Алехин В.А.

Наименование дисциплины:

# Встраиваемые вычислительные системы

Для магистрантов
По направлению подготовки 09.04.01

Москва, 2018

Оглавление

## Список обозначений

| | |
|---|---|
| BPM | Business Process Modeler - модуль построения моделей бизнес-процессов в форме диаграмм потоков данных |
| BPwin | CASE-средство , реализующее в качестве методологии IDEF0 |
| CAD | Computer Aided Design – автоматизированное проектирование |
| CAE | Computer Aided Engineering– поддержка инженерных расчетов |
| CALS | Continuous Acquisition and Life cycle Support - непрерывная информационная поддержка поставок и жизненного цикла |
| CAM | Computer Aided Manufacturing - компьютерная поддержка изготовления |

| | |
|---|---|
| CAN | Controller Area Network — сеть контроллеров) — стандарт промышленной сети, ориентированный, прежде всего, на объединение в единую сеть различных исполнительных устройств и датчиков. |
| CAPP | Computer Aided Process Planning - средства планирования технологических процессов |
| CASE | Computer-Aided Software/System Engineering – разработка программного обеспечения/программных систем с использованием компьютерной поддержки |
| CMMI | |
| COM | Component Object Model – компонентная модель объектов |
| CORBA | Common Object Request Broker Architecture – общая архитектура с посредником обработки запросов объектов |
| DCOM | Distributed COM – распределенная COM |

| | |
|---|---|
| DFD | data flow diagrams — диаграммы потоков данных |
| ECAD | Electronic CAD- системы автоматизированного проектирования (САПР) для радиоэлектроники |
| EDA | Electronic Design Automation – автоматизированное проектирование электроники |
| EDM | engineering data management – управление инженерными данными |
| ERD | entity-relationship diagrams |
| ERwin | CASE-средство, которое в качестве методологии использует IDEF1X |
| ERX | Entity-Relationship eXpert - модуль концептуального моделирования данных |
| ER-модель | entity-relationship model, модель «сущность — связь» |
| ESD | Entity Structure Diagram - диаграммы структур сущностей |

| | |
|---|---|
| ICAM | Integrated Computer-Aided Manufacturing - интегрированная компьютеризация производства |
| IDEF | Integrated DEFinition |
| IDEF0 | метод функционального моделирования |
| IDEF1 | метод моделирования информационных потоков |
| IDEF1X | |
| IDEF1X | метод моделирования данных и проектирования реляционных баз данных |
| IDEF2 | метод динамического моделирования систем |
| IDEF3 | метод получения описания функционирования системы и моделирования как причинно-следственных связей |
| IDEF4 | метод объектно-ориентированного проектирования. |
| IDEF5 | метод получения онтологического описанияи исследования сложных систем |

| | |
|---|---|
| ISO | International Organization for Standardization- международная организация по стандартизации |
| IT | Информационные технологии |
| MSF | Microsoft Solutions Framework — методология разработки программного обеспечения |
| OLE I | Object Linking and Embedding – связывание и внедрение объектов |
| OSTD | (ObjectStateTransitionDescription)– «внешнее описание» |
| PDM | Product Document Management – система управления данными об изделии |
| PFD | Process Flow Description - «внутреннее описание» и описания переходов из одного состояния в другое |
| PIM | Product information management – управление информацией об изделии |
| PLM | Product Lifecycle Management – термин |

| | |
|---|---|
| | используется для обозначения процесса управления полным циклом изделия |
| RAD | Rapid Application Development – быстрая разработка приложений |
| RDM | Relational Data Modeler - модуль реляционного моделирования |
| RUP | Rational Unified Process - рациональный унифицированны процесс. |
| SADT | Structured Analisys and Design Technique - технология структурного анализа и проектирования |
| SWEBOK | Software Engineering Body of Knowledge - Сфера знаний в области разработки программного обеспечения |
| TDM | technical data management - управление техническими данными |
| TIM | technical information management - управление |

| | |
|---|---|
| | технической информацией |
| UML | Unified Modeling Language - унифицированный язык моделирования |
| WRM | Workgroup Repository Manager - менеджер репозитория рабочей группы |
| АПК | Аппаратно-программный комплекс |
| АС | Автоматизированная система |
| ИПЭ | Информационная поддержка эксплуатации |
| ПАК | Программно-аппаратный комплекс |
| ПО | Программное обеспечение |
| ПС | Программные средства |

# Глава 1. Понятие программно-аппаратных комплексов и их применение

## 1.1. Определения программно - аппаратных комплексов

**Программно-аппаратный комплекс (ПАК)** - это ***набор технических и программных средств,*** работающих совместно для выполнения одной или нескольких сходных задач.

**Аппаратно-программный комплекс (АПК)** - ***техническое решение концепции алгоритма работы сложной системы***, управление которой, осуществляется, как правило, исполнением кода из определённого базового набора команд (системы команд), описанных в документации.

Состоит, соответственно, из двух основных частей:

**Аппаратная часть** - устройство сбора и/или обработки информации.

**Аппаратное обеспечение** - комплекс электронных, электрических и механических устройств, входящих в состав

системы или сети. Аппаратное обеспечение включает: - компьютеры и логические устройства; - внешние устройства и диагностическую аппаратуру; - энергетическое оборудование, батареи и аккумуляторы.

В компьютерном сленге часто используется слово «hard» (жёсткий, твёрдый), произошедшее от английского слова «hardware».

**Программная часть** - специализированное ПО (как правило, написано компанией - производителем аппаратной части), обрабатывающее и интерпретирующее данные, собранные аппаратной частью.

**Программное обеспечение** - совокупность программ системы обработки информации и программных документов, необходимых для эксплуатации этих программ (ГОСТ 19781-90).

***Програ́ммное обеспе́чение*** (ПО) — программа или множество программ, используемых для управления компьютером (ISO/IEC 26514:2008).

Другие определения из международных и российских стандартов:

- Совокупность программ системы обработки информации и программных документов, необходимых для эксплуатации этих программ (ГОСТ 19781-90);
- все или часть программ, процедур, правил и соответствующей документации системы обработки информации (ISO/IEC 2382-1:1993);
- компьютерные программы, процедуры и, возможно, соответствующая документация и данные, относящиеся к функционированию компьютерной системы (IEEE Std 829—2008).

Программное обеспечение является одним из видов обеспечения вычислительной системы, наряду с техническим (аппаратным), математическим, информационным, лингвистическим, организационным, методическим и правовым обеспечением.

Академические области, изучающие программное обеспечение,— это информатика и программная инженерия.

В компьютерном сленге часто используется слово «софт», произошедшее от английского слова «software».

Область применения ПАК

Область применения ПАК охватывает широкий спектр деятельности, как социальную сферу, так и объекты предпринимательства:

- оптовая и розничная торговля;
- информационно-библиотечные ресурсы;
- кредитные организации;
- образовательные учреждения;
- производственная сфера;
- оборонная сфера;
- логистические предприятия;
- предприятия коммуникаций и связи;

- организации здравоохранения;
- туристическая деятельность и т.д.

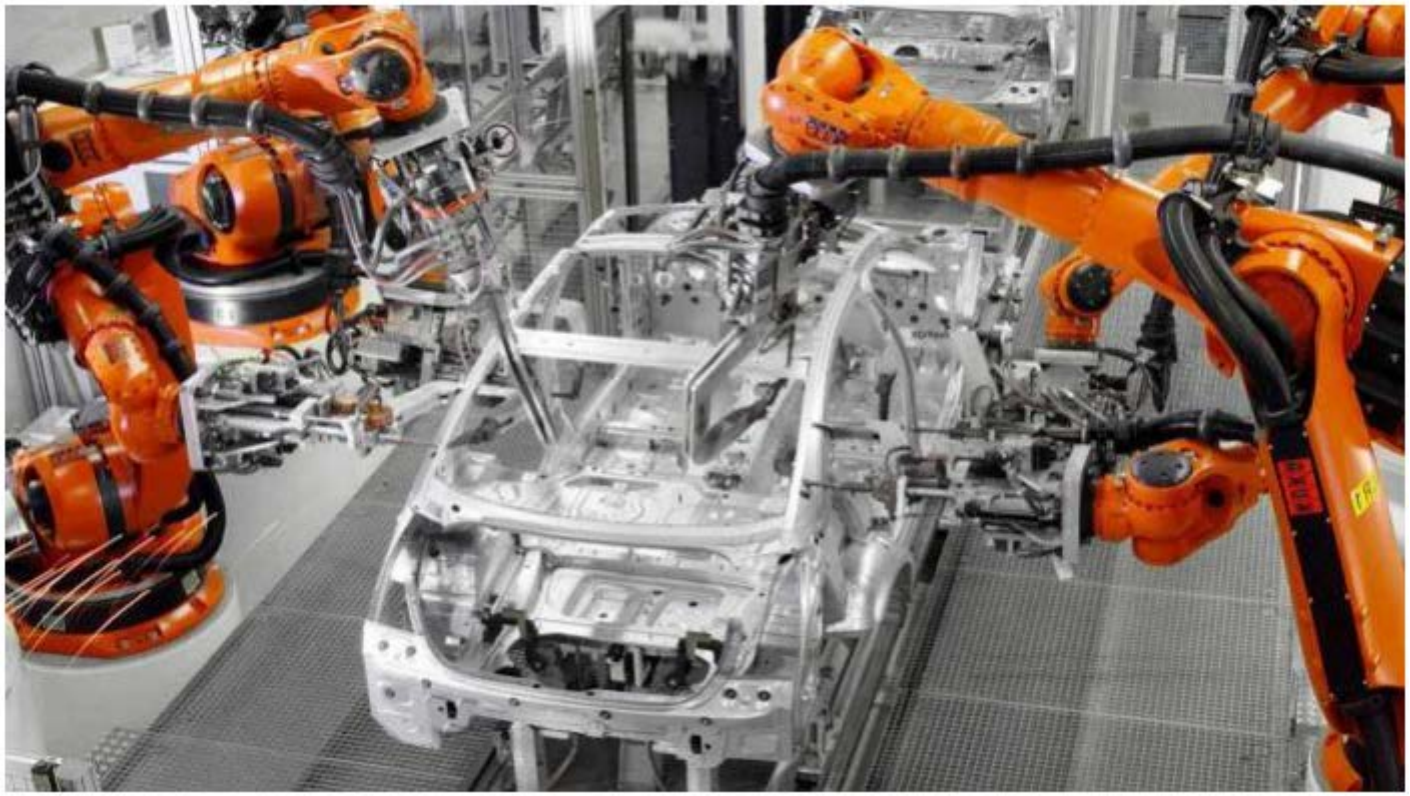
Для решения конкретных задач вышеперечисленных отраслей созданы следующие программно-аппаратных комплексов:

- автоматизированное рабочие место;
- электронная очередь;
- видеонаблюдения и контроля доступа;
- системы распознавания образов;
- медицинской диагностики;
- защиты конфиденциальной информации;
- автоматической книговыдачи;
- имитационное моделирование и постановка эксперимента;
- решение управленческих задач;
- системы видеоанализа;
- системы маскировки передачи защищаемой информации и т.д.

## 1.2. Примеры применения ПАК

### 1.2.1. Комплексы программно-аппаратные: назначение и принцип работы

Компьютеры позволяют автоматизировать огромное количество операций, которые ранее были под управлением людей. Скорость обработки данных компьютерами раскрывает широкие возможности, применимые в различных сферах человеческой деятельности, оставляя людям лишь самые важные функции по настройке и управлению различных цифровых систем. Современные комплексы программно-аппаратные (ПАК) позволяют оптимизировать не только отдельные операции, но и целые рабочие процессы, состоящие из множества действий.



## 1.2.2. Определение ПАК

Формально под определение «комплексы программно-аппаратные» подпадает огромное количество различных цифровых систем, начиная от простых персональных компьютеров, заканчивая

объемными хранилищами данных и комплексными системами безопасности. Однако среди IT-специалистов есть определённое разделение. Минимальной частью системы является устройство, в зависимости от сложности оснащённое встроенным ПО или просто соединённое с центром обработки сигналов. Несколько устройств, объединённых общей задачей, с единым узлом сбора и обработки информации, уже можно назвать полноценным ПАК. Многопользовательские сети с распределением функций поддержки и управления, постоянным обслуживанием которых занимаются специалисты, называются автоматизированными системами, в состав которых могут входить как ПАК, так и различные устройства с распределением степени доступа.

### 1.2.3. Сферы применения.

IT-специалист способен разработать и развернуть собственный комплекс аппаратных и программных средств, позволяющих обмениваться и обрабатывать практически любые виды информации

для технологических задач, выполнение которых требует автоматизации. Также существуют и готовые решения от различных производителей, активно внедряемые в различные сферы деятельности. Области применения ПАК довольно обширны на сегодняшний день. Перечислим основные из них: Управление базами данных. Системы обработки запросов от многочисленных пользователей к общему хранилищу информации могут быть как разработаны работниками IT-отдела, так и приобретены готовыми у производителей. Медицинские исследования. Всё больше методов диагностики человеческих заболеваний автоматизируются для повышения скорости и качества обследований. Среди них - и простейший программно-аппаратный комплекс, позволяющий за мгновения получить рентгеновский снимок на экране в кабинете врача, и сложнейшие комплексы ведения огромного количества больничных карт пациентов.

## 1.2.4. Производственная сфера.

Автоматическое управление широко внедряют во множество производственных станков и аппаратов, это позволяет исключить человеческий фактор и снизить затраты на контроль, который можно осуществлять прямо в процессе изготовления продукции. Системы безопасности. Широкий спектр различных решений для осуществления пожарной защиты и прочей охранной деятельности материальных ценностей, а также множество комплексов по защите информации от уничтожения или кражи.

## Преимущества и недостатки готовых ПАК перед комплексами собственной сборки

**Плюсы:** Готовое оборудование, на развёртывание которого уйдёт минимум времени. Все компоненты комплекса проходят

многоступенчатое тестирование на совместимость и отказоустойчивость. Специалисты компаний, выпускающих ПАК, используют оригинальное ПО для максимальной оптимизации взаимодействия всех частей комплекса. Минимальное количество обслуживающего персонала. Многие ПАК не требуют от пользователя глубоких знаний IT-технологий, а техподдержка от производителей способна помочь в решении большинства типовых проблем без вызова специалиста. Гибкость. Готовый комплекс аппаратных и программных средств, позволяющих выполнять расширенный спектр операций в своей области привлекает покупателей универсальностью.

**Минусы:** Стоимость готовых решений может быть выше, чем разработанные для конкретного процесса комплексы программно-аппаратные от IT-специалистов внутри компании. Гибкость сборных ПАК может быть недостаточна для решения определённых задач, где требуется уникальный подход. Несанкционированное вмешательство

в аппаратную или программную части комплекса зачастую приводит к ограничению гарантийного обслуживания.

### 1.2.5. Примеры ПАК от специалистов Oracle

Компания Oracle — признанный во всём мире авторитет в вопросах обслуживания баз данных. Начиная с 2008 года компания выпускает Exadata — комплекс аппаратных и программных средств, позволяющих компьютерам и рабочим станциям пользователей одновременный разноуровневый доступ к определённой информации. Отличительной особенностью Exadata является применение ячеек — отдельных автономных серверов, что позволяет быстрее обрабатывать запросы с большим объёмом информации. Среди других готовых решений от Oracle существует ПАК для быстрого развёртывания системы — Database Appliance, специально предназначенный для минимизирования затрат и времени на установку и настройку необходимого ПО. Также есть комплексы для использования облачных технологий (Private Cloud Appliance), для

хранения большого объёма информации (Big Data Appliance) и даже для защиты целостности данных (Zero Data Loss Recovery Appliance).


## Принципы работы ПАК Oracle

Exadata представляет собой программно-аппаратный комплекс предназначенный для централизованного управления базами данных и распределения хранимых файлов между отдельными ячейками. Тогда как основная система работает с большинством запросов, отсеивая ненужную информацию ещё в системе хранения, ячейки

могут работать напрямую, передавая значительные объёмы сведений. Данные распределяются на несколько ячеек, что позволяет увеличить производительность запросов. Zero Data Loss Recovery Appliance освобождает ресурсы центральной системы, затрачиваемые на резервное копирование данных. Этот комплекс разработан намеренно для баз данных под управлением приложений Oracle и легко интегрируется для защиты нескольких баз, предоставляя возможности сквозной проверки и защиты критичной информации.

### 1.2.6. Медицинские ПАК

Наглядно простые комплексы программно-аппаратные можно встретить в специально оборудованных «центрах здоровья». Эти отделения российских поликлиник и больниц открываются для популяризации здорового образа жизни, согласно приказу Минздравсоцразвития с 2009 года. Они позволяют быстро провести общую диагностику здоровья на основе показаний основных

параметров тела. Биоимпендансметр АВС-01 «Медасс», входящий в состав ПАК «Здоровье-Экспресс», позволяет за считанные минуты оценить психосоматическое и физическое состояние пациента, вывести результаты на компьютер и назначить рекомендации по улучшению здоровья.

## 1.2.7. ПАК на производстве

Применение шаговых двигателей позволяет выполнять производственные задания с точностью, недоступной человеческим рукам. Оснащение производственных аппаратов встроенными компьютерами — это и есть применение специализированных ПАК. Принцип действия таких комплексов довольно прост: квалифицированный рабочий задаёт схему изделия, используя встроенное программное обеспечение, после чего станок выполняет заданное количество запланированных действий

## 1.2.8. Системы мониторинга и безопасности

Различные анализаторы, тестеры и датчики чаще всего объединены в общую систему мониторинга, в том числе и программно-аппаратный комплекс. Системы видеонаблюдения, контроля доступа, как физического, так и информационного, - всё это примеры применения ПАК. Универсализация протоколов взаимодействия различных приборов позволяет создавать комплексы любой сложности и функциональности. Производители не стоят на месте, предлагая множество готовых решений для организаций и производств

## 1.2.9. ПАК в образовании

Необходимость регулярного сбора и классификации информации о современных программно-аппаратных средствах не вызывает сомнений. На её основе можно проводить грамотный и научно

обоснованный анализ полученной информации, используя современные методы оценки качества и классификации программно-аппаратных средств. Это, в свою очередь, даёт возможность делать заключения на предмет целесообразности внедрения различных программно-аппаратных комплексов в вузах России.

Классификационные признаки образовательных программно-аппаратных комплексов (ПАК) представляют собой следующую совокупность: дидактическая направленность, программная реализация, техническая реализация и предметная область применения (рис.).


*Алехин В.А. Встраиваемые системы. РТУ – МИРЭА, 2019*

Анализ современного состояния компьютерных информационных технологий в высшей школе показывает, что с появлением персональных компьютеров начали появляться тестирующие и контролирующие программы, поддерживающие традиционные технологии обучения в первую очередь в области естественнонаучных дисциплин, где методики обучения носят исторически устоявшийся характер. Затем с развитием технических возможностей персональных компьютеров компьютерные учебные программы (КУП) стали оснащаться возможностями моделирования.

Примеры: MultiSim, TINA, OrCAD, Proteus, SystemC.

Последние достижения в программном обеспечении позволяют перейти к использованию элементов объектно-ориентированного программирования (например, с использованием языков, входящих в программный продукт Microsoft Visual Studio).

### 1.2.10. Выгодны ли ПАК?

Итак, программно-аппаратные комплексы действительно способны сэкономить время и обеспечить надёжную работу в нужной сфере. Может показаться, что необходимость в ITспециалистах отпадает, но разобраться самим во всех тонкостях готовых ПАК не так уж легко. Безусловно, о целесообразности применения любых автоматизированных систем может судить лишь профессионал с достаточной квалификацией. Скорее, ПАК способны значительно ускорить работу специалиста, нежели заменить его совсем

## 1.3. Аппаратно-программные комплексы информационной поддержки эксплуатации

Начиная с 2005 г., Компанией АО «ИК «НЕОТЕК МАРИН» совместно с предприятиями ОАО «Коломенский завод», ООО

«Уральский дизель-моторный завод», ОАО «Пролетарский завод», ОАО «Адмиралтейские верфи», ПАО «ОДК-Сатурн» выполнен ряд опытно-конструкторских работ по разработке бортовых **переносных комплексов информационной поддержки эксплуатации (ИПЭ)** энергетического оборудования, **обучения и тренажа** личного состава.

Аппаратно-программные комплексы типа ИПЭ поставлены на современные корабли ВМФ РФ (более 10 проектов) и успешно показали себя как удобный инструмент, обеспечивающий решение следующих задач информационного обеспечения процессов интегрированной логистической поддержки сложного наукоёмкого оборудования:

- представление в электронном виде обслуживающему персоналу информационно-справочной и процедурно-технологической информации по содержанию, адекватной

эксплуатационной документации, поставляемой на бумажном носителе;

- визуализация процедур регламентного технического обслуживания (ТО) и ремонта;
- информационная поддержка обслуживающего персонала по обнаружению типовых неисправностей объекта и способам их оперативного устранения;
- мониторинг технического состояния функциональных комплексов с использованием переносных диагностических приборов и информации, регистрируемой бортовыми средствами контроля и управления;
- сбор и анализ статистики процессов эксплуатации энергетического оборудования на объекте;
- информационная поддержка управления запасами, заказом запасных частей и расходных материалов;

- информационная поддержка планирования материально-технического обеспечения, технического и сервисного обслуживания энергетического оборудования;
- обеспечение информационной поддержки выбора допустимых и рациональных режимов использования энергетического оборудования с учетом различных внешних условий;
- автоматизированная оценка эффективности эксплуатации оборудования и топливоиспользования;
- обучение обслуживающего персонала составу, устройству и алгоритмам функционирования изделия, правилам его эксплуатации и регламентного обслуживания;
- тренаж обслуживающего персонала по использованию энергетического оборудования в нормальных и аварийных ситуациях;
- тестирование личного состава на предмет допуска к использованию оборудования.


В процессе разработки используется программный инструментарий собственной разработки — линейка продуктов НЕО

*Алехин В.А. Встраиваемые системы. РТУ – МИРЭА, 2019*

ЭКСПЕРТ. Это позволяет эффективно разрабатывать и использовать такие информационные объекты как интерактивные и анимационные схемы, интерактивные трехмерные модели, трехмерные интерактивные анимации операций технического обслуживания и ремонта, виртуальные отсеки и помещения, а также другие сложные информационные объекты.

В процессе эксплуатации комплексов действует система послепродажного обслуживания (система интегрированной логистической поддержки), которая работает на основе следующих принципов:

- Обеспечение строгого соответствия информационной базы и прикладного ПО на всех этапах жизненного цикла энергетического оборудования (учет всех эксплуатационно-значимых изменений в КД и ЭД);
- Дальнейшее наращивание функциональных возможностей для повышения эффективности базовых и типовых задач

эксплуатации энергетического оборудования в создаваемых и сопровождаемых средствах **ИПЭ**;

- Обновление ранее разработанных версий прикладного ПО в предыдущих образцах ИПЭ до уровня функциональных возможностей последнего модернизированного образца;
- Целенаправленное и систематическое обеспечение подготовки ***личного состава кораблей и специалистов соединений*** использованию по назначению разработанных и поставленных Заказчику комплексов ИПЭ;
- Совершенствование предприятием-разработчиком комплексов ИПЭ единой системы учета, сопровождения и поддержки информационных баз и прикладного ПО на предприятии-изготовителе энергетического оборудования, в КБ-проектанте и на объекте использования программной продукции.


*Алехин В.А. Встраиваемые системы. РТУ – МИРЭА, 2019*


*Алехин В.А. Встраиваемые системы. РТУ – МИРЭА, 2019*

Аппаратные средства переносных комплексов выполнены в защищённом исполнении и предназначены для эксплуатации в экстремальных условиях.

Следует отметить, что разработанное прикладное программное обеспечение также эффективно используется для создания береговых учебных центров в целях подготовки экипажей кораблей. Специальный программный инструментарий НЕО ЭКСПЕРТ.ПРЕПОДАВАТЕЛЬ и НЕО ЭКСПЕРТ.КОП позволяет легко формировать на базе разработанного в рамках ОКР контента компьютерные обучающие программы и системы, позволяющие в автоматизированном режиме проводить обучение личного состава с использованием современных аппаратных средств для проведения занятий, а также осуществлять тестирование на предмет допуска к использованию энергетического оборудования судов (НЕО ЭКСПЕРТ.ЭКЗАМЕНАТОР).

# Глава 2. Встраиваемые вычислительные системы

## 2.1. Введение

Стремительный рост потребности во встраиваемых вычислительных системах (ВсС) различного назначения заставляет разработчиков активно совершенствовать методы и средства проектирования. Встраиваемые (или встроенные) системы и сети (embedded systems & networks) можно определить как специализированные (заказные) микропроцессорные системы, непосредственно взаимодействующие с объектом контроля или управления и объединённые с ним конструктивно.

Активно растет доля ВсС со сложной внутренней организацией, которая проявляется в таких особенностях, как многопроцессорная гетерогенная архитектура, распределённый характер вычислений, широкий диапазон потенциально доступных разработчику вычислительных ресурсов.

Большинство сегодняшних ВсС составляют распределённые информационно-управляющие системы (РИУС), в которых доля технических решений, характерных для иных классов вычислительных систем (ВС), не является доминирующей. Это позволяет сделать вывод об актуальности поиска и развития всего многообразия технических решений в области ВсС (а не только ограниченного их числа в рамках ряда канонических аппаратно-программных платформ), а также методов и средств их проектирования.

**Под аппаратной платформой** понимают группу совместимых микропроцессорных систем (компьютеров), которые могут выполнять одинаковые программы и служат как основа или база для создания близких по назначению систем. Взаимозаменяемым понятием часто выступает аппаратная среда (environment) – определенная конфигурация аппаратных средств. “a platform is a set

of rules and guidelines of the hardware and software architecture, together with a suite of building blocks that fit into that architecture.”

**Платформа –** это набор правил и руководств по архитектуре аппаратного и программного обеспечения, вместе с набором блоков, которые входят в архитектуру.

Ключевой характеристикой всех встроенных систем является то, что они спроектированы для выполнения специфической задачи или функции. Программное обеспечение, спроектированное для платформы, является специфичным для всей этой функции устройства. Во многих случаях свойства компьютера системы не видимы для пользователя, т.е. только функция, обеспечиваемая системой, является его содержанием. Конечный пользователь встроенных систем обычно сам не устанавливает в них приложения (хотя возможности по замене приложения на новую версию могут иметь место). Платформы встроенных систем покрывают огромное число устройств от домашних беспроводных роутеров до сложных

навигационных и мультимедиа систем легковых автомобилей и промышленных управляющих систем роботизированной сборки таких автомобилей. Характеристики платформ образуют множество различных требований к их интерфейсам и производительности. При разработке встроенной системы наиболее вероятным является выбор между различными SoC-устройствами (системы на кристалле), включающими только те периферийные компоненты, которые необходимые конкретному приложению. На рис. 2.1 приведена характерная платформа на основе SoC.

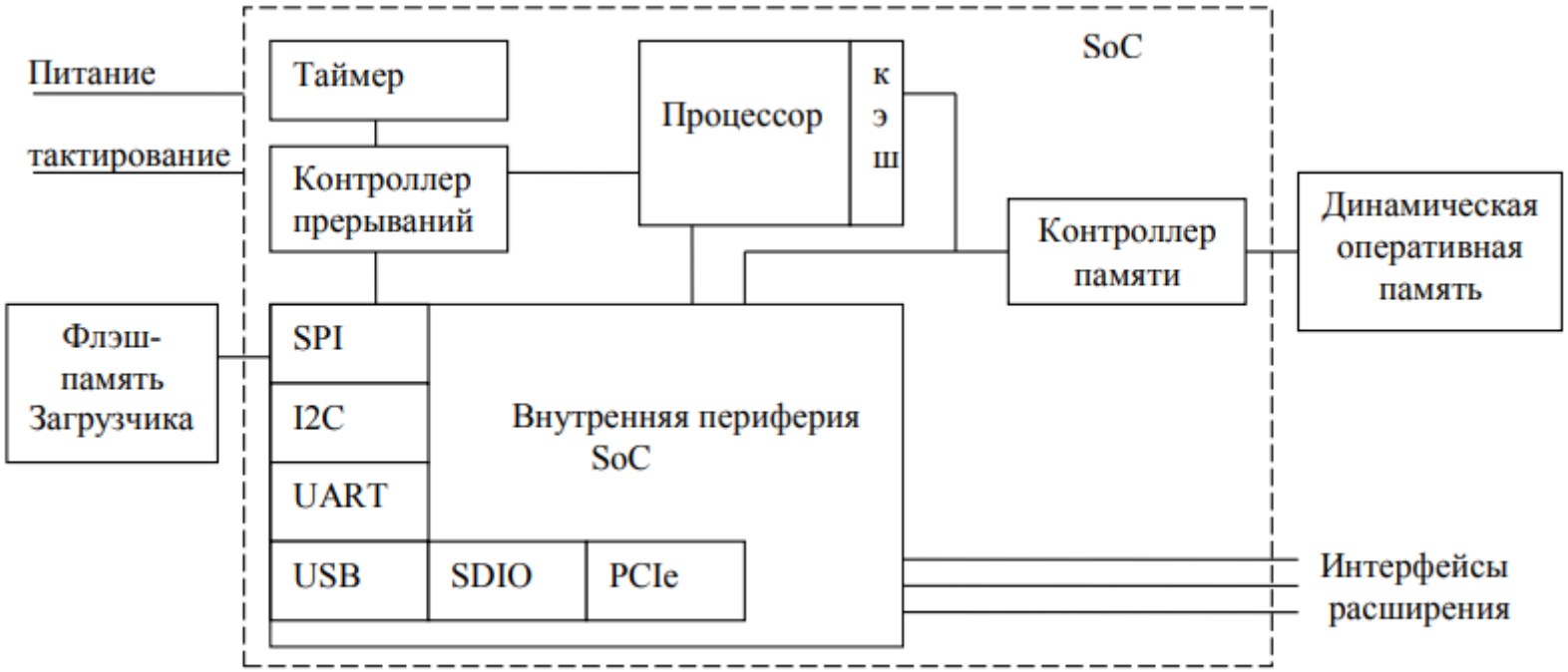


Рис. 1.1. Пример платформы на основе SoC

Рис. 2.1. Платформа на основе SoC

Процесс создания ВсС характеризуется высокой сложностью. Это определяется сочетанием таких условий проектирования, как нестандартность задачи, требование технической оптимальности решений (модель ограниченных вычислительных ресурсов), минимальные временные и финансовые бюджеты разработки,

присутствие большого числа дополнительных требований и ограничений (надежность, ограничения реального времени, тяжелые условия эксплуатации и многое другое).

