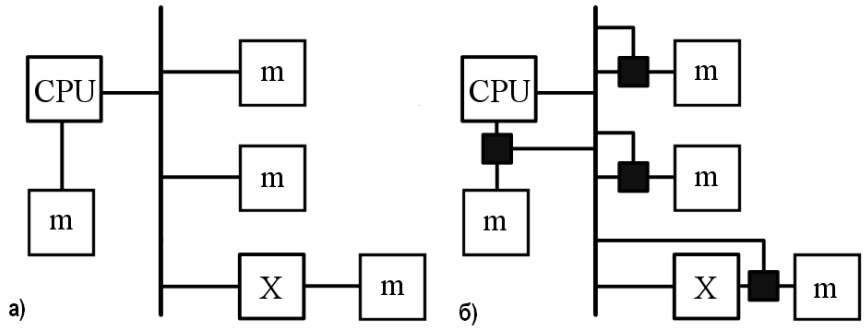


Рис. 5.7. Встраивание саботажников и подключение к системной шине (а – система до встраивания, б – после)

Система внесения сбоев в память может быть обозначена следующим множеством:

$FI_M = (M, S, I_{CPU}, C, A_{SW})$, где (5.1)

- $S_M = (S, I_{CPU}, C)$ M - множество оснащаемых блоков памяти;
- S – множество саботажников, подключенных к блокам памяти M;
- ICPU – интерфейс процессорного ядра;
- C – множество соединений между саботажниками S и интерфейсом ICPU;
- ASW – программные реализации алгоритмов проведения экспериментов.

Предложен следующий порядок действий при встраивании средств внесения неисправностей в систему:

1. Выбор в иерархии проекта части системы, подлежащей исследованию.
2. Поиск блоков памяти *M* в выбранной части системы.
3. Оснащение выбранных блоков памяти *M* саботажниками *S*.
4. Подключение саботажников *S* к интерфейсу процессора I_{CPU} посредством множества соединений *C*.

Поиск блоков памяти в устройстве осуществляется методами, описанными в параграфе 4.1. Критерии поиска определяются задачами тестирования. Оснащение блоков памяти саботажниками S и их подключение к I_{CPU} производится аналогично алгоритму встраивания тестовых агентов, рассмотренном в том же параграфе. Особенностью подхода является то, что для проведения самодиагностики используются встроенные в устройства процессорные блоки.

Реализация алгоритма оснащения памяти

Существующие расширения PHRT позволили решить следующие задачи, возникающие при встраивании саботажников в исследуемую систему:

- импорт/экспорт данных в формате EDIF, VHDL, XML;
- базовые операции над гибридной моделью: добавление и удаление элементов, переименование, доступ к свойствам и пр.;
- соединение компонентов на разных уровнях иерархии;

– анализ: оценка площади кристалла, временной анализ и пр.;
– интеграция с САПР Quartus II для прототипирования на ПЛИС: импорт/экспорт проектов, частичный синтез компонентов устройства.

Для реализации алгоритма оснащения для PHRT разработано расширение memfault_inject. На рис. 5.8 приведена схема применения разработанного расширения. Все приведённые шаги выполняются автоматически под управлением сценариев на языке TCL. Данные сценарии не входят в состав расширения и могут быть адаптированы пользователями для используемых ими конфигураций устройств и памяти.

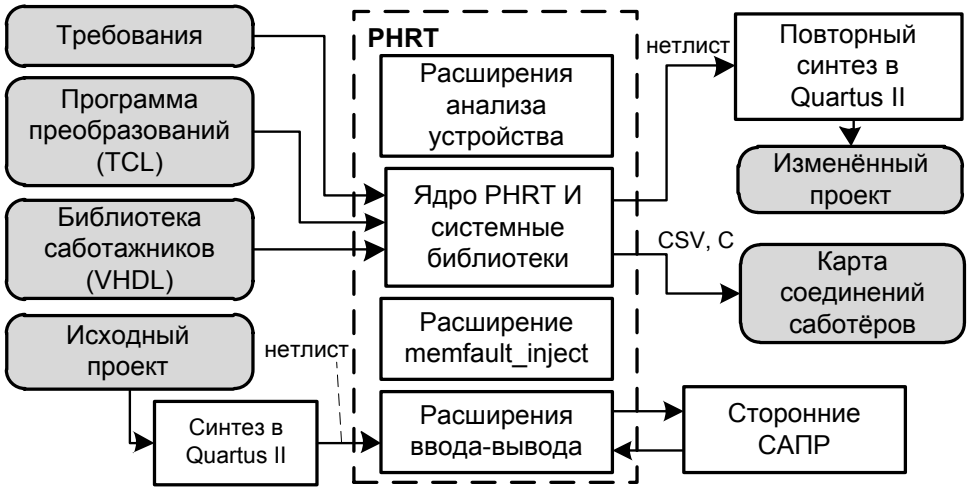


Рис. 5.8. Процесс преобразования устройства при встраивании саботажников

Процесс начинается с синтеза исходного устройства в САПР Quartus II и получения низкоуровневого структурного описания. Затем осуществляется импорт этого описания в PHRT и построение гибридной модели на базе исходного высокоуровневого описания и полученного низкоуровневого описания. После этого программа производит поиск блоков памяти по имени (“altsyncram”) и добавляет саботажники ко всем требуемым блокам. Далее все саботажники подключаются к внешнему интерфейсу управления, связанному с процессорным ядром. На рис. 5.9 приведена часть сценария, который реализует перечисленные задачи посредством вызова операций из расширений PHRT.

Для выполнения модификации устройства в сценарии вызывается расширение memfault_inject. Расширение оперирует параметризованным компонентом саботажника на языке VHDL и включает операции добавления

саботажника к заданному интерфейсу памяти. Саботажник имеет статические параметры, поэтому САР осуществляет частичный синтез элемента с помощью внешней САПР, получая в результате структурное описание саботажника с указанными значениями параметров. Далее происходит оснащение блоков памяти из списка и подключение саботажников к процессору. В конце преобразования САР формирует список адресов саботажников в адресном пространстве процессора, используемый для написания программы экспериментов по внесению неисправностей.

```
# Load extensions and import device
phrt::system::load_extensions \
    {quartus vhdl_netlist memfault_inject }
phrt::vhdl_netlist::import openRISC1200.vhd /test/

# Find memory by mask and instrument it
set mem_list \
    [phrt:find_refs /test/or1200_top -R ALTSYNCRAM]
phrt::memfault_inject:instrument $mem_list

# Output modified netlist
phrt::vhdl_netlist::export /test out.vhd
```

Рис. 5.9. Пример сценария для оснащения памяти средствами внесения однократного сбоя посредством PRHT

Результаты исследования

Полученные результаты подтверждают применимость предложенного подхода для оценки устойчивости системы к сбоям в памяти для реальных процессорных систем. Апробация на реальных устройствах подтвердила, что подход обеспечивает высокую эффективность по аппаратным ресурсам и скорости проведения экспериментов. Предложенная в параграфе 4.1 методика встраивания СТ была успешно применена для микропроцессорных СнК.

Корректность преобразований подтверждена за счёт верификации системы посредством различных наборов тестов, в том числе специализированных тестовых наборов для верификации встроенной памяти и банков регистров процессорного ядра. Результаты работы применяются для оценки устойчивости

цифровых систем к однократным сбоям памяти в соответствии с рассмотренными в [7] и [6] подходами.

5.4. Применение средств реинжиниринга устройств в маршрутах проектирования с непрерывной интеграцией

Непрерывная интеграция – современная методология, используемая при проектировании сложных систем, состоящих из большого числа компонентов. Отличительной особенностью подхода является регулярное выполнение функциональных и интеграционных тестов систем параллельно с разработкой. Это позволяет снизить время обнаружения ошибок и тем самым снизить стоимость их исправления и стоимость дальнейшего сопровождения продукта [84]. Поскольку регулярное выполнение задач реализации компонентов системы и проведения тестов требуют больших трудозатрат, то используются специальные САПР, называемые системами непрерывной интеграции.

Одной из наиболее популярных систем непрерывной интеграции является система Jenkins CI, представляющая собой распределённое клиент-серверное приложение с возможностью запуска задач на различных машинах. Система Jenkins CI широко применяется при проектировании аппаратных средств; особенности применения системы для задач тестирования прототипов на ПЛИС рассматривались д'Анжу и Фитчем в [79, 83]. Греем и МакГрегором также рассмотрено применение Jenkins CI для автоматизации полного цикла проектирования систем. В рамках работы на предприятиях автором диссертации было реализовано несколько маршрутов непрерывной интеграции с использованием Jenkins CI, основные результаты и рекомендации по применению данной системе при проектировании и реинжиниринге аппаратных средств изложены в [85].

На рис. 5.10 приведена схема подключения агента Jenkins CI к отладочной плате с ПЛИС XUPV5-lX110T компании Xilinx [87]. При этом используется два интерфейса для прошивки конфигурации ПЛИС через JTAG и последующего подключения тестового интерфейса. Подсистема тестирования генерируется

при помощи PHRT с использованием алгоритмов, предложенных в четвёртой главе. Трансформацию устройств в PHRT, последующий синтез устройства и тестирование предлагается запускать как задачи Jenkins.

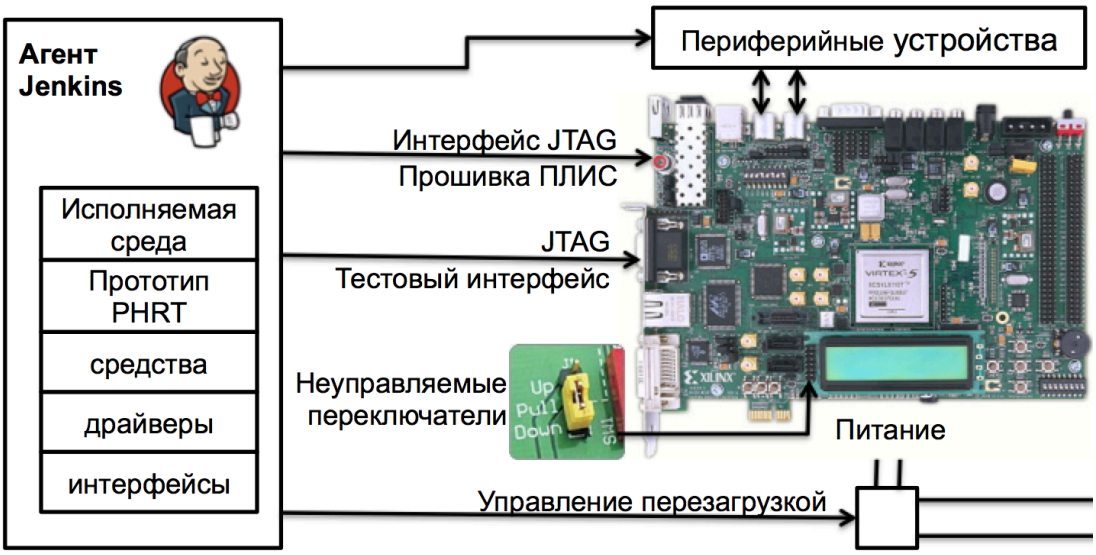


Рис. 5.10. Подключение агента системы непрерывной интеграции Jenkins CI к отладочной плате с ПЛИС

Для эффективной интеграции PHRT в систему непрерывной интеграции Jenkins CI разработан расширение системы Jenkins, который позволяет запускать скриптовый язык PHRT как один из шагов сборки. При этом не требуется развёртывание дополнительных инструментариев и расширений PHRT, так как ядро PHRT встраивается в плагин. Разработанное расширение обладает следующими характеристиками:

- возможность встраивания в маршруты верификации аппаратных продуктов на базе системы непрерывной интеграции Jenkins CI;
- использование инструментария реинжиниринга устройств PHRT, основанного на гибридной модели представления устройств;
- возможность автоматического встраивания средств внутрисхемного тестирования и тестовых интерфейсов в прототипы на ПЛИС;
- возможность описания алгоритмов анализа, преобразования и верификации проектов с помощью скриптового языка Tcl;
- возможность вызова внешних САПР для выполнения операций синтеза и прошивки устройств на ПЛИС;

- возможность экспорта синтезируемых описаний в формате EDIF после выполнения преобразований;
- поддержка версий системы Jenkins CI 1.509.3 и выше.

На рис. 5.11 приведён алгоритм, реализующий реинжиниринг и внутрисхемное тестирование устройства с использованием прототипа PHRT в системе Jenkins CI, которая контролирует условия запуска автоматической сборки и управляет процессом тестирования. Условием начала алгоритма могут быть ручной запуск пользователем, срабатывание таймера или появление новой версии исходных кодов устройства в системе контроля версий.

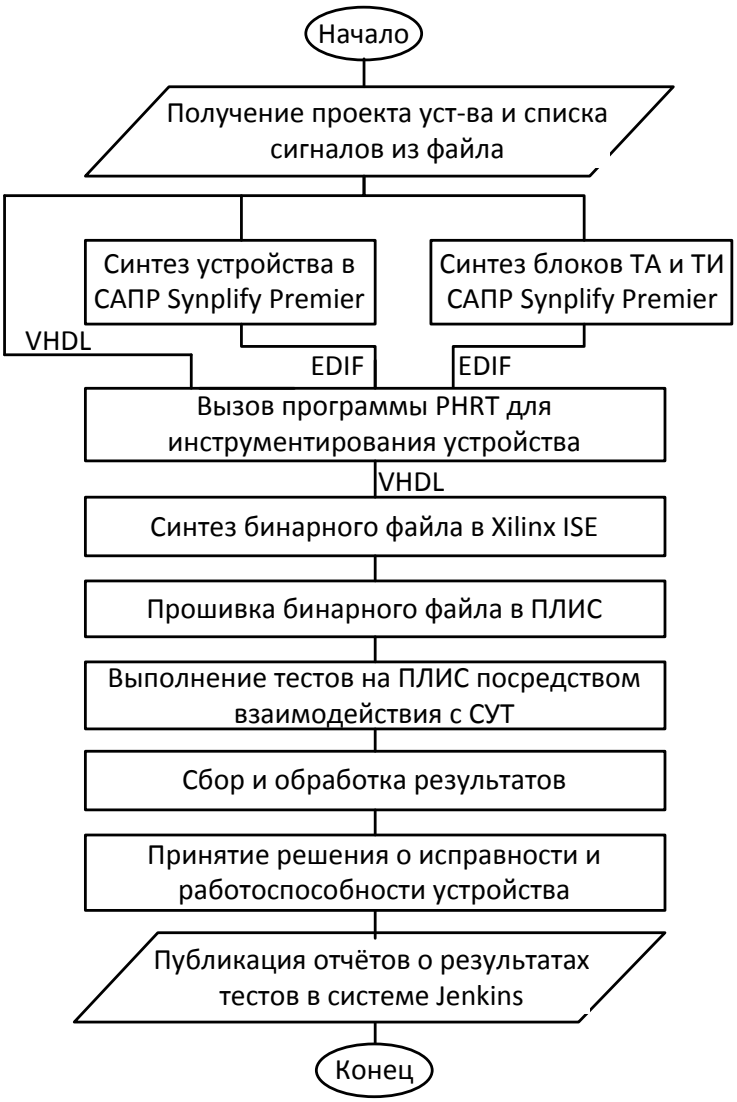


Рис. 5.11. Алгоритм проведения тестирования устройства на ПЛИС в системе непрерывной интеграции Jenkins CI

Приведённый пример демонстрирует один из наиболее распространённых случаев запуска цикла непрерывной интеграции в системе Jenkins. В случае успешного выполнения тестов отчёты и полученные бинарные файлы могут быть опубликованы в системе и переданы в отделы тестирования для выполнения детальных проверок. При этом возможно полностью автоматизировать внутрисхемное тестирование и снизить время обнаружения ошибок за счёт частого проведения тестов и возможности триангуляции причины ошибки по полученным отчётам. Предложенный выше подход к непрерывной интеграции был внедрён в ООО «Синопсис СПб», где подтвердил свою эффективность. Результаты подтверждены актом о внедрении.

5.5. Возможности применения модели и методов других областях реинжиниринга устройств

Разработанная модель и методики могут быть применены для различных задач реинжиниринга цифровых устройств. В соответствии с поставленными ограничениями модель предназначена для структурных описаний, что может затруднить её применение для ряда задач, связанных с высокоуровневыми и поведенческими описаниями. Элементы функций и метаданные позволяют применить модель для решения подобных задач.

Перенос элементов модели между представлениями

Независимость от языка описания устройства является одним из главных свойств гибридной модели. Расширяемость модели позволяет вводить новые элементы, которые не являются переносимыми между различными описаниями. Актуальна задача поддержки подобных элементов в универсальных библиотеках (например, в расширениях PHRT для ввода-вывода нетлистов). Предлагается следующий подход:

1. Для большинства задач достаточно использовать **функции**, для которых возможно конвертация в требуемое представление посредством частичного синтеза (“динамическая генерация” недостающих элементов).

2. При использовании **блоков** аналогичная функциональность может быть реализована через регистрацию обработчика изменений в гибридной модели с последующей обработкой в алгоритме реинжиниринга.
3. Перед выводом устройства все метаданные должны быть сконвертированы средствами программы управления реинжинирингом.

Рефакторинг цифровых устройств

В некоторых источниках рефакторинг понимается как частная задача реинжиниринга, но при этом он требует сохранения информации об устройстве, которая не участвует в синтезе устройств. Выделяют следующие категории подобной информации: комментарии к исходным кодам форматирование исходных описаний устройства, порядок следования элементов в описаниях и несинтезируемые компоненты (например, описанные на VHDL тесты).

Предлагаемая модель не предоставляет элементов для хранения перечисленной информации, но может быть расширена следующим образом:

1. Вся информация хранится в метаданных элементов.
2. Модификация метаданных осуществляется по мере выполнения преобразований модели за счёт обработчиков обратного вызова.
3. При экспорте описания устройства из модели метаданные используются для восстановления исходных описаний.

Предотвращение восстановления архитектуры из нетлистов

Нетлисты IP-ядер наиболее уязвимы к реверс-инжинирингу, поэтому актуален вопрос защиты нетлистов от подобных действий [27]. Типовым подходом является искажение (“обфускация”) исходных кодов на HDL и нетлистов, при помощи которой вносятся нефункциональные изменения как в само описание (переименование компонентов, разделение шин на сигналы, и т.д.), так и в архитектуру устройства (удаление иерархии, оптимизация логических цепей, внесение средств контроля памяти). Подобные изменения затрудняют извлечение архитектуры устройств, тем самым ограничивая возможности их реверс-инжиниринга.