Ключевой особенностью создания ВсС является необходимость комплексного проектирования, охватывающего практически все уровни организации ВС. Однако сегодня в достаточной степени формализованы и автоматизированы лишь конечные и часть средних этапов маршрутов проектирования.

Таким образом, первоочередное значение приобретает развитие методов и средств высокоуровневого (архитектурного, HLD – High Level Design) проектирования ВсС, где центральное место занимает формирование цельного взгляда на организацию всех фаз вычислительного процесса, как собственно на цель проектирования.

При разработке ВсВ особое внимание должно быть уделено архитектурному проектированию встраиваемых вычислительных систем на уровне архитектурных абстракций (вычислительных и

невычислительных), элементов архитектурного проектирования, рассмотрению проектного пространства ВсС и организации вычислительного процесса. Значительное место уделяется аспектной модели процесса проектирования ВсС, аспектной технологии сквозного проектирования ВсС на основе понятия архитектурных агрегатов.

## 2.2. Проблемы проектирования встраиваемых вычислительных систем

### 2.2.1. Понятие встраиваемой вычислительной системы

Постоянно растущая потребность в информационно-управляющих системах (ИУС) различного назначения на современном этапе заставляет разработчиков вычислительной техники активно совершенствовать способы и средства их проектирования.

Значительную долю ИУС составляют встраиваемые системы и сети (embedded systems & networks), которые по функциональному

назначению и конструктивному исполнению тесно связаны с объектом контроля или управления. Такие системы называют встраиваемыми или встроенными, мы будем рассматривать эти термины как синонимы, с сокращённым обозначением «ВсС».

Встраиваемые вычислительные системы и сети (или просто встраиваемые системы, ВсС) находят широкое применение в бытовой электронике, промышленной автоматике, на транспорте, в телекоммуникационных системах, медицинском оборудовании, в военной и аэрокосмической технике, в других областях. Сфера применения ВсС постоянно расширяется и в том или ином виде эти системы в ближайшее время проникнут во все области деятельности человека.

В общем случае ВсС являются для разработчиков вычислительной техники одним из наиболее сложных объектов проектирования. Даже поверхностный анализ типовых требований и

ограничений, которые необходимо учитывать при создании ВсС, подтверждает это. Вот некоторые примеры.

Характеристика реактивных систем реального времени:

- реагируют на состояние внешней среды;
- постоянный цикл взаимодействия со средой;
- в идеале, выполняют бесконечный целевой алгоритм;
- должны учитывать внешние временные ограничения (реальное время).

Характеристика мобильных массовых ВсС:

- сложный набор функций;
- работа в режиме реального времени;
- низкая стоимость производства;
- низкое энергопотребление;
- проектируются в сжатые сроки часто малыми рабочими группами;

- программирование в рамках модели «с учетом вычислительных ресурсов», в отличие от подхода «неограниченные ресурсы».

Особую группу составляют ВсВ, созданные по технологии «Система на кристалле»

Характеристика ВсС, выполняемых по технологии «система на кристалле» (SoC):

- сборка «готовых компонентов», зачастую приобретённых у сторонних
- производителей («интеллектуальная собственность»);
- иерархия «черных ящиков»;
- проектирование и верификация выполняются больше на системном
- уровне, чем на логическом;
- акцент на взаимодействие компонент;
- большая важность программного обеспечения.

В значительной степени сложность создания ВсС подтверждается и отсутствием в литературе чёткого определения для этого класса вычислительных систем.

Диапазон реализаций ВсС, действительно, очень велик. В него попадают и простейшие устройства уровня домашнего таймера, и сложнейшие распределенные иерархические системы, управляющие критически важными объектами. Важно что, проектируя ВсС, разработчик всегда создает *специализированную вычислительную систему* независимо от степени соотношения готовых и заново создаваемых решений. Он должен анализировать все уровни организации системы. Он имеет дело не с созданием приложения в готовой операционной среде при наличии мощных и удобных инструментальных средств, а с созданием новой специализированной ВС в условиях жёстких ограничений самого разного плана.

Тенденция усложнения ВсС проявляется, прежде всего, в том, что большинство систем реализуются в виде многопроцессорных распределённых ВС или контроллерных сетей. Это дополнительно усложняет задачу проектировщика. Рассмотрим основные свойства современных распределённых ВсС.

- Множество взаимодействующих узлов: более двух; интерес сегодня представляют системы с единицами тысяч взаимодействующих встроенных компьютеров.
- Работа в составе систем управления без участия человека. В таких системах оператор может присутствовать, он может получать информацию и частично иметь возможность воздействовать на работу системы, однако основной объем управления выполняет распределённая ВсС. Степень функциональной и пространственной децентрализации управления может меняться в широких пределах.

- Вычислительные элементы ВсС выполняют задачи, отличные от задач вычислений и коммуникаций общего назначения.
- Распределенные ВсС используются в составе больших по масштабу технических объектов (например, инженерное сооружение, объект энергоструктуры, транспортная система, летательный аппарат) или взаимодействуют с объектами естественной природы (например, комплексы мониторинга окружающей среды).
- Распределённые ВсС могут характеризоваться узлами с ограниченным энергопотреблением, иметь фиксированную или гибкую топологию, выполнять критичные для жизнедеятельности человека функции требовать высокотехнологичной реализации или создаваться как прототип.

Встраиваемые вычислительные системы «глубоко интегрированные» с объектами физического мира. Их элементы практически всегда ограничены по ресурсам. ВсС системы длительного жизненного цикла, часто автономные. Масштаб ВсС по размерам и сложности меняется в очень широких пределах. Эти системы рассчитаны часто на непрофессиональных (в вычислительной технике) пользователей. ВсС часто выполняют критически важные функции.

Вот как эти системы определяются в Оксфордском словаре по вычислительной технике:

- *Система реального времени (СРВ)*: любая система, в которой время формирования выходного воздействия является существенным.

Примеры СРВ: управление технологическими процессами, встроенные вычислительные системы, кассовые торговые системы и т.д.

- *Встроенная вычислительная система (ВсС):* любая система, которая использует компьютер как элемент, но чья основная функция не есть функция компьютера. Примеры ВсС: DVD-проигрыватель, светофорный объект, банкомат, паркомат и т.д.

Важно проследить эволюцию сегодняшнего понятия «встраиваемые системы» на протяжении времени развития вычислительной техники (рис. 2.1):

1. Информационно-управляющие системы, 60-е годы (УВК, БЦВМ).

2. Встроенные вычислительные системы (embedded systems) – ВсС (ES), конец 70-х годов.

3. Распределенные встроенные системы управления (networked embedded control systems / распределенные информационно-управляющие системы) – NECS / РИУС, конец 90-х годов.

4. Cyber Physical Systems – CPS (кибер-физические системы), примерно с 2006г.

ИУС (1) последовательно «переросли» в ВсС (2) по мере пространственного объединения ВС с объектом контроля / управления благодаря, в первую очередь, технологическим достижениям в электронике.

На сегодняшний день подавляющее большинство ВсС по-факту являются распределёнными (3). Представление ВсС как распределённых или включение в эту категорию, так называемых, контроллерных сетей или распределённых информационно-управляющих систем для многих отечественных специалистов является по-прежнему спорным. В зарубежной научно-технической литературе, напротив, достаточно чётко декларируется, что ВсС - это не только малогабаритные или моноблочные, одномодульные устройства, но и пространственно и/или архитектурно распределённые системы, которые непосредственно сопряжены с

объектами большого масштаба, пространственно распределёнными и т.д. Нецелесообразность выделения РИУС в отдельный класс ВС диктуется едиными условиями, принципами и инструментами их проектирования с «традиционными» ВсС.

Ожидающийся в ближайшем будущем качественный переход в восприятии ВсС и методах их проектирования демонстрирует понятие «кибер-физические системы» (4). Это могут быть электронно-механические системы, гидравлические, биологические и т.д. Суть последнего определения состоит в том, что проектирование объекта управления и системы управления для этого объекта должно выполняться в едином ключе, в едином комплексе, в рамках определённых или, как минимум, очень тесно взаимодействующих инструментальных средств. На это нацелен тезис (4), в соответствии с которым в мире реально развёрнуты работы по созданию технологий и инструментов для

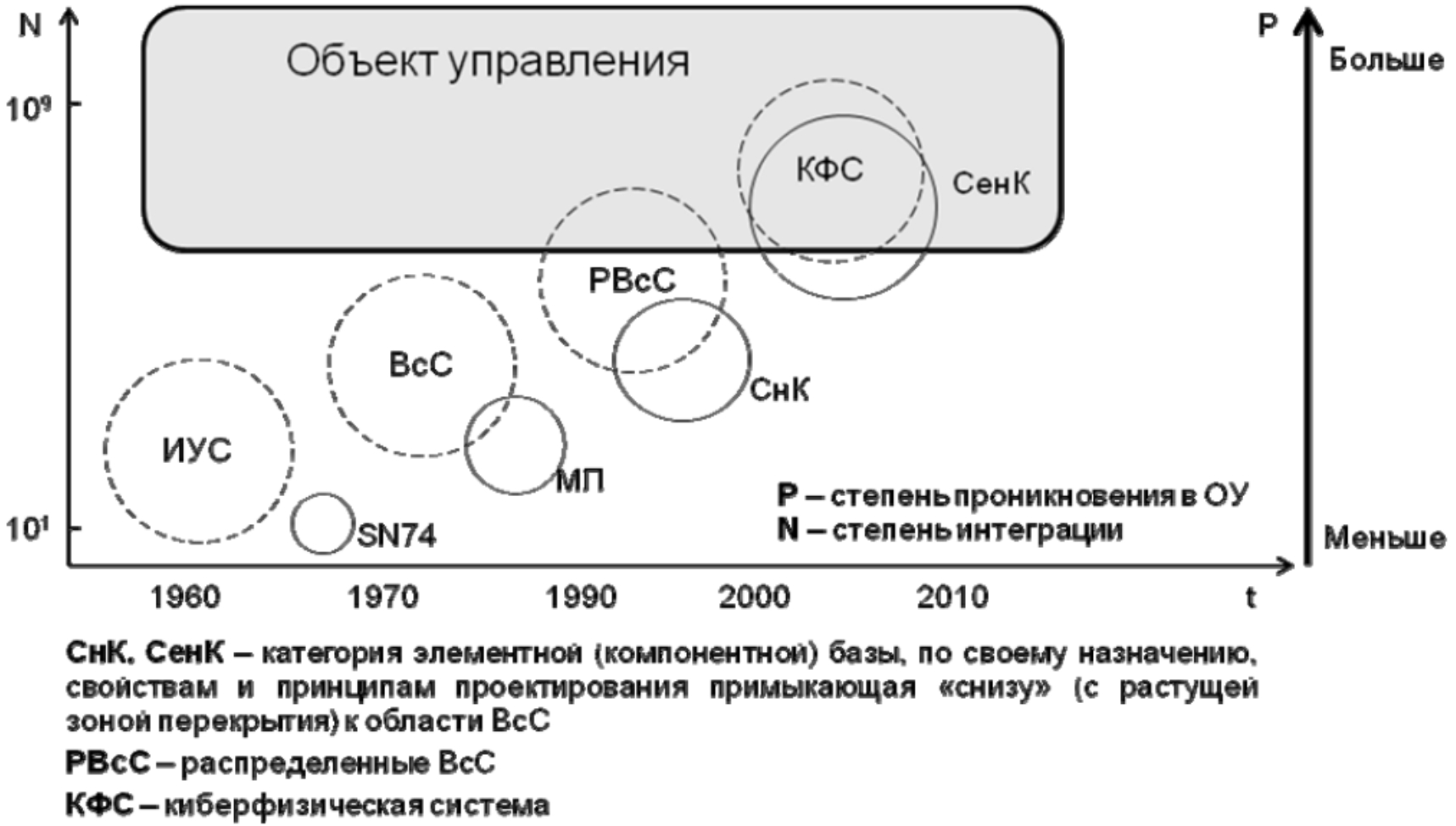
автоматизированного проектирования гетерогенных технических систем.

На рисунке (Рис. 2.2) приведён пример типичной структуры CPS. Она состоит из объекта управления, «физической» части — той части, которая не реализуется вычислительной техникой или цифровыми сетями, и может включать механику, биологические или химические процессы, даже

людей-операторов. Управление этой частью CPS осуществляют одна или несколько платформ, которые состоят из датчиков, исполнительных механизмов, одного или нескольких процессоров и, возможно, одной или нескольких операционных систем. Сетевая структура обеспечивает механизмы взаимосвязи платформ, образуя вместе с ними «кибернетическую» часть CPS.

Область проектирования ВсС на сегодняшний день тесно смыкается с областью проектирования СБИС. В своём развитии электронная компонентная база, достигнув уровня цифровых

***систем и сетей на кристалле*** (СнК и СенК), по своим основным характеристикам приблизилась к отдельным классам ВсС, что позволяет рассматривать единые методы проектирования как непосредственно для ВсС так и для систем и сетей на кристалле (рис. 2.1).




**СнК, СенК** – категория элементной (компонентной) базы, по своему назначению, свойствам и принципам проектирования примыкающая «снизу» (с растущей зоной перекрытия) к области ВсС

**РВсС** – распределенные ВсС

**КФС** – киберфизическая система

Рис. 2.1. Эволюция понятия «Встраиваемые вычислительные системы»

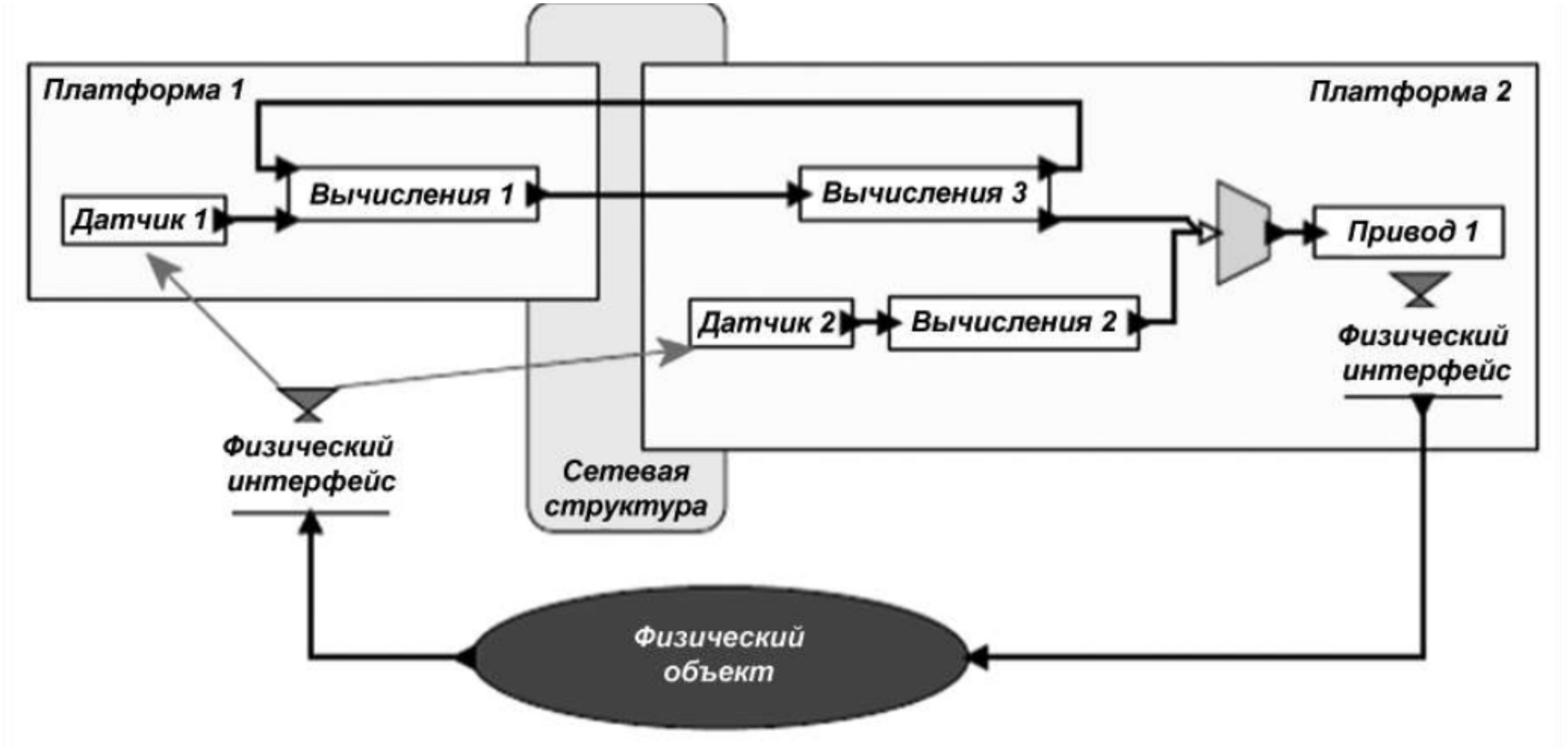



Рис. 2.2. Пример структуры кибер - физической системы

Суммируя приведенные выше трактовки ВсС и учитывая широкий круг возможных архитектурных решений для таких систем, определим ВсС как специализированные (заказные) вычислительные

системы (ВС), непосредственно взаимодействующие с объектом контроля или управления [и объединенные с ним единой конструкцией].

Это позволяет:

- Объединить большое число категорий вычислительных систем по ключевому общему признаку.
- Устранить проблему влияния вторичных относительно «вычислительной сути» факторов (размеры, конструкция, топология, конкретное целевое назначение и др.).
- Обеспечить свободный выбор реализации (ранее «выпадали» многие возможные архитектурные решения).
- Унифицировать «нишевые» методики проектирования.
- Развивать новые «активности», в первую очередь, в высокоуровневом проектировании (например, применять платформно-ориентированное проектирование, аспектное проектирование и др.).

В дальнейшем в учебном пособии ВсС будут рассматриваться с позиции данного определения.

### 2.2.2. Примеры реализации ВсС


Рис. 2.3. Сотовые телефоны

**Рис. 1.1.** Роботы, такие как марсоход, являются встроенными системами. Фотография с разрешения NASA/JPL-CALTECH

Рис. 2.4. Робот марсоход


Рис. 2.5. Примеры гражданских применений ВсС

Рис. 2.5. Электрическая зубная щетка

# Применение BcC в военных системах


Рис. 2.7. Авионика для военно-воздушных сил


Рис. 2.8. Управление беспилотными летательными аппаратами


Рис. 2.9. Виртуальные тренировочные платформы


Рис. 2.10. Игровая техника для обучения

| Таблица 1.1. Примеры встроенных систем | |
|---|---|
| Авиационные & Военные системы | Автопилоты самолетов, авионика и навигационные системы, системы автоматической посадки, системы наведения, управление двигателем. |

| | |
|---|---|
| Биомедицинские системы | Системы компьютерной томографии и ультразвукового исследования, мониторинг пациентов, кардиостимуляторы. |
| Автомобили | Управление двигателем, антиблокировочные тормозные системы, противобуксовочная тормозная система, управление подушками безопасности, управление системой обогрева и кондиционирования воздуха, навигация GPS, спутниковое радио, системная диагностика. |
| Коммуникация | Коммуникационные спутники, сетевые маршрутизаторы, коммутаторы, концентраторы. |
| Потребительская электроника | Телевизоры, духовки, посудомоечные машины, плееры DVD, стереосистемы, системы безопасности, управление поливом газонов, термостаты, фотокамеры, радиочасы, автоответчики, декодеры кабельного телевидения, |

| | |
|---|---|
| | другие устройства. |
| Устройства в/для компьютера | Клавиатуры, мыши, принтеры, сканеры, дисплеи, модемы, устройства жестких дисков, устройства DVD, графические платы, устройства USB. |
| Электронные инструменты | Системы сбора данных, осциллографы, вольтметры, генераторы сигналов, логические анализаторы. |
| Промышленное оборудование | Управление лифтами, системы наблюдения, роботы, станки с ЧПУ, программируемые логические контроллеры, промышленные системы автоматизации и управления. |
| Офисные машины | Факс-аппараты, копиры, телефоны, калькуляторы, кассовые аппараты. |
| Персональные устройства | Сотовые телефоны, переносные плееры MP3, видео-плееры, персональные цифровые |

| | |
|---|---|
| | помощники (PDA), электронные наручные часы, портативные видеоигры, цифровые камеры, системы GPS. |
| Роботы | *Промышленные роботы*, автономные транспортные средства, космические исследовательские роботы (например, роботы-*марсоходы*) |
| Игрушки | Системы видеоигр, игрушки роботы типа "Aibo", "Furby", и "Elmo". |

## 2.3. Киберфизические системы

### 2.3.1. Определение киберфизической системы

Киберфизические системы (Cyber-Physical System, CPS) — это системы, состоящие из различных природных объектов, искусственных подсистем и управляющих контроллеров, позволяющих представить такое образование как единое целое. В CPS обеспечивается тесная связь и координация между вычислительными и физическими ресурсами. Компьютеры осуществляют мониторинг и управление физическими процессами с использованием такой петли обратной связи, где происходящее в физических системах оказывает влияние на вычисления и наоборот.

Сложность такого рода задач приводит к мысли о том, что речь не идёт о создании автоматизированных систем, более крупных, чем существующие, где компьютеры интегрированы или встроены в те или иные физические устройства или системы. Речь о гармоничном сосуществовании двух типов моделей. С одной стороны - это

традиционные инженерные модели (механические, строительные, электрические, биологические, химические, экономические и другие), а с другой - модели компьютерные.

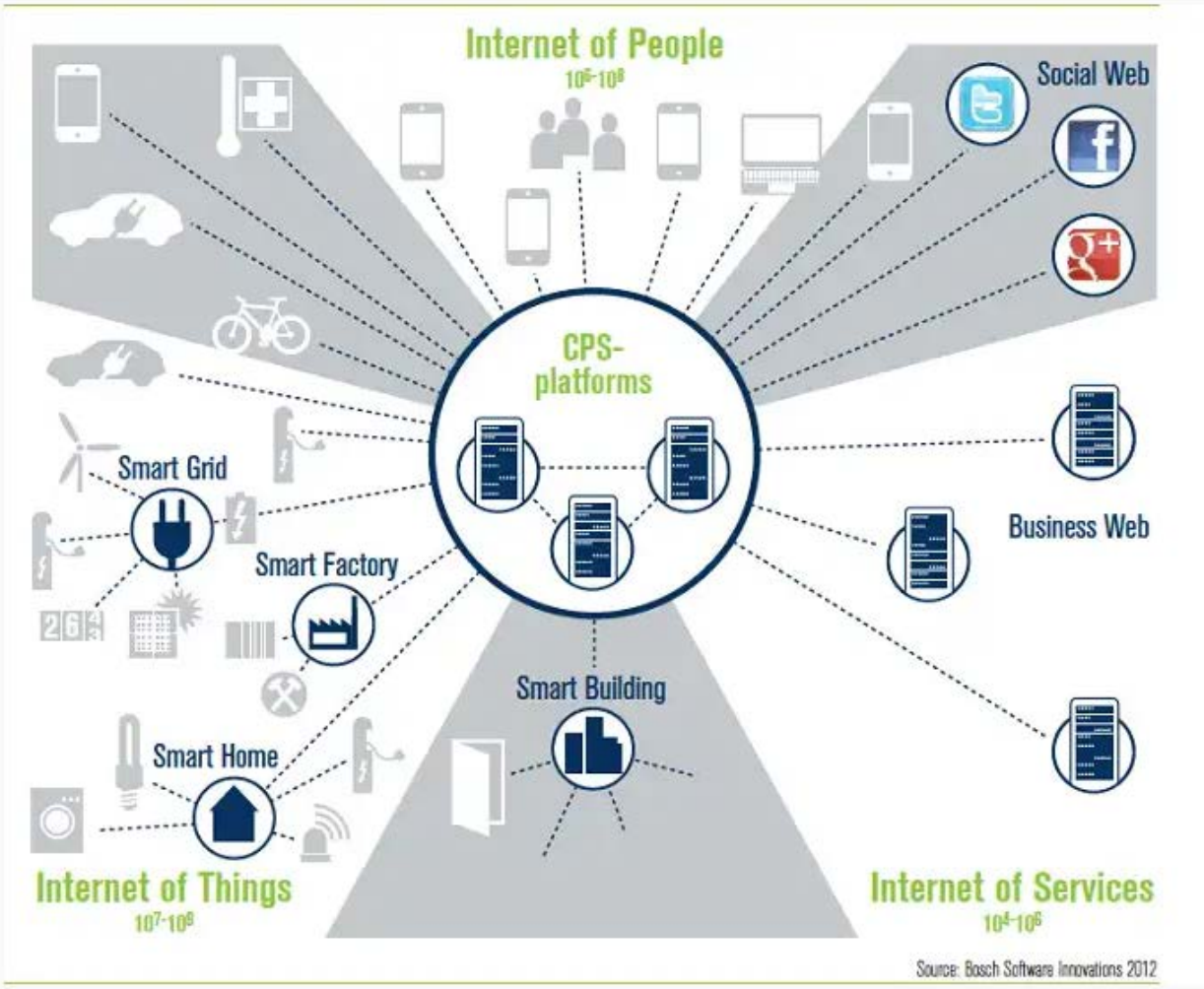
**Предшественниками CPS можно считать встроенные системы реального времени,** распределённые вычислительные системы, автоматизированные системы управления техническими процессами и объектами, беспроводные сенсорные сети.

С технической точки зрения CPS имеют много общего со структурами типа грид, реализуемыми посредством интернета вещей (Internet of Things, IoT), Индустрии 4.0, промышленного интернета вещей (Industrial Internet), межмашинного взаимодействия (Machine-to-Machine, M2M), туманного и облачного компьютинга (fog и cloud computing). Но этими техническими средствами ни в коем случае нельзя ограничивать представление CPS. Для этих сложных систем требуются новые кибернетические подходы к моделированию,

поскольку именно модели являются центральным моментом в науке и инженерии.

Немецкая академия Acatech уже говорит о перспективах национальных киберфизических платформ, которые складываются из трёх типов сетей:

- Интернет людей
- Интернет вещей
- Интернет сервисов




Киберфизические системы возникают на стыке интернета людей, вещей и сервисов

Рис. 2.11. Применение киберфизических систем

По мнению немецких академиков, перспективы появления киберфизических систем и формирования на их основе Индустрии 4.0 затрагивают интересы общества в целом, поэтому должны рассматриваться не только в техническом, а в более широком социокультурном аспекте, с учетом демографических и других происходящих изменений.

Область применения CPS распространяется практически на все виды человеческой деятельности, включая все многообразие промышленных систем, транспортные, энергетические и военные системы, все виды систем жизнеобеспечения от медицины до умных домов и городов, а также многие экономические системы.

Создание полноценных систем CPS в перспективе приведёт примерно к таким же изменениям во взаимодействии с физическим миром, как те, к которым привела в своё время Всемирная сеть.

### 2.3.2. Предпосылки появления киберфизических систем

Можно говорить о нескольких основных технических предпосылках, сделавших CPS возможными:

**Первая —** рост числа устройств со встроенными процессорами и средствами хранения данных:

сенсорные сети, работающие во всех протяжённых технических инфраструктурах;

медицинское оборудование;

умные дома и т.д.

**Вторая —** интеграция, позволяющая достигнуть наибольшего эффекта путем объединения отдельных компонентов в большие системы:

- Интернет вещей IoT),
- World Wide Sensor Net,
- умные среды обитания (Smart Building Environment),
- оборонные системы будущего.

**Третья** — ограничение когнитивных способностей человека, которые эволюционируют медленнее, чем машины. В этой связи непременно наступает момент, когда люди уже не в состоянии справиться с объёмом информации, требуемой для принятия решений, и какую-то часть действий нужно передать CPS, выведя человека из контура управления (human out of loop).

В то же время в ряде случаев CPS могут усилить аналитические способности человека, поэтому есть потребность в создании интерактивных систем нового уровня, сохраняющих человека в контуре управления (human in the loop).

Киберфизическая система (англ. cyber-physical system) — информационно-технологическая концепция, подразумевающая интеграцию вычислительных ресурсов в физические процессы. В такой системе датчики, оборудование и информационные системы соединены на протяжении всей цепочки создания стоимости, выходящей за рамки одного предприятия или бизнеса. Эти системы

взаимодействуют друг с другом с помощью стандартных интернет-протоколов для прогнозирования, самонастройки и адаптации к изменениям.

### 2.3.3. Мобильные киберфизические системы

Мобильные киберфизические системы, в которых у физической рассматриваемой системы есть врожденная подвижность, являются видной подкатегорией киберфизических систем. Примеры мобильных физических систем включают мобильную робототехнику и электронику, транспортируемую людьми или животными. Повышение популярности смартфонов увеличило интерес к области мобильных киберфизических систем.

Платформы смартфона делают идеальные мобильные киберфизические системы по ряду причин, включая:

Значительные вычислительные ресурсы, такие как обработка способности, местное хранение.

Многократные сенсорные устройства ввода-вывода, такие как сенсорные экраны, камеры, GPS, громкоговорители, микрофон, светочувствительные датчики, датчики близости.

Коммуникационные механизмы, такие как WiFi, 3G, Bluetooth для подключения устройств или к Интернету, или к другим устройствам

Языки программирования высокого уровня, которые позволяют быстрое развитие мобильного программного обеспечения узла CPS, (Java, C, C++,C#).

Легко доступные прикладные механизмы распределения, такие как Android Market и App Store Apple.

Для задач, которые требуют, больше ресурсов, чем доступно в местном масштабе, один общий механизм для быстрого внедрения основанных на смартфоне мобильных киберфизических системных узлов использует сетевое соединение, чтобы связать мобильную систему или с сервером или с облачной средой, позволяя решать

сложные задачи обработки, которые невозможны при местных ограничениях ресурса.

### 2.3.4. Примеры CPS

В киберфизических системах вычислительные элементы взаимодействуют с датчиками, которые обеспечивают мониторинг киберфизических показателей, и с исполнительными элементами, которые вносят изменения в киберфизическую среду. Зачастую киберфизические системы ориентированы на то, чтобы каким-либо образом управлять окружающей средой. Киберфизические системы объединяют информацию от интеллектуальных датчиков, распределенных в физической среде, для лучшего понимания среды и выполнения более точных действий.

В физическом контексте исполнительные элементы на основе получаемых данных вносят изменения в среду обитания пользователей. В виртуальном контексте киберфизические системы применяются для сбора данных о виртуальных действиях

пользователей, таких как использование социальных сетей, блогов и сайтов электронной коммерции. Затем киберфизические системы определенным образом реагируют на такие данные, прогнозируя действия или потребности пользователей в целом. Используя такие программные продукты, как IBM WebSphere Sensor Events, можно анализировать данные и события, поступающие от датчиков в реальном времени, и встраивать их в интеллектуальные решения.

Приведем несколько примеров практического применения киберфизических систем:

**В производственной среде**: киберфизические системы могут улучшить производственные процессы, обеспечивая обмен информацией реального времени между промышленным оборудованием, производственной цепочкой поставок, поставщиками, системами управления бизнесом и клиентами. Кроме того, киберфизические системы могут повышать эффективность этих процессов благодаря автоматическому мониторингу и контролю

всего производственного процесса и адаптации производства для удовлетворения предпочтений клиентов. Киберфизические системы повышают прозрачность и управляемость цепочек поставок, улучшая отслеживаемость и безопасность товаров.

**В здравоохранении**: киберфизические системы используются для дистанционного мониторинга физических показателей пациентов в реальном времени с целью уменьшения потребностей в госпитализации (например, пациентов с болезнью Альцгеймера) или для улучшения ухода за инвалидами и пожилыми людьми. Кроме того, киберфизические системы применяются в нейробиологических исследованиях для изучения функций организма человека с использованием интерфейсов между мозгом и оборудованием и терапевтической робототехники.

**В возобновляемой энергетике**: интеллектуальные энергосети представляют собой киберфизические системы, в которых датчики и

другие устройства обеспечивают мониторинг сети для целей контроля, повышения надежности и энергоэффективности.

**В интеллектуальных зданиях**: совместная работа интеллектуальных устройств и киберфизических систем позволяет сократить энергопотребление, повысить безопасность и защищенность, а также создать более комфортные условия для жителей. Например, киберфизические системы могут поддерживать мониторинг энергопотребления и использование систем регулирования для реализации концепции дома с нулевым потреблением электроэнергии. Кроме того, их можно использовать для определения степени ущерба для зданий в результате непредвиденных событий и предотвращения разрушения конструкций.

**На транспорте**: транспортные средства и инфраструктура могут взаимодействовать между собой, обмениваясь в реальном времени информацией о дорожном движении, местоположении и проблемах,

предотвращая транспортные инциденты и дорожные пробки, повышая безопасность и в конечном итоге экономя время и деньги.

**В сельском хозяйстве**: киберфизические системы могут использоваться для создания более современного и эффективного сельского хозяйства. Они могут собирать важную информацию о климате, почве и другие данные для более точного управления сельскохозяйственными работами. Датчики киберфизических систем могут вести постоянный мониторинг различных показателей, таких как орошение почвы, влажность воздуха и здоровье растений, для поддержания оптимальных окружающих условий.

**В вычислительных средах**: киберфизические системы позволяют лучше понимать поведение систем и пользователей для повышения производительности и более эффективного управления ресурсами. Например, можно оптимизировать работу приложений с учетом контекста и действий пользователей или отслеживать доступность ресурсов. Кроме того, популярные социальные сети и

сайты электронной коммерции хранят информацию о действиях пользователей и затребованном контенте, анализируют эту информацию, чтобы предсказывать, что может быть интересно пользователям, и предлагать рекомендации в отношении друзей, публикаций, ссылок, страниц, событий или продуктов.

**Разумные города** можно рассматривать как масштабные киберфизические системы — с датчиками, которые отслеживают вычислительные и физические показатели, и исполнительными элементами, которые определенным образом меняют сложную городскую среду. Правительства, организации и технологические отрасли заняты решением задач, порождаемых растущим уровнем урбанизации, в целях улучшения городской жизни, например, путем повышения эффективности энергоснабжения и качества услуг.

## 2.3.5. Проблемы в развитии CPS

С технической точки зрения предстоит еще решить множество сложных проблем — как минимум эффективным и применимым в реальных условиях способом. Вот некоторые из таких проблем:

**Разнородность данных.** Разнородность данных — это серьезная проблема, которая может негативно влиять на эффективность взаимодействий и разработку коммуникационных протоколов. Системы должны быть способны поддерживать большое количество различных приложений и устройств.

**Надежность.** Киберфизические системы можно использовать в таких критически важных областях, как здравоохранение, инфраструктура, транспорт и многие другие. Основными требованиями являются надежность и безопасность, поскольку исполнительные элементы оказывают влияние на окружающую среду. Фактически влияние исполнительных элементов может быть необратимым, поэтому вероятность их непредвиденного поведения

должна быть сведена к минимуму. Кроме того, окружающая среда непредсказуема, поэтому киберфизические системы должны быть способны продолжать работу в непредвиденных обстоятельствах и адаптироваться в случае сбоев.

**Управление данными.** Необходимо хранить и анализировать большие данные, поступающие от различных сетевых устройств, обрабатывать их и в реальном времени выводить результаты. Данными можно управлять с использованием отложенной или оперативной потоковой обработки, в зависимости от назначения системы. При использовании потоков в реальном времени информация может часто меняться и обработка основывается на адаптивных и постоянных запросах.

**Конфиденциальность.** Проблема заключается в поддержании баланса между сохранением конфиденциальности и защитой персональных данных — и доступностью данных для предоставления более качественного обслуживания. Поскольку

киберфизические системы управляют значительными объемами данных, включающих такую конфиденциальную информацию, как здоровье, пол, вероисповедание и множество других персональных сведений, возникают серьезные проблемы конфиденциальности данных. Для киберфизических систем необходимы политики обеспечения конфиденциальности, поэтому нужен инструмент обезличивания данных, позволяющий удалять персональную информацию перед обработкой данных системой.

**Безопасность.** Киберфизические системы должны обеспечивать безопасность коммуникаций, поскольку все действия координируются между устройствами в реальном времени. Киберфизические системы расширяют масштаб и объем взаимодействия между физическими и вычислительными системами, что усложняет задачи обеспечения безопасности. Для решения этой проблемы недостаточно традиционных инфраструктур обеспечения безопасности, и нужно искать новые решения.

Необходимо защищать как поступающие данные, так и хранимые данные, собранные для использования в будущем. И наконец, киберфизические системы основываются на разнородных приложениях и беспроводных коммуникациях, что зачастую усложняет обеспечение безопасности.

**Реальное время.** Киберфизические системы управляют значительными объемами данных, получаемых от датчиков. Вычислительная обработка должна быть эффективной и своевременной, поскольку физические процессы продолжаются независимо от результатов вычислений. Для удовлетворения этого требования киберфизические системы должны обладать пропускной способностью или мощностью, необходимой для поддержки немедленной обработки, поскольку невыполнение своевременных действий может привести к долгосрочному ущербу.

Перед нами стоит задача освоения результатов технологической эволюции, которую Интернет вещей (и в том числе киберфизические

системы) привносит в нашу повседневную жизнь. Эти технологии будут повышать качество обслуживания и в конечном итоге работать на благо окружающей среды по мере появления разумных городов по всему миру.

Киберфизические системы, являющиеся движущей силой инноваций, охватывают множество различных дисциплин. Сотрудничество различных отраслей может сделать их важной производственной силой. Кроме того, для киберфизических систем нужны высококвалифицированные кадры, поэтому необходимы сотрудничество и взаимодействие отраслей и университетов. И наконец, киберфизические системы имеют огромный потенциал для изменения и совершенствования каждого аспекта жизни людей, помогая решать критически важные для нашего общества проблемы и превосходя современные распределенные системы в плане безопасности, производительности, эффективности, надежности, удобства использования и по многим другим показателям.

## 2.4. Рынок специалистов в области встроенных систем

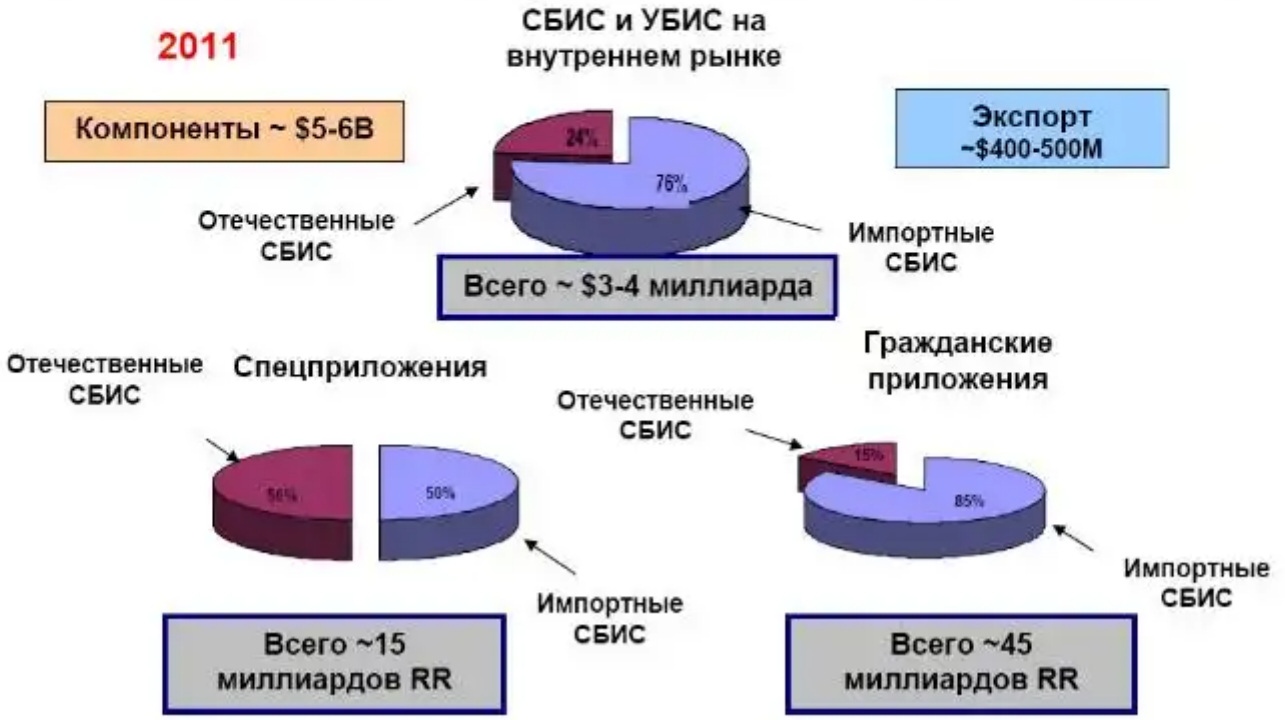

### Рынок специалистов в области встроенных систем

- Потребители - **десятки государственных и частных организаций и фирм** только в СПб
  **Распространенные заблуждения:**
  **нужен программист, "сетевик", "интегратор"**
- Требуемый профиль специалиста:
  **- "системщик" - интегральный специалист в области ИТ очень высокой квалификации**
  - большое число узких специалистов по разделам (ПЛИС, RTL, встроенное ПО, SCADA, ПЛК …)

- Движение по возрождению и развитию электроники СБИС и УБИС (центры наноэлектроники)

# Консервативный прогноз российского рынка электроники

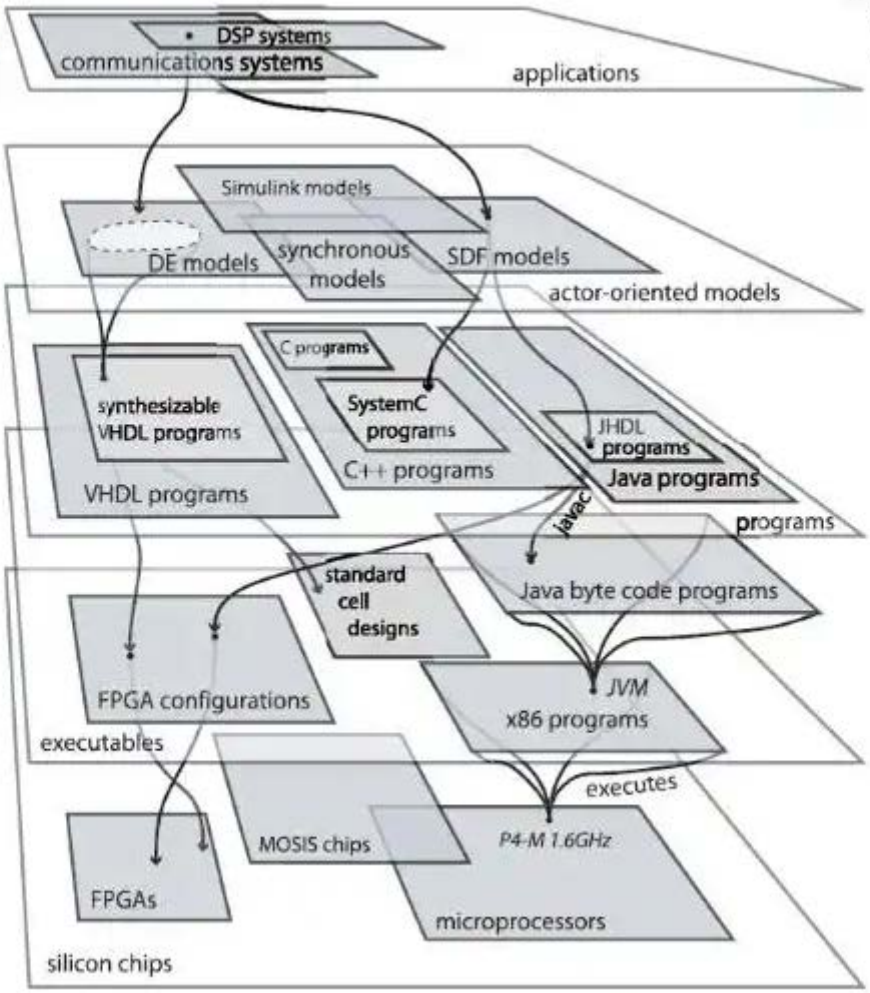
*Экстраполяция структуры российского рынка компонентов и СБИС на 2011 год*



# Внутренний рынок: достаточно пространства для роста

- Коммуникации, промышленная и потребительская электроника → 30%
- Военная электроника → 25%
- Цифровое ТВ и связное оборудование → 15%
- Автомобильная электроника → 15%
- СМАРТ-карты и электронные удостоверения → 9%
- Глобальная система спутниковой навигации ГЛОНАСС → 6%
- Другие приложения: структура рынка быстро меняется
- **Устойчивый рост потребностей рынка:**
  - многомиллиардные национальные проекты в здравохранении, образовании, строительстве и т.п. требуют огромное количество электронных устройств, предпочтительно, местных производителей

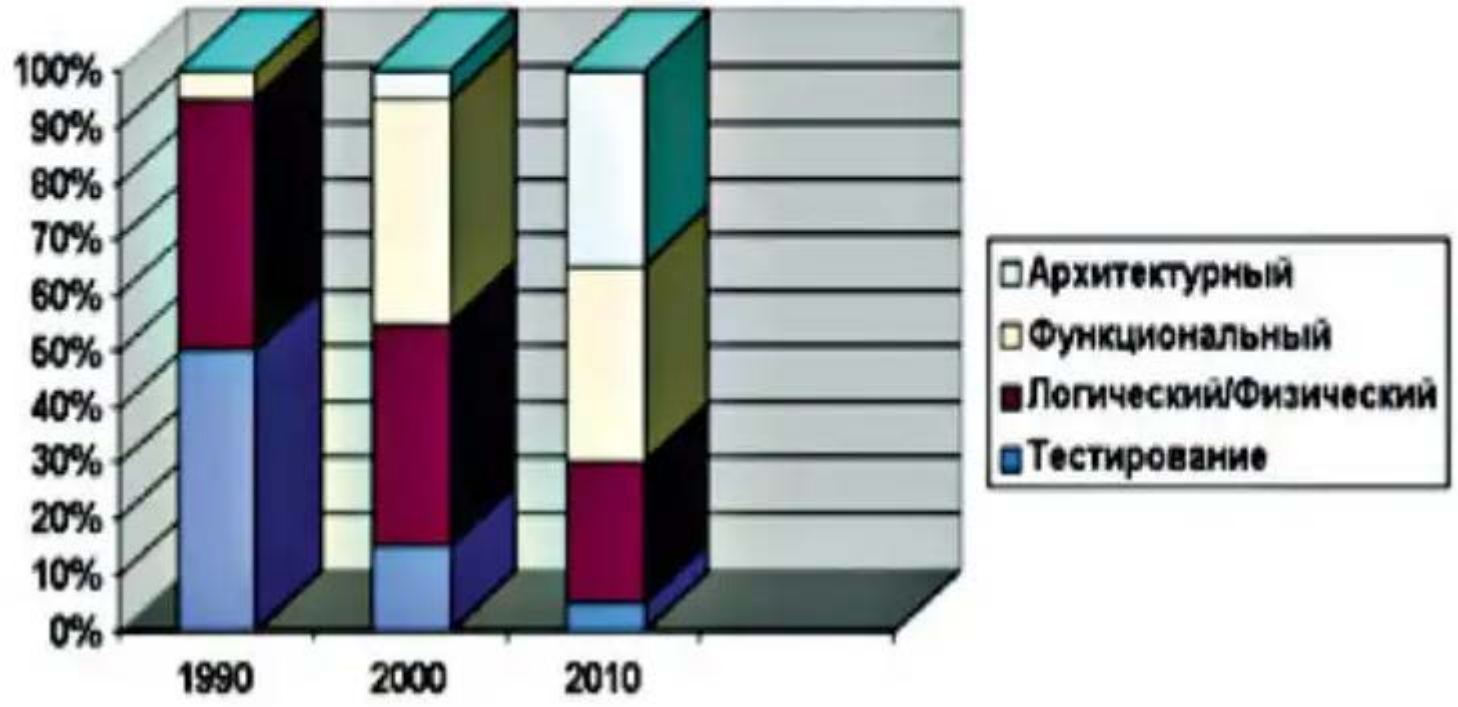
# Платформы проектирования по Edward Lee



# "Центр тяжести" в проектировании встроенных систем

- "Проектирование встроенных вычислительных систем требует знаний в различных прикладных областях – от телекоммуникаций, системотехники и программирования до физики полупроводниковых процессов.
- **Системная составляющая проектирования приобретает первостепенное значение**, поскольку именно на этом этапе определяются все базовые характеристики будущей системы, а цена принятых решений самая высокая.
- Так, если **в 1990 году реализация проекта** (начиная с логического уровня) **занимала 90%** всего объема проектных работ, то к **2010 году проектирование на архитектурном и функциональном уровнях будет составлять 70%** в общем объеме работ и только 30% придется на реализацию

# Трудоемкость проектирования на различных уровнях абстракции при создании встроенных систем




# Идея направления (чему и как учить)

Производители заказных и полузаказных УБИС

Производители СнК и встроенных систем

Производители лицензируемых IP-компонент

Производители библиотек цифровых и аналоговых элементов и IP-элементов

## Подготовка "системообразующего специалиста"

- в области технологий **высокоуровневого проектирования** встроенных систем и комплексов,
- а также в области технологий проектирования интегральных вычислительных компонент встраиваемых систем

**Системообразующий специалист** – это ***сегодня*** специалист, проектирующий и реализующий целевую вычислительную систему, обеспечивающую функционирование разрабатываемого объекта.

# Глава 3. Организация встраиваемых систем

## 3.1. Программное и аппаратное обеспечение

Как правило, ВсС представляют собой полностью самодостаточные устройства. Это цельные ВС, функционирующие практически независимо и обладающие возможностью влиять на своё окружение тем или иным образом (или хотя бы реагирующие на события внешнего мира изменениями собственного внутреннего состояния). В подавляющем большинстве случаев такие системы реализуются как электронные компоненты, смонтированные на печатных платах и связанные при помощи проводящих дорожек определённым образом, с некоторыми интерфейсами для связи с объектами управления и внешним миром.

Иногда ВсС выполняются в корпусах и зачастую имеют собственные автономные источники питания.

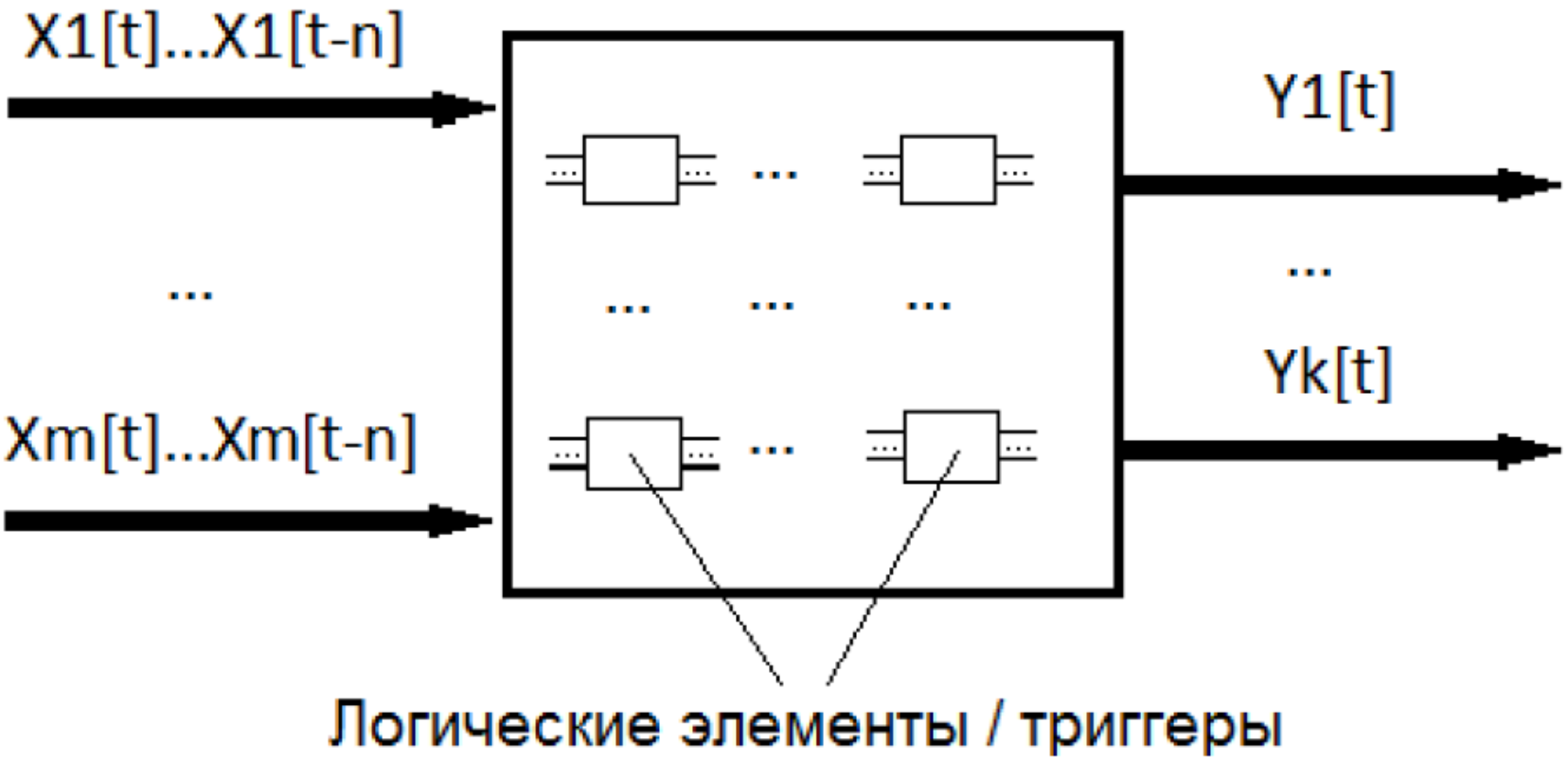



*Рис. 11. Схематичное представление некоторой электронной схемы.*

Рис. 3.1. Схематическое представление электронной схемы

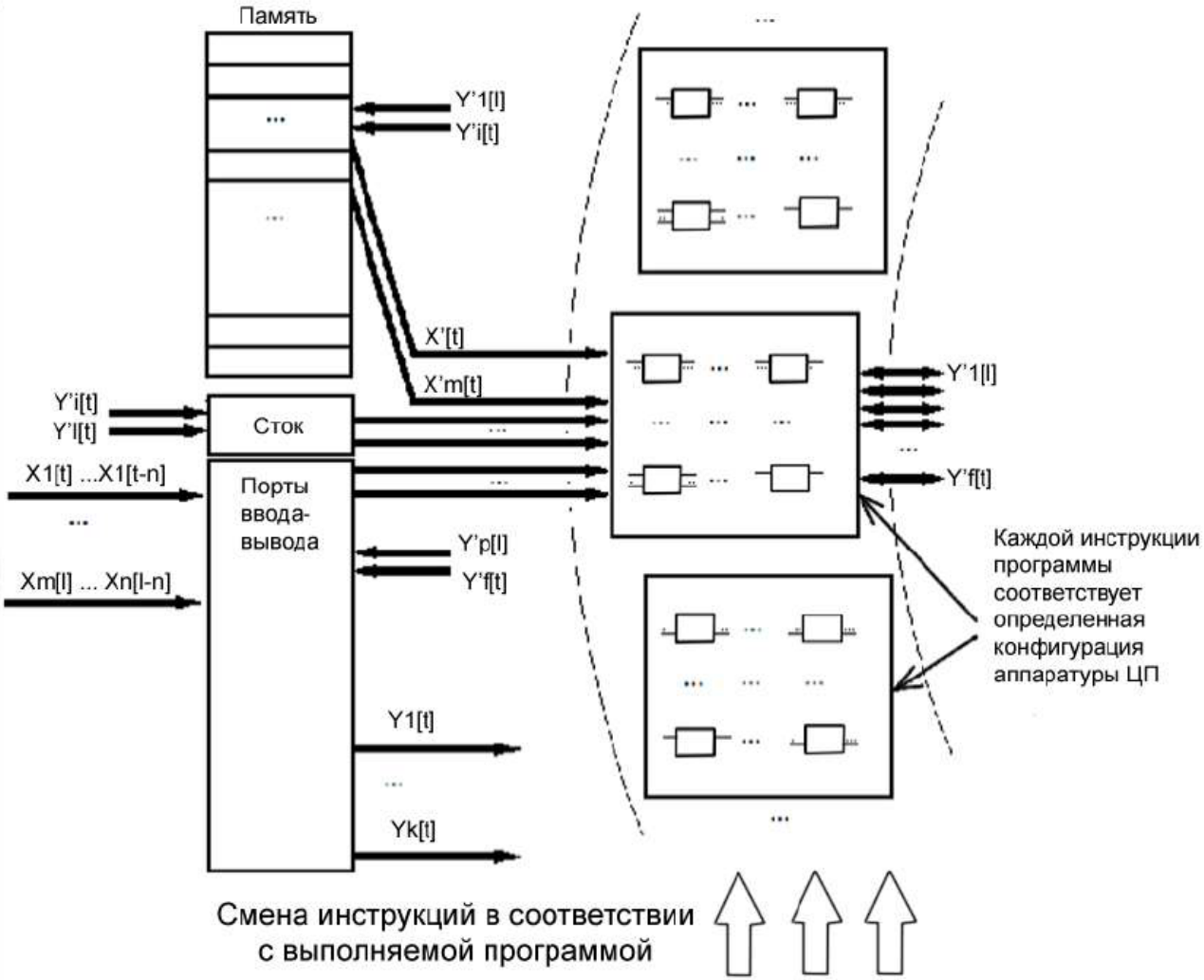

Память
Y'1[l]
Y'i[t]
X'[t]
X'm[t]
Y'i[t]
Y'l[t]
Сток
X1[t] ...X1[t-n]
Порты ввода-вывода
Y'p[l]
Y'f[t]
Xm[l] ... Xn[l-n]
Y1[t]
Yk[t]
Y'1[l]
Y'f[t]
Каждой инструкции программы соответствует определенная конфигурация аппаратуры ЦП
Смена инструкций в соответствии с выполняемой программой

Рис. 3.2.

Более распространёнными в современной вычислительной технике являются ВС с использованием фон-Неймановской архитектуры.

Простейшие ВС, функционирующие на этих принципах, можно представить, как небольшую фиксированную универсальную схему, которая способна выполнять несколько простейших действий (операции сложения, записи в память и чтения из памяти, логические операции и т.д.). Выбор конкретного действия из всех возможных определяется поданной на вход управляющей последовательностью. Значения управляющей последовательности в заданном порядке выгружаются из памяти, оттуда же берутся также состояния входов схемы и туда записываются результаты выполнения действий с выходов схемы.

Координация вычислений и всего функционирования схемы производится управляющей логикой, контроллером.

Набор управляющих последовательностей, изменяющих состояние схемы, называется программой.

Используют понятия аппаратного и программного обеспечения. В широком смысле, к аппаратному обеспечению (hardware) относят любые ВС — и различные разновидности фон-Неймановских машин, и аппаратные блоки, реализующие потоковую обработку данных, и возможные промежуточные и гибридные варианты, которые, на самом деле, являются комбинациями этих двух.