Предложенные в работе методики восстановления архитектуры устройств из низкоуровневых описаний могут быть применены для извлечения информации из поставляемых третьими сторонами готовых блоков (“IP-ядер”) и последующего внесения в них изменений. Например, в подпараграфе 3.2.1 для нетлистов рассмотрены подходы к устранению сквозных сигналов и удалению согласующих каскадов. Сформированы следующие рекомендации для тех IP-блоков, где используются внешние САПР синтеза и моделирования:

1. Шифрование исходных описаний и нетлистов при передаче между САПР в маршруте проектирования. При отсутствии такой возможности, необходима обфускация нетлистов.
2. Искусственное нарушение иерархии элементов в устройстве посредством введения сквозных ссылок.
3. Переименование всех сигналов устройства, удаление комментариев.
4. Использование особенностей интерпретации HDL целевыми средствами синтеза для формирования описаний, несинтезируемых в других САПР.
5. Встраивание средств шифрования и контроля критических для безопасности системы данных посредством рассмотренных в главе 4 методик встраивания средств тестирования и самодиагностики.

Для защиты от применения предложенных методик восстановления архитектуры (пункт 2 списка) рекомендуется использовать случайно генерируемые замкнутые контуры из сквозных соединений и нефункциональных логических элементов, подключённых к логическим сигналам устройства. Подобная конфигурация компонентов противоречит условиям целостности модели, и архитектура не может быть восстановлена в общем случае. Алгоритмы реверс-инжиниринга придётся разрабатывать индивидуально для требуемых типа устройств и формата нетлистов.

Предлагаемые действия по обфускации нетлистов и исходных описаний могут быть выполнены как шаг реинжиниринга устройств.

5.6. Анализ результатов апробации подходов

По результатам апробации и внедрения разработок подтверждена применимость разработанных моделей и методик для решения частных задач реинжиниринга устройств и встраивания СТ. Тем не менее, выявлен ряд ограничений, которые затрудняют её использования для произвольных задач реинжиниринга устройств. Данные ограничения рассмотрены ниже. Также предложены дальнейшие пути развития работы с целью расширения области применения разработанных моделей и методик.

Гибридная модель является основой всех предлагаемых в работе методик и алгоритмов реинжиниринга СнК, при разработке и апробации которых выявлены следующие ограничения модели:

1. Специализация модели имеет ряд ограничений, которые затрудняют её применение для произвольных устройств (см. параграф 2.1).
2. Недостаточное внимание в модели уделено поддержке высокоуровневых HDL (SystemC, SystemVerilog). Значительная часть информации должна представляться в виде метаданных или функций.
3. Сложность работы с сигналами и соединениями при работе с шинами сигналов и при подключении сигналов к шинам и блокам.

При разработке гибридной метамодели в параграфе 2.1 введены ограничения, которые сужают класс задач реинжиниринга, для которых могут быть применены предлагаемые в работе модели и методики. Несмотря на формальную совместимость модели с HDL и частично успешную апробацию модели для задач реинжиниринга СнК с использованием языков SystemC и SystemVerilog, полученные результаты свидетельствуют о недостаточной поддержке языковых конструкций из данных HDL. Возможна специализация модели для поддержки данных языков в модели, но при дальнейшей разработке целесообразно рассмотреть расширение типизации модели.

Разработанная структура сигналов и соединений в модели проста для задач анализа устройств, так как на низком и высоком уровнях предоставляется

информация обо всех соединениях. Однако, для сложных топологий в модели отсутствуют средства для поддержки таких операций, как объединение сигналов в шину, изменения размерности шины сигналов и т.д. Реализация всей логики ложится на алгоритмы реинжиниринга, что приводит к высокой сложности операций и риску ошибок. Проблема может быть решена путём доработки модели или разработки общих методик, которые будут использоваться в расширениях набора операций.

5.7. Выводы по главе

В целом модель показала свою применимость для рассмотренных классов задач: встраивания средств тестирования, внесения структурной избыточности и других преобразований. Основным направлением возможного развития исследований является расширение областей применения гибридной модели и предложенных методик для решения задач реинжиниринга

Совместный реинжиниринг поведенческих и структурных описаний

Разработанная гибридная модель поддерживает поведенческие описания через блоки функций. Однако, такой подход затрудняет реализацию алгоритмов реинжиниринга, направленных на прямой анализ и преобразование поведенческих описаний без частичного синтеза. При помощи специализации моделей можно предоставить дополнительные элементы для поведенческих описаний, но при этом усложняется задача реинжиниринга из-за необходимости одновременной обработки нескольких типов представлений.

Наибольшую проблему вызывает анализ связей сигналов в дереве. Если в структурном описании связи фиксированные, то для поведенческого представления они возникают в зависимости от условий. Не проводя частичный синтез элементов из поведенческого описания, тяжело реализовать совместный анализ, причём задача ложится на алгоритмы реинжиниринга. Сделан вывод, что в чистом виде использование двух равноправных описаний невозможно, и требуется неявное приведение одного типа к другому. Использование многоуровневых моделей позволяет частично решить данную проблему,

поэтому актуальна задача разработки методик работы с поведенческими описаниями в гибридной модели.

Поддержка аналоговых и смешанных устройств

Для описания цифро-аналоговых устройств и систем существуют специализированные языки описания, например, VHDL AMS (Analog Mixed-Signal) или Verilog AMS. Чаще всего подобные языки являются расширениями HDL для цифровых устройств. При разработке архитектуры САР и методов внутреннего представления не было использовано ни одной предпосылки о том, что средство является цифровым. Предложенные подходы за счёт специализации модели можно распространить на цифро-аналоговые системы, но для точного вывода требуется провести дополнительные исследования и апробацию подходов на подобных задачах. В случае перехода к цифро-аналоговым описаниям, скорее всего, потребуется расширить списки параметров сигналов и соединений, а также добавить новые типы сигналов. Перечисленные изменения касаются расширяемых элементов представления, что подтверждает простоту поддержки цифро-аналоговых описаний.

Совместный реинжиниринг для верификации аппаратно-программных систем

Современные СнК часто включают микропроцессорные компоненты, которые в дополнение к аппаратной составляющей включают программную часть. Преимуществом подхода является совмещение гибкости программного обеспечения с высокой скоростью обработки на специализированных аппаратных вычислителях [61]. Совместная разработка программной и аппаратной составляющих таких систем является одним из направлений исследований в областях САПР. В дополнение к разработке, возникает вопрос реинжиниринга подобных систем посредством гибридной модели. В задачах встраивания СТ данная задача наиболее актуальна, так как при реинжиниринге устройств потребоваться соответствующей адаптации или генерации тестовых наборов. При решении данной задачи целесообразно использовать обобщённую модель системы.

ЗАКЛЮЧЕНИЕ

В результате исследования получены значимые результаты в научной области реинжиниринга однокристальных цифровых устройств и систем на кристалле. Разработанные модели устройств и методы имеют высокую научную и практическую значимость, так как они позволяют строить новые инструментальные средства реинжиниринга, снижающие затраты на прототипирование, разработку, реинжиниринг и контроль устройств за счёт алгоритмизации и повторяемости основных операций при анализе и модификации архитектур устройств. На основании проведённых исследований получены следующие новые научные результаты:

1. Проведён обзор существующих подходов и моделей для обработки описаний устройств в САР, сформированы критерии оценки, проведён сравнительный анализ подходов. Введена новая классификация средств поддержки реинжиниринга.
2. Разработана новая гибридная метамодель цифровых устройств, входящая в класс многоуровневых моделей, отличающаяся от других совмещением двух уровней представления и применимая в широком классе задач анализа, модификации и верификации архитектур цифровых устройств.
3. Предложены и обоснованы функционально-полный набор базовых операций над моделью, а также методики построения модели из исходных описаний и специализации модели, позволяющие эффективно описывать алгоритмы реинжиниринга устройства.
4. Разработаны методы встраивания средств внутрисхемного тестирования и самодиагностики, которые снижают затраты на тестирование и риск возникновения ошибок, а также могут быть интегрированы в типовые маршруты проектирования однокристальных цифровых устройств.
5. Разработана методика совместной верификации модели устройств в системе моделирования и аппаратных прототипов на базе единого набора тестов, в отличие от других не требующая процессорных блоков.

6. Разработан метод встраивания средств внесения однократных сбоев в память для контроля устойчивости СнК к однократным сбоям памяти, отличительной особенностью которого являются высокая скорость проведения экспериментов и использования процессорного ядра системы для снижения требований по аппаратным ресурсам.

Для подтверждения применимости предложенных моделей и методик разработан прототип САПР PHRT (Programmable Hardware Reengineering Toolkit), включающий разработанный инструментарий и набор средств для решения задач реинжиниринга и встраивания средств тестирования. Разработанная архитектура прототипа имеет самостоятельную практическую ценность, но не рассматривается в работе как научный результат и не выносится как основное положение на защиту. С использованием прототипа PHRT в работе решён ряд задач реинжиниринга и внутрисхемного тестирования устройств, разработки внедрены в компаниях ООО «Синопсис СПб» и ООО «ЭсДиСи». Также прототип использован для автоматизации встраивания средств внесения однократных сбоев в память. При решении которых удалось значительно снизить временные затраты на подготовку проектов устройств к внутрисхемному тестированию, что подтверждается соответствующими актами о внедрении. Результаты внедрения подтвердили свою применимость подходов и их эффективность при решении задач внутрисхемного тестирования.

Разработанные модели и методы могут быть использованы при разработке новых моделей цифровых устройств, инструментальных средств и САПР для решения задач реинжиниринга. Развитием работы может быть улучшение гибридной метамодели устройств и методик работы с ней, расширение области применения разработок в задачах верификации цифровых систем и снятие ограничений, введённых при разработке исходной гибридной метамодели. Также возможна разработка новых методов встраивания средств внутрисхемного тестирования и повышения отказоустойчивости цифровых устройств, основанных на гибридной метамодели.

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

АСГ	-	Абстрактный Синтаксический Граф
АСД	-	Абстрактное Синтаксическое Дерево
ИС	-	Информационная Система
КСПТ	-	кафедра Компьютерных Систем и Программных Технологий
ПЛИС	-	Программируемая Логическая Интегральная Схема
ПО	-	Программное Обеспечение
САПР	-	Система Автоматизированного Проектирования
САР	-	Система Автоматизации Реинжиниринга
СБИС	-	Сверхбольшая Интегральная Схема
		(в тексте - интегральная схема)
СнК	-	Система на Кристалле
СППР	-	Средство Поддержки Принятия Решений
СПР	-	Средство Поддержки Реинжиниринга
СТ	-	Средство тестирования
СУТ	-	Система Управления Тестированием
AMS	-	Analog Mixed-Signal
		Смешанные аналого-цифровые сигналы
API	-	Application Programming Interface
		Интерфейс программирования приложения
ASG	-	Abstract Semantic Graph
		Абстрактный семантический граф (АСГ)
AST	-	Abstract Syntax Tree
		Абстрактное синтаксическое дерево (АСД)
BIST	-	Built-In Self-Test
		Встроенное самотестирование
DAG	-	Directed Acyclic Graph
		Направленный ациклический граф

EDA - Electronic Design Automation
Автоматизация проектирования электронных приборов

EDIF - Electronic Design Interchange Format
Формат обмена проектами электронных приборов

GUI - Graphical User Interface
Графический интерфейс пользователя

HDL - Hardware Description Language
Язык описания устройства

IDE - Integrated Development Environment
Интегрированная среда разработки

ICT - In-Circuit Testing
Внутрисхемное тестирование

IEC - International Electrotechnical Commission
Международная электротехническая комиссия

IP - Intellectual Property (IP-module)
Сторонние функциональный модуль устройства

ISO - International Organization for Standardization
Международная организация по стандартизации

RTL - Register Transfer Level
Описание устройства на уровне регистровых передач

SoC - System on Chip
Система на кристалле (СнК)

TCL - Tool Command Language
Командный язык инструментов

UML - Unified Modeling Language
Унифицированный язык моделирования

VHDL - VHSIC (Very high speed integrated circuits) HDL
Язык описания высокоскоростных интегральных схем

XML - eXtensible Markup Language – Расширяемый язык разметки

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Ахтырченко К.В., Сорокваша Т.П. Методы и технологии реинжиниринга ИС // Труды Института Системного Программирования РАН. 2003. т. 4. 141–162 с.

2. Бадд Т. Объектно-ориентированное программирование в действии. СПб: Питер, 1997. 464 с.

3. Васильев А.Е. Микроконтроллеры. Разработка встраиваемых приложений. СПб: БХВ-Петербург, 2008. 302 с.

4. Головина Е. и др. Метод создания и отладки комплексных тестов для функциональной верификации СнК, ориентированный на их повторное использование на всех этапах проектирования // Проблемы разработки перспективных микро-и наноэлектронных систем-2014 (МЭС-2014). Москва, 2014. с. 45–50.

5. Зайцев В.С., Степанец В.Я. Аппаратное ускорение цифрового моделирования // Материалы трудов ITS2012. Минск, 2012. с. 206–207.

6. Максименко С.Л., Мелехин В.Ф. Анализ надежности цифровых устройств со структурным резервированием и периодическим восстановлением работоспособного состояния узлов // Информационно-Управляющие Системы. 2013. № 3 (64). 16–22 с.

7. Максименко С.Л., Мелехин В.Ф., Филиппов А.С. Анализ проблемы построения радиационно-стойких информационно-управляющих систем // Информационно-Управляющие Системы. 2012. № 2 (57). 18–25 с.

8. Ненашев О.В. Разработка методов и средств автоматизации реинжиниринга устройств, заданных HDL-спецификациями. СПб: Изд-во СПБГПУ, 2011. 200 с.

9. Поляков А.К. Языки VHDL и VERILOG в проектировании цифровой аппаратуры. Солон-Пресс, 2003. 320 с.

10. Путря Ф.М. Особенности использования возможностей объектно-ориентированного программирования SystemVerilog для функциональной верификации многоядерных СнК // Проблемы разработки перспективных микро-и наноэлектронных систем-2012 (МЭС-2012). , 2012. с. 83–88.

11. Синицын С., Налютин Н. Верификация программного обеспечения // М БИНОМ. 2008. т. 6. 368 с.
12. Харари Ф., Козырев В.П., Гаврилов Г.П. Теория графов. Москва: Либроком, 2009. 302 с.
13. Akehurst D.H. и др. Compiling UML State Diagrams into VHDL: An Experiment in Using Model Driven Development // Proceedings of ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems (formerly the UML series of conferences). Genova, 2007. с. 219–224.
14. Alfred V., Sethi R., Jeffrey D.U. Compilers: Principles, Techniques and Tools. Addison-wesley, 1986. 870 с.
15. Altera Corporation. Introduction to the Quartus® II Software v10.0. San Jose: Altera Corporation, 2010. 136 с.
16. Antognetti P., Massobrio G., Massobrio G. Semiconductor device modeling with SPICE. McGraw-Hill, Inc., 1993.
17. Arnold R. Software reengineering. Los Alamitos Calif.: IEEE Computer Society Press, 1993. 645 с.
18. Ashenden P.J., Wilsey P.A., Martin D.E. SUAVE: Extending VHDL to improve data modeling support // Des. Test Comput. IEEE. 1998. т. 15. № 2. 34–44 с.
19. Barbero M. и др. A practical approach to model extension // Model Driven Architecture-Foundations and Applications. Springer, 2007. с. 32–42.
20. Batory D. Multilevel models in model-driven engineering, product lines, and metaprogramming // IBM Syst. J. 2006. т. 45. № 3. 527–539 с.
21. Baxter I.D., Pidgeon C., Mehlich M. DMS: Program Transformations for Practical Scalable Software Evolution // Proceedings of the 26th International Conference on Software Engineering ICSE ’04. Washington, DC, USA: IEEE Computer Society, 2004. с. 625–634.
22. Bergey J. и др. A Reengineering Process Framework. Pittsburgh: Software Engineering Institute, 1995. 840 с.
23. Bjesse P. и др. Lava: hardware design in Haskell // ACM SIGPLAN Notices. ACM, 1998. с. 174–184.

24. Brand M. Van den, Klint P., Verhoef C. Reverse engineering and system renovation—an annotated bibliography // ACM SIGSOFT Softw. Eng. Notes. 1997. т. 22. № 1. 57–68 с.
25. Burmester S. и др. Incremental design and formal verification with UML/RT in the FUJABA real-time tool suite // Proceedings of the International Workshop on Specification and Validation of UML Models for Real Time and Embedded Systems, SVERTS2004, Satellite Event of the 7th International Conference on the Unified Modeling Language, UML. Citeseer, 2004.
26. Carbine A., Feltham D. Pentium (R) Pro processor design for test and debug // Test Conference, 1997. Proceedings., International. IEEE, 1997. с. 294–303.
27. Chakraborty R.S., Bhunia S. Hardware protection and authentication through netlist level obfuscation // Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design. IEEE Press, 2008. с. 674–677.
28. Charest L., Aboulhamid E.M. A VHDL/SystemC comparison in handling design reuse // System-on-Chip for Real-Time Applications. Springer, 2003. с. 41–50.
29. Chikofsky E.J., Cross J.H. Reverse engineering and design recovery: A taxonomy // IEEE Softw. 1990. 13–17 с.
30. Chiou D. и др. FPGA-accelerated simulation technologies (fast): Fast, full-system, cycle-accurate simulators // Proceedings of the 40th Annual IEEE/ACM international Symposium on Microarchitecture. IEEE Computer Society, 2007. с. 249–261.
31. Devos H. и др. Finding and applying loop transformations for generating optimized FPGA implementations // Trans. High-Perform. Embed. Archit. Compil. I. 2007. 159–178 с.
32. Edwards S.A. The challenges of synthesizing hardware from C-like languages // Des. Test Comput. IEEE. 2006. т. 23. № 5. 375–386 с.
33. Fowler M. Domain-specific languages. Pearson Education, 2010. 640 с.
34. Frank U. Multilevel modeling // Bus. Inf. Syst. Eng. 2014. т. 6. № 6. 319–337 с.