Программное обеспечение (software) относится уже только к программируемым компьютерам. Под понятие ПО подпадают именно последовательно выполняемые инструкции (а заодно, и вообще все те данные, которые, загруженные в программируемый компьютер, определяют его функционирование — например, таблицы рассчитанных заранее значений). К ПО относят также документацию, позволяющую его использовать.

Как правило, в современных разработках ПО для ВсС пишется на языках высокого уровня (ЯВУ), в основном на С и С++.

В процессе компиляции ПО из понятной человеку формы на ЯВУ транслируется в форму, понятную компьютеру, т.е. в набор кодов инструкций, своим выполнением позволяющих реализовывать на данной аппаратуре заданный программистом алгоритм.

Часто в качестве вычислительных ядер ВсС, берущих на себя основную нагрузку по обработке информации и управлению, используются так называемые микроконтроллеры — относительно маломощные и достаточно стандартные процессоры с некоторым набором периферийных устройств и встроенной памятью, выполненные в виде одной микросхемы и способные решать некоторые простые задачи. Для многих понятие ВсС является синонимом вычислительной системы именно на этой базе, хотя современные технологии позволяют создавать более эффективные по многим параметрам решения.

Существует некоторое число различных вариантов реализаций функциональности проекта, которые могут быть использованы для любой части некоторого алгоритма. Каждая функция системы может быть выполнена как:

1) программа для виртуальной машины на произвольном процессоре;

2) программа на процессоре общего назначения (CPU);

3) программа на процессоре с расширенным набором инструкцийдля конкретного алгоритма (ASIP);

4) программа на цифровом сигнальном процессоре (DSP);

5) микрокод на специализированном процессоре с очень длинными командами (VLIW);

6) конфигурируемая аппаратура на программируемых логических схемах (FPGA);

7) алгоритм, полностью реализованный в аппаратуре (ASIC).

Каждый из данных подходов к реализации имеет свои характеристики энергопотребления, производительности и гибкости. При разделении функциональности проектируемой системы на части, которые используют эти целевые платформы, или при отображении всей задачи на какую-то выбранную конкретную платформу — ищется баланс между производительностью и энергопотреблением с одной стороны, и возможностью изменять поведение при помощи программирования с другой. Немаловажную роль также играют такие факторы, как стоимость изготовления, трудоёмкость проектирования и другие.

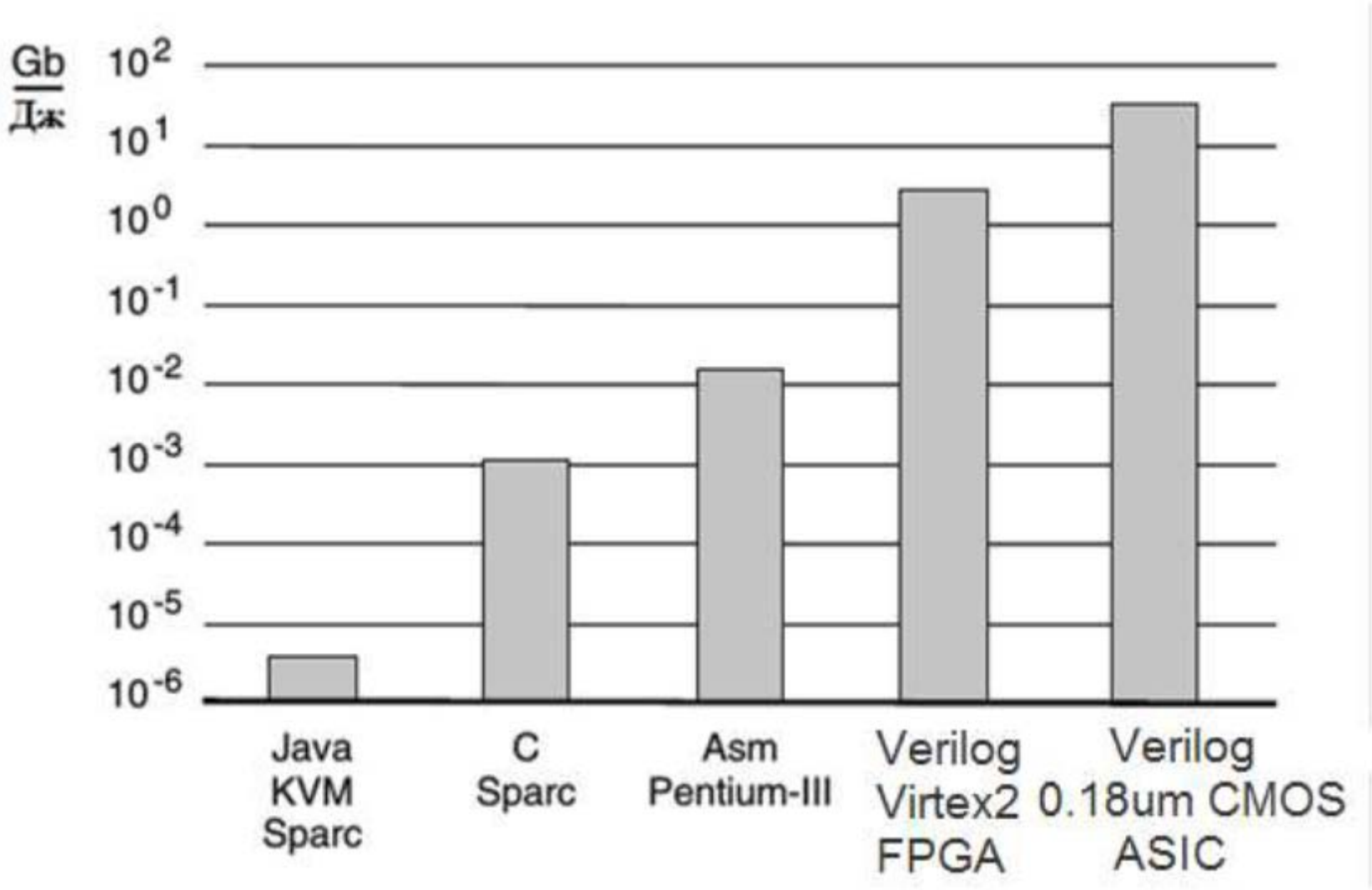



Рис. 3.3. Энергетическая эффективность

Рис. 3.3 даёт представление, насколько могут различаться по эффективности разные подходы к реализации. На нём приведены характеристики энергопотребления для реализации одного и того же приложения — шифрования по стандарту AES на различных целевых платформах. Слева направо это:

- Java, запущенная на Java-машине, которая запущена на микроконтроллере;
- C-программа на микроконтроллере;
- оптимизированный код Ассемблера на процессоре Pentium-III; код на языке Verilog, запущенный на Virtex-II FPGA;
- ASIC-реализация.

Гибкость реализации изменяется слева направо, от очень легко и быстро программируемой Java-машины до трудоёмкой в проектировании и неизменяемой после изготовления заказной микросхемы. Ось Y показывает число гигабит информации, которые могут быть закодированы каждой из платформ, используя один

Джоуль энергии. Масштаб логарифмический, и можно видеть, что разница в эффективности различается на порядки.

## 3.2. Системы на кристалле

Система на кристалле (System on Chip, SoC, СнК) — система, построенная на едином кристалле, в которой интегрируются такие элементы, как процессор (процессоры, в том числе специализированные), некоторый объем памяти, ряд периферийных устройств и специализированных вычислительных блоков, и их соединения. Всё вышеперечисленное составляет оптимальный набор для некоторого заранее известного приложения — например, обработки и передачи видеоданных. Выражение "система на кристалле" не является, строго говоря, термином. Это понятие отражает общую тенденцию к повышению уровня интеграции за счёт интеграции функций.

СнК часто используются в качестве компонентов ВсС и в предельных случаях фактически ими же и являются, для ВсС и СнК используются одни методы проектирования. Реализация в виде СнК исключает необходимость применения многочисленных отдельных интегральных схем и реализации интерфейсов связи между ними. От обычных микроконтроллеров СнК отличается спецификой организации, направленной на решение конкретных задач, в то время как микроконтроллер реализует универсальный набор функций. Также отличием от микроконтроллеров является то, что в СнК обычно используются более мощные процессоры, пригодные для запуска тяжеловесных операционных систем (Linux, Windows), которые зачастую требуют для эффективной работы больших анков внешней памяти и развёрнутую периферию. В некоторых реализациях микросхемы СнК имеют контакты как на нижней своей части для соединения с платой, так и на верхней — для установки дополнительных блоков памяти.

Типовая встраиваемая система, построенная на базе SoC, может содержать такие наборы интерфейсов и контроллеров, как:

- системная шина и контроллеры шин LPC/ISA, PCI, PCMCIA;
- контроллеры управления NOR/NAND Flash, SDRAM, SRAM, DDR;
- контроллер Ethernet;
- последовательные интерфейсы UART, SPI/SSP/uWire, RS-232, RS-422/RS-485, CAN;
- беспроводные интерфейсы WiFi/IEEE802.11, ZigBee, Bluetooth, IrDA;
- интерфейсы поддержки Flash-карт памяти: SD/MMC, CompactFlash, MemoryStick;
- контроллер LCD STN/TFT/OLED;
- контроллер матричной клавиатуры;
- модули беспроводной передачи данных GSM/GPRS, CDMA;

- модули приема сигналов спутниковых навигационных систем GPS, Glonass;
- аппаратные поддержки плавающей точки, шифрования, DRM и т.п.;
- аудио- и видео- интерфейсы;
- источники опорной частоты;
- регуляторы напряжения и стабилизаторы питания.

Преимущества "систем на кристалле" перед классическими "системами на печатной плате":

1) Миниатюризация. Как правило, устройство, созданное на базе SoC, состоит из одной (максимум 2-х) СБИС и ограниченного набора дискретных компонент, которые по технологическим причинам не могут быть интегрированы внутрь ИС.

2) Снижение потребляемой мощности. СБИС типа "система на кристалле" изготавливаются по технологии "глубокого субмикрона" (DSM — Deep Submicron) 0.35 мкм и ниже, что позволяет снизить

напряжение питания и, как следствие, существенно уменьшить потребляемую мощность.

3) Повышение надёжности. Объединение нескольких компонент (IP-блоков) на одной пластине кремния позволяет существенно уменьшить число паяных соединений.

4) Снижение стоимости для больших партий. ВсС, выполненных на базе СнК, при налаженном производстве являются гораздо более дешёвыми решениями, чем обычные системы на платах.

5) Упрощение монтажа. При использовании технологии СнК требуется устанавливать меньше корпусов на плату — как правило, сам вычислительный элемент.

Также, необходимо помнить, что процессу проектирования аппаратуры обычно сопутствует процесс создания ПО, драйверов, позволяющих «оживить» создаваемую микросхему. Кроме того, производительность приборов класса СнК в значительной мере зависит от эффективности взаимодействия всех встроенных

компонентов и от эффективности их взаимодействия с внешним, относительно прибора, миром. В первую очередь это связано с различием в быстродействии встроенных компонентов, в особенности организации интерфейсов.

При всех достоинствах технологии проектирование и отладка СнК как единой сложной системы является гораздо более трудоёмким и сложным процессом, чем проектирование её компонентов в виде отдельных микросхем. Исходя из этого, одной из основных проблем при разработке СнК является борьба со сложностью и трудности анализа протекающих внутри микросхемы процессов.

Помимо этого, учитывая высокую стоимость исправления ошибок в конечных реализациях, с одной стороны, значительную долю процесса проектирования СнК всегда занимает верификация и всестороннее тестирование получаемого продукта, а с другой —

ищутся пути внесения изменений в законченное и уже выпущенное с производства изделие.

Последний вариант привлекателен ещё и тем, что может увеличить гибкость устройства, позволяет лучше настраивать его под конкретное приложение.

Существует отдельный класс СнК, представляющих однокристальное конфигурируемое или программируемое решение, которое допускает оперативное изменение своей внутренней аппаратной структуры и конечного предназначения на этапе производства или в полевых условиях, непосредственно в проекте. Такие интегральные схемы были отнесены к группе изделий системного уровня интеграции, но получили другое название — Configurable System on a Chip или CSoC. Поскольку термин CSoC не стандартизован, то существуют и другие названия изделий этого класса — System on Programmable Chip (SoPC), Programmable System

on a Chip (PSoC) или просто SoC, что определяется вкусом и желаниями конкретного производителя микросхем.

Помимо этого, перспективным на данный момент направлением развития СнК считаются сети на кристаллах — СнК, в которых соединения между отдельными компонентами реализуются не в виде шин и прямых соединений, а в виде сетей. Это позволяет улучшить масштабируемость однокристальных решений, а также, в больших проектах — энергопотребление. Кроме того, ведутся исследования по использованию беспроводных соединений для связи в рамках чипа, с целью обхода ограничений традиционных электрических соединений, связанных с задержками передачи сигналов, их рассинхронизацией и энергетическими потерями в цепях.

## 3.3. Методология проектирования SystemC

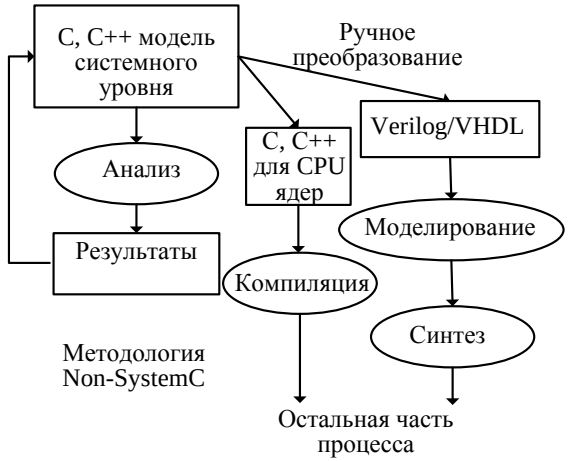
Для повышения эффективности проектирования «систем на кристалле» и устранения ошибок применяют моделирование электронных систем на языке SystemC (рис. 3.4).

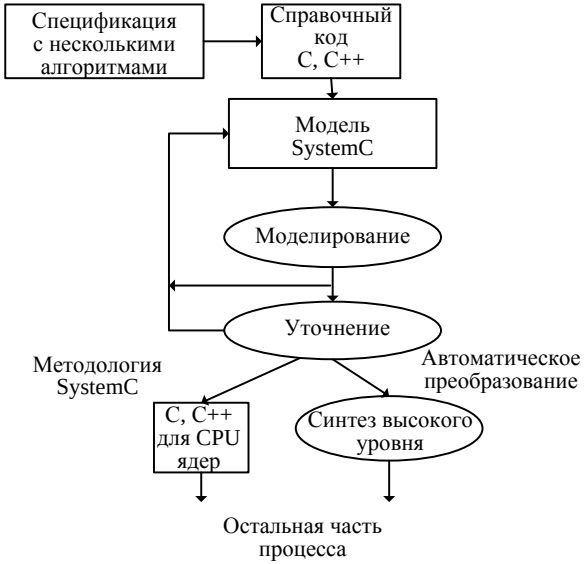

Рис. 3.4. Учебное пособие по SystemC

Чтобы понять методологию проектирования SystemC и ее преимущества, важно сначала понять методологию, отличную от SystemC.

В методологии non-SystemC (рис. 3.5) разработчики системы должны написать исполняемые спецификации в C или C ++, а затем проверить и отладить этот проект. Обнаружив, что эта конструкция удовлетворяет всем техническим требованиям, она передается в группы разработки RTL. Затем группа проекта RTL перезаписывает проект в RTL, чтобы синтезировать его для вентилей. В такой методологии функциональное RTL-описание иногда зависит от выполняемых спецификаций и, следовательно, становится склонным к ошибке. Кроме того, возникает реальная проблема, если на уровне RTL обнаруживается, что что-то в концептуальной модели не может быть реализовано, так как нет общей среды проектирования между разработкой системы и ее реализацией.

В методологии SystemC разработчику системы нужно только написать модель SystemC. Разработчик может итеративно уточнять исполняемые спецификации вплоть до уровня регистровой передачи, который все еще находится в SystemC до синтеза. Испытательные программы (Testbench) можно использовать повторно, чтобы гарантировать, что итерационный процесс не имеет ошибок. Если во время реализации RTL обнаружено, что что-то концептуально неправильно, гораздо проще его переписать.




Non SystemC SystemC

Рис. 3.5. Сравнение методологии проектирования

SystemC использует следующие типы моделей:

- архитектурная модель системы;
- модель производительности системы;
- модель уровня транзакций (TLM);
- функциональная модель;

• модель передачи на уровне регистров (RTL).

Некоторые проекты SystemC начинаются как функциональные модели, в то время как большая часть кода SystemC делается на уровене TLM. Почти всегда TLM выступает в качестве исполняемой платформы, которая является достаточно точной для запуска программного обеспечения.

Главной причиной использования SystemC является значительное увеличение производительности симуляции на уровне TLM по сравнению с исполняемыми платформами, смоделированными на уровне RT с использованием Verilog или VHDL. Модели SystemC TL достаточно быстры, чтобы служить платформой для разработки программного обеспечения, что позволяет выполнить раннюю разработку программного обеспечения и совместное моделирование аппаратного и программного обеспечения. Обе TL и функциональные модели достаточно быстры для архитектурного моделирования и анализа на системном уровне.

Типичные числа для модели System on Chip составляют от ~ 1K циклов в секунду до более высоких ~ 300-400K циклов в секунду. Этот диапазон зависит от того, как процессор моделируется в системе. Если использован симулятор набора инструкций (ISS), то производительность обычно составляет от 1 до 10 тыс. циклов. Если процессор моделируется как прямое подключение к системной шине, то производительность переходит в диапазон 100K и выше.

Вторая причина использования SystemC - функциональная проверка. Одна и та же исполняемая платформа, которая используется для разработки программного обеспечения, часто используется и для проверки всей системы. Эта проверка происходит на раннем этапе проекта, и TLM становится прекрасной проверкой для всей системы.

Поскольку SystemC - это C ++, у него есть ряд неотъемлемых свойств, таких как классы, шаблоны и наследование, которые поддаются проверке. Эти возможности дополняются SystemC

Verification Library (SCV), что делает SystemC мощным языком проверки, а также языком моделирования.

## 3.4. Многоядерные системы

Развитие компьютерных архитектур в настоящий момент тормозится наличием следующих основных системных проблем:

1) Энергетическая стена – транзисторы дешевы, энергия дорога;

2) Стена памяти – памяти много, но доступ к памяти сравнительно медленный;

3) Стена параллелизм**а** – комплексная проблема, начинающая с параллелизма на уровне команд и кончающаяся распараллеливанием на уровне процессорных ядер;

4) Стена образования – классическое высшее образование с достаточно жесткой привязкой к архитектуре Фон-Неймана делает научно-техническое сообщество более инертным и мешает развивать и внедрять новые архитектуры систем.

Первая стена не позволяет увеличивать дальше тактовую частоту процессорных ядер, а остальные три стены затрудняют увеличение количества ядер. Кроме того, увеличение количества ядер не всегда приводит к пропорциональному приросту производительности, вследствие следующих ограничений:

1) Закон Амдала: «В случае, когда задача разделяется на несколько частей, суммарное время её выполнения на параллельной системе не может быть меньше времени выполнения самого длинного фрагмента» (см. рис. 3.6);

2) Неэффективная схема связей между ядрами, вытекающая из технологических ограничений по организации связей между узлами сетей на кристалле или кристаллами на печатной плате и не позволяющая добиться нужной степени распараллеливания для данной задачи;

3) Сложность программирования.

Тем не менее, несмотря на описанные выше проблемы, многоядерные системы являются основным и едва ли не единственным на данный момент реально осуществимым подходом к созданию высокопроизводительных, «high-end» процессоров. Причём, число вычислительных ядер в последних процессорах исчисляется десятками (Рис. 14). То, что такие компании- гиганты, как Intel и Amd, применяют именно эту технологию в своих перспективных разработках, говорит само за себя. Активно используются многоядерные системы и в области BcC.

Концепция многоядерных систем подразумевает как минимум три аспекта:

1) наличие нескольких вычислительных ядер;

2) наличие коммуникаций между этими ядрами;

3) наличие связи с внешним миром – памятью, периферийными устройствами и т.д.

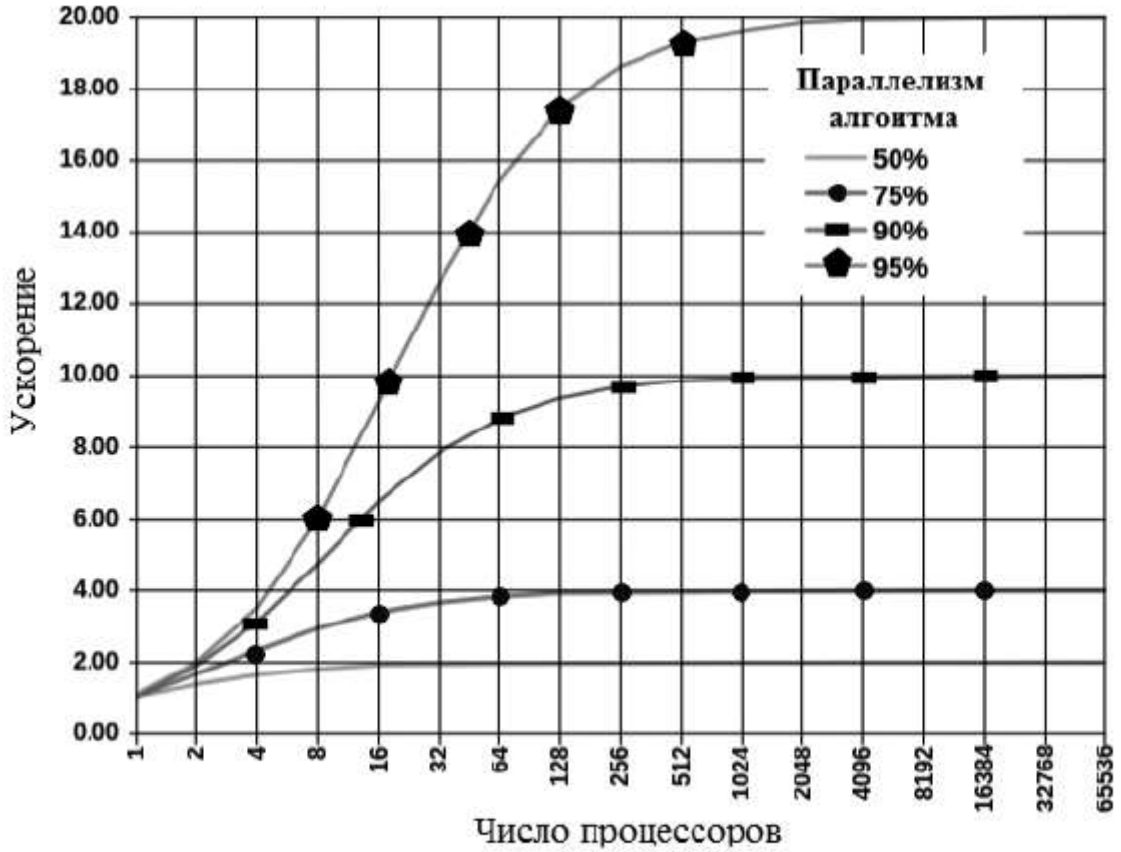



*Рис. 14. Закон Амдала.*

Рис. 3.6. Закон Амдала

Процессоры, которые образуют многоядерные системы, могут быть гомогенными – когда на всех ядрах могут запускаться одни и те же бинарные файлы. Такие процессоры имеют одинаковые наборы команд и производительность. При этом современные архитектуры могут управлять частотой тактирования для каждого ядра, что помогает в вопросах энергосбережения или, наоборот, даёт возможность временного ускорения какого-то приложения. Вообще, большинство многоядерных процессоров общего назначения являются гомогенными. Кроме этого, в большой их части реализовано глобальное адресное пространство, общее для всех ядер, и полная когерентность кэш-памяти (когерентность предполагает, что у каждого процессора свой кэш, но в него отображаются все изменения в глобальной памяти). Таким образом, с программной точки зрения, одно гомогенное ядро практически ничем не отличается от другого.

Гетерогенные архитектуры включают как минимум два разных типа ядер, различающихся по набору инструкций, функциональности или производительности.

Кроме физической многоядерности, на каждом процессоре может использоваться технология SMT, позволяющая запускать одновременно несколько вычислительных потоков, на самом деле выполняя их поочерёдно. Это достигается за счёт дополнительных наборов регистров и контроллеров прерываний при общих остальных ресурсах. Благодаря этому, в момент простоя одного потока вычислений управление передаётся другому — например, в момент ожидания освобождения какого-то ресурса, или завершения долгой, многотактовой команды. В процессорах производства Intel эта технология, называемая «Hyper-threading», позволяет получать два логических процессора для каждого физического, в процессорах Sun UltraSPARC — восемь. Однако, это не является многоядерностью в прямом смысле слова.

Обеспечение непротиворечивости памяти при возможности доступа к ней нескольких параллельно работающих ядер находится в числе фундаментальных проблем создания многоядерных систем. Это приводит к необходимости введения методов синхронизации, обеспечивающих когерентность использующихся несколькими ядрами данных. При этом механизмы блокировки или синхронизации могут стать причиной конфликта блокировок между несколькими потоками или процессами, конкурирующими за доступ к данным (или другому ресурсу).


*Рис. 15. Структура типового многоядерного процессора* [20].

Рис. 3.7. Структура типового многоядерного процессора

Многоядерные системы представляют собой очень мощный и при этом чрезвычайно сложный инструмент. Это проявляется как в их использовании, с точки зрения программистов, так и в проектировании, с точки зрения разработчиков микропроцессоров.

Остаются нерешёнными задачи оптимизации доступа к памяти, снятия ограничений масштабируемости многоядерных систем, обеспечения соединений между ядрами. При этом в большинстве случаев современное ПО пишется в стандартной, последовательной манере, без учёта возможностей многопоточного исполнения.

## 3.5. Реконфигурируемые системы

Реконфигурируемые вычислительные системы (РВС) — это системы, имеющие возможность менять свою модель вычислений, или, иначе говоря, позволяющие вносить существенные изменения в свою аппаратную часть. РВС занимают нишу между двумя парадигмами создания ВС: чисто программными системами на некотором процессоре с фиксированной аппаратной архитектурой и

специализированными аппаратными, реализующими, как правило, потоковую модель вычислений.

Необходимость таких систем обусловлена тем, что классическая вычислительная система на базе архитектуры фон-Неймана имеет множество недостатков. Одна из её проблем – взаимодействие процессора с памятью, так как современная память работает значительно медленнее процессора. Несмотря на наличие механизмов, ускоряющих этот процесс (многоуровневая кэш память, конвейерное исполнение команд, предсказание ветвлений), в большинстве случаев АЛУ процессора простаивает, что снижает эффективность системы в целом.

Эффективность работы процессора немного возрастает при использовании механизма гипертрединга, когда к одному блоку памяти обращается два потока, работающие с одной и той же областью памяти в рамках одного процесса. Но, тем не менее рост производительности, необходимый для высокопроизводительных

вычислений, и экономичность, важная во ВсС, плохо достижимы в рамках классических архитектур. Чем больше разрыв между скоростью работы памяти и процессора, тем ниже фактически достигаемая эффективность работы вычислительной системы.

Кроме этого, 95% времени выполнения инструкции (см. Рис. 3.8) приходится на всевозможные вспомогательные и подготовительные действия, и лишь 5% на саму вычислительную операцию – то, ради чего всё, собственно, и организуется.


*Рис. 16. Примерное соотношение времени работы различных вспомогательных действий при выполнении инструкции микропроцессора с архитектурой Фон-Неймана* [21].

Рис. 3.8. Соотношение времени выполнения вспомогательных действий

Использование РВС позволяет достигать производительности вычислений большей, чем при использовании ПО и фиксированного процессора (но всё же, обычно, меньшей, чем с аппаратно реализованными потоковыми вычислениями), при этом — с некоторой степенью гибкости и возможностью изменения исполняемых функций в определённых рамках, ограниченных заданными ресурсами. Таким образом получается, фактически, изменяемый аппаратный блок, обладающий преимуществами аппаратной реализации и в дополнение возможностью перестраиваться под разные приложения.

Наиболее актуальны РВС в области создания суперкомпьютеров и встраиваемых систем. В первом случае, реконфигурация позволяет получать выигрыш в виде высокой производительности при решении различных специфических ресурсоёмких задач, где уже достигнут предел повышения производительности, полученный стандартными экстенсивными методами, или же — обходится слишком дорого. Во

втором — обеспечивается необходимая вычислительная мощность в, наоборот, небольших вычислительных комплексах, которые работают в рамках постоянного дефицита ресурсов: габаритных размеров, массы, энергии, стоимости, что усугубляется также требованиями реального времени. То есть, это два граничных случая для всех ВС.

Аппаратные вычислительные элементы РВС могут быть реконфигурированы однажды, когда система выпускается, раз в какое-то количество лет, для устранения багов, а также для обновлений (статическая реконфигурация), или же раз в несколько часов, чтобы адаптироваться под текущую задачу (динамическая/run-time реконфигурация). Часто реализуется частичная реконфигурация — динамическая реконфигурация некоторой части аппаратуры при неизменной структуре остальной аппаратуры.

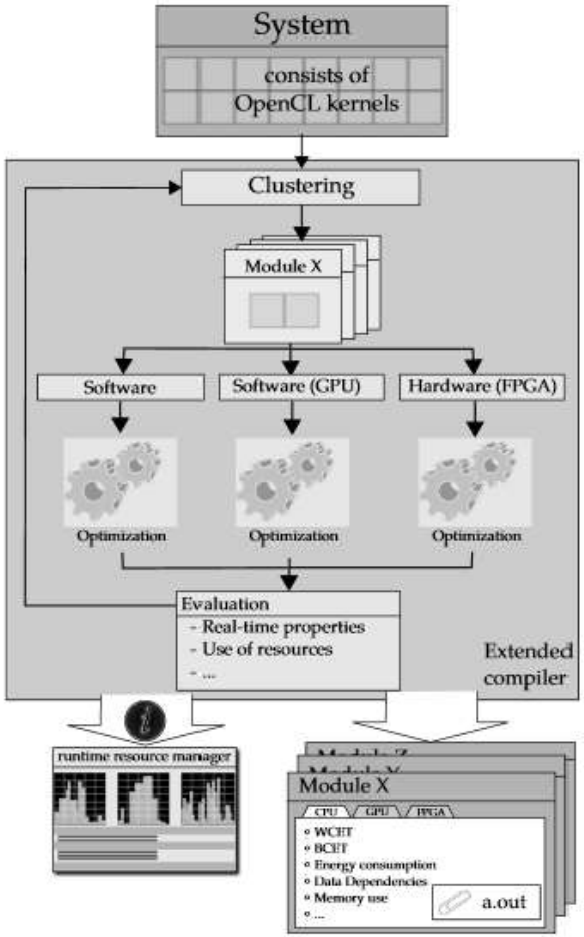



*Рис. 17. Организация динамической реконфигурации с компиляцией вычислительных ядер перед выполнением для всех аппаратных блоков [22].*

# Рис. 3.9. Организация динамической реконфигурации

РВС чрезвычайно разнообразны по архитектуре и требуют сложных технологий создания конфигурационного обеспечения (configware), что делает задачу их проектирования нетривиальной. Это подтверждается опытом создания большого числа как экспериментальных, так и серийных ВсС с реконфигурируемой архитектурой.

Важными задачами, требующими решения, являются также определение частей алгоритма, которые требуют оптимизации, и выделение их из изначального вычислительного процесса. Один из подходов предполагает наличие менеджера ресурсов, который перераспределяет вычислительные задачи между различными блоками (CPU, DSP, GPU, FPGA) в зависимости от требуемой для решаемой задачи вычислительной мощности.

Основными задачами, решаемыми при создании РВС являются:

- создание аппаратных блоков, обладающих свойством реконфигурируемости;

- выявление участков приложения, которые требуется оптимизировать;
- отображение оптимизируемых частей приложения на реконфигурируемую аппаратуру (создание оптимизированных аппаратных реализаций);
- реконфигурация – способ записи новой конфигурации в аппаратные реконфигурируемые блоки;
- включение реконфигурируемых блоков в вычислительный процесс.

Ключевая проблема, связанная с реконфигурируемой аппаратурой – сложность её программирования. Традиционные процессы разработки: для аппаратуры – синтез, размещение и трассировка, для ПО – компиляция, запуск и отладка – не поддерживают реконфигурируемые системы.

Динамическое реконфигурирование может быть рассмотрено как гибрид между ПО, где ЦПУ "реконфигурируется" с выполнением

каждой инструкции, а объём памяти большой, но пропускная способность доступа к ней ограничена, и аппаратурой, где реконфигурация происходит редко (например, записью в конфигурационные регистры UART), памяти мало, но доступ к ней имеет потенциально высокую пропускную способность.

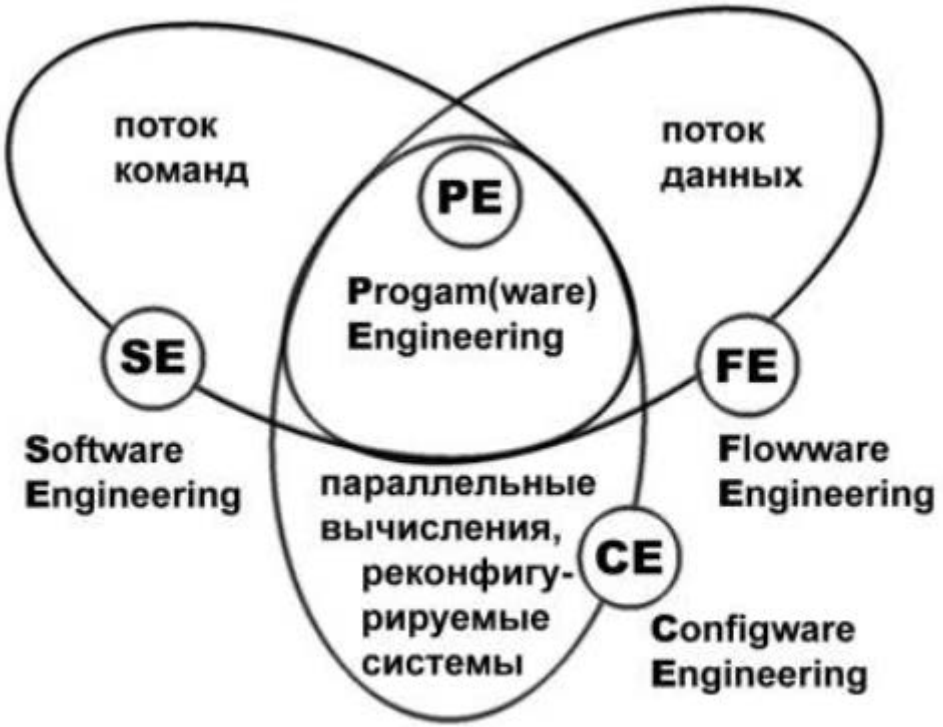



*Рис. 19. Современное представление вычислительной техники* [21].

Рис. 3.10. Современное представление вычислительной техники

## 3.6. Проблемно-ориентированные процессоры

Проблемно-ориентированный процессор, заказной процессор — это специализированный процессор, проектируемый под конкретную задачу.

Вообще, под названием «application-specific processor (ASP)» в англоязычной литературе понимаются и ASIC, реализующие решение задачи на жёсткой логике, и цифровые сигнальные процессоры (DSP), и процессоры с расширенными системами команд (application-specific instruction-set processor, ASIP). Однако, в русскоязычной литературе под названием «проблемно-ориентированный процессор» понимаются именно специализированные процессоры, имеющие не стандартную систему команд, которая вместе с поддерживающей её аппаратурой создаётся для решения некоторой заранее известной и специализированной задачи.

ASIP являются компромиссом между универсальностью и программируемостью процессоров общего назначения, производительностью и энергетической эффективностью ASIC-решений, которые, обычно, серьёзно оптимизируются, но под конкретные приложения. Как правило, данный подход используется в технологии СнК.

Архитектура ASIP включают статическую логику, в которую входит некоторый набор стандартных инструкций, и конфигурируемую логику для расширений, изменяемую в процессе функционирования или при синтезе архитектуры. Таким образом, запущенное на ASIP ПО может использовать как стандартные возможности процессоров общего назначения, так и улучшенные аппаратно.

В отличие от аппаратных ускорителей на жёстко заданной логике, решения на основе ASIP обладают значительно большей гибкостью и лучше приспособлены для повторного использования.

Проектирование ASIP включает следующие шаги:

1) Исследование пространства возможных архитектурных решений;

2) Реализация архитектуры;

3) Создание ПО для полученной архитектуры;

4) Системная интеграция.

ASIP в большинстве случаев моделируются при помощи более абстрактных языков описания архитектуры (Architecture Description Languages, ADL), которые содержат информацию не только об архитектуре аппаратуры процессора, но и о его наборе инструкций, модели памяти и других аспектах, важных для программирования.

# Глава 4. Проектирование программно-реализованных встраиваемых систем

## 4.1. Особенности проектирования встраиваемых систем

Изучение и проектирование ВсС влечёт за собой некоторые определённые проблемы, которые являются редкими в универсальных вычислениях и обычных настольных компьютерах.

В ВсС время, которое требуется, чтобы выполнить задачу, может быть критичным для правильного функционирования системы, ведь происходящие в реальном мире физические процессы редко являются процедурными, и промедление в обработке каких-то событий для критических областей может повлечь непоправимые последствия.

Принципиальное отличие информационных систем от систем реального времени, к которым, в большинстве случаев, относятся ВсС, состоит в трактовке параметра «реакция вход-выход»: «The

right answer late is wrong» («Правильный ответ поздно – неправильный»).

Система должна успевать обрабатывать изменения значений на своих входах и выставлять актуальные значения управляющих выходов в соответствии с заложенными алгоритмами, влияя тем самым на окружающую среду и подчинённый объект (объекты) управления. Таким образом, осуществляется управление динамикой процессов в окружающей среде по принципу обратной связи: за счёт измерений состояния контролируемых процессов и организации действий, которые влияют на данные процессы через некоторые исполнительные механизмы.

Может присутствовать необходимость одновременного выполнения нескольких действий, каждое из которых является важным – ведь процессов, которые необходимо контролировать, часто бывает больше одного. Исходя из этого – параллелизм внутренне присущ ВсС.

Подход, когда каждая задача ждёт своей очереди, и производится простой последовательный перебор по порядку, в большинстве случаев неприемлем в алгоритмах управления.

При проектировании ВсС необходимо добиваться максимально быстрого и параллельного с точки зрения внешней среды поведения ВС, несмотря на последовательную природу большинства современных компьютеров.

Желательно, чтобы ВсС функционировала так, как будто является полностью параллельной системой. Многие из технических проблем при разработке и анализе встроенного программного обеспечения микропроцессорных систем как раз и происходят вследствие потребности соединять последовательные вычисления с непрерывно и одновременно изменяющимися во внешнем мире процессами.

Зачастую такие системы применяются в ответственных областях, что влечёт за собой повышенные требования к надёжности и всесторонней защите проектируемых устройств. Если выходит из

строя бортовой компьютер автомобиля или самолёта, медицинское оборудование, блок управления лифтом, светофором или каким-нибудь станком – это может вызвать причинение вреда людям, окружающей среде, чьему-нибудь имуществу.

Для обеспечения достаточной степени надёжности работы устройств нужно учитывать множество факторов: как природные, естественные воздействия, так и последствия человеческой деятельности.

Причём последнее включают в себя случайные факторы и целенаправленные вредоносные воздействия – вандализм, попытки несанкционированного доступа, взлома устройств.

Учитывая, что набирающий силу киберфизический подход к созданию ВсС подразумевает в большинстве случаев подключение к сети интернет, а также то, что понятие «интернет вещей», когда каждое устройство подключается к глобальной сети и становится доступным извне, уже становится реальностью – появляется всё

больше мишеней для вероятных атак извне и всё острее встаёт проблема обеспечения безопасности.

Кроме внешних факторов, влияющих на работу системы уже во время её функционирования, обязательно следует учитывать и возможность возникновения ошибок во время процесса создания продукта.

Поэтому процессу создания ВсС в широком смысле – учитывающем сначала разработку, затем изготовление – обычно сопутствуют процессы верификации и тестирования.

Разработчики соблюдают два правила.

Первое – то, что максимально надёжное является максимально простым.

Второе следствие – то, что ВсС, состоящие из большого количества тесно взаимодействующих «элементов» (в качестве которых можно выделить ПО, в том числе операционные системы, библиотеки и драйвера, интегральные схемы, все электронные

элементы, устанавливаемые на платах, со связями между ними, датчики, исполнительные механизмы, соединительные кабели и т.д.), априори имеют большой потенциал для возникновения ошибок при проектировании и отказов – уже в процессе работы. Даже при полностью отлаженных отдельно ПО и аппаратуре, в их совместной работе могут возникать непредвиденные эффекты, вызванные взаимодействием этих базовых элементов.


Рис. 4.1. Взаимное влияние компонентов ВсС и внешних факторов, сказывающееся на сложности и надёжности.

Сама по себе разработка ВсС представляет собой результат междисциплинарного проектирования, в котором условно можно выделить три основных составляющих:

- Этап решения задачи на прикладном уровне, когда необходимо найти правильные методы и алгоритмы без деталей реализации. Это сфера деятельности прикладных специалистов из соответствующих областей (физика, энергетика, медицина,
- лингвистика, биология и др.).
- Процесс проектирования, в ходе которого требуется отобразить полученное прикладное решение на технологическую базу информатики и вычислительной техники (ВТ). Это работа специалистов из области информатики, сегодня все чаще ее называют архитектурным, высокоуровневым или системным проектированием.

- Фаза реализации, в ходе которой инженеры, программисты и прикладные специалисты обеспечивают выполнение ранее сформулированных требований, таких как необходимая функциональность, динамика поведения, надежность и безопасность функционирования, габариты, энергопотребление, стоимость и технологичность при тиражировании.

После этапов разработки следуют обязательные шаги по верификации и валидации. Иногда они постоянно сопутствуют процессу проектирования, позволяя оценивать промежуточные результаты до получения конечного продукта, что позволяет вовремя отследить отклонения от нормы и допускаемые в процессе ошибки. Помимо этого, в процесс проектирования обязательно входят шаги по анализу возможных решений.

Таким образом, **разработка ВсС требует обширных знаний в разных областях** – понимания вычислительных систем,

программного обеспечения, сетей и физических процессов, умения оценивать их совместную динамику. Чтобы заставить ВсС работать с физическими процессами, требуется технически сложное, *низкоуровневое проектирование.* Разработчики встраиваемого ПО, если система реализуется на микропроцессорной базе, вынуждены бороться с контроллерами прерываний, архитектурами памяти, программированием на уровне ассемблера (для использования специализированных функций или для точного контроля времени выполнения), разработкой драйверов устройств, сетевых интерфейсов и политик планирования. *Разработчики аппаратуры должны учитывать многочисленные ограничения, контролировать реальные задержки сигналов, огромное количество ресурсов тратится на верификацию полученных продуктов*. Кроме того, очень важно иметь чёткое представление о предмете проектирования, доступных методах и средствах его создания. На системном уровне требуется контролировать множество факторов,

координировать усилия отдельных разработчиков, что тоже является отдельной и важной задачей.

Проектируя ВсС, разработчик всегда создает *специализированную вычислительную систему* независимо от степени соотношения готовых и заново создаваемых решений.

## 4.2. Современное проектирование ВсС

На сегодняшний день подавляющее большинство ВсС являются распределенными (РИУС, NECS). Представление ВсС как распределенных, включение в категорию ВсС контроллерных сетей или распределенных информационно-управляющих систем для многих специалистов (особенно отечественных) является спорным. В зарубежной научно-технической литературе, напротив, доминирует представление о ВсС как о широком классе вычислительных систем, включающем как малогабаритные или одномодульные устройства, так и распределенные системы, которые непосредственно сопряжены с объектами большой протяженности,

большого масштаба, пространственно распределенными и т.д. И в этом смысле определенные изменения в восприятии, в мышлении специалистов, связанных с этой областью, в нашей стране обязательно должны произойти. Иначе мы будем продолжать разговаривать в области встраиваемых вычислительных систем на разных языках. Перспективной является эволюция понятия ВсС в сторону комплексных кибер-физических систем (к ним можно отнести электронно-механические, гидравлические, биологические и т.д. системы с вычислительной составляющей). Суть термина CPS состоит в том, что проектирование объекта и системы управления этим объектом должно выполняться в едином ключе и в едином комплексе в рамках объединённых или как минимум очень тесно взаимодействующих инструментальных средств. Четкого понимания и применения требуют отмеченные выше варианты трактовки ВсС как завершенной (целевой) прикладной системы или как проектной платформы (с различной степенью универсальности). Автор считает

первый вариант более значимым в плане осмысления и организации процессов проектирования конечных систем и рассматривает второй (платформенный) вариант определения ВсС более узким. Проектирование вычислительной платформы удобно рассматривать в качестве одного из сценариев проектирования в области ВсС. Договорённость между специалистами по данному вопросу необходима для корректной оценки видов и объёмов работ, выбора маршрутов проектирования, определения зон ответственности исполнителей, собственно, для выполнения проектов. Важным в проектировании ВсС является анализ базовых сценариев проектирования, т.к. они во многом определяют постановку задачи или постановка задачи определяет выбор сценария и необходимую методику проектирования. В рамках сценариев методики сильно различаются.

Выделяются три сценария проектирования ВсС:

а) заказной или целевой, когда создается специализированная ВсС «под ключ»;

б) создание вычислительной (проектной) платформы для целевых ВсС;

в) модификация существующих систем или платформ.

Основными являются первые два сценария. В качестве иллюстрации сложности и важности сценариев можно отметить бурное развитие методологии PBD (платформно-ориентированное проектирование), первоначально явно направленной на создание СнК. В обозримой перспективе развитие СнК может преодолеть барьер, когда удельный вес реализации целевой задачи был значительно меньше затрат на создание платформы. Мы получим возможность в рамках данной технологии приблизиться (или перейти) к сценарию заказного проектирования. В таком случае акценты (и технологии) в PBD необходимо будет менять кардинально. Понятие вычислительной или (более широко)

проектной платформы играет важную роль в создании ВсС. Действительно, существует значительное число подходов или направлений в их проектировании. Коллективы специалистов в различной степени привержены конкретным направлениям и проектным платформам. Часто выбор платформы определяется вторичными, не совсем техническими, факторами, например традициями коллектива, освоенными архитектурами и инструментарием и т.д. Для разработчика ВсС очень важным является широкое видение и обоснованный выбор вариантов решения задачи. Проектная платформа, как решение архитектурного уровня, в традиционных технологиях создания ВсС действительно характеризует практически весь маршрут проектирования. Поэтому на основе этого понятия возможна классификация архитектурных решений для ВсС. Перечень широко используемых в настоящее время и перспективных категорий архитектурных платформ в области ВсС представлен ниже:

- платформы промышленных ПК;
- платформы программируемых логических контроллеров;
- полуфабрикаты от мультимедиа-индустрии (КПК и сотовые телефоны с мобильными операционными системами, системы прикладного программирования в стиле micro-DotNet и т.д.);
- микроконтроллерные модули со средствами программирования (на «железе», RTOS, виртуальные машины и т.д.);
- DSP-платформы; - ПЛИС, ПСнК;
- ASIC, ASSP, SOC и т.д.;
- «свободная» кремниевая компиляция.

Данный перечень лишь в упрощенном и частично косвенном виде представляет вычислительные архитектуры ВсС. Например, в нем не отражен важнейший аспект распределенности и параллельности вычислений. В основу классификации в данном случае положено не назначение, не функциональность проектируемой системы, а архитектурная особенность,

архитектурная направленность — парадигма, являющаяся фундаментом проектирования той или иной ВсС. Платформа в данном отношении рассматривается как нижний фиксированный в проекте уровень ресурсов/сервисов, «гранулярность» которых различна. Поясним этот тезис. Например, платформы промышленных персональных компьютеров и программируемых логических контроллеров позволяют относительно просто и быстро создать прикладную систему, однако эта эффективность проявляется только в рамках «стандартных» технических заданий. Проектные платформы микроконтроллеров и сигнальных процессоров предоставляют большую свободу разработчику. Они имеют свою специфику, в первую очередь в организации системного программного обеспечения, в степени открытости архитектуры и т.д. В качестве платформ с успехом используются ПЛИС и программируемые СнК, сочетающие гибкость программных и аппаратных средств. Безусловно, можно говорить об использовании

ветки заказного проектирования микросхем большой степени интеграции — ASIC. Нельзя упускать из виду и предельный вариант, когда в качестве платформы выступает весь электронный логический базис в рамках принципа свободной кремниевой компиляции. Последний вариант — в некотором роде из области фантастики, а перечисленные выше — наши реалии в проектировании.

### 4.3. Высокоуровневое проектирование встраиваемых систем

Ранее проектирование цифровых устройств начиналось с размещения транзисторов для реализации нужной функции. Очевидно, что при таком ручном методе проектирования гибкость заключается в выборе размеров транзисторов и способе проведения проводных соединений, однако при этом достигается оптимальная реализация заданной функции.

По мере усложнения проектов возникла необходимость в еще более высоком уровне абстракции, при котором количество элементов уменьшилось бы, по сравнению с числом вентилей.

Основное внимание на этом уровне абстракции сосредоточено на передаче данных между регистрами, логическими узлами и шинами. Поэтому он и называется уровнем регистровых передач (RTL).

С течением времени проекты еще более усложнились, и возникла необходимость в переходе на более высокий, по сравнению с регистровыми передачами, уровень абстракции. В настоящее время этот уровень, называемый уровнем электронной системы (Electronic System Level – ESL), или системным уровнем, является наивысшим. На системном уровне разработчик заботится только о функционировании разрабатываемой системы и описывает алгоритм, который должен быть реализован. Алгоритм описывается с помощью процедурного языка, подобного языку программирования C. Описание системы на этом уровне не содержит синхросигналов или временных задержек вентильного уровня.

Средства проектирования системного уровня включают средства ввода, моделирования и, конечно, программы генерации аппаратной

части. Генерация аппаратуры из описания системного уровня может выполняться одним из двух возможных способов. Первый способ аналогичен применяемому на других уровнях абстрагирования и заключается в трансляции описания системного уровня на низший уровень абстракции, то есть на уровень регистровых передач. Альтернативным способом является путь, когда процедурное описание системного уровня может компилироваться для выполнения на заданном процессоре. Этот метод становится возможным только на системном уровне, потому что описание системного уровня является процедурным, и для этого используется язык описания программной части системы, подобный языку С.

Именно последний вышеупомянутый метод генерации аппаратуры из описания системного уровня и должен стать методом проектирования встраиваемых систем. Традиционную методику, то есть трансляцию описания на системном уровне в описание на уровне регистровых передач, часто называют С-синтезом, или

синтезом системного уровня. На рис. 4.2 приведены обсуждаемые здесь уровни абстракции описания цифровой системы.


*Рис. 1.1. Уровни абстракции описания цифровой системы*

Рис. 4.2. Уровни абстракции описания цифровой системы

На сегодняшний день в цикле работ по созданию BcC все большее значение приобретает этап высокоуровневого проектирования (High Level Design, HLD). На рис. 4.3 показаны подуровни системного уровня представления/проектирования вычислительных систем традиционной Y-диаграммы. Системный уровень, который представляется обычно единым верхним уровнем, в настоящее время разделяется на три части:

спецификация;

архитектура (макроархитектура);

микроархитектура.


Рис. 4.3. Y – модель

Y-модель позволяет наглядно представить уровни абстракции для аппаратуры ВС (Рис. 4.3). Две верхние оси отображают поведенческий и структурный аспекты представления системы,

нижняя – её физическую реализацию. С увеличением расстояния от начала оси увеличивается уровень абстракции. Фактически, крайние точки, изображённые на рис. 4.3, и представляют собой системный уровень представления ВС.

Процесс проектирования, отображённый на Y-модель при помощи перемещений между осями, представлен на Рис. 36. Стрелки слева направо соответствуют задачам синтеза на различных уровнях абстракции, стрелки справа налево – представляют собой анализ полученных решений. Далее, стрелки сверху вниз – генерация из описаний аппаратуры конкретных реализаций, стрелки снизу-вверх – извлечение описаний из существующих реализаций. стрелки вдоль осей показывают увеличение уровня абстракции или детализацию, закруглённая стрелка – символизирует оптимизацию полученного структурного представления без изменения уровня абстракции.

В Y-модели не представлен временной аспект, аспект коммуникаций между объектами и абстракции данных. Кроме того, хотя такое представление можно расширить и на представление ПО, но в явном виде оно в модели не представлено. Более точной, пусть и несколько громоздкой, является Rugby-модель. Она изображает идеализированный процесс проектирование системы от некоторой абстрактной идеи до реализации через повышение абстракций ортогональных аспектов:

- вычислений,
- коммуникаций,
- данных и временного аспекта.

SLD (System Level Design) — это работа с абстрактными описаниями некоторой ВС, предполагающая их создание, анализ, модификацию и дальнейшее использование в процессе проектирования. Т.е., недостаточно просто создать некоторую модель будущей системы: это должна быть достаточно детальная и

одновременно не очень громоздкая модель, отражающая именно те свойства, которые важны на системном уровне. Кроме того, это должна быть модель оптимального для данной задачи решения, действительно эффективного и адекватного имеющимся ресурсам. И, наконец, при проектировании должны присутствовать понимание и инфраструктура, позволяющие связать высокоуровневое описание с процессом реализации и с оценкой промежуточных результатов, т.е., процесс проектирования должен завершиться получением именно предсказанного результата — что, собственно, является также задачей высокоуровневого проектирования.


*Рис. 36. Y-модель* [30].

Рис. 4.4. Y - модель

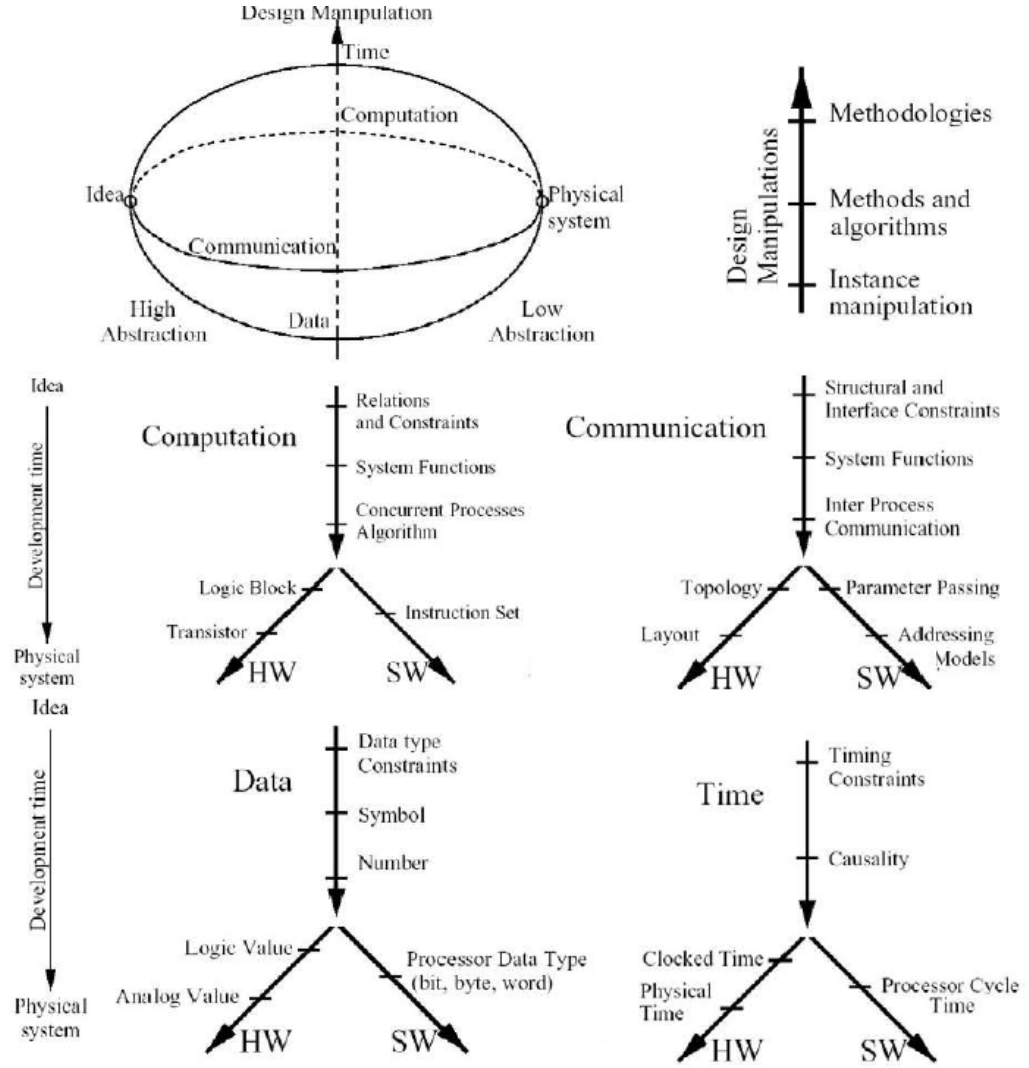



Рис. 4.5. Rugby-модель

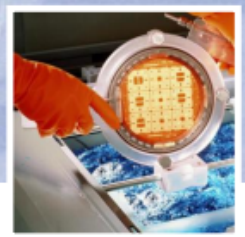
Уровень исходной спецификации предполагает создание, в первую очередь, поведенческого представления ВсС непосредственно на основе технического задания. Уровень макроархитектуры содержит общее представление об устройстве ВсС, охватывая все ее компоненты, независимо от того, как они в дальнейшем будут реализованы. Уровень микроархитектуры должен содержать информацию о концептуальном устройстве отдельных вычислителей, входящих в состав ВсС, независимо от способа реализации вычислителя (например, в виде кремниевого процессора или виртуальной машины). На практике сегодня большинство разработчиков ВсС либо вообще не использует деление на макро- и микроархитектуру, либо если микроархитектуру выделяют, то связывают ее исключительно с аппаратно реализованными процессорами (процессорными ядрами). Перечислим основные задачи этапа высокоуровневого проектирования ВсС:

- концепция решения целевой задачи, исходные спецификации;

- организация вычислительного процесса (модели вычислений — MoC);
- организация инфраструктуры проекта;
- архитектура BcC, ее оценка и верификация;
- выходные спецификации для этапа реализации.

Важную роль в данном отношении играет понятие модели вычислений (Model of Computation), которое позволяет создавать абстрактное представление BcC и выполнять комплексное и поэлементное моделирование. Инфраструктура проекта, которая определяет маршруты проектирования, инструментальные средства и многое другое, в случае BcC часто по-прежнему организуется в рамках ограниченного числа шаблонов. Это препятствует анализу разработчиками многих потенциально возможных архитектурных решений, снижая эффективность проектирования. Одна из важных проблем, которая пока продолжает усугубляться на фоне процессов ускорения проектирования BcC, состоит в отсутствии эффективных

технологий и инструментов сквозного цикла проектирования. Наиболее ощутимо она проявляется именно на высокоуровневых этапах проектирования (архитектурное, функциональное, логическое проектирование). Существует наличие семантического разрыва между архитектурными описаниями и их восприятием исполнителями, выполняющими последующие этапы разработки.