35. Gonzalez I. и др. Classification of Application Development for FPGA-Based Systems // Aerospace and Electronics Conference, 2008. NAECON 2008. IEEE National. с. 203–208.
36. Gupta R.K., Liao S.Y. Using a programming language for digital system design // IEEE Des. Test Comput. 1997. т. 14. № 2. 72–80 с.
37. Haldar M. и др. FPGA hardware synthesis from MATLAB // VLSI Design, 2001. Fourteenth International Conference on. IEEE, 2001. с. 299–304.
38. Hammer M. Reengineering the corporation: a manifesto for business revolution. New York NY: Harper Business, 1994. вып. 1. 680 с.
39. Hartman A. Software and hardware testing using combinatorial covering suites // Graph Theory, Combinatorics and Algorithms. Springer, 2005. с. 237–266.
40. Hennessy J., Patterson D. Computer organization and design : the hardware software interface. San Francisco Calif.: Morgan Kaufmann Publishers, 1998. вып. 2nd ed. 751 с.
41. Jamieson P. и др. Odin II-an open-source verilog HDL synthesis tool for CAD research // Field-Programmable Custom Computing Machines (FCCM), 2010 18th IEEE Annual International Symposium on. IEEE, 2010. с. 149–156.
42. Jerraya A. Behavioral synthesis and component reuse with VHDL. Boston: Kluwer Academic Publishers, 1997. 265 с.
43. Jervan G. и др. A hybrid BIST architecture and its optimization for SoC testing // Quality Electronic Design, 2002. Proceedings. International Symposium on. , 2002. с. 273–279.
44. Kaiping Z., Huss S.A. RAMS: A VHDL-AMS Code Refactoring Tool Supporting High Level Analog Synthesis // IEEE Computer Society Annual Symposium on VLSI: New Frontiers in VLSI Design (ISVLSI’05). Tampa, FL, USA. с. 266–267.
45. Kazman R., Woods S.G., Jeromy Carriere S. Requirements for integrating software architecture and reengineering models: CORUM II // Reverse Engineering, 1998. Proceedings. Fifth Working Conference on. IEEE, 1998. с. 154–163.
46. Lajolo M., Prevostini M. UML in an electronic system level design methodology // Proceedings of DAC’04 UML for SoC workshop. , 2004.

47. Leventis P. и др. Cyclone™: a low-cost, high-performance FPGA // Proceedings of the IEEE Custom Integrated Circuits Conference. IEEE; 1999, 2003. с. 49–52.
48. Mamoutova O.V., Nenashev O.V., Filippov A.S. In-circuit Emulation of Memory Fault Injection // Recent Adv. Electr. Eng. Comput. Sci. 2014. 105–107 с.
49. Melham T.F. Higher order logic and hardware verification. Cambridge: Cambridge University Press, 2009. 165 с.
50. Miczo A. Digital logic testing and simulation. John Wiley & Sons, 2003.
51. Nahir A. и др. Scheduling-based test-case generation for verification of multimedia SoCs // Proceedings of the 43rd annual Design Automation Conference. ACM, 2006. с. 348–351.
52. Oliver I. Model Driven Embedded Systems. // ACSD. , 2003. с. 5.
53. Ousterhout J.K., Jones K. Tcl and the Tk toolkit. Chennai: Pearson Education, 2009. 816 с.
54. Patti R.S. Three-dimensional integrated circuits and the future of system-on-chip designs // Proc. IEEE. 2006. т. 94. № 6. 1214–1224 с.
55. Peng Z., Kuchcinski K. Automated transformation of algorithms into register-transfer level implementations // Comput.-Aided Des. Integr. Circuits Syst. IEEE Trans. On. 1994. т. 13. № 2. 150–166 с.
56. Pohl C., Paiz C., Porrmann M. vMAGIC—automatic code generation for VHDL // Int. J. Reconfigurable Comput. 2009. т. 2009.
57. Project Management Institute. A guide to the project management body of knowledge (PMBOK® Guide). Newtown Square Pa.: Project Management Institute, 2008. вып. 4th ed.
58. Ramos R., Sampaio A., Mota A. Transformation laws for UML-RT // Formal Methods for Open Object-Based Distributed Systems. Springer, 2006. с. 123–137.
59. Rosenberg L.H., Hyatt L.E. Software re-engineering. Unisys Federal Systems, 1998. 17 с.
60. Saleh R. и др. System-on-chip: reuse and integration // Proc. IEEE. 2006. т. 94. № 6. 1050–1069 с.

61. Sangiovanni-Vincentelli A., Martin G. Platform-based design and software design methodology for embedded systems // IEEE Des. Test Comput. 2001. т. 18. № 6. 23–33 с.

62. Sankaran R. и др. Decoupling interaction hardware design using libraries of reusable electronics // Proceedings of the 3rd International Conference on Tangible and Embedded Interaction. ACM, 2009. с. 331–337.

63. Schürr A. Reengineering & Refactoring with Graph Transformations // TU Sarmstadt. 2004. 16 с.

64. Sethuram R. и др. Zero Cost Test Point Insertion Technique to Reduce Test Set Size and Test Generation Time for Structured ASICs // Test Symposium, 2006. ATS'06. 15th Asian. IEEE, 2006. с. 339–348.

65. Shen J., Abraham J.A. An RTL abstraction technique for processor microarchitecture validation and test generation // J. Electron. Test. 2000. т. 16. № 1-2. 67–81 с.

66. Singh R. International Standard ISO/IEC 12207 software life cycle processes // Softw. Process Improv. Pract. 1996. т. 2. 35–50 с.

67. Tessier R., Giza H. Balancing logic utilization and area efficiency in FPGAs // Field-Programmable Logic and Applications: The Roadmap to Reconfigurable Computing. Springer, 2000. с. 535–544.

68. Thomas D.E., Moorby P.R. The Verilog hardware description language. Springer, 2002.

69. Tong J.G., Anderson I.D., Khalid M.A. Soft-core processors for embedded systems // Microelectronics, 2006. ICM'06. International Conference on. IEEE, 2006. с. 170–173.

70. Vanderperren Y., Mueller W., Dehaene W. UML for electronic systems design: a comprehensive overview // Des. Autom. Embed. Syst. 2008. т. 12. № 4. 261–292 с.

71. Vidal J. и др. A co-design approach for embedded system modeling and code generation with UML and MARTE // Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE'09. IEEE, 2009. с. 226–231.

72. Wawrzynek J. Components and Design Techniques for Digital Systems. Lection 6: Hardware Description Languages // Berkeley University. 2003. 42 c.
73. Willis J. AIRE/CE. Internal Intermediate Representation (IIR) Specification Version 4.6 Including Digital VHDL & VHDL-AMS support // University of Cincinnati. 1999. 573 c.
74. Wilsey P.A., Benz D.M., Pandey S.L. A model of VHDL for the analysis, transformation, and optimization of digital system designs. University of Cincinnati, 1995. c. 611–616.
75. Yoder J.W., Razavi R. Metadata and adaptive object-models // Object-Oriented Technology. Springer, 2000. c. 104–112.
76. VLSI test principles and architectures: design for testability / под ред. L.-T. Wang, C.-W. Wu, X. Wen. Amsterdam ; Boston: Elsevier Morgan Kaufmann Publishers, 2006. 777 c.
77. Баринов В.А. Реинжиниринг: сущность и методология// Институт Независимой Оценки [Электронный ресурс]. URL: http://www.ippnou.ru/article.php?idarticle=002369 (дата обращения: 09.05.2011).
78. AMIQ company. DVT System Verilog User Guide [Электронный ресурс]. URL: http://www.dvteclipse.com/help.html?documentation/sv/Tips_and_Tricks.html (дата обращения: 22.05.2011).
79. Anjou M. d'. FPGA Continous Integration with Jenkins// Synopsys User Group Archives [Электронный ресурс]. URL: http://www.synopsys.com/news/pubs/snug/2013/canada/a1_danjou_paper.pdf (дата обращения: 04.01.2015).
80. Brigham Young University. FPGA Reliability Studies - BYU EDIF Tools [Электронный ресурс]. URL: http://reliability.ee.byu.edu/edif/ (дата обращения: 08.06.2011).
81. Cadence Design Systems, Inc. Cadence Products A - Z [Электронный ресурс]. URL: http://www.cadence.com/products/Pages/all_products.aspx (дата обращения: 24.05.2011).

82. Elgris Technolodgies. EDIF overview// Elgris Technolodgies [Электронный ресурс]. URL: http://www.elgris.com/content/edif_overview.html (дата обращения: 15.05.2011).
83. Fitch A. Continuous Integration for FPGA Design and Verification// Verification Futures 2015 [Электронный ресурс]. URL: http://www.testandverification.com/conferences/verification-futures/2015-europe/speaker-alan-fitch-ericsson-tv-ltd/ (дата обращения: 03.05.2015).
84. Fowler M. Continuous integration// Martin Fowler's Personal Site [Электронный ресурс]. URL: http://www.martinfowler.com/articles/continuousIntegration.html (дата обращения: 02.05.2014).
85. Nenashev O. Jenkins-Based Continuous Integration for Heterogeneous Hardware and Software Projects// Jenkins User Conference 2015, London [Электронный ресурс]. URL: https://www.cloudbees.com/jenkins/juc-2015/abstracts/europe/02-01-1130-nenashev (дата обращения: 06.01.2015).
86. Venners B. Refactoring with Martin Fowler [Электронный ресурс]. URL: http://www.artima.com/intv/refactor.html (дата обращения: 04.11.2010).
87. Xilinx Inc. ML505/ML506/ML507 Evaluation Platform User Guide [Электронный ресурс]. URL: http://www.xilinx.com/support/documentation/boards_and_kits/ug347.pdf (дата обращения: 09.02.2014).
88. SAMATE team. Tool Taxonomy// SAMATE project [Электронный ресурс]. URL: http://samate.nist.gov/index.php/Tool_Taxonomy.html (дата обращения: 17.05.2011).
89. Semantics Designs inc. DMS Software Reengineering Toolkit [Электронный ресурс]. URL: http://www.semdesigns.com/Products/DMS/DMSToolkit.html (дата обращения: 04.11.2010).
90. Sigasi nv. Sigasi HDT for VHDL [Электронный ресурс]. URL: http://www.sigasi.com/sigasi-hdt (дата обращения: 22.05.2011).

ПРИЛОЖЕНИЯ

В приложениях представлены используемые в работе термины и определения, материалы по применению методик в частных задачах реинжиниринга, дополнительная информация о CAP и результатах его апробации. В таблице 1 приведён перечень приложений к диссертации.

Таблица 1
Список приложений к диссертации

№ приложения	Число страниц	Название/Описание
A	8	Классификация средств реинжиниринга цифровых устройств. Рассмотрены классификации инструментариев по различным признакам.
B	6	Обзор средств инструментариев реинжиниринга устройства. Проведён анализ используемых методик и моделей на основании открытой информации.
C	14	В приложении описан прототип PHRT и его расширения, разработанные при апробации предложенных моделей и методик на примерах частных задач реинжиниринга.
D	8	Набор операций над гибридной моделью в прототипе CAP. Приведено краткое описание набора базовых операций над предлагаемой моделью, которые предложены для CAP.
E	7	Рассмотрены особенности использования VHDL-нетлистов, которые выбраны в прототипе PHRT в качестве входного формата описания устройств.
F	8	Введение структурной избыточности в устройство для рассмотренного в пятой главе примера реинжиниринга.
G	3	Акты о внедрении полученных в исследовании результатов в ООО «Синопсис СПб», ООО «ЭсДиСи» и ФГАОУ ВО «СПбПУ»

ПРИЛОЖЕНИЕ А. КЛАССИФИКАЦИЯ СРЕДСТВ РЕИНЖИНИРИНГА ЦИФРОВЫХ УСТРОЙСТВ

В данном приложении приведена классификация средств автоматизации реинжиниринга, разработанная автором при подготовке магистерской диссертации [8]. Для САПР существует множество различных классификаций (например, в [35] или [88]), которые можно применить и для средств поддержки реинжиниринга. Однако данные классификации не отражают специфику СПР с точки зрения постановки задачи реинжиниринга. В работе введена новая классификация средств реинжиниринга, которая приведена в данном приложении.

Классификация СПР по решаемым задачам реинжиниринга

В СПР могут решаться любые задачи реинжиниринга. Если опираться на модель “подковы” (см. п. 0) то можно выделить следующие группы задач:

- средства восстановления архитектуры (реверс-инжиниринга);
- средства поддержки принятия решений (СППР);
    - анализаторы;
    - экспертные системы;
- средства трансформации представлений;
    - модификации архитектуры;
    - рефакторинга;
- средства контроля качества системы;
    - средства тестирования;
    - верификаторы;
- средства формирования выходных данных;
- средства разработки общего назначения.

Средства реверс-инжиниринга

Данные средства извлекают описание архитектуры объекта из имеющихся представлений: исходных кодов, нетлистов и т.п. В общем случае реверс-реинжиниринг можно рассматривать как преобразование объекта из одного

описания в другое, более удобное для использования в процессе реинжиниринга. Они нужны в том случае, если разработчик изначально не имеет доступа к описанию архитектуры. Такое может быть, если он использует недокументированный legacy-код или же закрытый IP-модуль.

Средства поддержки принятия решений

Реинжиниринг системы требует её детального анализа. Поэтому, на первый план выходят средства поддержки принятия решений (СППР), которые помогают разработчику выбрать оптимальные способы трансформации системы. Существует целый пласт подобных систем в области экономики, но данные системы можно отразить и на технические задачи.

В группу СППР входят анализаторы, которые позволяют выявить проблемы в текущей архитектуре системы. Например, в IDE (Integrated Development Environment – интегрированная среда разработки) Quartus II, используемой при разработке устройств на ПЛИС фирмы Altera, при компиляции проекта производится оценка временных характеристик результирующего устройства. С помощью данного средства можно выявить “узкие места”, которые ограничивают повышение тактовой частоты и производительности устройства.

Экспертные системы, по сути, представляют собой базу знаний, которая содержит рекомендации для решения тех или иных проблем. Например, в случае недостатка производительности вычислительного модуля подобная система могла бы порекомендовать реализовать “конвейерную обработку”. Подобные системы позволяют накапливать историю проблем и их типовых решений, что в первую очередь полезно для разработчиков с малым опытом работы в области поставленной задачи.

Средства трансформации представлений

Задачей данных средств является преобразование архитектуры системы на основании исходной архитектуры и принятых решений по реинжинирингу. В параграфе 1.3 выделены следующие группы задач:

– рефакторинг представлений устройств;
– трансформация архитектуры;
– перенос архитектуры между представлениями.

Из всех средств, рассматриваемых в данной классификации, только задачи трансформации представления относятся к самому реинжинирингу, а не поддержке данного процесса.

Средства контроля качества системы

Средства тестирования и верификации, по сути, являются анализаторами, но используются для проверки соответствия системы её спецификации. В процессе трансформации системы в неё вносятся модификации, которые могут внести в систему ошибки. Поэтому, контроль соответствия системы её спецификации является такой же неотъемлемой составляющей процессов реинжиниринга и разработки.

Средства формирования выходных описаний

После трансформации архитектуры требуется преобразовать её в требуемое выходное представление. Данный этап совпадает с реализацией в процессе разработки, поэтому в нём можно использовать обычные САПР. Выходным представлением могут быть исходные коды на HDL, нетлисты, текстовые спецификации, фотошаблоны и пр.

Ещё выходным форматом могут быть отчёты о результатах обработки (статистика, рекомендации к трансформации и пр.).

Средства разработки общего назначения

К данной группе в классификации относятся те средства, которые обычно применяются в процессе разработки, но могут быть применены и при реинжиниринге устройства. Из текущей классификации к данной группе можно отнести средства верификации и формирования выходных данных.

Классификация СПР по области применения

Средства поддержки реинжиниринга могут применяться в самых различных задачах. Средства могут быть универсальными или

специализированными, а также ориентироваться на области, не связанные с реинжинирингом цифровых устройств. На рисунке 1 приведена более подробная классификация, которая будет пояснена далее.

Универсальным средством является то, которое охватывает все этапы процесса реинжиниринга системы и пригодно для решения широкого класса задач реинжиниринга. Специализированными средствами называются те, которые решают только некоторые задачи из списка.

Под узкоспециализированными средствами понимаются те, которые реализуют лишь один алгоритм преобразования (например, заменяют все элементы памяти на их отказоустойчивые аналоги). В других же задачах данные средства неприменимы.

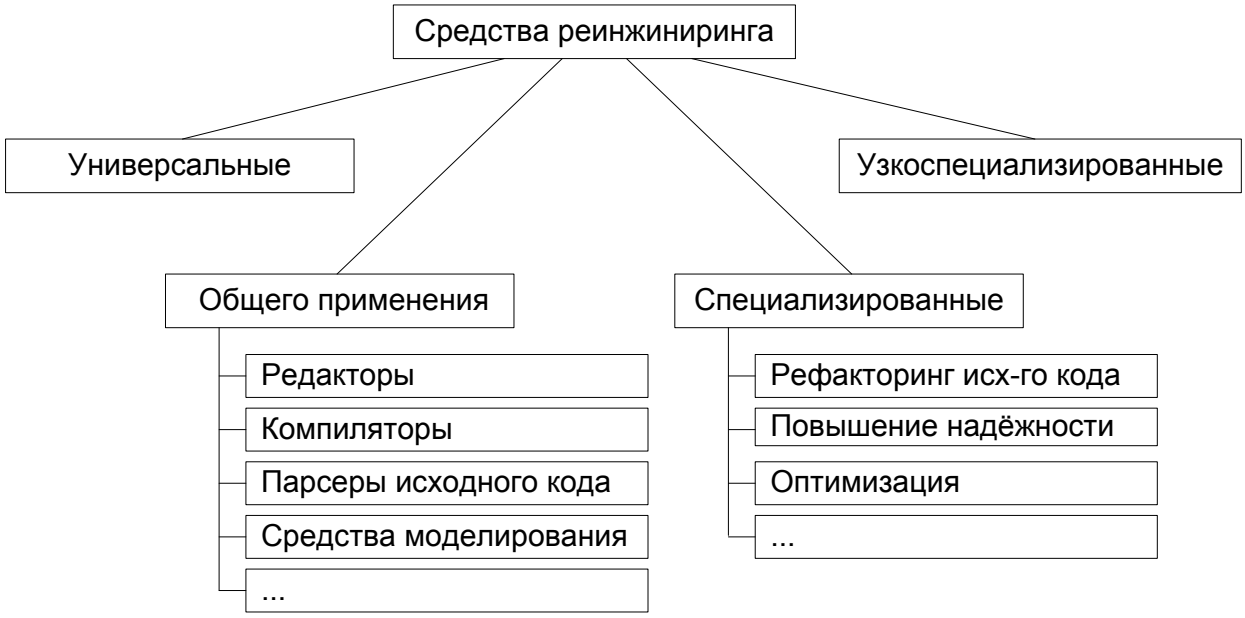


Рис. 1. Классификация СПР по области применения

Классификация СПР по интеграции с другими средствами

Средства поддержки реинжиниринга могут быть как завершёнными средствами (т.е. полнофункциональными модулями), так и набором некоторых компонентов, на базе которых может быть построено полноценное средство для решения пользовательских задач (см. рисунок 2).

Завершённые СПР могут быть отдельностоящими средствами, когда пользователю приходится все операции выполнять в одной среде. В то же время возможно часть функциональности связать со средой разработки,

позволив тем самым использовать функциональность самой среды (например, для визуализации архитектуры или компиляции). При этом из самой СПР возможен быстрый доступ к средствам реинжиниринга (например, актуально для рефакторинга кода).

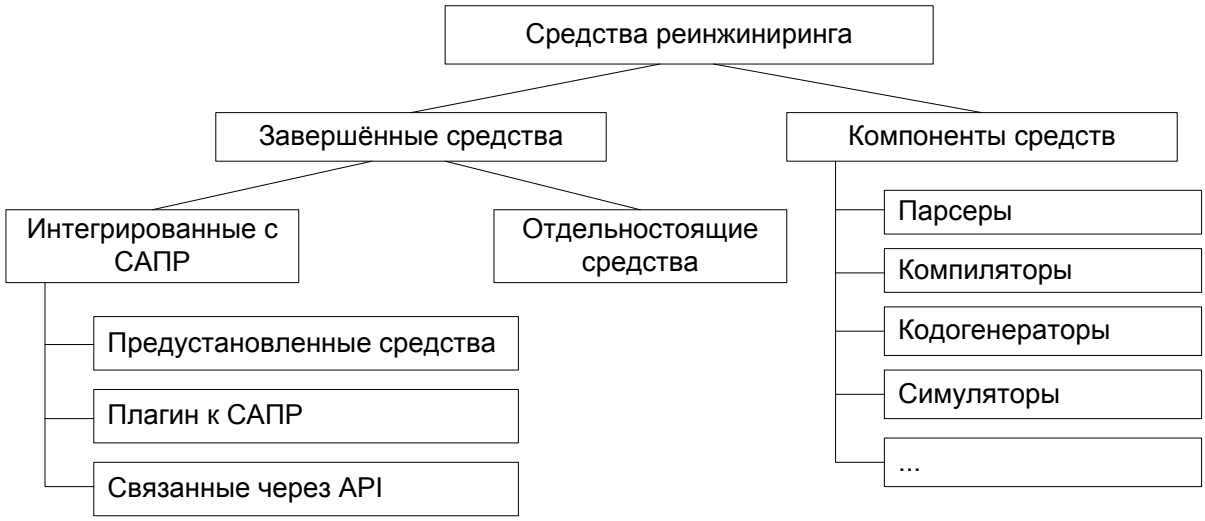


Рис. 2. Классификация по возможности интеграции с другими средствами

СПР может быть встроенным, когда некоторые базовые функции реинжиниринга предоставляет сама среда разработки. Возможно реализовать средство реинжиниринга в виде отдельного включаемого модуля (плагина), который может быть добавлен в среду разработки при необходимости. Таким образом, среда предоставляет некоторый межпрограммный интерфейс (в англ. Application Programming Interface или API), через который модуль реинжиниринга получает доступ к её средствам. Возможен и обратный подход, когда API предоставляется самим СПР, а уже среда разработки взаимодействует с ним. Такой подход распространён в модульных средствах, когда этапы разработки реализуются отдельными средствами, а задачей среды является вызов данных обработчиков в соответствии с конфигурацией.

Классификация СПР по степени автоматизации

Средства поддержки реинжиниринга могут быть:

- частично автоматические (автоматизированные);
- автоматические;
- программируемые.

Частично-автоматические средства автоматизируют лишь некоторые этапы реинжиниринга. Например, для реинжиниринга устройства можно использовать любую САПР. Этим можно полностью автоматизировать этап компиляции устройства, но трансформация архитектуры и модификация исходных кодов должна производиться пользователем. Например, пользователь может вызвать команду “троировать все элементы памяти” в одном из блоков, после чего все необходимые операции будут выполнены автоматически.

Автоматические средства реинжиниринга могут выполнить некоторые задачи без вмешательства пользователя. Ключевой особенностью данных средств является то, что автоматизируется принятие решений, а пользователю лишь нужно задать требования. Например, он может дать команду “минимизировать вероятность ошибок в памяти”, а система уже сама принимает решение на основании имеющихся системных ресурсов.

Программируемые средства являются, на взгляд автора, отдельным классом СПР. В них возможен любой уровень автоматизации, но сначала требуется задать алгоритм для выполнения пользовательской команды. Но после этого пользователь получает средство, в точности соответствующее его задачам, и в дальнейшем он всегда может его модифицировать.

Классификация СПР по внутреннему представлению (модели) устройства

Для выполнения задач реинжиниринга требуется обработка архитектуры устройства. Поэтому, необходимо выбрать такое внутреннее представление, которое обеспечило бы возможность выполнения поставленных задач реинжиниринга. Возможны следующие подходы:

- использование входного представления;
- использование специализированного представления;
    - связь с исходным представлением;
    - отсутствие связи с исходным представлением;
- использование специального представления для каждой задачи;
    - связь с исходным представлением;
    - отсутствие связи с исходным представлением.

Входные представления используют средства рефакторинга, которые вносят изменения в файлы исходных кодов. Для подобных средств возможно каскадное применение, когда над одними и теми же данными последовательно выполняются несколько различных типов преобразований. Тем не менее, у подобных средства много ограничений, связанных с необходимостью опираться на представление в исходных файлах. Например, при переименовании сигнала в текстовых HDL нужно найти все упоминания в исходных файлах, проверить отсутствие конфликтов и только потом изменить имена. С усложнением преобразований увеличивается сложность связей, что требует использования специализированного формата представления данных.

Используя специализированное внутреннее представление, можно относительно легко выполнить все требуемые задачи, после чего сгенерировать выходное представление архитектуры в требуемом формате. Подобный подход реализуем, но имеет следующие недостатки:

- требуется обеспечить экспорт представления в выходной формат;
- разрывается связь между входными и выходными данными.

Иногда сохранение связи с входным представлением является одним из основных требований к модели представления. Например, при рефакторинге исходных кодов разработчик ожидает получить ограниченные задачей изменения. Если не сохранять исходное представление, то могут быть потеряны порядок следования объявлений, табуляция кода, комментарии и пр. Несмотря на сохранение функциональности, пользователь получит новое описание, что затруднит дальнейшую разработку. Такой подход применим для средств и преобразований разомкнутого цикла разработки, не требующих соответствия выходного представления входному: анализаторов, оптимизаторов и т.п.

Компромиссным вариантом является использование внутреннего представления, которое каким-либо образом ссылается на входное. Тогда связь сохраняется на протяжении всего преобразования, и на выходе можно получить представление, близкое к исходному. Такой подход требует модификации сразу

двух представлений в процессе обработки, что в ряде случаев может быть достаточно сложным.

В СПР для каждой задачи может использоваться специализированное представление. Такой подход позволяет добиться более простого решения отдельных задач, но плохо подходит для расширяемого средства реинжиниринга, где решаемые задачи заранее неизвестны. Альтернативным подходом может быть наличие нескольких представлений, когда пользователь может выбрать то из них, которое ему более подходит. При этом задачи синхронизации представлений между собой должны решаться самим средством.

Другим возможным подходом является использование многоуровневых представлений, когда в одном описании одновременно сочетаются несколько форматов представления, что позволяет использовать различные методы и алгоритмы реинжиниринга для более широкого класса задач. Представленная в диссертации гибридная модель является многоуровневой, поэтому представленный в параграфе 5.1 прототип относится к данному классу средств.

ПРИЛОЖЕНИЕ В. ОБЗОР СРЕДСТВ АВТОМАТИЗИРОВАННОГО РЕИНЖИНИРИНГА УСТРОЙСТВА

При предварительном исследовании предметной области был проведён обзор существующих средств автоматизации реинжиниринга (CAP). Выделены следующие категории:

- средства разработки общего назначения;
- средства рефакторинга представления устройства;
- универсальные программируемые CAP;
- узкоспециализированные CAP.

Для каждой из групп в работе рассмотрено несколько примеров средств разработки, для которых в научных работах приведены результаты апробации.

Средства общего назначения

К данным средствам относятся те средства, которые используются в процессе разработки: редакторы, компиляторы, симуляторы и так далее. Они не ориентированы на реинжиниринг, но могут быть использованы для отдельных задач процесса. Обычно средства разработки строятся по модульному принципу, после чего объединяются в цепочки средств (toolchain), каждое из которых решает отдельную задачу. Подобные пакеты программ выпускают все ведущие компании-производители САПР: Synopsys, Cadence, Mentor Graphics, Altera, Xilinx и пр.

Перечисленные выше компании выпускают средства для всех этапов проектирования и различных уровней сложности проектов, их номенклатура очень велика. Например, Cadence предлагает более 50 различных продуктов [81]. Даже такие комплексные CAP не могут решить все возможные задачи разработки, поэтому они имеют интерфейсы для подключения внешних средств автоматизации разработки (Electronic Design Automation или EDA), которые могут решать отдельные задачи процесса разработки. Например, интегрированная среда разработки Quartus II фирмы Altera поддерживает следующие группы средств:

- синтез устройств из проектов на языках VHDL, Verilog и SystemVerilog;
- моделирование устройств посредством ModelSim Altera;
- временной анализ;
- формальная верификация;
- синтез исходных кодов [15].

Инструменты взаимодействия с EDA

Для упрощения взаимодействия оболочки со сторонними средствами автоматизации разработки устройства (Electronic Design Automation или EDA) последние должны обладать универсальными интерфейсами для передачи команд и данных.

Проведённый обзор показал, что в реальности механизмы взаимодействия не стандартизированы, и существует широкий спектр форматов, многие из которых поддерживаются лишь одним средством. Наиболее распространёнными форматами являются три типа нетлистов: EDIF, Verilog- и VHDL-нетлисты.

Средства рефакторинга

Автоматизация рефакторинга исходных кодов является одной из наиболее распространённых задач реинжиниринга. Поэтому существует достаточно много средств, которые включают в себя те или иные возможности рефакторинга. В большинстве подобные средства входят в состав САПР.

Sigasi HDT

Sigasi HDT – это полноценная среда разработки для VHDL, поставляемая компанией Sigasi. На официальном web-сайте проекта основными преимуществами называются динамическое выявление ошибок, контекстная подстановка имён из базы знаний, а также наличие модуля рефакторинга кода [90]. Данный модуль реализует следующие возможности:

- переименование элементов;
- инкапсуляция выражений в пакеты (package);
- добавление портов в Entity;

– присоединение Entity к внешнему сигналу (с созданием порта).

Первые две задачи в соответствии с принятой в работе классификацией относится к рефакторингу, остальные – к внесению изменений в функциональные характеристики устройств.

AMIQ Design and Verification Tool (DVT)

AMIQ DVT – это среда разработки на SystemVerilog и e (внутренний объектно-ориентированный HDL), выполненная в виде плагина к IDE Eclipse. Среда поддерживает такие простейшие функции рефакторинга, как переименование элементов описания и инкапсуляцию имён. Кроме того, для SystemVerilog среда поддерживает рефакторинг под управлением скриптового файла, что является одним из требований к универсальному CAP [78].

Скриптовый XML-файл содержит список элементарных действий, которые следует выполнить в процессе рефакторинга. Действия включают переименования различных элементов устройства, имён исходных файлов, добавление меток (например, TODO) и комментариев. Формат скриптового файла не поддерживает сложную выборку объектов для преобразования (например, через регулярные выражения), поэтому возможности подобного рефакторинга крайне ограничены. Тем не менее, при помощи данного средства можно, например, произвести перенос устройства между некоторыми базовыми библиотеками со схожими интерфейсами.

Таким образом, примитивное средство рефакторинга наподобие DVT может решить и некоторые, очень узкие задачи реинжиниринга. Однако, сложность реализации подобных преобразований невелика, и в случае возможности следует использовать подобные средства.

RAMS

Рефакторинг описаний на HDL возможен не только для цифровых, но и для смешанных устройств. К примеру, средство RAMS позволяет производить рефакторинг описаний на языке VHDL-AMS [44]. В процессе преобразований RAMS использует объектное дерево, которое генерируется из АСГ (см. рисунок

1). Разбор грамматики производится с стандартными библиотеками с открытыми исходными кодами, RAMS реализует только рефакторинг.

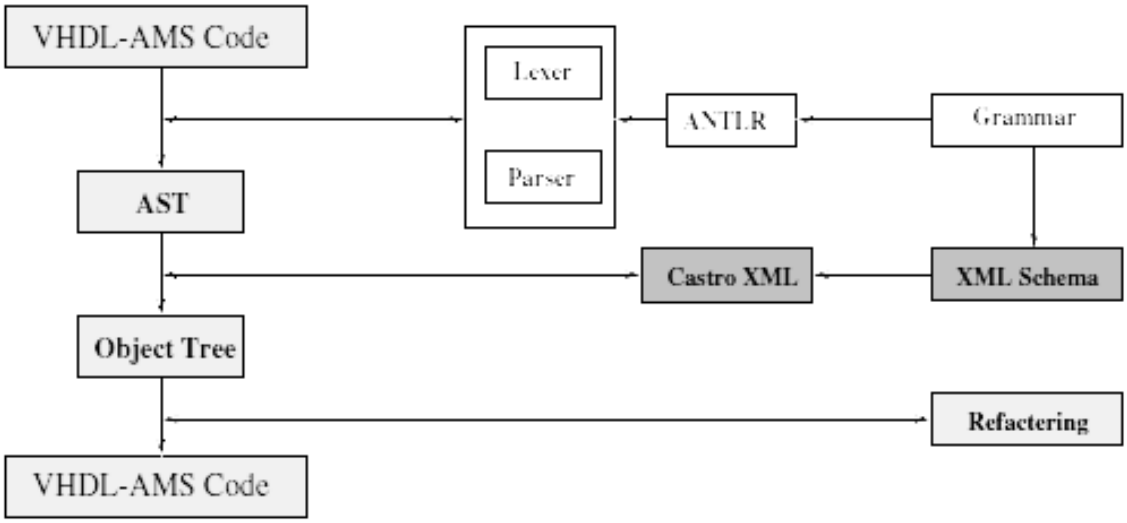


Рис. 1. Процесс рефакторинга VHDL-AMS с использованием RAMS [44]

Программируемые средства реинжиниринга устройства

DMS Software Reengineering Toolkit

Данный продукт заявлен как универсальное средство реинжиниринга программного обеспечения [89]. Средство поддерживает более двадцати языков программирования, в том числе VHDL и Verilog. Архитектура средства (см. рисунок 2) имеет модульную структуру. Средство состоит из четырёх основных модулей: синтаксического анализатора исходных кодов, семантического анализатора, модуля трансформации и генератора выходных файлов.

Автоматизация в средстве достигается за счёт набора правил обработки, которые компилируются отдельным модулем (Компилятор правил) из пользовательских программ обработки, которые могут быть написаны на множестве языков (в т.ч. C++, Python, Java). DMS SRT позволяет полностью автоматизировать обработку и реинжиниринг устройства, если пользователь корректно специфицирует правила преобразования. Тем не менее, у средства есть ряд ограничений:

- отсутствие интерфейса к сторонним средствам;
- представление устройства в виде семантического графа.

Главным недостатком DMS SRT является его закрытость. Пользователь может лишь косвенно управлять процессом преобразования, программируя правила. Но при этом невозможно встроить вместо модуля-анализатора некий частный модуль со специфичным внутренним представлением.

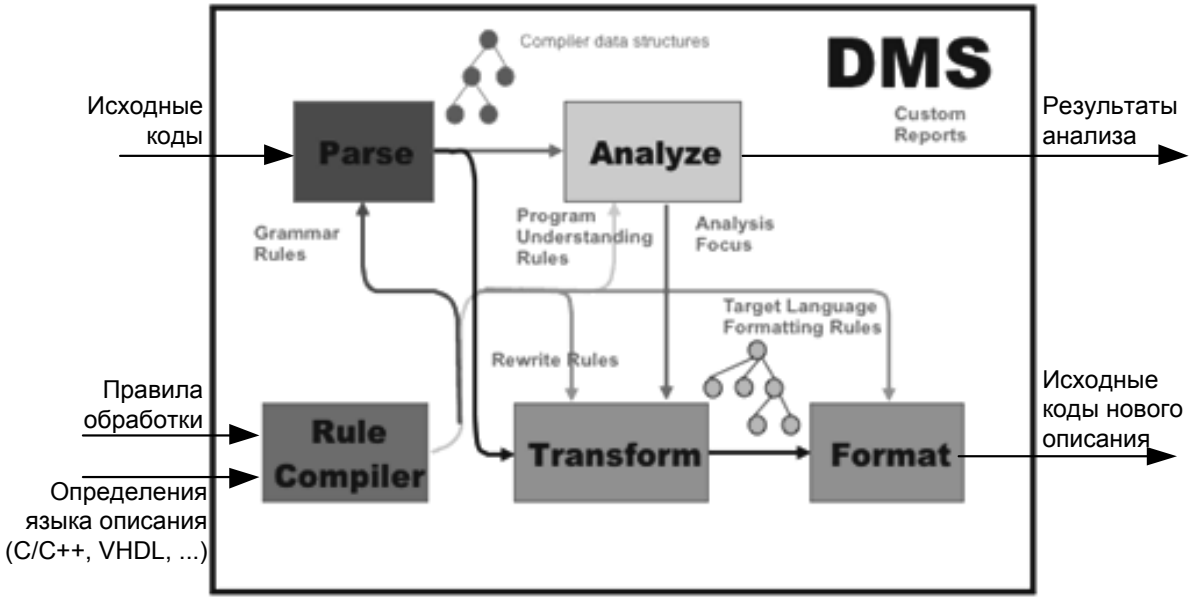


Рис. 2. Схема работы DMS Software Reengineering Toolkit [89]

В случае DMS Software Reengineering Toolkit универсальность подхода и поддержка многих языков являются скорее недостатком, так как средство ориентировано не на реинжиниринг целевого объекта, а на реинжиниринг его исходных кодов. Тем не менее, оно является единственным из рассмотренных средств, на базе которого может быть построено полностью автоматическое средство реинжиниринга устройства с программируемой функциональностью.