# Семантический разрыв HLD/MLD

| Проект | Год начала/ длит. проекта | Людей в проекте | Примерное количество строк кода (C,C++,VHDL,...), тыс. строк | Проблемы | Стоимость ошибки семантического разрыва HLD/MLD, % от общего времени проектирования |
|---|---|---|---|---|---|
| КТС ТРАКТ | 1996/5 | 20 | 350 | Непонимание рядовых исполнителей | 35 |
| КТЖ-2 | 2000/2 | 8 | 85 | Ошибки в архитектуре сети, непонимание рядовых исполнителей | 50 |
| АР3000 | 2002/2 | 10 | 130 | Сложности в понимании при передаче проекта в другую фирму | 35 |
| СУНО-ЛУЧ-1 | 2003/3 | 9 | 75 | Непонимание рядовых исполнителей, высокая сложность проекта | 75 |
| СУМЭ | 2006/2 | 8 | 187 | Сложности с передачей проекта в другую рабочую группу | 50 |
| MiniLab | 2008/2 | 10 | 50 | Непонимание рядовых исполнителей | 50 |

Этот разрыв обусловлен неадекватностью языка представления вычислительной архитектуры, недостаточно отработанной системой архитектурных вычислительных абстракций, которые должны использовать разработчики всех уровней ВсС, и естественно вытекающим из этого отсутствием инструментальных средств, поддерживающих такого рода абстракции. Безусловно определённые абстракции имеются, они используются разработчиками, но эти абстракции не увязаны друг с другом, т.к. де-факто архитектор, аппаратчик и программист в области ВсС разговаривают на разных языках. Потери рабочего времени проектировщиков из-за семантического разрыва на маршрутах проектирования ВсС средней сложности по нашим оценкам составляют до 50%. Анализ проводился на выборке 10 проектов со следующими характеристиками: разработка ВсС на основе распределённых микропроцессорных архитектур со специальными механизмами повышения производительности и/или надежности; примерное

количество строк кода проекта на C, C++, Verilog и др. языках — 50—350 тыс.; число основных исполнителей — 10—30 чел.). Типичные ошибки:

а) непонимание локальной задачи рядовыми исполнителями;

б) ошибки в архитектуре системы;

в) сложности в понимании при передаче проекта в другую рабочую группу или фирму.

В таблице 1 приведены инструментальные средства архитектурного и системного проектирования. Видно, что индустриальные средства, которые позволяют эффективно работать на системном уровне проектирования BcC, на сегодняшний день практически отсутствуют. Существует определённое число академических проектов исследовательского характера, которые развиваются в первую очередь в университетах. Эти средства, по отзывам не только представителей ведущих фирм по созданию инструментальных средств для индустрии, но и самих разработчиков

BcC, пока довольно далеки от эффективного коммерческого использования.

Таблица 1

Таблица 1. Инструментальные средства этапа высокоуровневого проектирования BcC

| Разработчик | Инструмент | Назначение | Абстракции | Категория |
|---|---|---|---|---|
| Компания Mentor Graphics | SystemVision | Моделирование смешанного сигнала и высокоуровневое моделирование | VHDL-AMS, Spice, C | Промышленность |
| Компания National Instruments | LabView | Разработка контрольно-измерительных и испытательных приложений | Язык программирования LabView | Промышленность |
| Университет Калифорнии (Беркли) | Ptolemy II | Моделирование и разработка распределенных встраиваемых систем реального времени | Все модели вычислений | Научная |
| Университет Канзаса | Rosetta | Объединение неоднородных спецификаций в единую декларативную семантическую среду | Все модели вычислений | Языки |
| Компания Mentor Graphics | Nucleus | Семейство ОС реального времени и средства разработки | Программное обеспечение | Промышленность |
| Организация Open SystemC Initiative | SystemC | Обеспечение аппаратно-ориентированных структур в рамках контекста C++ | Уровень транзакций в RTL | Языки |
| Консорциум Object Management Group | Unified Modeling Language | Спецификация, визуализация и документирование моделей программных систем | Объектно-ориентированные диаграммы | Языки |
| Компания Cadence | Incisive | Интегрированная инструментальная платформа верификации, включая симуляцию, формальные методы и эмуляцию | RTL и утверждения SystemC | Промышленность |
| Компания Synopsys | System Studio | Алгоритмическое и архитектурное проектирование, оценка производительности | SystemC | Промышленность |
| Университет Калифорнии (Беркли) | Metropolis | Функциональное и архитектурное проектирование, отображение, детализация и верификация | Все модели вычислений | Научная |
| Университет Калифорнии (Беркли) | Mescal | Программирование специализированных платформ | Расширенный комплекс моделирования Ptolemy II, сетевые процессоры | Научная |

Исследования в области совершенствования технологий высокоуровневого проектирования BcC в мире активно развиваются. Разработчики рассматривают в качестве средства решения многих проблем в проектировании BcC реализацию в методиках и инструментах следующих принципов:

- унификация взгляда (представления, использования и т.д.) на hardware/software (HW/SW);

- формирование согласованной системы высокоуровневых (архитектурных) абстракций для описания (представления) ВсС;

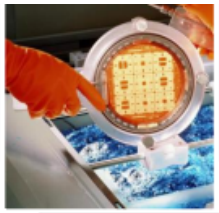
- широкое использование принципов платформно-ориентированного проектирования, в т.ч. на уровне ключевых решений в проекте;

- выделение функциональных и нефункциональных требований технического задания в аспекты процесса проектирования; - динамический баланс «альтернатив»: проектирование/реализация, HW/SW, design/run time и др.

Инструментами реализации вычислительного процесса могут выступать различные вычислительные абстракции, часть из которых перечислена ниже:

- концепции проектирования (платформно-ориентированное, модельно-ориентированное, акторно-ориентированное и др.);

- модели вычислений;
- виртуальные машины;
- платформы (вычислительные, инструментальные, протоколов взаимодействия и т.д);
- механизмы (технические решения из различных областей-аспектов в абстрактном представлении);
- элементная база (как множество реализаций механизмов).



# Инструментальные средства HLD

**A Platform-Based Taxonomy for ESL Design.**
A.S.-Vincentelli, University of California, Berkeley

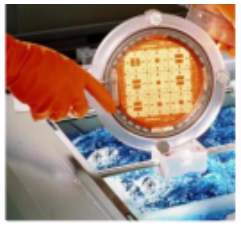
| Provider | Tools | Focus | Abstraction | Category |
|---|---|---|---|---|
| Mentor Graphics | SystemVision | Mixed-signal and high-level simulation | VHDL-AMS, Spice, C | F: Industrial |
| National Instruments | LabView | Test, measurement, and control application development | LabView Programming language | F: Industrial |
| Univ. of California Berkeley | Ptolemy II | Modeling, simulation, and design of concurrent, real-time, embedded systems | All MoCs | F: Academic. |
| Univ. of Kansas | Rosetta | Compose heterogeneous specifications in a single declarative semantic environment | All MoCs | F: Languages. |
| Mentor Graphics | Nucleus | Family of real-time operating systems and development tools | Software | P: Industrial |
| Open SystemC Initiative | SystemC | Provide hardware-oriented constructs within the context of C++ | Transaction level to RTL | FP: Languages. |
| Object Management Group | Unified Modeling Language | Specify, visualize and document software system models | Object-oriented diagrams | FP: Languages. |
| Cadence | Incisive | Integrated tool platform for verification, including simulation, formal methods, and emulation | RTL and SystemC assertions | PM: Industrial |
| Synopsys | System Studio | Algorithm and architecture capture, performance evaluation | SystemC | FPM: Industrial |
| Univ. of California, Berkeley | Metropolis | Operational and denotational functionality and architecture capture, mapping, refinement, and verification | All MoCs | FPM: Academic. |
| Univ. of California, Berkeley | Mescal | Programming of application-specific programmable platforms | Extended Ptolemy II, network processors | FPM: Academic. |

- **F – Functionality, P – Platform, M - Mapping**

10

В рамках данного подхода организация вычислительного процесса, определенного в техническом задании на ВсС,

рассматривается как инструмент реализации целевой встраиваемой системы. На рис. 4.6 показано место и значение архитектурной платформы в процессе проектирования ВсС, и перечислены основные элементы объектно-событийной модели вычислений РИУС (ОСМВ), которая разработана в ходе наших исследований.



# Место и значение архитектурной платформы и Объектно-событийная модель вычислений РИУС (ОСМВ) в процессе проектирования ВсС

**Архитектурная платформа:**

- Перечень аспектов проектирования
- Модель (модели) вычислений
- Внешняя логика взаимовлияния аспектов (критерии проектирования)
- Перечень зафиксированных шаблонов повторного использования
- Элементная база

**ОСМВ:**

- Объект, функциональный блок (ФБ)
- Событие
- Порт
- Узел

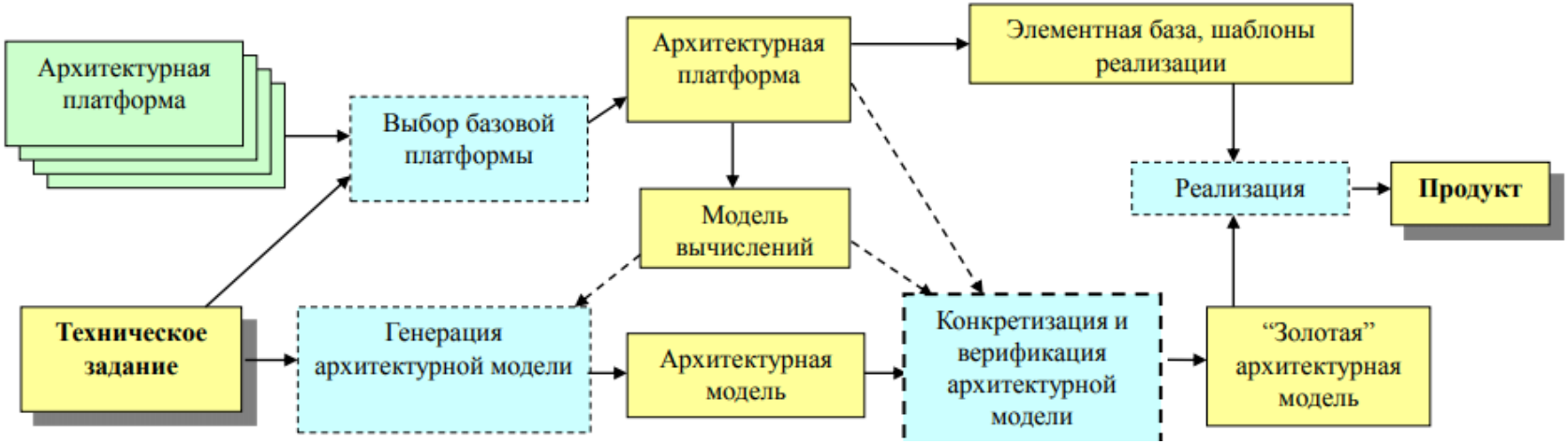



Рис. 4.6.

## 4.4. Платформы проектирования в вычислительной технике

Многообразие платформ, применяемых в вычислительной технике, представлено на рис. 4.7. Под проектной платформой понимается уровень абстракции в маршруте проектирования, который скрывает несущественные на данном этапе детали множества возможных способов реализации.

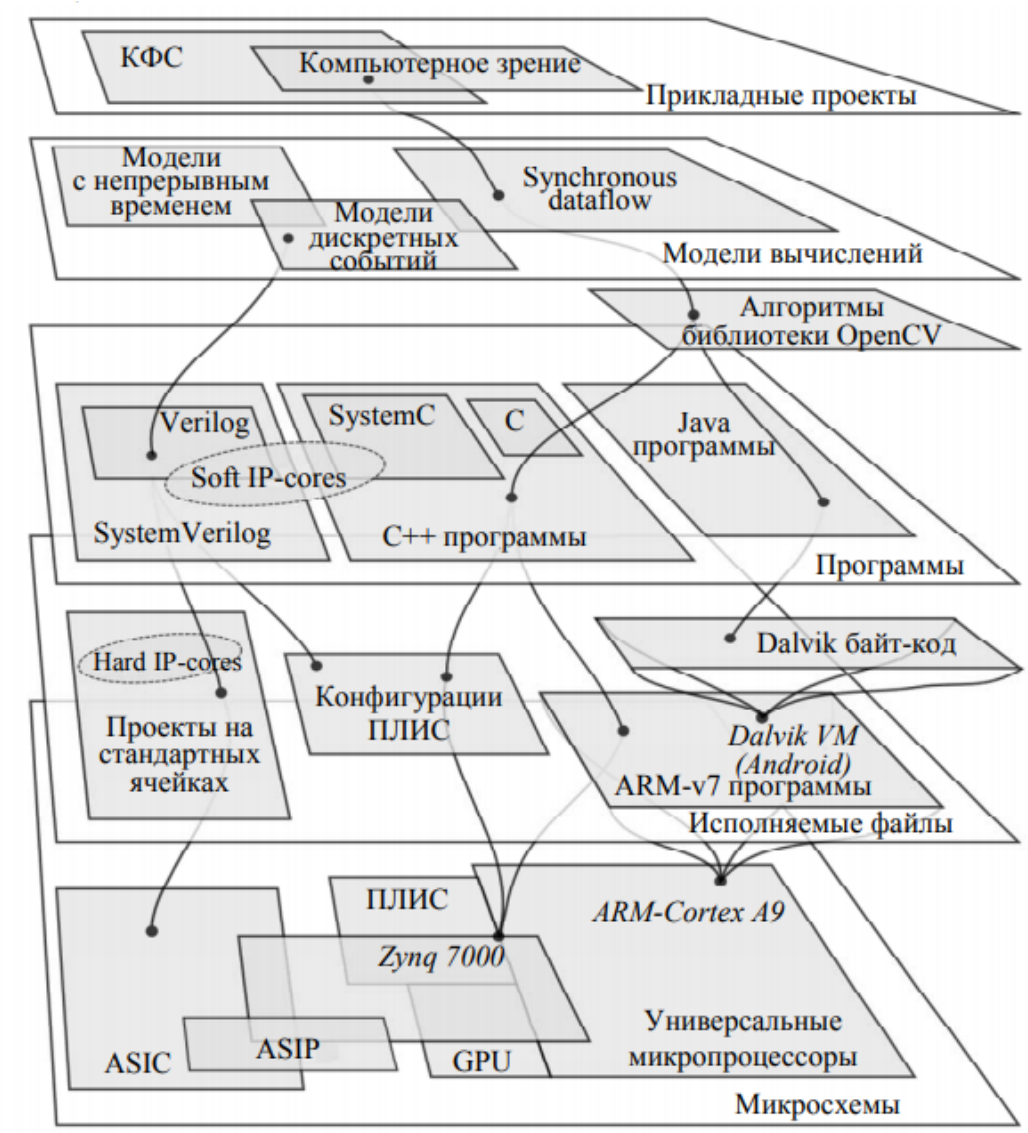



Рис. 4.7. Платформы вычислительной техники

Процесс проектирования на основе платформ, называемый в литературе платформоориентированным проектированием (platform-based design, PBD), заключается в поэтапном Проектирование вычислительных платформ киберфизических систем  уточнении высокоуровневой начальной спецификации до готового продукта с использованием платформы на каждом этапе проектирования. Это позволяет повысить эффективность проектирования за счет абстрагирования и повторного использования. Анализ работы показывает, что почти каждая абстракция (платформа), применяемая для преодоления сложности в проектировании вычислительной техники, скрывает важные для ВсС и КФС характеристики физического мира.

ы Существует многообразие вариантов, технологий и платформ, используемых при создании ВсС. Владеть ими всеми на достаточном уровне невозможно, поэтому предлагается большое количество шаблонов, стереотипов, стандартов де-факто и настоящих

отраслевых стандартов, нацеливающих на использование конкретных решений в конкретных случаях (или предписывающих их применение). Как было отмечено, назначение ВсС — решение конечной (прикладной) задачи средствами ВТ. Это означает, что основу проектирования составляет вопрос организации целевого вычислительного процесса аппаратными и программными средствами. Многообразие вариантов реализации вытекает из многоуровневой организации ВсС — своеобразного «слоеного пирога».

Его слои частично соответствуют физической организации ВсС, но в большей степени такое деление определяется логической организацией, архитектурой системы. Одни и те же функции потенциально могут быть реализованы на любом уровне «пирога». На практике же выбор уровня должен быть компромиссом между стоимостью и получаемыми характеристиками. Этот выбор является одной из центральных проблем проектирования ВсС. На практике он

обычно сводится к вопросу распределения функциональности между программной и аппаратной составляющими. В плане архитектуры сегодняшние ВсС достаточно разнообразны. Они обладают различными по принципу работы процессорами, число которых составляет от одного до многих тысяч, причём иногда различных типов. Пространственно ВсС могут быть сосредоточенными или распределенными (сети компьютеров, контроллеров, сети на кристалле). В них могут использоваться различные операционные системы (ОС), средства виртуализации вычислений, стеки протоколов сетевого взаимодействия. Вычисления могут строиться на базе большого числа разнообразных моделей вычислений, которые составляют основу языков программирования (например C, Java, Prolog, Haskell, IEC 61131-3) и проектирования (например Verilog, SystemC, AHDL, UML.

. Выполняя требования технического задания на ВсС, разработчики могут в качестве основы использовать

«вычислительные полуфабрикаты» (будем называть их платформами), которые потребуют различных затрат на проектирование, так как предоставят существенно разные уровни «развитости» или «завершенности» ВС, от которых надо будет строить «вверх». Например, аппаратная платформа ПК позволяет решать прикладную задачу программным путем с использованием в качестве вариантов стандартную ОС, ОС реального времени (ОСРВ), программирование «на голом железе». Создаваемые системы будут существенно отличаться по реактивности (реакции на события реального времени), надежности и предсказуемости работы, требуемой квалификации разработчиков, стоимости работ. Анализ уровней «погружения» в вычислительную организацию ВС в рамках проекта ВсС показывает, что на практике выполняются работы на всех уровнях, причем в различных сочетаниях. Ниже перечислены основные категории таких проектных работ: выбор или разработка прикладного ПО; выбор или разработка системного ПО;

определение состава периферийных модулей (УСО) из готового набора; разработка УСО; определение состава центральных вычислительных ресурсов ВС (процессоров, памяти, интерфейсов); разработка центральных вычислительных ресурсов ВС на готовой компонентной базе; разработка компонентной базы. Проектная платформа, как решение архитектурного уровня, в традиционных технологиях создания ВсС определяет практически весь маршрут проектирования и разработки. Поэтому именно она может выступать в качестве классификационного признака технологий создания ВсС и определять уровень «погружения» в вычислительную иерархию, ожидающий разработчика. Технологии решения вычислительной задачи и платформы, которые фиксируют в значительной степени эти технологии, следует рассматривать в качестве основных шаблонов проектирования ВсС. Наиболее широко используемыми сегодня в индустрии являются следующие платформы:

промышленные ПК;

программируемые логические контроллеры (ПЛК, PLC) и программируемые контроллеры автоматизации (ПАК, PAC);

мобильные и интернет-устрой ства (смартфоны и планшеты);

контроллерные (Fieldbus) и сенсорные сети;

микроконтроллеры; сигнальные процессоры (DSP);

программируемая логика — ПЛИС (PLD, FPGA);

заказные СБИС (ASIC, ASIP, SoC, Network on Chip – NoC).

В основу классификации положен нижний уровень ресурсов/сервисов/элементов ВсС, который закрепляет тип основного вычислительного элемента в проекте. Платформы промышленных ПК и ПЛК/ПАК позволяют относительно просто и быстро создать прикладную систему, однако эта эффективность проявляется только в рамках типовых технических заданий. Проектные платформы микроконтроллеров и сигнальных процессоров предоставляют больше свободы разработчику. Они имеют свою специфику, в первую очередь в организации системного

ПО и в степени открытости архитектуры. В качестве платформ с успехом используются ПЛИС, сочетающие гибкость программных и аппаратных средств. Существуют проекты, включающие создание специальной компонентной базы, в первую очередь SoC, ASIP, ASIC. Особое положение занимают платформы мобильных и интернет-устройств, которые начинают активно использоваться в качестве мобильных терминалов ВсС, в том числе со SCADA «на борту», и сетевые контроллерные платформы, которые выступают системообразующими решениями в ВсС с распределенной организацией. При движении по списку проектных платформ/шаблонов вниз в целом растет достижимая оптимальность проектных решений и суммарная сложность проектирования. В таблице приведены примеры фирм-производителей, сравнительные данные по областям применения, типовые характеристики и требуемые компетенции проектирования ВсС для каждой из перечисленных выше проектных платформ.

ТАБЛИЦА. ОСНОВНЫЕ ПРОЕКТНЫЕ ПЛАТФОРМЫ ВСТРАИВАЕМЫХ СИСТЕМ

| Тип платформы | Производители | | Примеры продуктов | Область применения | Тактовая частота | Объемы памяти | Система ввода-вывода |
|---|---|---|---|---|---|---|---|
| | Отечественные | Иностранные | | | | | |
| Промышленные ПК | Fastwel, «ГРАНИТ-ВТ», «МЦСТ» | Advantech, Panasonic, AMP, Siemens | SIMATIC Box PC, IS-1U-SYS6, FRONT Station 432.87 | Промышленность (SCADA, управление ПЛК), пользовательские терминалы, связь | Соответствует производительности офисных ПК. Сотни МГц, единицы ГГц | От сотен Мбайт до единиц Гбайт | HMI, расширение УСО посредством PCI, USB |
| ПЛК, ПАК | Fastwel, ОВЕН, ЛМТ, Segnetics | Siemens, Mitsubishi Electric, Mean Well, VIPA | System 300S, ADAM-5000, Simatic S7-300, APAX-5520CE | Автоматизация технологических процессов (промышленность, энергетика, транспорт, ЖКХ) | Сотни МГц | От десятков кбайт до сотен Мбайт | Дискретные, аналоговые, релейные, цифровые |
| Мобильные и интернет-устройства | teXet, «Код безопасности» | Nokia, Samsung, Apple, HTC, Getac, Panasonic, Advantech | PWS-8033M, iPad, Континент Т-10, Nokia Lumia 900, Advantech P37B | Пользовательские интерфейсы: промышленность, военные, транспорт, спасательные службы, экология | От сотен МГц до единиц ГГц | От сотен Мбайт до единиц Гбайт | Сенсорные HMI, беспроводные сетевые и периферийные интерфейсы |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| МК | «Мета», «Мультиклет», «МЦСТ», «ЭЛВИС» | NXP Semiconductors, Atmel, Microchip, Freescale, STMicroelectronics | LPC2000, Atmel AVR PIC-32, Coldfire, STM32 | Любые электронные изделия малой и средней серийности | От десятков кГц до единиц ГГц | От десятков кбайт до сотен Мбайт | Интерфейсы «машина–машина», примитивные HMI, дискретные, аналоговые, релейные |
| Сигнальные процессоры | «ЭЛВИС», НТЦ «Модуль» | Analog Devices, Freescale, Texas Instruments | SigmaDSP, ADSP-21xx, Blackfin, StarCore, DSP56K, TMS320, KeyStone, DaVinci, Л1879ВМ1 (NM6403), 1892ВМ2Я | Обработка звука, видео, связь, радиолокация, сонары | От сотен кГц до единиц ГГц | От десятков кбайт до сотен Мбайт | Цифровые интерфейсы |
| ПЛИС | – | Altera, Xilinx, Actel, Lattice | Stratix V, Cyclone V, Virtex-6, Zynq-7000, Spartan-6, RTAX-S, Axcelerator, IGLOO, LatticeECP3 | Мелкосерийные изделия, приборы, прототипирование | До десятков ГГц | До сотен Мбайт | Цифровые интерфейсы (определяется техническим заданием) |
| Заказные СБИС | «ЭЛВИС», «МЦСТ», НТЦ «Модуль», «Ангстрем», НИИМЭ, «Микрон» | Broadcom, Qualcomm, Realtek, Rockchip, Allwinner, Lucent, Freescale | – | Любые крупносерийные электронные изделия: контроллер ввода/вывода, процессор, RAM, SoC | Десятки ГГц | От единиц байт до единиц Гбайт | Определяется задан |

## 4.5. Проектирование электронных устройств в САПР Cadence

В настоящее время Cadence предоставляет широкий набор программных средств для проектирования современных электронных устройств от интегральных схем и ПЛИС до систем на кристалле (СНК) и Интернета вещей.

На рис. 4.8 показаны этапы проектирования систем на кристалле, включающие:

Системное проектирование;

Аппаратное проектирование;

Проектирование топологии интегральной схемы (ИС);

Проектирование корпуса ИС;

Проектирование печатной платы;

Разработка программных средств;

Отладка и тестирование системы.


Рис. 4.8. Этапы проектирования СНК

На рис. 4.9.. показан маршрут проектирования сложной электронной системы. Сначала проводится системное проектирование на языках C++ и SystemC с использованием библиотек, стандартов, сложных заказных готовых IP (Intellegence Properties) блоков. Затем проводят одновременно аппаратное и программное проектирование цифровых и смешанных аналогово-цифровых и заказных блоков с использованием языков Verilog, VHDL, AMS, выполняют логический синтез FPGA (field-programmable gate array - программируемая логическая интегральная схема (ПЛИС)) и ASIC (application specific integrated circuit – интегральная схема специального назначения). Проводят физическое прототипирование, создают библиотеку производителя, проектируют топологию, выполняют верификацию топологии с возвратом для уточнения на системное прототипирование, эмуляцию и

системное программирование. Такой замкнутый цикл может повторяться многократно, пока не будут достигнуты надежные требуемые параметры устройства. Только после этого проект передают в производство, корпусирование и разработку печатной платы.

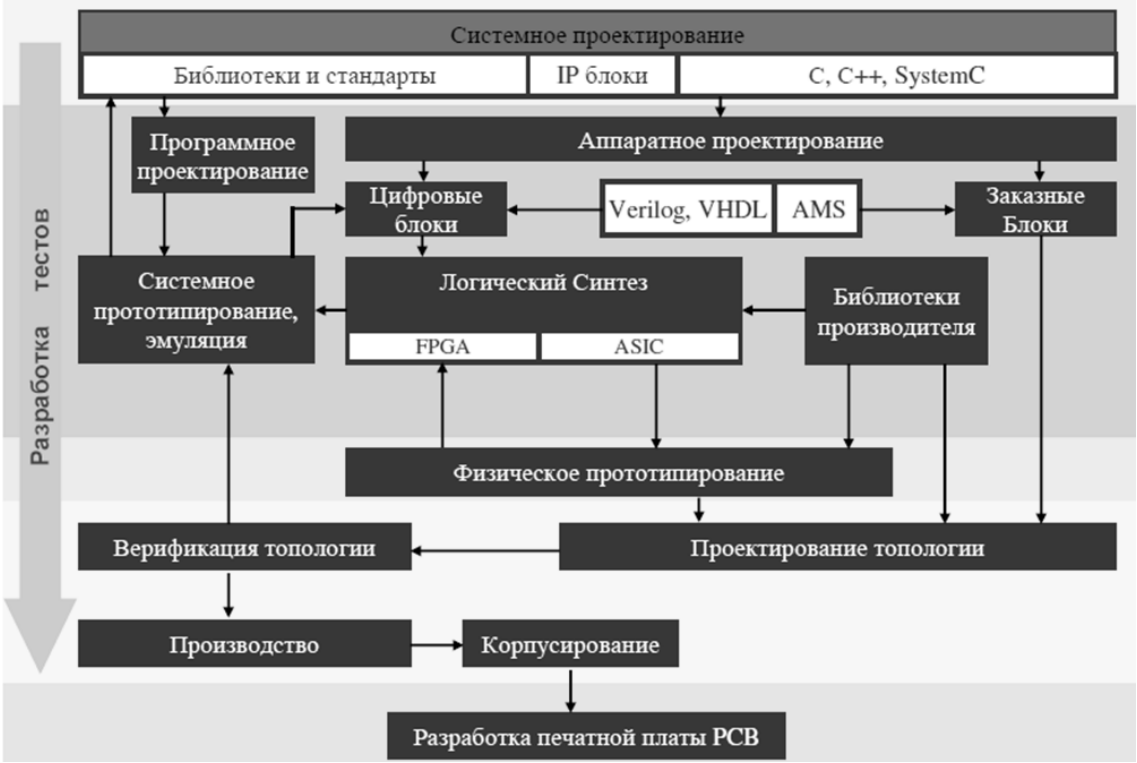



Рис. 4.9. Маршрут проектирования Cadence

Чтобы научиться этому, надо начать с простых электронных устройств и изучить OrCAD 17.2.

### 4.5.1. Шаблоны проектирования BcC

Выделение шаблонов, некоторых элементарных действий и шагов позволяет значительно облегчить труд человека (в идеале — за счёт создания средств автоматизации проектирования (САПР), полностью снимающих с проектировщика отдельные задачи).

Появление САПР является прямым и наиболее очевидным следствием формализации процесса проектирования. САПР призваны снизить долю рутинной работы и уменьшить количество ошибок, допускаемых из-за присутствия в процессе проектирования человеческого фактора. Все это помогает сделать соответствующую технологию доступной для использования в индустрии, так как при этом увеличивается вероятность получения продукта заданного качества. В области проектирования встраиваемых систем

поддержку со стороны САПР можно увидеть только в отдельных областях, в основном – для создания СнК и печатных плат.

Примеры формализации маршрута проектирования и выделения из него шаблонов приведены на Рис. 4.10, Рис. 4.11.

Аббревиатуры ФТ, НФТ обозначают функциональные и нефункциональные требования, ЯОФ /ТЯОФ – языки описания функциональности / трансляторы ЯОФ, ВПл – вычислительная платформа.