Специализированные средства автоматизации реинжиниринга

Специализированные средства автоматизации реинжиниринга составляют большинство существующих средств. Причиной этому является то, что в большинстве случаев разработчику (даже в рамках фирмы) требуется реализовать небольшое число алгоритмов обработки. В этом случае разработка универсального средства отдельному разработчику (да и целому предприятию) невыгодна из-за больших затрат.

Покупка и адаптация универсальных средств возможна, если их суммарная стоимость владения будет ниже, чем стоимость разработки нового

специализированного средства. К сожалению, подобное условие редко выполняется даже для областей с более широким рынком (как в случае с математическими пакетами). Получается, что разрабатываются внутренние средства, которые решают лишь узкие задачи, специфичные для определённой компании. Подобные решения, если и выходят на рынок, то имеют низкий спрос и мало поддерживаются, в результате чего на их последующую универсализацию рассчитывать не приходится. Ниже приведён пример для одного из специализированных средств.

BYU EDIF Tools

Инструментарий BYU EDIF Tools разработан научной группой надёжности ПЛИС из университета Бригама Янга. Он предоставляет САПР и набор библиотек для задач редактирования и анализа EDIF-нетлистов [80]. Инструментальное средство поддерживает следующие функции реинжиниринга:

- синтаксический анализатор EDIF-нетлистов, реализованный на Java;
- генератор синтезируемых кодов на языке JHDL;
- объединение нескольких устройств и цепей в один нетлист;
- троирование элементов для повышения отказоустойчивости;
- минимальная библиотека для синтеза устройств для ПЛИС Xilinx.

Преимуществом данного САПР является открытый исходный код, что позволяет при необходимости расширять инструментарий и вносить в него поправки при возникновении ошибок в процедурах реинжиниринга или синтеза. Средство использует внутреннее представление в виде АСД, поэтому, несмотря на наличие API, его применение для задач реинжиниринга затруднительно. Ещё одним ограничением является использование JHDL в качестве выходного представления, так как поддержка данного языка средствами разработки ограничена.

ПРИЛОЖЕНИЕ С. ПРОТОТИП СРЕДСТВА АВТОМАТИЗАЦИИ РЕИНЖИНИРИНГА PHRT

В приложении приведены выдержки из заявки на регистрацию программного средства PHRT, которое было построено как прототип CAP с целями апробации и подтверждения практической применимости предложенных гибридной модели устройств и методик её обработки. Несмотря на статус прототипа, разработанный инструментарий был успешно внедрён в ряде проектов по разработке электронных устройств и СнК, поэтому он может рассматриваться как завершённое средство.

Общая концепция

Общий подход предлагаемой архитектуры предлагает модульный подход. При этом формируется некоторое “ядро”, которое включает минимально необходимую функциональность средства и динамический контекст преобразования. Всё остальное выносится в дополнительные модули-расширения, которые могут загружаться ядром в случае необходимости.

Ядро и расширения вместе образуют средство реинжиниринга, которое, в свою очередь, может быть использовано в пользовательских целях: интегрировано с САПР, адаптировано для специфических задач и т.п. На рисунке 1 приведена схема средства реинжиниринга с указанием основных его составляющих. Подробно все перечисленные модули будут рассмотрены далее.

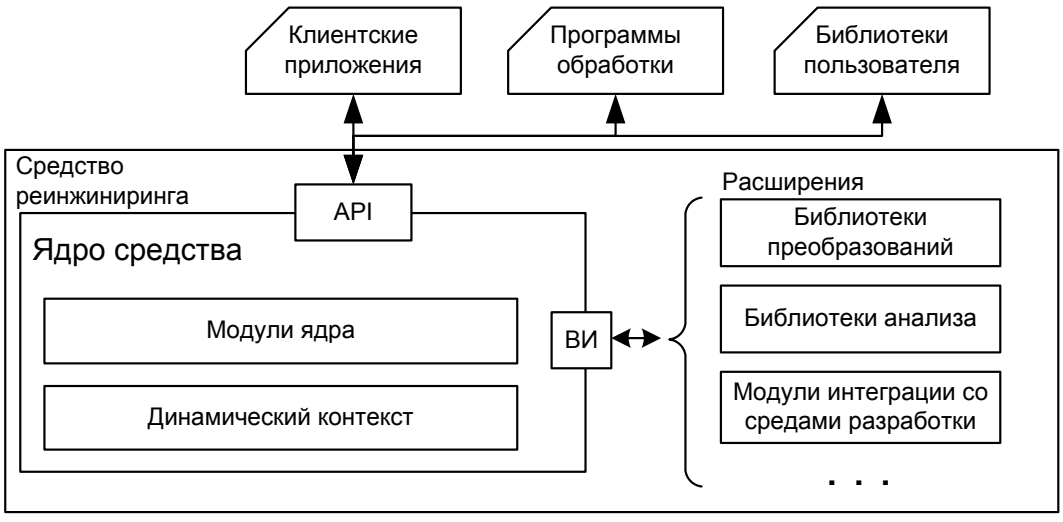


Рис. 1. Концепция построения средства реинжиниринга

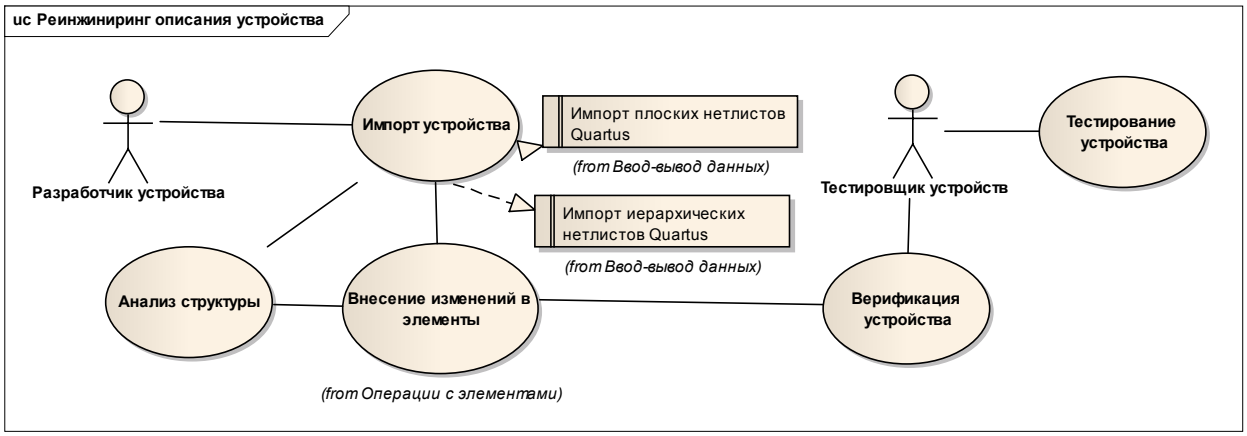


Рис. 2. Порядок реинжиниринга в прототипе

Контекст средства

Контекст САР хранит всю динамическую информацию, которая используется при выполнении задач реинжиниринга. В него входят как внутренние представления устройств, так и пользовательские элементы.

Контекст средства реинжиниринга включает следующие элементы:

- внутреннее представление устройства;
- информация о загруженных библиотеках и расширениях;
- списки обработчиков событий;
- указатели на текущее положение в дереве (для косвенной адресации к элементам).

Для управления элементами контекста будут использоваться специальные модули, называемые менеджерами.

Пользовательские расширения

Под пользовательскими расширениями понимаются любые модули, которые подключаются через внутренний интерфейс (ВИ). Само ядро, кроме интерфейса, содержит механизмы для загрузки модулей и организации их взаимодействия. Чтобы исключить конфликты с библиотеками элементов, предлагается называть данные расширения библиотеками ядра.

Ядро средства

Ядро CAP является основной частью, через которую связываются все прочие составляющие системы. Данный модуль, с одной стороны, должен быть максимально компактен, но при этом он должен поддерживать все ранее рассмотренные возможности. Можно выделить следующие основные элементы, которые должны присутствовать в ядре CAP:

- средство хранения контекста преобразования;
- API для подключения пользовательских библиотек;
- механизм генерации событий;
- библиотека системных команд для управления ядром.

На рис. 5.2 приведена схема взаимодействия модулей ядра с внешними элементами. Для упрощения контроля над преобразованиями предлагается весь доступ к контексту преобразования производить через совокупность компонентов-менеджеров, вместе называемых “исполняемой средой”. Каждый менеджер отвечает за отдельную часть контекста исполняемой среды.

Кроме необходимых элементов, в ядро средства решено добавить ещё два элемента: интерпретатор команд внутреннего языка и библиотеку элементарных преобразований. Основанием к этому стала необходимость использования данных модулей в большинстве задач реинжиниринга. Далее мы рассмотрим общие подходы к построению каждого из элементов ядра.

Команды преобразований

Работа ядра средства реинжиниринга основана на передаче команд между его модулями. С учётом требований и архитектуры, объект команды в средстве должен включать следующее:

- реализацию алгоритма трансформации устройства;
- описание параметров команды;
- описание правил разбора командной строки;
- алгоритмы отмены выполнения команды;
- методы проверки параметров;

- справочную информацию для генерации отчётов.

Под командами в средстве понимаются самые разные группы операций, которые оказывать различное влияние на архитектуру или не оказывать её вовсе. Средство реинжиниринга может не иметь возможности проанализировать алгоритмы и оценить их влияние, поэтому предлагается ввести в описание команды следующие флаги:

- команда модифицирует дерево элементов;
- команда может быть отменена;
- команду требуется фиксировать в истории команд.

Менеджеры исполняемой среды

Менеджеры исполняемой среды используются для организации взаимодействия между командами из библиотек ядра и деревом элементов. Также они представляют шлюз к внутренней функциональности для внешних приложений (через API). Менеджеры ядра тесно связаны друг с другом, и разделение идёт по решаемым задачам. Сами задачи будут рассмотрены далее.

Интерфейсы ядра

В архитектуре средства реинжиниринга предусмотрено два типа интерфейсов:

- межпрограммные интерфейсы (API);
- внутренние интерфейсы для расширений.

Межпрограммные интерфейсы, в соответствии со сформированными требованиями, должны обеспечить передачу команд в CAP и возвращение результатов. Кроме этого, должен присутствовать механизм для оповещения внешнего средства о возникновении событий. Внутренние интерфейсы используются для предоставления расширенного доступа к функциональности библиотекам ядра. Основной задачей интерфейсов ядра является предоставление контролируемого доступа к менеджерам исполняемой среды, которые обеспечивают исполнение передаваемых команд.

Интерпретатор внутреннего языка управления преобразованиями

Интерпретатор команд является дополнительным модулем, который позволяет считывать и исполнять программы, реализованные на внутреннем языке. Модуль решает следующие задачи:

- разбор команды (определение имени команды, подготовка параметров);
- формирование вызова менеджера команд для исполнения;
- последовательное исполнение группы команд (например, внутри скрипта).

Основным элементом модуля, используемым в интерпретаторе, является конфигурируемый синтаксический анализатор команды, который должен провести разбор и анализ командной строки с учётом того, что каждая команда имеет свои опции и аргументы. Модуль синтаксического анализатора, в частности, используется менеджером команд при исполнении своих вызовов.

Хранение контекста

В требованиях к средству реинжиниринга предусмотрена навигация по дереву элементов косвенной адресации к элементу, хранение истории команд. Задачи управления контекстом выделены в специальный модуль, который исполняет следующие задачи:

- хранение ссылки на текущий выбранный элемент;
- хранение истории переходов между элементами;
- хранение и обновление истории команд;
- выявление изменений в дереве элементов и генерация сообщений.

Менеджер контекста необходим для формирования событий об изменении дерева, так как в самих командах обработки может быть недостаточно данных для формирования отчёта. В итоге, при изменении элемента сначала вызывается менеджер контекста, который передаёт всю необходимую информацию в менеджер событий.

Выполнения команд в ядре

Всякая программа управления преобразованиями устройства разбивается на некоторую последовательность вызовов, запускающих те или иные команды трансформации. В ядре средства должен быть модуль, с помощью которого осуществляется управление исполнением программ. Этот модуль предлагается называть менеджером команд. В его задачи входят:

- хранение списка доступных команд и описаний их параметров;
- проверка поступающих команд на исполнимость (существование обработчика команды, корректность параметров);
- вызов обработчика команды;
- контроль порядка исполнения команд и исключение конфликтов;
- возвращение результатов выполнения команды;
- генерация отчётов об ошибках при выполнении команд.

При хранении команд предлагается определять их имена в менеджере команд, а не в самих обработчиках. Это позволит при необходимости изменить имя команды, чтобы избежать конфликта имён между двумя командами из различных библиотек (например, заменить конфликтные команды A на *lib1.A* и *lib2.A*). Также возможно привязать одну команду сразу к нескольким вызовам.

Перед выполнением любой команды менеджер команд при помощи синтаксического анализатора извлекает имя команды, после чего загружает её описание из своей базы. Менеджером производится извлечение и проверка параметров команды, после чего запускается функция преобразования. Результаты выполнения команды возвращаются наружу. Загрузка списка задач в менеджер команд производится при инициализации ядра или же менеджером расширений при добавлении новой библиотеки.

Механизм событий

Механизм событий должен информировать внешние модули CAP об изменениях в контексте, чтобы те могли адаптировать модель и свои

внутренние представления под текущий контекст ядра CAP. Можно выделить следующие группы событий:

- изменение структуры дерева элементов;
- изменения параметров элемента;
- изменение выбранного элемента;
- загрузка и отключение расширений;
- возникновение исключительных ситуаций в ядре.

В таблице 1 приведён список используемых в PHRT событий. В колонке слева приведены описания событий и их символьные идентификаторы событий, которые предлагается использовать в дальнейшем.

Для передачи событий в сторонние модули предлагается использовать механизм, когда сами модули предоставляют некоторый API, через который им может быть передана информация о событии. Абстрагируясь от интерфейса передачи, будем называть подобный интерфейс “обработчиком событий”.

Таблица 1

Примеры событий, генерируемых ядром

Имя события	Описание
NODE_ADDED	Добавлен новый элемент
NODE_DELETED	Из модели удален элемент
NODE_CHANGED	Произведена модификация элемента
PARAM_ADDED	В элемент добавлен новый параметр
PARAM_DELETED	Из элемента удалён параметр
PARAM_CHANGED	Изменено значение параметра
LIB_LOADED	Добавлена новая библиотека ядра
SEL_CHANGED	Изменён выбранный элемент для навигации
CORE_EXIT	Прекращение работы средства реинжиниринга

Для обеспечения взаимодействия с обработчиками в ядре средства предлагается реализовать специальный модуль – менеджер событий. Данный модуль должен решать следующие задачи:

- регистрация и настройка обработчика события;
- отключение обработчика событий;
- вызов обработчика при возникновении события;
- предоставление списка текущих обработчиков;

Модули сами должны регистрировать свои обработчики, так как само ядро о них может не знать. Аналогично, должна быть возможность отключения обработчика, чтобы при изменении состояния модуля (например, при отключении), не тратить системные ресурсы на обработку событий. Для данной задачи в API должна быть предоставлена соответствующая функциональность.

Число генерируемых событий при реинжиниринге велико, поэтому предлагается их каким-либо образом группировать, чтобы при возникновении конкретного события не приходилось вызывать все обработчики, а только те, которым действительно требуется данное событие. При текущем списке типов событий логично реализовать

Механизм расширений ядра

Разработка библиотек ядра является основным путём расширения функциональности CAP. Вместе с библиотекой ядра в средство могут быть загружены следующие элементы:

- новые команды обработки;
- расширения базовых элементов модели;
- модули поддержки ввода-вывода в различные форматы представления;
- модули поддержки внешних языков программирования;
- дополнительные внешние API для средства;
- пользовательские библиотеки элементов (в виде библиотек элементов).

Таким образом, механизм расширений является крайне важным с точки зрения встраиваемости, программируемости, расширяемости разрабатываемого

средства. Поэтому, при реализации средства реинжиниринга он должен быть реализован в первую очередь.

Менеджер расширений

Управление расширениями производится при помощи отдельного модуля, называемого менеджером библиотек. В модуле решаются следующие задачи:

- хранение списков загруженных и доступных расширений;
- загрузка новых расширений;
- начальная инициализация расширений (загрузка их компонентов);
- редактирование и сохранение списка доступных библиотек.

Задача выгрузки библиотеки крайне сложна, так как модель может содержать элементы, построенные с использованием расширений, что может привести к непредсказуемым последствиям. Поэтому, при разработке архитектуры решено, что единственным путём отключения библиотеки будет перезапуск средства PHRT.

Зависимости между расширений

Расширения ядра могут зависеть друг от друга. При этом, от менеджера библиотек требуется обеспечить загрузку всех необходимых библиотек. Для реализации подобного механизма требуется:

- добавить контроль версий в библиотеках;
- добавить в описание библиотеки список ссылок на используемые библиотеки (с указанием диапазона версий);
- добавить в средство механизм выбора порядка загрузки библиотек с учётом зависимостей;

Контроль версий необходим, так как библиотеки могут обновляться независимо друг от друга, в результате чего может оказаться, что загруженная по зависимости библиотека не содержит нужной команды или, что хуже, её поведение изменилось

Состав библиотеки

Исходя из вышесказанного, сущность библиотеки должна включать следующие элементы:

- идентификатор библиотеки (тип + версия);
- список используемых библиотек;
- программа инициализации библиотеки;
- пользовательское наполнение (команды, элементы, …).

Внутренние библиотеки ядра

В ядре средства реинжиниринга предусмотрено две внутренних библиотеки: библиотека системных команд и библиотека базовых операций. Данные библиотеки добавлены в ядро, так как им требуется расширенный доступ к составляющим ядра, а их использование необходимо практически во всех операциях. Подробное описание команд из данных библиотек приведено в приложении D. Внутренние библиотеки загружаются при инициализации средства реинжиниринга, так как в противном случае не будет выполняться требование функциональной полноты системы команд.

Библиотека системных команд (SystemLib)

Библиотека системных команд реализует команды управления средством реинжиниринга. Команды данной библиотеки реализуют управление менеджерами среды и средством в целом. Поэтому, данной библиотеке требуется предоставить более широкие права доступа, чем для пользовательских библиотек. Функционирование CAP без данной библиотеки невозможно, так как через её средства подключаются другие задачи реинжиниринга.