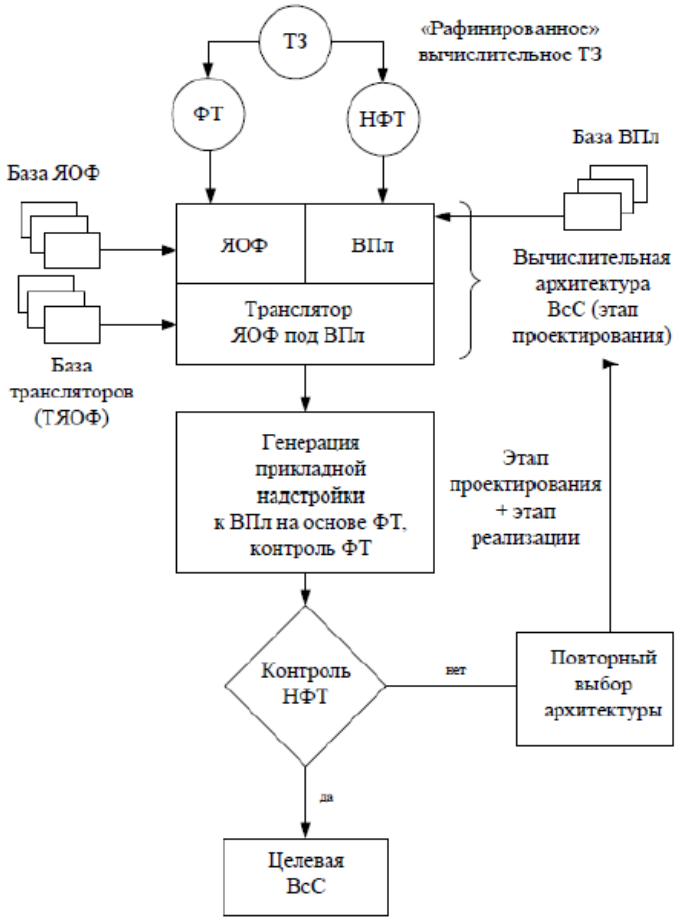



*Рис. 38. Шаблоны проектирования ВсС на основе традиционной схемы проектирования [61].*

# Рис. 4.10. Шаблоны проектирования ВсС на основе традиционной схемы проектировнаия

SLD влечёт за собой большие затраты на этапе анализа и предварительного проектирования. Но они окупаются ввиду роста качества конечного продукта, снижения рисков. В первую очередь, конечно, это справедливо для сложных комплексных систем — однако практика показывает, что подход оправдан также и для простых задач.

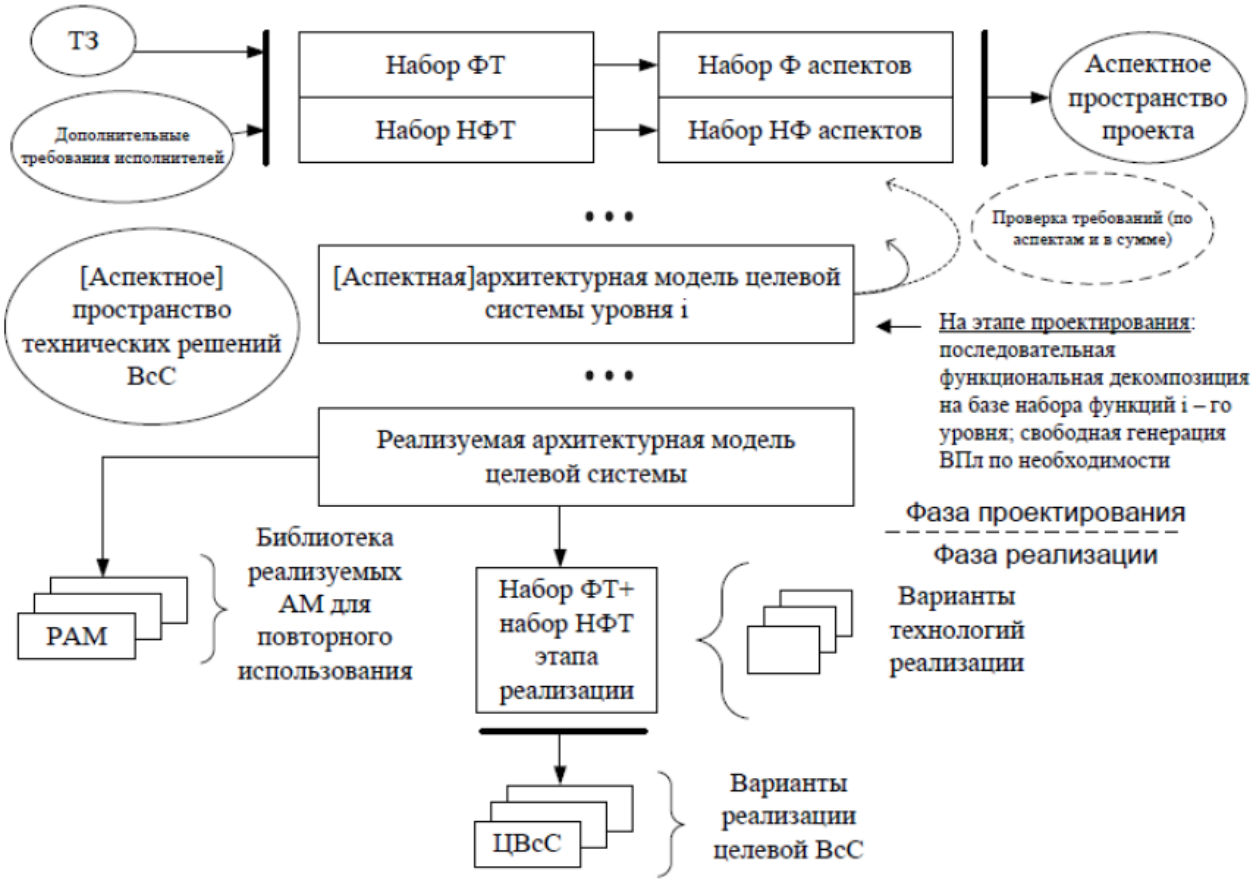



*Рис. 39. Шаблоны проектирования ВсС на базе аспектной модели и композиции ВПл [61].*

Рис. 4.11. Шаблоны проектирования ВсС на базе аспектной модели и композиции ВПл.

Проектирование «сверху вниз» позволяет снизить возникающие в процессе проектирования риски, в первую очередь интеграционные. Иначе говоря, уменьшается вероятность того, что на конечном этапе проектирования обнаруживается невозможность или сложность взаимодействия некоторых составляющих компонентов системы, а также вероятность появления критичных ошибок и «узких мест» (пример – недостаточная производительность канала передачи данных).

### 4.5.2. Архитектурные абстракции

Использование абстракций является одним из основных методов работы со сложными, комплексными объектами. Это обусловлено особенностью нашего мышления — ограничением на количество объектов/понятий/сущностей, с которыми мозг может одновременно работать (по разным оценкам, это число в среднем равно четырём-семи).

Поэтому способность быстро вычленять суть, значимую для решения конкретной задачи, из общего массива информации – является одной из важнейших составляющих успешного решения любой проблемы.

Взгляд на вычислительную технику и инфокоммуникационные технологии (ВТ&ИКТ) с «абстрактных» высот позволяет:

- определять (ограничивать снизу и сверху) область деятельности – зону ответственности ВТ&ИКТ-специалиста;
- делать «прозрачным» представление того, что называют «технологиями» во всех областях ВТ&ИКТ;
- взвешенно распределять влияние реализации на принцип решения задачи в ВТ&ИКТ – областях;
- позволять аккумулировать и свободно (осознанно и без «шор») развивать концептуальные технические решения,
- менять/расширять уровень технологий повторного использования;

- выявлять (проявлять) «болевые точки» в организации и проектировании ВТ&ИКТ – систем, следовательно, обеспечивать грамотную постановку задач.

Также, в контексте проектирования ВсС и СнК множество базовых архитектурных абстракций можно свести в следующие четыре группы:

1) базовые элементы ВС (вычислительный механизм, вычислительная платформа, архитектурный агрегат);

2) абстракции представления ВС на системном уровне (архитектура, архитектурная платформа, архитектурная модель, аспект);

3) абстракции процесса проектирования (проектирование ВсС, инфраструктура проекта, проектное пространство, аспектное пространство);

4) понятия для анализа и оценки существующих архитектурных решений (вычислительный процесс, виртуальная машина, модель вычислений).

Часть из перечисленных абстракций имеют ярко выраженную «вычислительную» специфику, их будем относить к категории вычислительных.

Следующие абстракции будем относить к категории невычислительных: архитектурный агрегат, аспект, проектирование ВсС, инфраструктура проекта, проектное пространство, аспектное пространство.

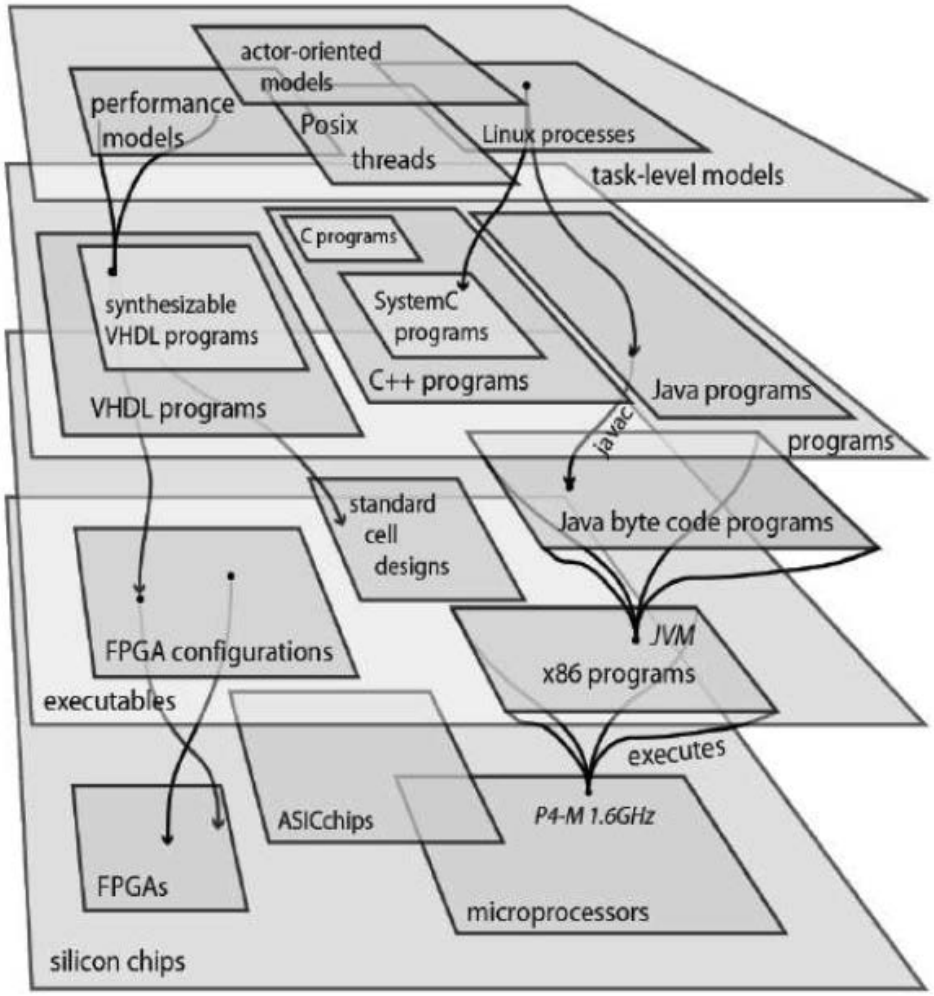



*Рис. 40. Многоуровневая организация BcC [34].*

Рис. 4.12. Многоуровневая модель BcC

Переход от высокоуровневого абстрактного представления вычислительного процесса к уровню физической реализации так называемого "не вычислительного базиса" (границей можно считать вентильный или транзисторный уровень) в силу сложности задачи требует большого числа промежуточных уровней и представлений. Эти уровни необходимы, прежде всего, разработчику для борьбы со сложностью задачи. На практике чаще всего выделение и реализация таких уровней в вычислительной системе выполняется различными коллективами разработчиков и на разных этапах проектирования. Причем в большинстве проектов значительная (основная) часть работы присутствует в повторно используемых компонентах, таких как процессоры, операционные системы, различные API, коммуникационные протоколы, трансляторы и т.д.

в проектное пространство ВсВвходят объекты или элементы, участвующие в процессе проектирования системы, которые позволяют описать, структурировать, зафиксировать

функциональность ВсС. Примерами таких абстракций выступают понятия вычислительной архитектуры, вычислителя, интерфейса, платформы, процесса, вычислительного механизма, виртуальной машины, программируемого интерпретатора, модели вычислений и другие.

### 4.5.3. Модели вычислений

Ключевой задачей в процессе проектирования является обоснованный выбор той или иной модели целевой системы. Рассмотрение системы с точки зрения той или иной модели автоматически привнесёт в систему свойства, присущие выбранной модели. При рассмотрении моделей ВсС, обычно говорят о так называемых “моделях вычислений” (МоС).

Некоторые направления исследований в области технологий проектирования встраиваемых систем обращают особое внимание на использование в проектировании этого понятия. При представлении

вычислительной системы как иерархии виртуальных машин каждому уровню такой модели соответствует своя МоС.

***МоС можно интуитивно представить, как набор правил, необходимых для построения вычислительного процесса системы.*** Это парадигма, описывающая протекание вычислительного процесса, способы обмена данными, взаимодействия между отдельными функциональными элементами. Кроме того, МоС предлагает терминологию и примитивы, в базисе которых требуется выражать и описывать целевую систему. МоС описывает природу потоков данных, элементов синхронизации, роль времени в процессе выполнения системой целевой функции. Различные МоС по-разному описывают одни и те же процессы, протекающие в целевой системе. Для больших и сложных систем совершенно нормальное положение дел, когда различные части системы представляются различными МоС.

Примерами МоС являются:

- модель с дискретными событиями,
- сеть обработки потоков данных,
- взаимодействующие конечные автоматы,
- синхронная модель вычислений,
- объектно-событийная модель вычислений
- и денотативно-объектная модель вычислений.

Модель вычислений может служить «точкой пересечения» понимания работы системы разными специалистами. Использование терминологии, предоставляемой моделью вычислений, также помогает коллективной разработке.

Проектирование ВсС с использованием понятия моделей вычислений представляется как проектирование вычислительного процесса.

Использование разных моделей вычислений позволяет математически точно описать определённые аспекты встроенной вычислительной системы.

Важной и неотъемлемой частью модели вычислений является язык МоС. С точки зрения языка его МоС представляет собой поведение некоторой абстрактной вычислительной машины (или просто абстрактного вычислителя) в рамках семантики языка. С этой точки зрения можно вести разговор о МоС традиционных языков программирования, таких как С, С++, Java, Pascal и т.д.

МоС системы на данном уровне абстракции, используя выразительные средства языка, описывает следующие аспекты системы:

1) Поведение вычислительных компонентов;

2) Взаимодействие вычислительных компонентов;

3) Способы передачи данных и синхронизацию вычислений;

4) Способы декомпозиции и агрегации вычислительных компонентов.

Кроме собственно описания целевой системы на заданном уровне абстракции, МоС должна обеспечивать разработчику средства

работы с этим описанием. Разработчик должен иметь возможность доказывать истинность или ложность определенных утверждений, относительно целевой системы, проверять соответствие определённым требованиям и ограничениям, накладываемым на целевую систему. Инструменты, предоставляемые моделью, должны позволять проводить оценку тех или иных характеристик целевой системы, проводить оптимизацию по выбранным параметрам. Все эти действия разработчика можно назвать моделированием целевой системы в терминах выбранной МоС.

### 4.5.4. Аспектное представление проекта

Комплексный характер проектов ВсС в сочетании с ростом их сложности требует создания методов и технологий проектирования, которые позволят эффективно учитывать, анализировать, синтезировать, отслеживать качество всех признанных существенными сторон организации ВсС и существующей вокруг нее инфраструктуры на протяжении всего жизненного цикла,

особенно на этапах создания и модификации. Выделение таких относительно самостоятельных сторон является процессом нетривиальным. *Такие локализованные стороны проекта или целевой системы называют аспектами.*

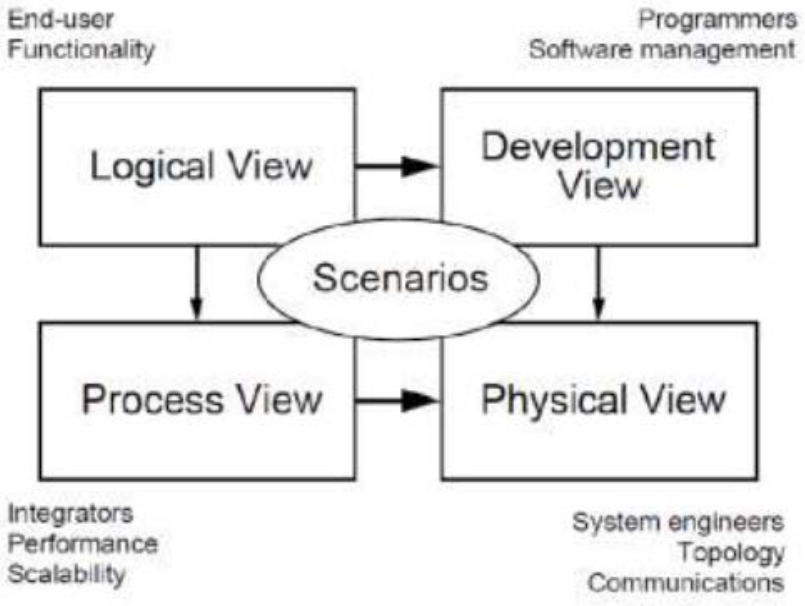



*Рис. 41. Аспектное представление проекта* [34].

Рис. 4.13. Аспектное представление проекта BcC

Аспект – это некоторая частная проблема проектирования в рамках задачи создания ВсС. Аспекты существуют не в рамках какого-либо этапа или шага развития проекта или целевой системы, а на протяжении всего процесса проектирования или всего жизненного цикла системы («вес» аспекта в проекте меняется во времени и может вырождаться до нуля).

Множество, включающее все аспекты проектирования, будем называть аспектным пространством процесса проектирования ВсС.

Множество, непосредственно принадлежащее проектируемой целевой системе, будем называть аспектным пространством целевой системы.

Список аспектов в проекте ВсС всегда конечный, но их общий перечень является открытым.

Типовыми и наиболее важными аспектами процесса проектирования ВсС можно считать:

- структурно-функциональный,

- конструктивно-технологический,
- энергетический,
- инструментальный,
- повторного использования,
- организационно-экономический,
- документный,
- надежностный,
- точностной.

Проработка аспекта в рамках проекта должна идти последовательно на всех стадиях и выражается в его специфицировании, проектировании, верификации, реализации и т.д.

Работа в рамках аспекта представляет собой мини-проект, который направлен на реализацию одного из свойств создаваемой системы.

Важнейшим фактором успеха проектирования ВсС является учет требований и ограничений, которые накладываются в явном и

неявном виде через ТЗ, а также вносятся самим коллективом проектировщиков, отражая его возможности, предпочтения, вторичные задачи и т.д.

### 4.5.5. Платформно-ориентированное проектирование

Платформно-ориентированное проектирование (platform-based design, PBD) – интенсивное повторное использование для ВсС готовых платформ, когда значительные части проекта основываются на предварительно разработанных и верифицированных компонентах, которое в жёстких рыночных условиях позволяет значительно повысить конкурентоспособность разработки. PBD – «ориентированный на интеграцию подход к проектированию сложных систем, при котором придаётся особое значение систематическому повторному использованию, и который основывается на заданных инструментальных платформах и виртуальных компонентах совместимых аппаратных средств и ПО с

целью снижения рисков разработки, стоимости и времени выхода на рынок».

Принципы PBD состоят из старта на самом высоком уровне абстракции, когда скрыты ненужные подробности реализации, суммирования важных параметров реализации в абстрактной модели, ограничения исследования пространства проектирования до набора доступных компонентов, и выполнения проектирования как последовательности детализирующих начальную абстракцию шагов, которые идут от начальной спецификации к конечной реализации, используя платформы на различных уровнях абстракции.

В концепции платформно-ориентированного системного проектирования основным понятием является платформа – некоторый компонент разрабатываемых систем, пригодный для повторного использования. Это могут быть платы или их части, драйверы, интерфейсы, операционные системы, API и т.п.). Аппаратная и программная платформы объединяются в так

называемую системную платформу. Функциональность – то, что требуется от разрабатываемого изделия – рассматривается отдельно от структурного представления системы. Процесс проектирования представляет собой поиск отображения для заданной функциональности на какой-то из существующих компонентов (может сравниваться несколько платформ и выбираться оптимальная).

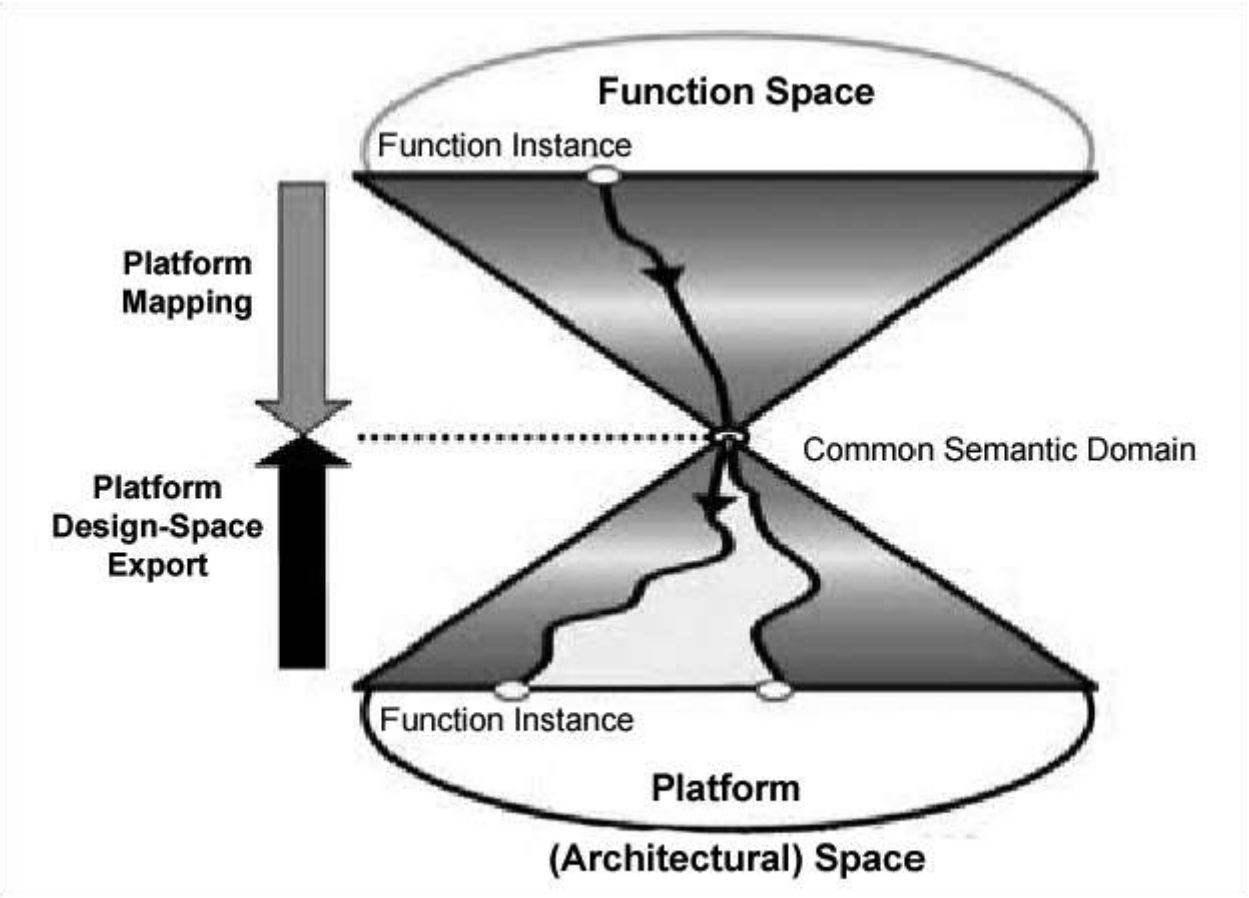



*Рис. 42. Платформно-ориентированное проектирование [4].*

Рис. 4.14. Платформно – ориентированное проектирование

Аппаратная платформа – это множество архитектур вычислительных машин, позволяющее решать поставленную задачу. При проектировании ограничения архитектуры обычно определяются в терминах производительности и размеров. Обычно в аппаратной платформе больше возможностей, чем требуется от проектируемой системы. Нет смысла использовать аппаратуру, пригодную для решения единственной задачи.

Программная платформа – абстрактный уровень взаимодействия программы с аппаратурой. Основная идея – это разработка платформно - независимого API. Т.е. между программой и аппаратурой вставляется программный слой, унифицирующий работу программы с некоторым набором аппаратуры.

К программной платформе относят:

- операционную систему (обычно ОСРВ), распределяющую
- аппаратный ресурс;

- драйверы устройств, обеспечивающие подсистему ввода/вывода;
- сетевую подсистему, обеспечивающую взаимодействие компонентов вычислительной системы.

Системная платформа, как уже говорилось, объединяет аппаратную и программную платформы. В начале проектирования обе платформы являются абстракциями. Причем, чем выше эта абстракция, тем лучше – больше свободы в выборе решений по конкретизации системы.

Нежелательно преждевременное разделение функций между аппаратурой и ПО.

Возможно и желательно рассмотрение нескольких платформ, пригодных для реализации одной и той же функциональности с целью выбора оптимальной.

Проектируемая система с добавлением ограничений (быстродействие, габариты, надежность, потребление энергии,

доступное API, наиболее подходящая ОСРВ и т.п.) уточняется (актуализируется) и вариантов ее реализации остается все меньше и меньше. В итоге получается единственное решение: однозначный выбор аппаратуры и определённая модель ПО.

Для обеспечения возможности повторного использования платформы должны быть хорошо верифицированы и документированы. В идеале, желательны стандарты, не определяющие их внутреннее устройство, но фиксирующие функциональность и интерфейсы для интеграции (шины, API и т.д.).

Повсеместное использование таких стандартов может значительно видоизменить всю индустрию электроники, позволив быструю и эффективную интеграцию существующих сторонних разработок. В качестве примера подобной стандартизации можно привести сферу персональных компьютеров, где стандартизованы внутренние соединения, устройства ввода вывода (мыши, клавиатуры), архитектура системы команд.

Применение подобного подхода в области проектирования электроники должно принести пользу как изготовителям платформ, так и тем, кто их использует.

Примером такой платформы из области BcC является платформа DaVinci производства компании Texas Instruments, представляющая собой линейку высокопроизводительных мультимедийных процессоров, программное обеспечение в виде готовых библиотек, позволяющих использовать их возможности, документацию и развитую сеть для поддержки быстрого и эффективного проектирования на основе данных процессоров.

### 4.5.6. Модельно-ориентированное проектирование

Под модельно-ориентированным проектированием (model-based design, MBD) обычно понимается парадигма, в соответствии с которой проектирование ПО ВС (и всей системы вообще) фокусируется на высокоуровневых исполняемых моделях проектируемой системы.

Наиболее популярным и широко распространённым примером реализации MBD является среда проектирования MATLAB/Simulink, которая повсеместно используется для проектирования систем автоматического управления (САУ), систем цифровой обработки сигналов (ЦОС, DSP), механических систем и т.д. Учитывая одновременное моделирование физических непрерывных процессов и дискретных вычислений, MATLAB/Simulink позволяет вполне реализовать киберфизический подход.

Кроме того, в области создания ПО известно и широко используется управляемое моделями проектирование (model-driven development, modeldriven engineering, model-driven architecture). В основе этого подхода лежит генерация программного кода из моделей (абстрактных описаний ПО), которые являются основными артефактами разработки.

Понятия MBD и MDD во многом пересекаются и фактически отражают один и тот же подход к проектированию, на основе синтеза реализаций из абстрактных моделей.

При более абстрактном видении проблемы имеется больше свободы для выбора более подходящего решения, и больше возможностей конечных реализаций, в противовес проектированию для одной и только одной платформы.

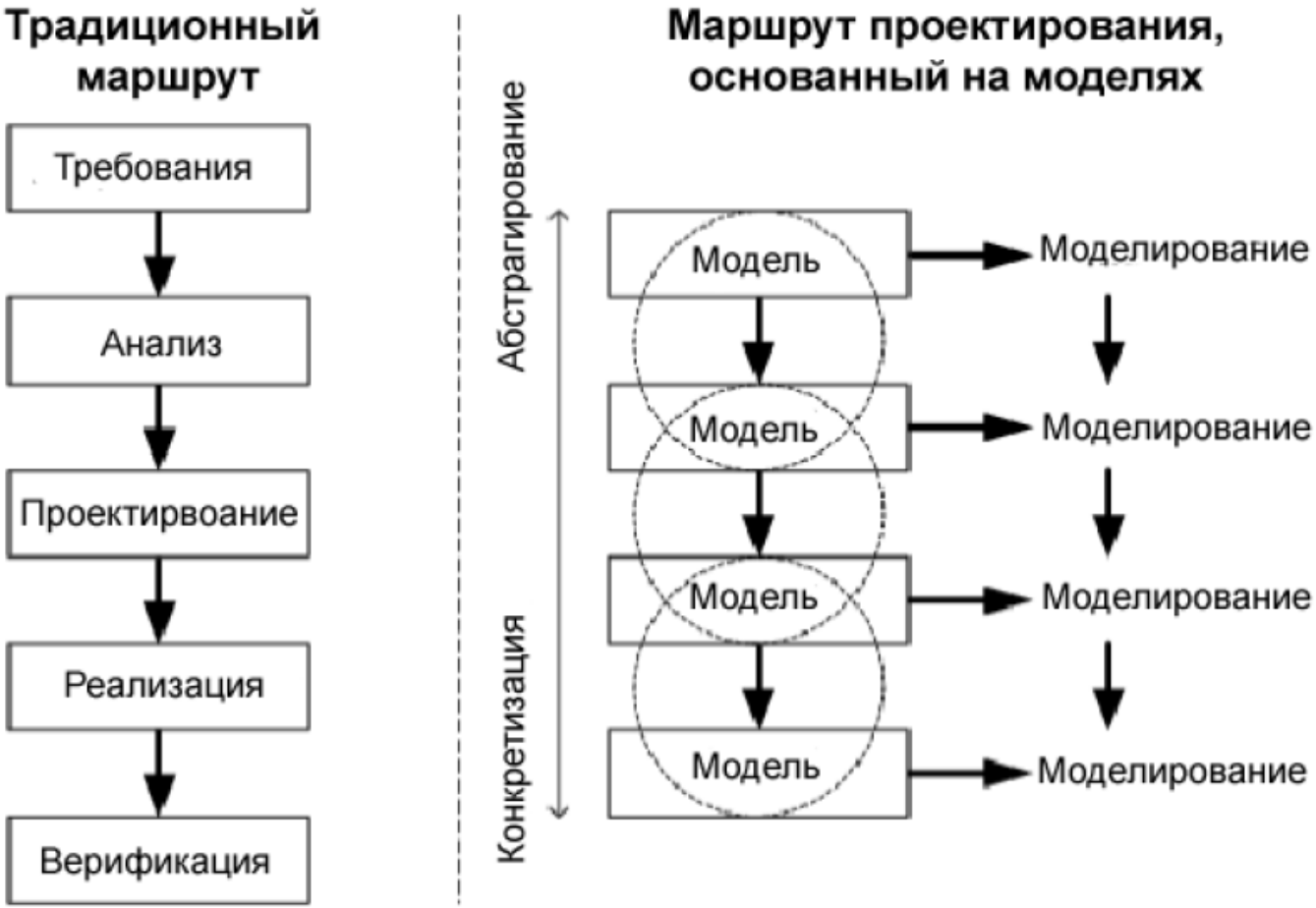



*Рис. 43. Традиционный и основанный на моделях маршруты проектирования* [35].