Библиотека базовых операций (BasicLib)

В библиотеку базовых преобразований входят команды модификации дерева элементов и получения информации об элементах, предоставляющих полный набор операций для работы с гибридной метамоделью (см. п. 2.5).

Библиотека базовых преобразований также требует расширенного доступа, но уже к дереву элементов. Через команды базовой библиотеки должны

осуществляться преобразования в пользовательских расширениях, сами же команды должны работать с деревом элементов напрямую. Также в группу добавлены команды вывода текстовых описаний устройства (см. команду ls).

Внешние библиотеки

Ниже кратко описаны основные расширения прототипа CAP, которые были разработаны в рамках исследований с целью апробации предложенных моделей устройств и методик работы с ними. В таблице 2 приведён полный список разработанных библиотек.

Таблица 2

Расширения PHRT, разработанные для апробации моделей и методик

Название расширения	Назначение
vhdl_basic	Базовая поддержка VHDL.
vhdl_netlist	Ввод-вывод VHDL-нетлистов, генерируемых САПР Quartus II
edif_netlist	Ввода-вывод нетлистов устройств, описанных в форматах EDIF 2 0 0 и EDIF 3 0 0.
Quartus	Частичный синтез компонентов модели в среде Quartus II.
signal_analysis	Специализированное расширение для анализа структуры нетлистов. Решает задачи выявления деревьев и цепей тактирования, зависимостей межу сигналами.
Reliability	Библиотека, которая решает задачу внесения структурной избыточности в устройство, описанную в пятой главе.
test_insertion	Библиотека для встраивания средств тестирования и диагностики в соответствии с методами, описанными в четвёртой главе.
memfault_inject	Библиотека для встраивания средств внесения сбоев в память устройство на основании алгоритмов, разработанных совместно с Мамутовой О.В. и описанных в пятой главе

Расширение для ввода-вывода представлений

В соответствии с двухуровневой схемой ввода-вывода был реализован программный модуль, который обеспечивает возможность ввода VHDL-нетлистов и вывода VHDL-описаний. Модуль ввода-вывода не входит в ядро, и вся его функциональность была реализована в виде двух библиотек ядра: VHDL и Netlist. Библиотека Netlist ссылается на VHDL, которая в свою очередь реализует базовые механизмы работы с VHDL-файлами, используя библиотеку vhdl_ast.

Библиотеки в настоящее время содержат только функции ввода-вывода данных. Если с выводом VHDL-файлов всё относительно просто, то организация ввода нетлистов потребовала решения большого числа частных задач, некоторые из которых перечислены ниже:

- исключение несинтезируемых сигналов из нетлистов;
- восстановление иерархии по объявлениям блоков;
- восстановление интерфейсов по неполным объявлениям компонентов;
- восстановление иерархии компонентов.

Подробнее о проблемах ввода нетлистов можно прочитать в приложении А, а о реализации алгоритма – в описании команд библиотеки. За время разработки прототипа не удалось обеспечить поддержку всех возможных конструкций в VHDL-нетлистах прототипа. Поэтому, разработанный модуль поддерживает не все возможности типы нетлистов, а считывание некоторых устройств может происходить с ошибкой.

Расширение сбора статистики по структуре нетлистов (signal_analysis)

Библиотека сбора статистики появилась в результате слияния функций анализа, которые изначально располагались в базовой библиотеке работы с деревом элементов, с командами, разработанными Егоровым И.В. в рамках научно-исследовательской работы на кафедре КСПТ.

Библиотека сбора статистики включает следующие функции:

- вывод всех ссылок на элемент;

– вывод всех элементов модели;
– получение числа элементов нижнего уровня различных в элементе;
– получение всех элементов, реализующих указанный интерфейс;
– получение списка всех элементов по интерфейсу;
– получение статистики по портам интерфейса;
– получение цепочек соединений между сигналами.

Перечисленный функционал, прежде всего, может быть полезен при программном управлении преобразованиями, когда решение о проведении операции принимается по результатам анализа дерева. В дальнейшем библиотека может быть расширена для структурного анализа устройства.

Расширение введения структурной избыточности

Данное расширение было запланировано как многосторонняя библиотека, которая решала бы задачи введения структурной избыточности в устройство, что может быть одним из путей повышения надёжности. В рамках проекта в расширении Reliability реализована команда введения избыточности:

– создано расширение с элементами повышения надёжности (отказоустойчивая память, голосователи для задач мажорирования);
– созданы методы для генерации элемента голосователя по заданным пользователем параметрам;
– реализован алгоритм внесения структурной избыточности (см. ниже).

Параметры вызова команды описаны в приложении D. Пример введения структурной избыточности с использование данной команды приведён в приложении F. Ввиду отсутствия специализированных команд модификации, многие операции в алгоритме пришлось выполнять “с нуля”. При помощи команд из стандартных библиотек решаются следующие задачи:

– копирование блоков;
– создание блоков по интерфейсам;
– подключение групп сигналов, добавления портов к блокам;
– переносы соединений от одного элемента к другому;

– добавление сигналов в блоки устройства.

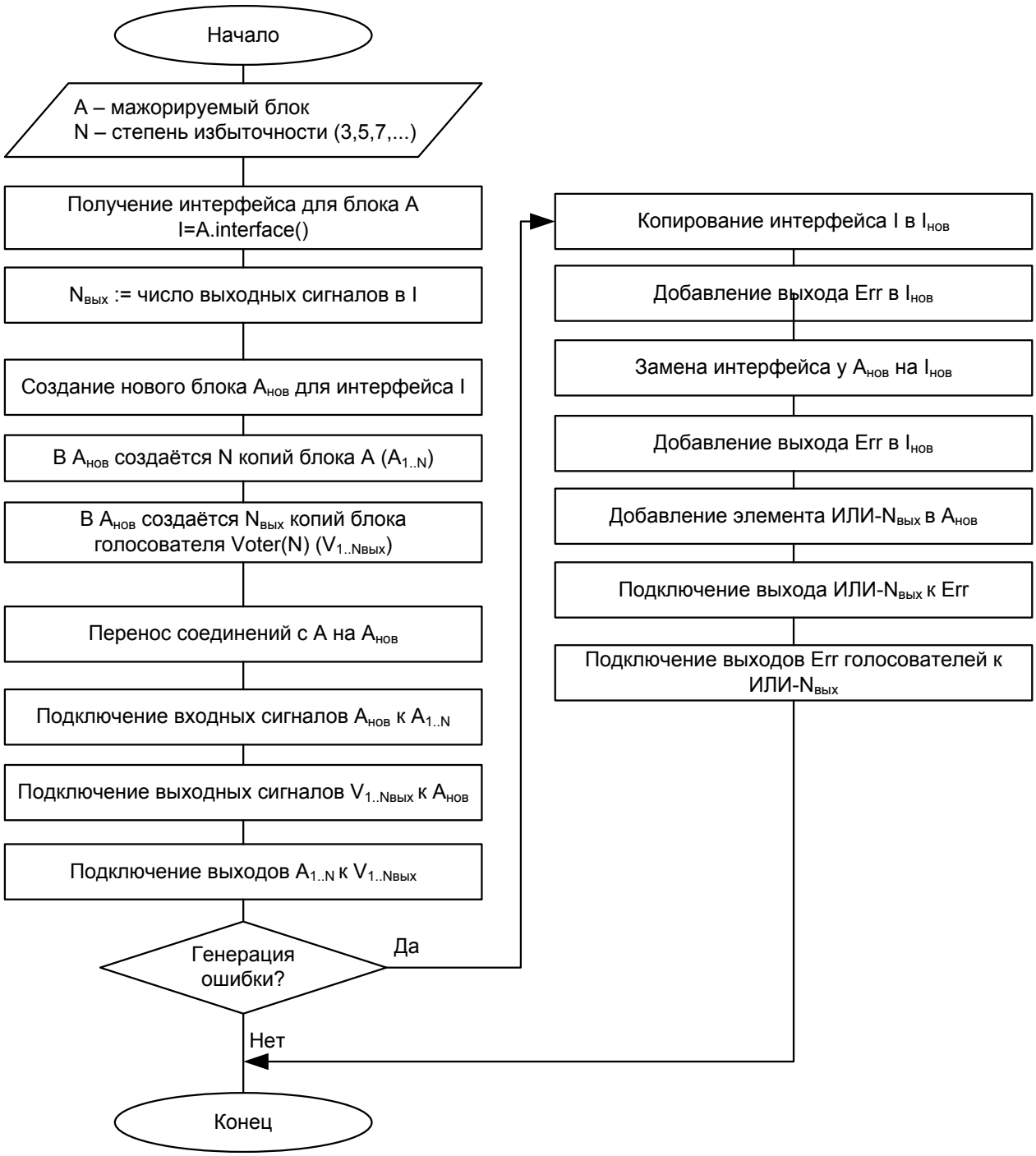


Рис. 4. Порядок действий при внесении аппаратной избыточности в устройство с использованием функций базовой библиотеки CAP

ПРИЛОЖЕНИЕ D. НАБОР ОПЕРАЦИЙ НАД ГИБРИДНОЙ МОДЕЛЬЮ В ПРОТОТИПЕ CAP

В данном приложении приведено краткое описание встроенного в прототип PHRT набора операций с моделью, который описан в параграфе 2.5. Команды из внешних расширений PHRT описываются частично.

Внутренние компоненты ядра

Библиотека системных операций SystemLib

Библиотека SystemLib включает команды управления средством реинжиниринга. В данную группу входят команды запуска программ, работы с библиотеками историей команд, а также команды выхода справки и выхода из средства реинжиниринга.

Команда вывода справки

Команда вывода справки предназначена для вывода информации о доступных командах средства и информации об отдельных командах. Кроме того, планируется добавление в команду базовой информации о механизмах адресации и прочих элементах. Команда help принимает один аргумент – имя команды. При отсутствии аргумента выводится список доступных команд из загруженных библиотек, иначе – справка по команде в следующем формате:

Листинг 1. Пример вывода справки по команде help

```
[R]root\>>help setparam
Sets new value for parameter
Usage:
        <nodepath> [-?|--help] [-s] <parampath> <newvalue>
Options:
  <nodepath>
      Path to device node
  [-?|--help]
      Prints this help
  [-s]
```

Prints subcommand reports

<parampath>

Path to parameter. Ex,"ports.port_A.signal_type"

<newvalue>

new value

Таблица 1

Список команд в библиотеке SystemLib

Команда	Описание	Флаги*		
		M	L	U
help, man, ?	Вывод справочной информации о команде или списка команд	-	-	+
undo	Отмена последнего действия, если возможно	+	-	-
redo	Повторение последнего отменённого действия	+	-	+
history	Отображение истории команд	-	-	-
save	Сохранение истории команд в скрипт по указанному пути	-	-	-
run	Запуск скрипта, содержащего команды PHRT	+	+	-
quit, exit	Выход из программы PHRT	+	+	-
system	Выполнение системной команды в операционной системе, где запущен PHRT	-	+	-
clibs_info	Вывод информации о доступных и загруженных библиотеках	-	-	+
clibs_load	Загрузка библиотеки в исполняемую среду PHRT	+	+	-

* Флаги M – модифицирует контекст; L – сохраняется в истории; U – операция может быть отменена командой undo (для команд модификации)

Команда отображения истории (history)

Команда отображения истории выводит список выполненных команд. Также выводятся команды, которые были отменены через undo.

Листинг 2. Пример вывода истории по команде history

```
pila:\>>history
Executed actions:
    run scripts\load.m
    cb pila:
    cb .\counter
    cb ..
Redo memory:
    cb .
    cb .\[S]gnd
Total commands number: 6
```

Сохранение истории (save)

Команда сохранения истории выводит историю (без отменённых команд) в виде скрипта, который затем может быть исполнен командой run.

Запуск скрипта (run)

Запускает исполнение скриптовой программы из указанного файла. Исполнение программы идёт последовательно вплоть до завершения файла или возникновения ошибки. Строки, начинающиеся на “#”, считаются комментариями и игнорируются.

Вызов системной операции (system)

Передача вызова в командную оболочку ОС. В функцию передаётся строка с вызовом команды и её аргументами. Следует помнить, что команды навигации в Shell не влияют на текущий каталог в прототипе САР.

Вывод списка расширений PHRT (clibs_info)

Команда выводит списки загруженных или доступных для загрузки (при ключе -a) расширениях. В выводимом списке приводится информация об именах и версиях доступных расширений.

Листинг 3. Пример вызова команды вывода списка доступных расширений

```
device:\>>clibs_info -a
System[0.0.0]
BasicLib[0.0.0]
VHDL[0.0.0]
Netlist[0.0.0]
Analysis[0.0.0]
Quartus[0.0.0]
Reliability[0.1.1]
```

Набор базовых операций над моделью (расширение BasicLib)

Прототип PHRT включает минимально необходимый набор операций для работы с моделью, обеспечивающий полноту операций над ней. Сюда входят команды преобразования дерева, получения информации об элементах и навигации по дереву. В таблице 2 приведён список команд данного расширения.

Таблица 2

Список поддерживаемых операций над гибридной моделью в PHRT

Команда	Описание	Флаги*		
		M	L	U
pwd, ls	Получение пути к текущему выбранному элементу	-	-	-
cb	Переход к элементу по указанному пути	-	+	+
add	Создание нового элемента	+	+	+
addref	Создание ссылки на указанный элемент	+	+	+
delete	Удаление элемента из модели	+	+	+
move	Перенос элемента по указанному адресу	+	+	+
copy	Копирование элемента	+	+	+
getparam	Получение значения параметров или атрибута	-	-	+
setparam	Установка значения параметра или атрибута	+	+	+
getlinks	Получение списка всех внешних список на элемент	-	-	-
getmeta	Получение значения поля метаданных по имени	-	-	+
setmeta	Установка нового значения поля метаданных	+	+	+
sethandler	Установка обработчика событий в модели. В PHRT он является сложным объектом, включающим отдельные методы для различных событий в модели	-	-	-
check	Контроль указанного элемента или всего дерева. При этом вызываются все обработчики событий, реализующие соответствующий метод	-	+	-
* Флаги: M – модифицирует модель устройства, L – сохраняется в истории, U – операция может быть отменена командой undo (для команд модификации)				

Внешние расширения PHRT

Расширение для работы с VHDL и VHDL-нетлистами

Расширение VHDL включает базовые средства для работы с VHDL и VHDL-нетлистами. Данное расширение подключает vhdl_ast и реализует основные методы считывания внутреннего представления из AST и методики экспорта. Расширение также включает пользовательскую команду для вывода устройства в виде единого VHDL-файла, структура которого близка к нетлисту, но при этом возможен синтез данного устройства, т.е. файл полностью соответствует стандарту VHDL. Формат вызова команды приведён ниже:

> vhdl_export путь_к_элементу имя_выходного_файла

Команда не модифицирует дерева элементов, но фиксируется в истории команд и может быть отменена. В случае, если выходной файл был перезаписан, то при отмене исходный файл не восстанавливается.

Расширение Netlist

Расширение Netlist предназначено для работы с VHDL-нетлистами и реализует команды импорта и экспорта нетлистов. Поддерживаются плоские и иерархические нетлисты. Для работы использует расширения для работы с описаниями на VHDL.

Команда import_netlist

Команда предназначена для считывания элементов из плоских и иерархических нетлистов различных типов. Считывание идёт в двухпроходном режиме. Сначала считываются объявления интерфейсов, а потом – всё остальное. Подробный алгоритм импорта нетлиста приведён в параграфе 3.2.

Команда export_netlist

Команда экспорта плоского или иерархического нетлиста. Была реализована до реализации вывода VHDL-нетлистов и с тех пор не поддерживается.

Таблица 3
Опции команды import_netlist

Команда	Описание
[-?|--help]	Вывод справки
[-s]	Вывод отчёта о выполнении подкоманд
[-f]	Флаг считывания плоских нетлистов устройства, не содержащих иерархии (по умолчанию - иерархические)
[-p <loadpath>]	Путь к элементу, в который будут сохранены считанные элементы. Целевым элементом могут быть в том числе устройства и библиотеки
[-o]	Разрешение перезаписи считываемых элементов для предотвращения ошибок при импорте устройства из нескольких входных описаний
[-i]	Игнорирование несинтезируемых сигналов и портов при импорте устройства (например, сигналы управления devoe, devpor, … для ПЛИС Altera)
<file>	Путь к нетлисту или VHDL-файлу, из которого идёт импорт гибридной модели устройства

Расширение Reliability – повышение отказоустойчивости устройств

Расширение Reliability реализует функции введения структурной избыточности в указанные блоки устройств с целью повышения их отказоустойчивости. В текущей реализации поддерживаются мажорирование указанного элемента и замена блоков памяти отказоустойчивыми аналогами.

Формат вызова команды внесения структурной избыточности: majorize [путь_к_элементу] [число элементов, по-умолчанию 3], где путь задаётся в соответствии с правилами адресации к элементам в гибридной модели. По умолчанию выполняется операция троирования, но опционально возможно использование только двух элементов для детектирования сбоя или же пяти и

более элементов для большей отказоустойчивости модуля. При этом голосователь генерируется автоматически с использованием элементов логических функций в синтезируемом базисе (N-И, N-ИЛИ, НЕ).

Для замены блоков памяти предоставляется универсальная команда

Порядок действий при внесении структурной избыточности данной командой приведён в приложении С.

Расширение Signal Analysis

Расширение сбора статистики (в средстве - Analysis) предназначено для извлечения дополнительной информации о дереве элементов. В таблице 4 приведён список команд, которые поддерживает библиотека.

Таблица 4

Список команд в библиотеке Analysis

Команда	Описание	Флаги*		
		M	L	U
stats	Получение статистики по числу элементов различных типов в иерархии указанного элемента.	-	-	+
ports	Получение информации о портах устройства, блока или функции.	-	-	+
comps	Получение списка всех блоков в модели, реализующих указанный интерфейс.	-	-	+
find_synced_elems	Получение списка всех блоков и функций, которым передаётся сигнал тактирования. При этом возможно учитывать производные сигналы, получаемые после прохождения сигнала через блоки функций.	-	-	-
* Флаги: M – модифицирует контекст, L – сохраняется в истории, U – может быть отменена командой undo (для команд модификации)				

ПРИЛОЖЕНИЕ Е. ОСОБЕННОСТИ ИСПОЛЬЗОВАНИЯ VHDL-НЕТЛИСТОВ В СРЕДЕ QUARTUS II И ПРОТОТИПЕ

В данном приложении рассмотрены вопросы использования VHDL-нетлистов, которые были выбраны в прототипе САР устройства в качестве входного формата описания. Перечислены параметры нетлистов, используемые в прототипе средства реинжиниринга. Описания в данном приложении ориентированы на версии 9 и 10 среды Quartus II.

Включение нетлистов в процесс разработки Quartus II

При разработке прототипа требовалось интегрировать его с внешней средой разработки, чтобы использовать её возможности по синтезу и анализу устройства. На рисунке 1 приведён пример маршрута проектирования в САПР Quartus II. Он допускает взаимодействие (в том числе автоматическое) с различными группами средств разработки через вызовы подкоманд на языке Tcl, поэтому возможно связать его с САР в единый маршрут проектирования.

Среда Quartus II поддерживает некоторое количество входных и выходных форматов. Для прототипа средства реинжиниринга в качестве входного формата решено взять VHDL-нетлисты, а выходного – структурное подмножество VHDL. Это позволяет использовать маршрут реинжиниринга с двойным синтезом, предложенный в подпараграфе 5.1.3.

Надо отметить, что для различных типов внешних EDA-средств IDE Quartus II поддерживает различные форматы выходных файлов, что связано с ориентацией САПР на уже существующие средства разработки. Поэтому, генерация VHDL-нетлистов возможна только для средств моделирования, и конфигурируется из соответствующей панели. При этом функциональность средства не ограничивается (т.е. возможно не только моделирование).

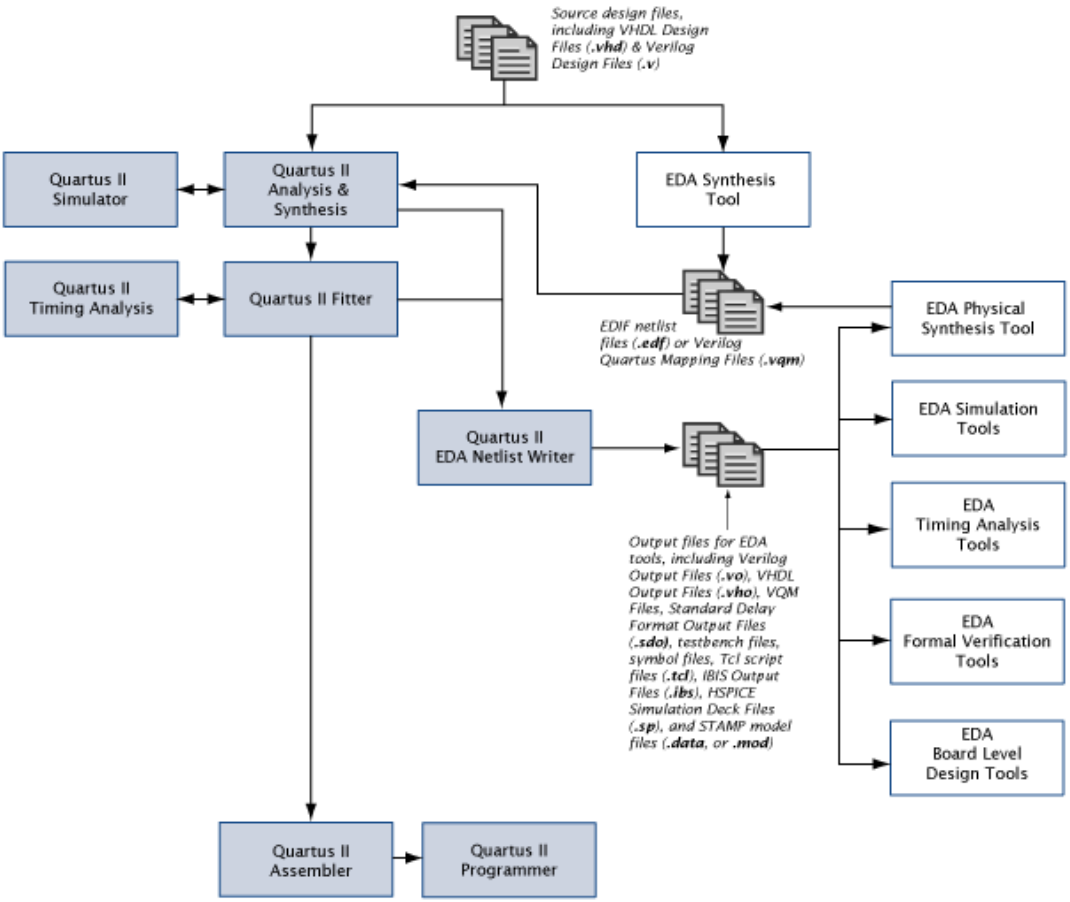


Рис. 1. Этапы процесса разработки устройства в среде Quartus II с использованием сторонних средств разработки [10]

Настройки генерации VHDL-нетлистов

Среда Quartus II поддерживает различные варианты формирования VHDL-нетлистов. В таблице 1 описаны основные настройки генерации выходных данных для симулятора, которые можно задать в меню Settings/EDA Tools/Simulation. Приведены только основные настройки, влияющие на выходное структурное описание устройства в нетлисте.

Для задания путей в уровнях иерархии расширяются имена элементов. При этом, для разделения уровней иерархии используются специальные метки (см. примеры в таблице 2). В случае, если включена опция “Map illegal HDL characters” все символы заменяются на “_a”. Это делает устройство потенциально синтезируемым, но исключает возможность анализа, так как комбинации “_a” могут встречаться и в именах элементов.

Таблица 1

Настройки формирования VHDL-нетлистов в среде Quartus II

Параметр	Тип	Описание
Format for output netlist	Список	Формат выходного нетлиста. Для генерации VHDL-нетлистов требуется выбрать
Architecture name in VHDL	Строка	Имя архитектуры в выходном файле
Maintain hierarchy	On/Off	Параметр задаёт, требуется ли отображать в нетлисте иерархию элементов устройства. Форматы нетлистов описаны в п. 2 приложения
Map illegal HDL characters	On/Off	Исключение из имени нетлистов символов, не поддерживаемых стандартом VHDL.*
Bring out device-wide set/reset signals as ports	On/Off	Добавление глобальных сигналов запуска и перезапуска в порты устройства
Flatten busses into individual nodes	On/Off	Развёртывание шин сигналов в отдельные элементы. Также влияет на добавление агрегаций в описания присвоений сигналов.
Don’t write top-level HDL entity	On/Off	Запрет вывода Entity для корневого элемента
Truncate long hierarchy paths	On/Off	Ограничение длины путей по иерархии в именах элементов

Таблица 2

Объявления элементов при различных значениях “Map illegal HDL characters”

Значение	Имя элемента в объявлении	Комментарий
On	SIGNAL \counter|s[1]~6\ : std_logic;	Объявление сигнала
	\counter|s[2]~7\ : cycloneii_lcell_comb	Добавление компонента
Off	SIGNAL counter_as_a1_a_a6 : std_logic;	Объявление сигнала
	counter_as_a2_a_a7 : cycloneii_lcell_comb	Добавление компонента

Форматы VHDL-нетлистов

VHDL-нетлисты представляют собой некоторое подмножество стандартного языка описания устройства VHDL. При этом форматы данных в этих файлах не полностью соответствуют стандартам языка, поэтому синтез нетлистов данного типа в среде Quartus II не всегда возможен. Основное применение формата – проведение моделирования устройства в САПР ModelSim-Altera, но может быть использован и как формат для обмена данными с другими САПР.

VHDL-нетлисты могут включать:

- использование библиотеки IEEE и пакета std_logic_1164;
- объявление библиотек и пакетов, содержащих компоненты для элементов нижнего уровня языка;
- объявления типов внутри пакетов (PACKAGE);
- описания сущностей, используемых в нетлисте:
- описания архитектур сущностей.

В сущностях и архитектурах могут использоваться сигналы и порты могут иметь типы std_logic, std_logic_vector или типы на их основе, определённые внутри нетлиста. Описания сущностей могут содержать не только порты, но и generic-параметры.

В архитектурах VHDL для нетлистов допускается использование лишь некоторого подмножества структурного описания устройств. По результатам анализа спецификации нетлистов, в архитектуре могут быть использованы:

- объявления компонентов;
- присвоения как сигналов и их шин;
- использование агрегаций в присвоениях;
- доступ к элементам шин по индексу;
- использование оператора NOT.

Плоские и иерархические нетлисты

Quartus II поддерживает генерацию нетлистов в двух форматах: с сохранением иерархии устройства и без неё. Данные типы нетлистов соответственно называются иерархическими и плоскими. В иерархических нетлистах каждый модуль устройства представляет собой отдельную сущность (entity) и архитектуру (architecture), которые связываются без дополнительного объявления компонентов. Данное описание наиболее близко к чистому VHDL, но имеется ряд отличий, который исключает синтез нетлиста без его обратного инжиниринга:

- допускаются соединения между сигналами различных уровней иерархии;
- допускаются глобальные сигналы;
- именование сигналов и компонентов нарушает спецификацию формата VHDL, так как может включать запрещённые специальные символы.

Плоские нетлисты являются упрощённой версией, где всё описание устройства приводится в виде одной entity и одной архитектуры. При этом в объявлениях сигналов и компонентов указывается полный путь в исходной иерархии, что позволяет восстановить её во время импорта.

Поддержка VHDL-нетлистов в модуле ввода-вывода прототипа PHRT

Разработанный прототип средства реинжиниринга поддерживает далеко не все комбинации приведённых выше настроек. В таблице 3 приведены списки настроек, поддерживаемых в прототипе. Кроме перечисленного, в текущей реализации модуль ввода-вывода не поддерживает следующие возможности:

- загрузку определений компонентов из указанных библиотек;
- обработку описаний внутри пакетов;
- обработку объявлений пользовательских типов.

В настоящей версии прототипа при загрузке игнорируются указания на используемые библиотеки элементов, и по-умолчанию всегда используется cyclone_ii, т.е. поддерживаются только ПЛИС семейства Cyclone II, производимые фирмой Altera. Прочие ограничения были выявлены при импорте устройств, использующих мегафункции Altera (конфигурируемые IP-блоки), генерируемые средой Quartus II. Для сложных устройств (в том числе

микропроцессорных ядер) в нетлистах объявления данных компонентов не встречались, поэтому проблема с поддержкой компонентов была выявлена лишь на этапе тестирования прототипа PHRT. Проблема не затрагивает устройства, реализованные на чистом VHDL или Verilog без использования средств автогенерации нетлистов, используемых Quartus для мегафункций.

Таблица 3

Требуемые настройки генерации нетлистов для прототипа CAP

Параметр	Тип	Описание
Format for output	Список	Всегда VHDL
Architecture name	Строка	произвольно имя
Maintain hierarchy	On/Off	Поддерживаются и плоские (Off), и иерархические (On) нетлисты.
Map illegal HDL characters	On/Off	Всегда Off. Спец. символы используются для распознавания уровней иерархии.
Bring out device-wide set/reset signals as ports	On/Off	Значение может быть любым, так как в модуле ввода-вывода прототипа предусмотрено исключение несинтезируемых сигналов devoe, devclrn и devpor.
Flatten busses into individual nodes	On/Off	Значение может быть любым, там как расширение vhdl_netlist поддерживает преобразование шин и агрегаций в группы.
Don’t write top-level HDL entity	On/Off	Должен быть off, иначе не будет считан интерфейс корневого элемента.
Truncate long hierarchy paths	On/Off	Должен быть off, иначе восстановление архитектуры окажется невозможным.

Перечисленные выше ограничения относительно легко устранить, так как разработанная методика представления поддерживает все необходимые механизмы для работы с библиотеками и типами. Доработка модуля ввода-вывода является одной из перспективных задач дальнейшей разработки, необходимой для реализации полноценной САПР.

ПРИЛОЖЕНИЕ F. ПРИМЕР ВВЕДЕНИЯ СТРУКТУРНОЙ ИЗБЫТОЧНОСТИ В УСТРОЙСТВО

В данном приложении приведён пример пошагового выполнения реинжиниринга с использованием прототипа средства. При тестировании ставится задача демонстрации полного цикла реинжиниринга: от синтеза исходного проекта в Quartus II до синтеза и тестирования устройства после реинжиниринга. Для проверки возьмём какое-либо простое устройство, в котором введём структурную избыточность для одного из элементов. Требуется выполнить следующие шаги:

1. Выбрать тестовый проект для проверки команды.
2. Синтезировать нетлист в Quartus II.
3. Импортировать нетлист в прототипе CAP.
4. Вызвать команду мажорирования одного из модулей устройства.
5. Провести верификацию синтезированного устройства в CAP.
6. Экспортировать устройство в VHDL.
7. Синтезировать полученное устройство.
8. Проверить корректность работы построенного устройства.

Перечисленные выше шаги полностью автоматизированы посредством расширений прототипа PHRT.

Подготовка тестового проекта

В связи с наличием недоработок в прототипе, требуется взять простое устройство. Его решено реализовать на VHDL при помощи поведенческого описания, чтобы продемонстрировать преимущества использования внешней среды для формирования структурных описаний.

В качестве тестируемого устройства возьмём счётчик с управляемым модулем счёта, разрешением счёта, возможностью внешнего сброса и автоматического сброса после достижения требуемого значения. Исходные коды данного устройства приведены ниже в листинге 1.

Листинг 1. Исходные коды тестируемого устройства

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

-- Управляемый счётчик с возможностью автоматического сброса
ENTITY sat_counter IS
	Generic (
		width: integer:=2
	);
	Port (
		-- Clock
		CLK:	IN std_logic;
		-- Reset if nRst=0
		nRST:	IN std_logic;
		-- Count enable
		ENA:	IN std_logic:='1';
		-- Automatic reset if code=Sat
		autoRST: IN std_logic:='1';
		-- Max code
		Sat:	IN std_logic_vector(width-1 DOWNTO 0):=(others => '1');
		-- Code output
		Q:	OUT std_logic_vector(width-1 DOWNTO 0)
	);
END sat_counter;

ARCHITECTURE arch OF sat_counter IS
	SIGNAL SUM: std_logic_vector(width-1 DOWNTO 0);
BEGIN
clk_ev:
```

```
    PROCESS(clk,nRST)
    BEGIN
        if (nRST='1') then
            SUM <= (others => '0');
        elsif (CLK'event and CLK='1') then
            if (SUM = Sat) then
                if (autoRST='1') then
                    SUM <= (others => '0');
                end if;
            elsif (ENA = ‘1’) then
                SUM <= std_logic_vector( unsigned(SUM) + 1 );
            end if;
        end if;
    END PROCESS;
    Q <= SUM;
END arch;
```

Был произведён синтез устройства в Quartus II со стандартными настройками проекта. Схема на уровне RTL, сформированная средством RTL Netlist Viewer, приведена на рисунке 1.

Для хранения накопленного результата были сгенерированы триггеры. Сравнение адреса идёт через компаратор, инкрементация – через сумматор. Код загрузки из сумматора проходит через серию мультиплексоров, которые управляются входными сигналами (в т.ч. ENA) и результатом сравнения. Асинхронный сброс подан на соответствующие входы сброса триггеров. Для проверки работоспособности устройства проведено его моделирование, результаты которого приведены на рисунке 2.

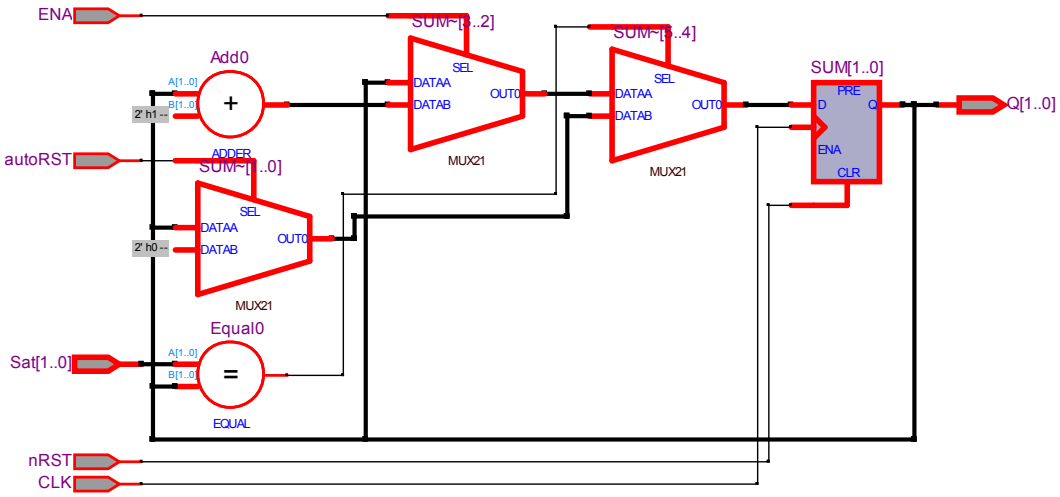


Рис. 1. RTL-схема синтезированного устройства

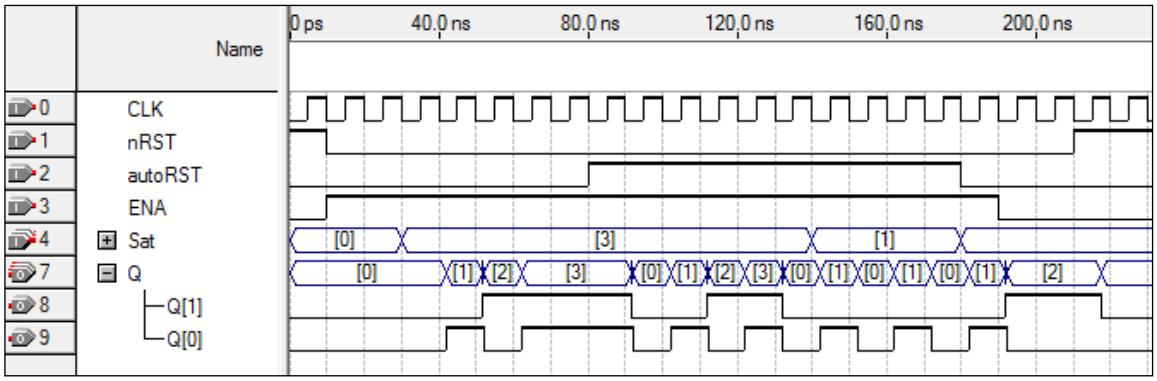


Рис. 2. Результаты моделирования устройства

Настройка вывода генерации нетлистов для реинжиниринга в среде Quartus II

Для проведения моделирования требуется подготовить VHDL-нетлисты с описанием исходного устройства. Для этого нужно изменить настройки генерации нетлиста в Project Options/EDA Tool/Simulation, а потом нажать на кнопку “More EDA Netlist Writer Settings” и в открывшемся окне настроить дополнительные параметры. Указания по настройке генерации нетлистов и обоснование опций приведены в приложении Е. На рисунках 3 и 4 приведены скриншоты обоих окон настройки.

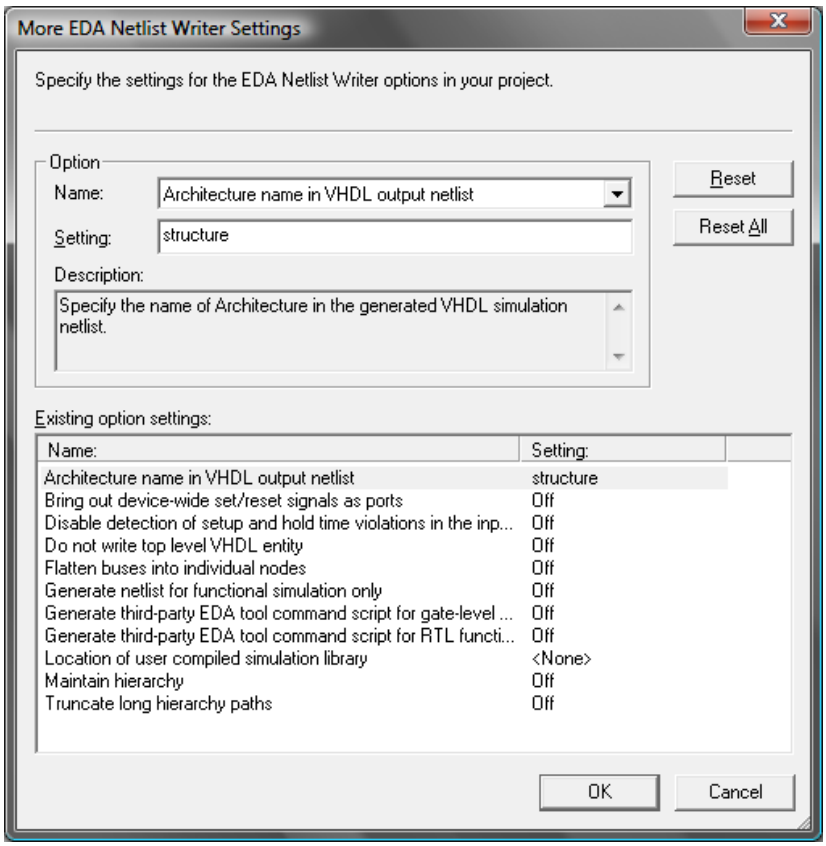


Рис. 3. Окно дополнительных настроек генерации нетлистов

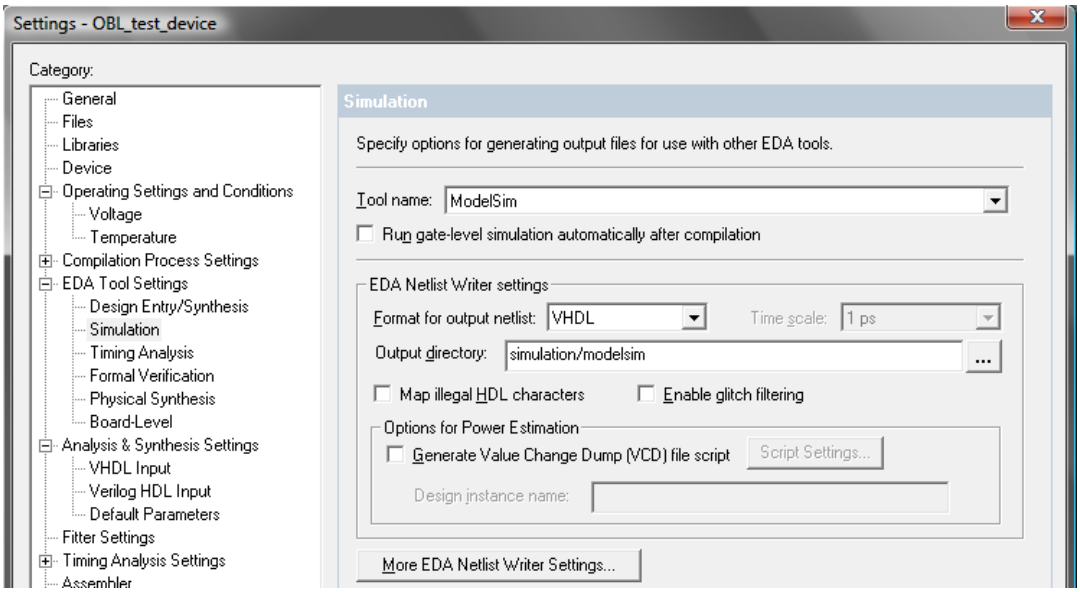


Рис. 4. Окно настроек EDA Tool/Simulation в Quartus

Проведение реинжиниринга

Для выполнения операции по внесению структурной избыточности была разработана скриптовая программа, листинг которой приведён ниже:

Листинг 2. Скрипт проведения преобразования

```
# Загрузка библиотек
```

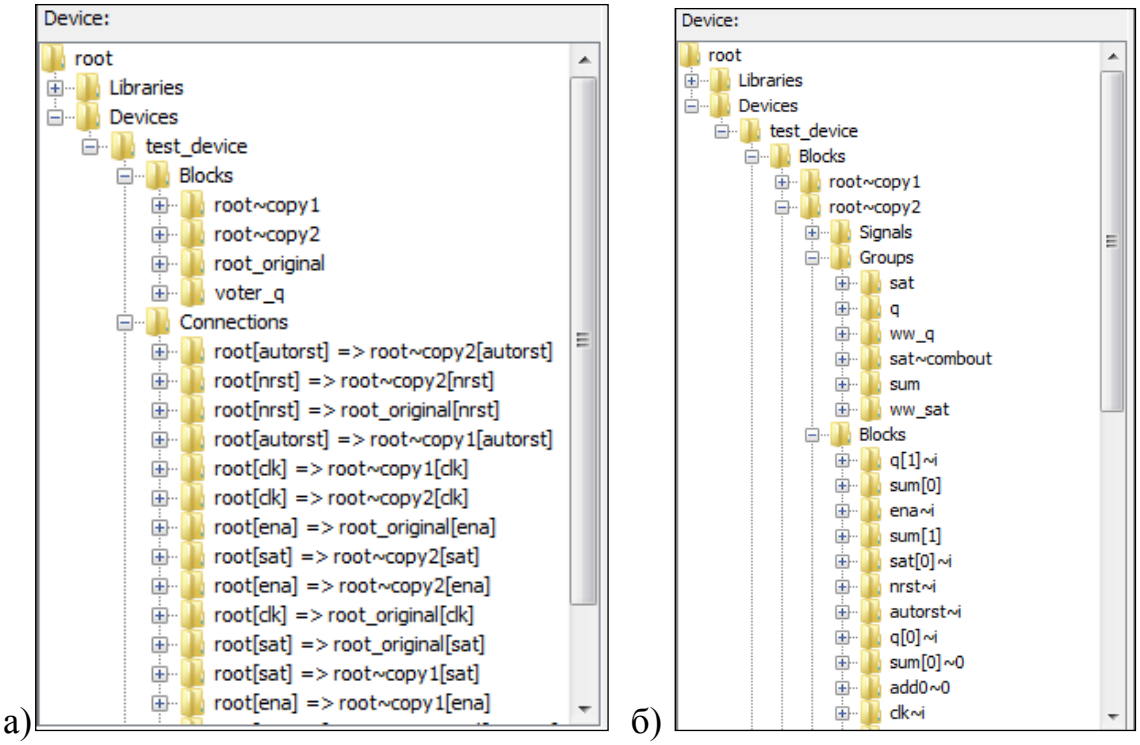
```
clibs_load Netlist
clibs_load Reliability
# Загрузка устройства
import_netlist -f -p test_device:\ E:\Temp\obl_demo\OBL_test_device.vho
# Мажорирование устройства
majorize test_device:
check test_device:
# Вывод результатов
export_vhdl -d E:\Temp\obl_demo\test_device.vhd test_device:
```

Данная программа была вызвана в консоли графического интерфейса прототипа при помощи системной команды run. Для команды был установлен флаг s, поэтому на дисплей были выведены отчёты по всем внутренним командам программы. В листинге 3 приведён сформированный отчёт.

Листинг 3. Отчёт о выполнении скриптовой программы

```
.\[R]root\>>run -s E:\Temp\obl_demo\demo.m
Subcommands:
 >>>clibs_load Netlist
   Library Netlist[0.0.0] was loaded
 >>>clibs_load Reliability
   Library Reliability[0.1.1] was loaded
 >>>import_netlist -f -p test_device:\ E:\Temp\obl_demo\OBL_test_device.vho
   Node loaded
 >>>majorize test_device:\
   Node test_device:\ majorized.
 >>>check test_device:\
   Errors: 0  Warnings: 0  Messages: 0
 >>>export_vhdl test_device:\
   Node test_device:\ has been exported.
Script executed
```

В графическом представлении модели устройства (рисунок 5) можно видеть, что созданы копии корневого элемента и добавлен голосователь для выходных сигналов. В списке соединений Connections присутствуют все необходимые подключения между элементами, таким образом алгоритм введения структурной избыточности выполнен верно.



а) б)

Рис. 5. Структура устройства после преобразования в PHRT

(а – структура устройства; б – структура скопированного элемента)

В листинге 4 приведён пример отчёта о структуре устройства, сформированный командой ls. Полный отчёт с рекурсией по уровням привести невозможно из-за большого числа элементов в иерархии.

Листинг 4. Структура корневого элемента построенного устройства

```
.\[R]root\>>ls test_device:
Device "test_device"
  -- Blocks:
      BlockReference "root~copy1" #REFERS test_device:\root_original\
      BlockReference "root~copy2" #REFERS test_device:\root_original\
      BlockInstance "root_original"
      VoterReference "voter_q" #REFERS [l]reliability\voter3\
```

```
-- Connections:
    SignalConnection "root[autorst] => root~copy2[autorst]"
    SignalConnection "root[nrst] => root~copy2[nrst]"
    SignalConnection "root[nrst] => root_original[nrst]"
    SignalConnection "root[autorst] => root~copy1[autorst]"
    …
-- Libraries:
    Library "test_device"
```

По построенному отчёту видно, что был создан только один голосователь, хотя устройство синтезировалось для двухразрядного счётчика. Таким образом, при выполнении теста в модуле ввода-вывода была обнаружена очередная ошибка. К сожалению, на момент написания диссертации устранена она не была. Чтобы закончить пример, в устройство вручную был добавлен ещё один голосователь. После исправления ошибки с соединениями и ряда модификаций устройство было успешно синтезировано. Результаты моделирования (см. рисунко 6) совпали с результатами для исходного устройства

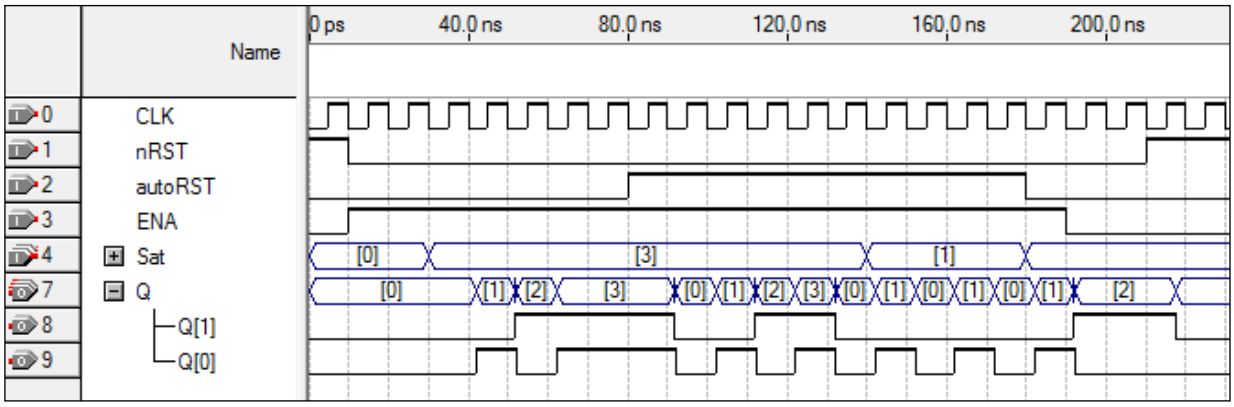


Рис. 6. Результаты моделирования устройства, сгенерированного PHRT

Таким образом, на простом примере встраивания структурной избыточности подтверждена работоспособность прототипа. Более сложные примеры его использования рассмотрены в диссертации.

ПРИЛОЖЕНИЕ G. АКТЫ О ВНЕДРЕНИИ

ООО «Синопсис СПб»

197376, г. Санкт-Петербург, ул. Профессора Попова, 23 литер д

Тел. +7 (812) 408-74-00
Факс +7 (812) 406-70-52

SYNOPSYS®

Акт о внедрении программного продукта

Расширение системы Jenkins CI для встраивания средств тестирования

Настоящий акт выдан Ненашеву Олегу Вячеславовичу для представления в диссертационный совет. Акт свидетельствует о том, что расширение test-point-insertion-plugin для системы непрерывной интеграции Jenkins CI внедрено в ООО "Синопсис СПб", являющегося российским представительством компании Synopsys Inc. (NASDAQ:SNPS) (www.synopsys.com). Процесс внедрения происходил с 24 мая 2014г. по 20 февраля 2015г.

Расширение test-point-insertion-plugin разработано аспирантом Санкт-Петербургского государственного политехнического университета Ненашевым Олегом Вячеславовичем. Заявлены следующие функциональные и технические характеристики расширения:

- возможность встраивания в маршруты верификации аппаратных продуктов на базе системы непрерывной интеграции Jenkins CI;
- использование инструментария реинжиниринга устройств PHRT, основанного на гибридной модели представления устройств;
- возможность автоматического встраивания средств внутрисхемного тестирования и тестовых интерфейсов в прототипы на ПЛИС;
- возможность описания алгоритмов анализа, преобразования и верификации проектов с помощью скриптового языка Tcl;
- возможность вызова внешних САПР (в т.ч. продуктов Synopsys Inc.) для выполнения операций синтеза и прошивки устройств на ПЛИС;
- возможность экспорта синтезируемых описаний в формате EDIF после выполнения преобразований;
- поддержка версий ядра Jenkins CI, начиная с 1.509.3;

test-point-insertion-plugin эксплуатируется в нескольких проектах внутри компании Synopsys Inc., связанных с автоматизацией процессов разработки и верификации процессорного IP ARC и отладочных плат, которые являются продуктами компании Synopsys Inc. Подтверждены заявленные характеристики расширения, новизна и практическая ценность использованных в программном продукте гибридной модели, методов и средств.

Генеральный директор _____ / Г.С. Никодимов /

31 марта 2015 года

Общество с ограниченной ответственностью «Синопсис СПб» Synopsys SPb Санкт-Петербург

ООО «ЭсДиСи»
СПб, ул Руднева д.4 лит.А, пом.14-Н, тел. +7 (981) 1293929, E-mail: antonov@scideco.com

Акт о внедрении

Инструментарий для задач реинжиниринга цифровых устройств PHRT

Настоящий акт выдан аспиранту Санкт-Петербургского политехнического университета Петра Великого Ненашеву Олегу Вячеславовичу для представления в диссертационный совет Д 212.229.18.

Акт свидетельствует о том, что инструментарий PHRT (Programmable Hardware Reengineering Toolkit), разработанный Ненашевым О.В., успешно апробирован в ООО ЭсДиСи. Инструментарий применялся для подготовки и модификации прототипов устройств с целью проведения тестирования на ПЛИС. Процесс апробации происходил с 16 июня 2014г. по 23 декабря 2014г. По результатам апробации подтверждены заявленные характеристики входящих в PHRT инструментальных средств:

- интеграция инструментария в среду разработки Eclipse;
- возможность описания алгоритмов анализа, преобразования и верификации проектов с помощью скриптового языка Tcl;
- использование гибридной модели устройств, сочетающей в себе структурные описания (нетлисты) и высокоуровневые компоненты из языков VHDL и Verilog;
- встраивание средств внутрисхемного тестирования и тестовых интерфейсов в прототипы на ПЛИС посредством расширения ict-test-insertion для инструментария PHRT;
- генерация тестовой инфраструктуры по списку тестовых сигналов;
- возможность вызова САПР Quartus II для синтеза устройств и их последующей прошивки на ПЛИС семейства Cyclone IV;

Применение инструментария PHRT позволяет значительно снизить затраты на организацию тестирования прототипов устройств на ПЛИС. Это позволяет расширить тестовые планы для устройств и выявить большее количество ошибок на ранних этапах проектирования устройств, снизить общую стоимость их устранения.

Генеральный директор, к.т.н. / А.П. Антонов /

"24" Декабря 2014г.

Общество с ограниченной ответственностью «ЭсДиСи» «SDC» Co Ltd Санкт-Петербург

УТВЕРЖДАЮ

Проректор по образовательной деятельности
ФГАОУ ВО "СПбПУ"
Разинкина Е.М.

"__" _____ 2015г.

Акт о внедрении в учебный процесс

результатов диссертационной работы Ненашева Олега Вячеславовича

Настоящий акт выдан аспиранту Санкт-Петербургского политехнического университета Петра Великого Ненашеву Олегу Вячеславовичу для представления в диссертационный совет.

Акт свидетельствует о том, что предложенные в диссертации Ненашева О.В. гибридные модели, методы реинжиниринга и внутрисхемного тестирования аппаратных средств, а также прототипы инструментальных средств внедрены в учебный процесс на кафедре компьютерных систем и программных технологий, входящей в Институт информационных технологий и управления Санкт-Петербургского политехнического университета Петра Великого. Изложенные в диссертации «Реинжиниринг цифровых устройств и встраивание средств тестирования на базе многоуровневых моделей» разработки используются в курсе «Проектирование аппаратных средств вычислительных систем», преподаваемом студентам 5-6 курсов в рамках магистерской программы 230100 Ф(М) О ((ФТК):КСиПТ).

Заведующий кафедрой КСПТ, д.т.н. _____ / В.Ф. Мелехин /

"30" 04 2015г.