Рис. 4.15. Традиционный и основанный на моделях маршруты проектирования

Традиционные методы проектирования (аппаратуры, ПО, систем в целом) представляют собой стадии разработки требований, анализа, проектирования, реализации и верификации.

Проектирование же на основе моделей вращается вокруг моделей и трансформаций между этими моделями. В процессе проектирования работа производится над некоторым набором моделей в каждый момент времени, что предоставляет обратную связь за счёт верификации. Верификация осуществляется сравнением поведения высокоуровневых, более абстрактных, и низкоуровневых, детализированных моделей.

Проектирование на основе моделей обычно представляется как ряд моделей, связанных между собой нисходящими трансформациями. Однако, в реальном мире ни такое, строго последовательное проектирование сверху вниз, ни проектирование снизу-вверх не работают, кроме очень специальных случаев.

Разработка должна производиться одновременно на разных уровнях абстракций и с учётом многих аспектов в одно и то же время.

Симуляции должны включать в себя обширные наборы технологий верификации и валидации, их результаты должны сравниваться с результатами симуляций более высокоуровневых моделей, проверяться на соответствие ограничениям. В течение процесса проектирования модели трансформируются автоматически или полуавтоматически в более платформно-ориентированные. Выделяются следующие модели:

1) Не зависящие от организации вычислительного процесса (Computation Independent Model). Это – модели использования системы, модели предметных областей, модели данных.
2) Не зависящие от платформы (Platform Independent Model). Это модели, выполненные в рамках основных моделей вычислений (конечный автомат, сети процессов, и др.).
3) Платформно-зависимые (Platform Specific Model).

4) Реализации спецификаций.

Использование при проектировании высокоуровневых моделей системы позволяет исследовать альтернативы реализации разрабатываемого продукта, упрощает коммуникации с заказчиком и исполнителями, удобнее для документирования.

### 4.5.7. Пример: Ptolemy

Ptolemy II представляет собой комплекс программ, библиотек Java-классов и механизмов. Комплекс предназначен для построения и исследования моделей различного рода систем, в основном для встраиваемых систем, а также для академических исследований в области моделирования.

На основе идей и подходов, реализованных в Ptolemy II, активно развиваются средства моделирования как во многих исследовательских HLD САПР, так и в продуктах индустриального назначения.

Ptolemy II создан и постоянно развивается группой исследователей Ptolemy Project из университета Калифорнии в Беркли, возглавляемой профессором Edward A. Lee.

Ptolemy II обеспечивает моделирование вычислительных систем различной природы. На основе комплекса созданы специализированные подсистемы для моделирования:

- беспроводных сетей и распределенных систем, объединенных радио-, оптическими или акустическими каналами связи (пакет VisualSense);
- гибридных систем, объединяющих в себе дискретную логику управления и модели с непрерывным временем. Пакет HyVisual включает в себя модели вычислений FSM (finite state machines) и CT (continuous-time), библиотеку акторов, поддерживающих эти модели вычислений и средства визуализации моделей.

Каждая подсистема представляет собой подмножество классов, механизмов и акторов, объединенное одной управляющей программой.

Ptolemy II создан для моделирования гетерогенных вычислительных систем, то есть систем, объединяющих компоненты, описываемые в разных моделях вычислений:

- дискретную и аналоговую технику,
- сети и телекоммуникационные протоколы,
- стохастические и криптографические системы,
- радиокомпоненты,
- компоненты, реализующие обработку сигналов и изображений и т.д.

Для этого комплекс поддерживает моделирование в рамках различных моделей вычислений и объединение отдельных моделей в гетерогенную иерархию с последующим комплексным моделированием.

В настоящее время комплекс поддерживает следующие модели вычислений:

• CT (continuous time) – модели с непрерывным временем;

• DE (discrete-event) – модели с дискретными событиями;

• DDE (distributed discrete events) – системы с частично упорядоченным множеством дискретных событий;

• FSM (finite-state machines) – конечные автоматы, в модели отсутствует понятие времени;

• PN (process networks) – сети процессов Кана (Kahn), в модели отсутствует понятие времени;

• SDF (synchronous dataflow) – модели для представления систем обработки сигналов, в моделях отсутствует понятие времени;

• DDF (dynamic dataflow) – расширение SDF, позволяющее компонентам изменять количество потребляемых и генерируемых элементов данных за одну итерацию в процессе исполнения модели;

• HDF (heterogeneous dataflow) – расширение SDF, позволяющее сохранить статическое планирование и другие свойства моделей в ее рамках и избегать ограничений на неизменность количества потребляемых и генерируемых компонентами элементов данных;

• PSDF (parameterized synchronous dataflow) – расширение SDF с параметризируемой дисциплиной планирования вычислений;

• DT (discrete time) – периодически запускаемые статически планируемые процессы с дискретным временем. Аналог SDF, в котором присутствует понятие времени;

• SR (synchronous-reactive) – синхронно-реактивная модель вычислений;

• CI (component interaction) – модель вычислений со стилем взаимодействия компонентов «push/pull»;

• CSP (communicating sequential processes) – взаимодействующие последовательно выполняемые процессы;

• Giotto – поддержка одноименного языка и его семантики. Один из вариантов синхронной модели, в котором компоненты могут активироваться с разной частотой;

• TM (timed multitasking) – вариант взаимодействия задач ОСРВ;

• Wireless – модель вычислений для исследования систем, объединенных беспроводными каналами связи;

• GR – поддержка 3-D графики.

Комплекс Ptolemy II является open-source-проектом, что способствует распространению и широкому использованию подходов и механизмов, реализованных в нем.

Система Ptolemy II включает в себя графическую среду разработки Vergil, позволяющую схематично представлять модели, аннотировать и конфигурировать их, проверять их корректность и запускать на исполнение (Рис. 4.16).

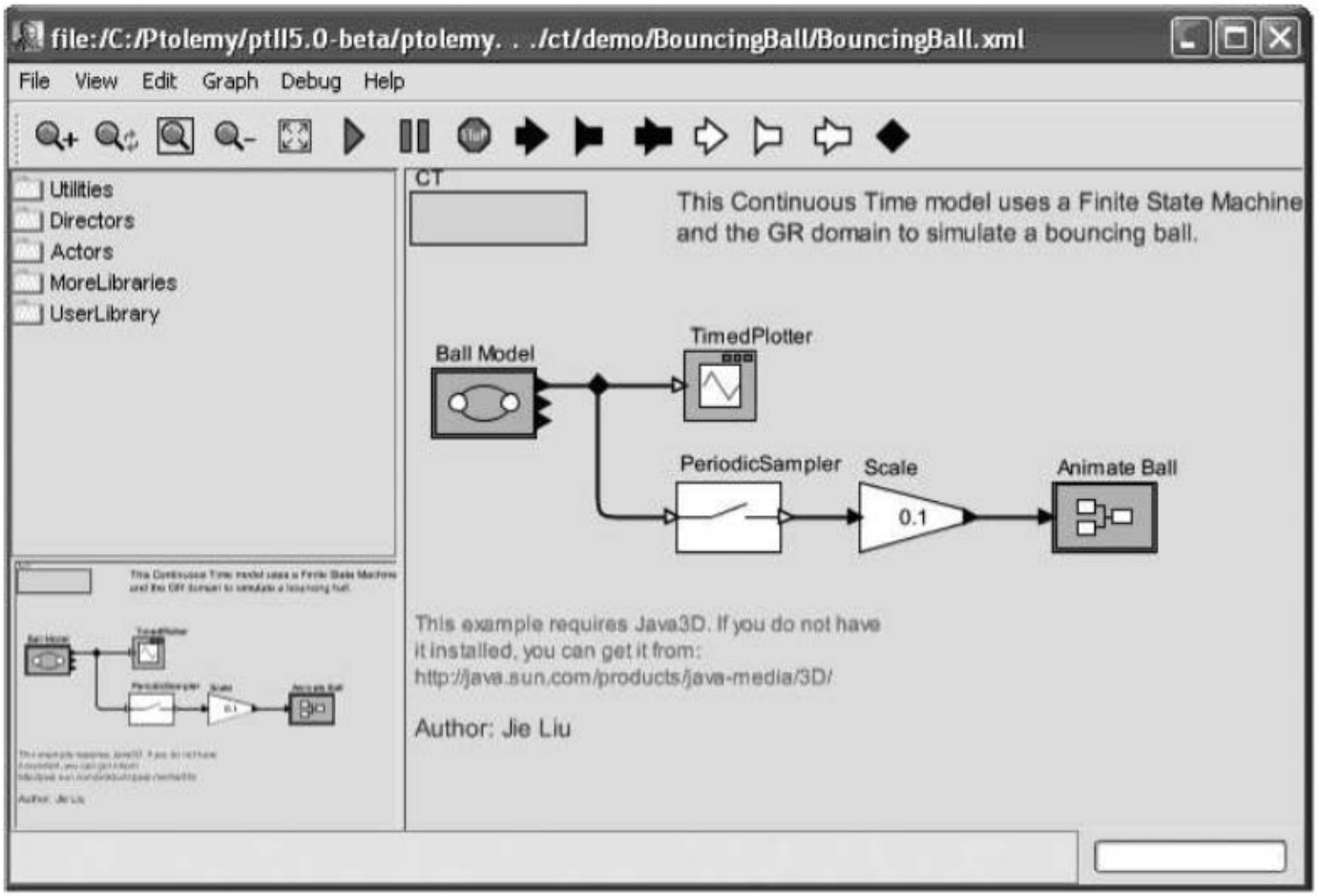



*Рис. 44. Редактор моделей Vergil.*

Рис. 4.16. Редактор моделей Vergil

Vergil включает редактор моделей в акторном представлении (Graph editor), редактор конечных автоматов (Рис. 4.17), текстовый редактор, редактор иконок акторов, позволяет просматривать HTML-страницы и редактировать текст файлов.

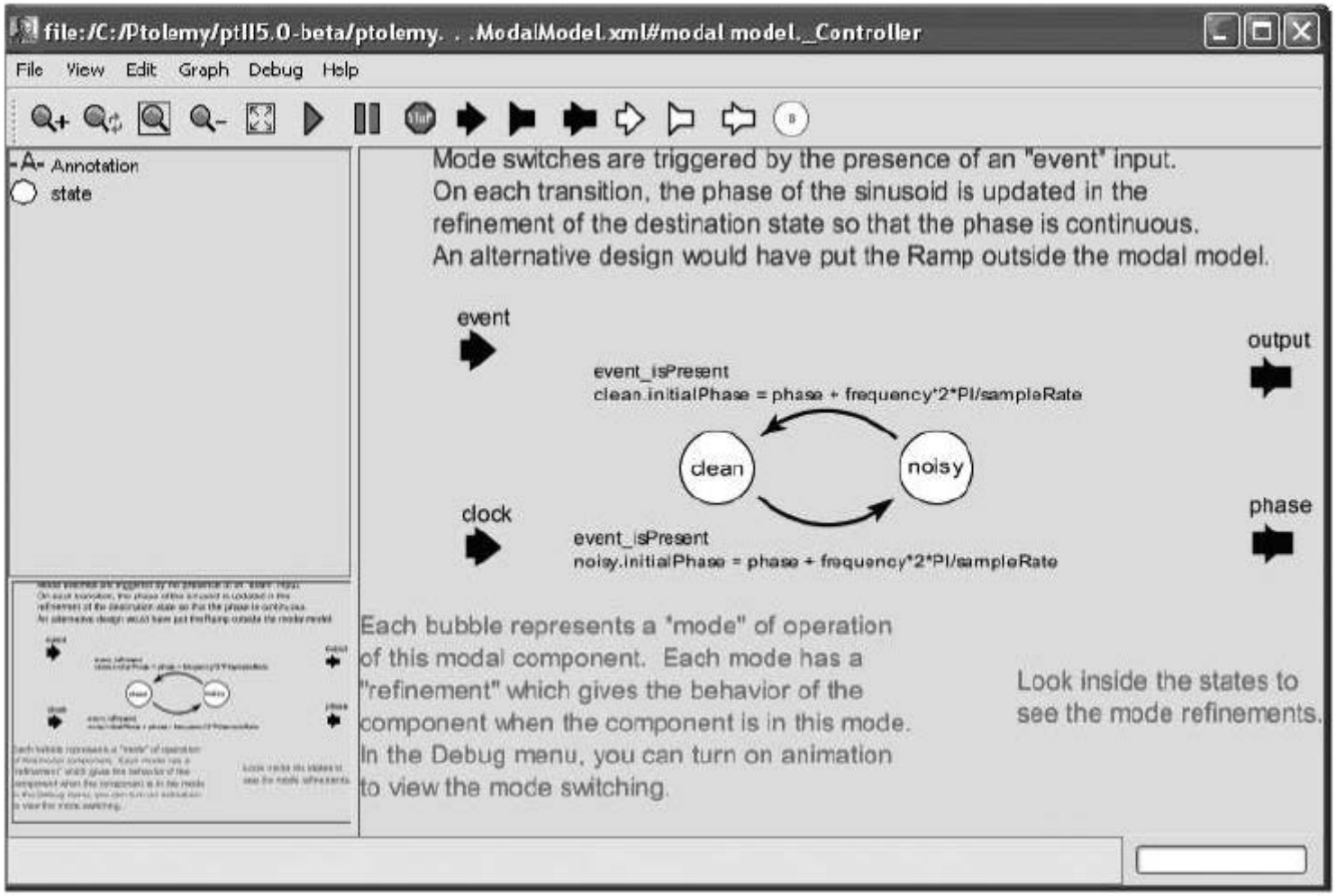



*Рис. 45. Редактор конечных автоматов Vergil.*

Рис. 4.17. Редактор конечных автоматов

Компонентами моделей Ptolemy II по большей части являются акторы, атомарные функциональные преобразователи сигналов, поведение которых описано на Java. Каждый такой актор представляет собой Java- класс, который может быть в процессе моделирования расширен и дополнен различными механизмами Ptolemy II. Акторы могут быть составными, то есть являться контейнерами других моделей, обеспечивая, таким образом, их иерархичность. Комплекс включает обширную библиотеку акторов, систематизированную по группам выполняемых ими преобразований (Рис. 4.18).

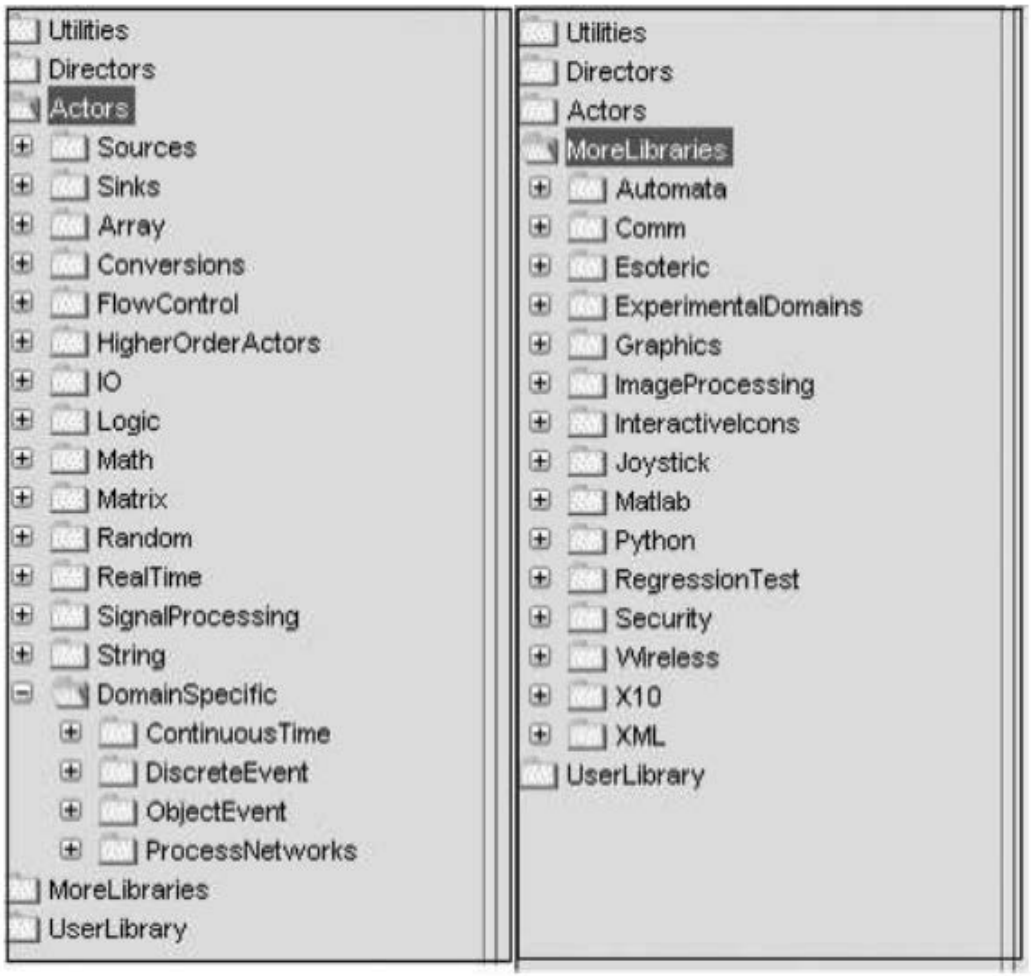
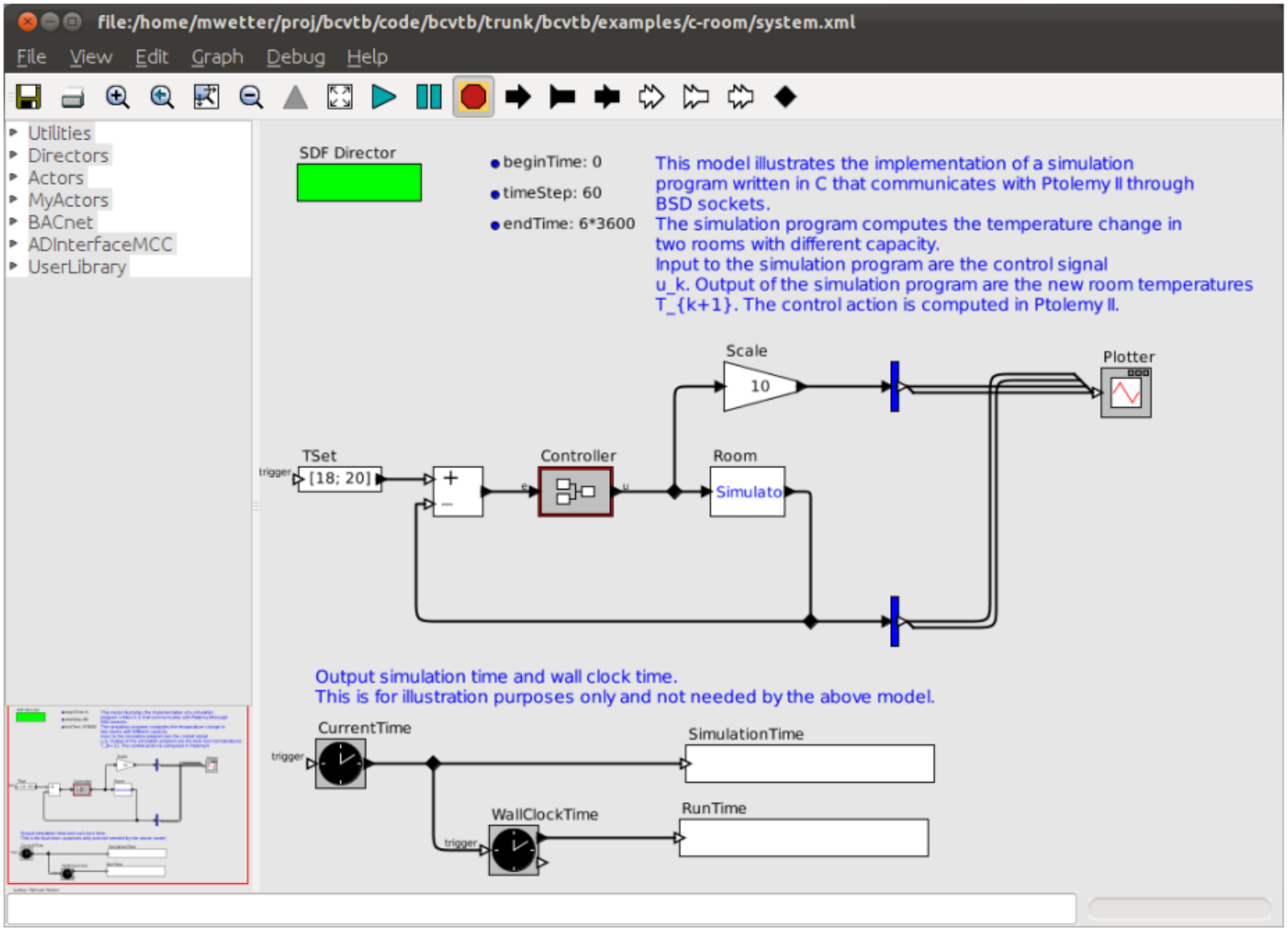



*Рис. 46. Библиотека акторов Ptolemy II.*

Рис. 4.18. Библиотека акторов

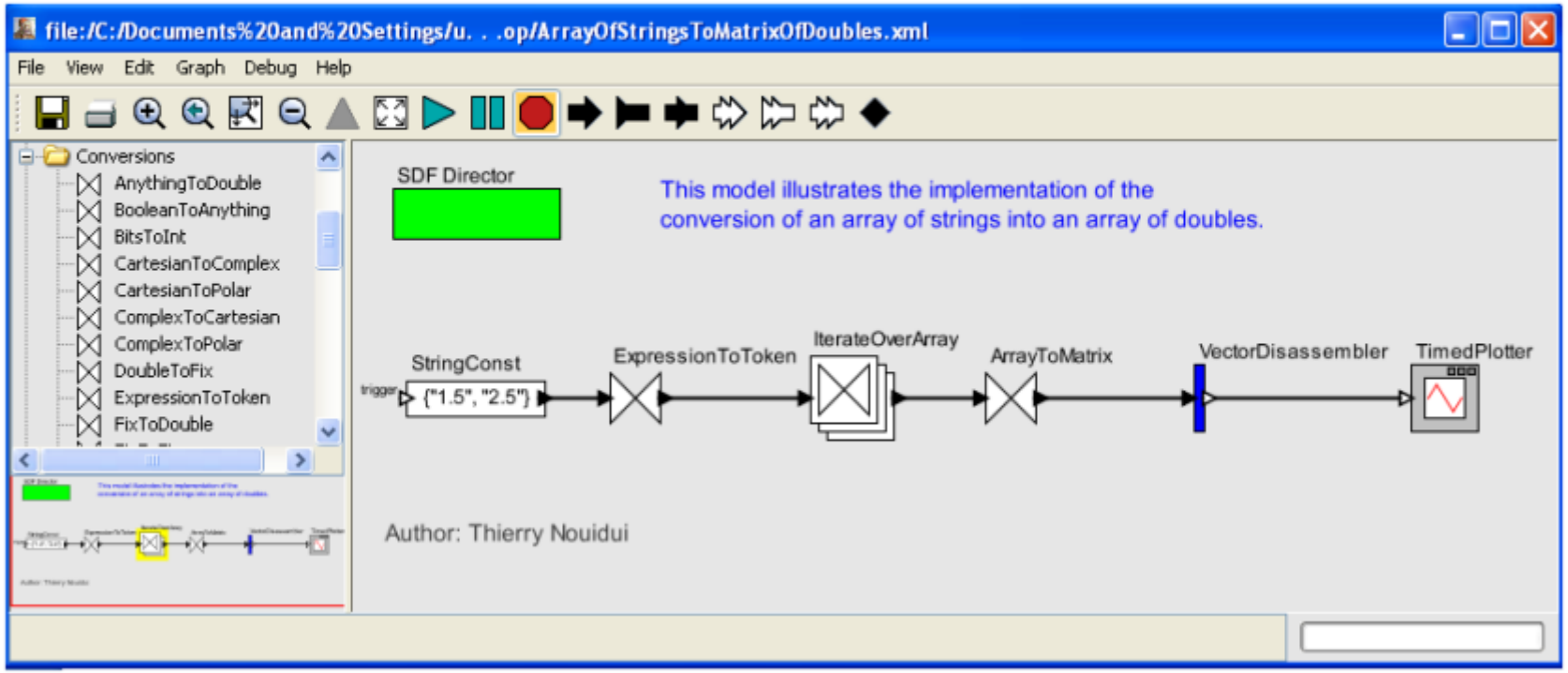
В Ptolemy II разработан довольно мощный язык выражений, позволяющий не только выполнять математические операции над данными, получая значение результата, но и создавать переменные различных типов, в том числе и произвольных составных, выполнять преобразования типов, выполнять сложные встроенные процедуры обработки данных и создавать собственные, производить ввод/вывод и управлять вычислительным процессом.

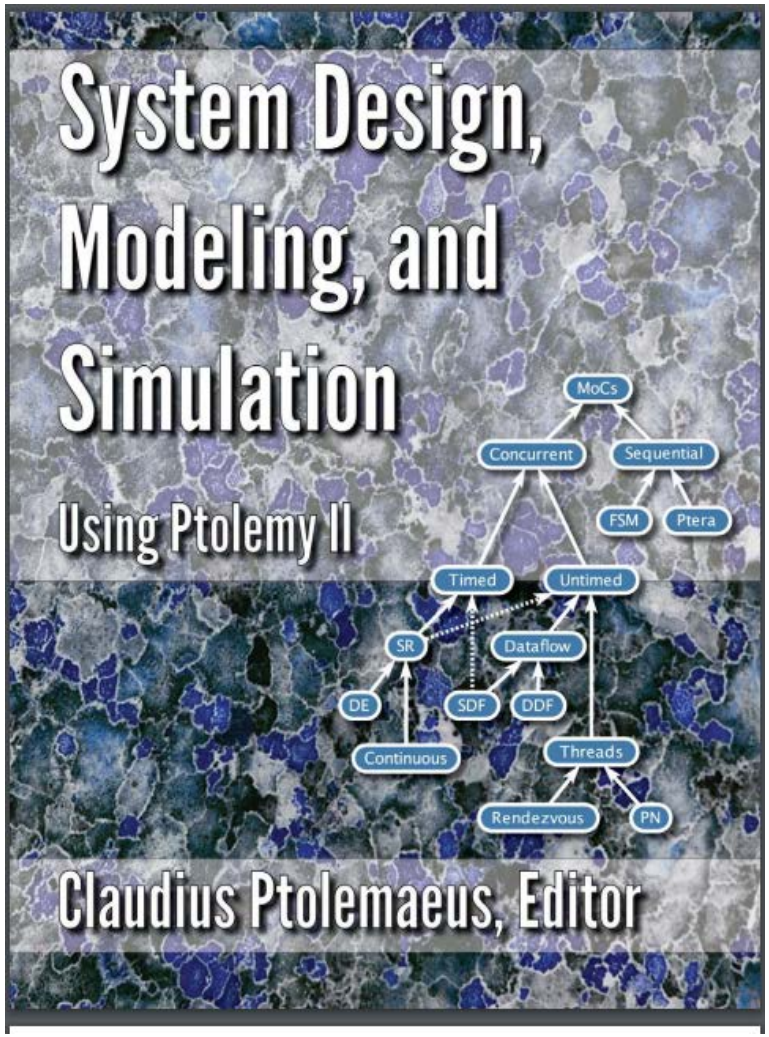
**Figure 5.1. Ptolemy II system model that connects a model of a controller and a room.**



### 5.2.2. Data Type Conversion

**Figure 5.6. Ptolemy II system model that converts an array of strings into an array of doubles.**






https://ptolemy.berkeley.edu/books/Systems/

**First Edition, Version 1.02**

**ISBN: 978-1-304-42106-7**

**Please cite this book as:**

Claudius Ptolemaeus, Editor,
*System Design, Modeling, and Simulation using Ptolemy II*, Ptolemy.org, 2014.
http://ptolemy.org/systems.

Программа
https://ptolemy.berkeley.edu/ptolemyII/ptII11.0/index.htm

# Глава 5. Моделирование встраиваемых систем в пакете программ Ptolemy II

## 5.1. Обзор пакета программ Ptolemy II

Ptolemy II представляет собой комплекс программ, библиотек Java-классов и механизмов. Комплекс предназначен для построения и исследования моделей различного рода систем, в основном для встроенных систем, а также для академических исследований в области моделирования.

### 5.1.1. История Ptolemy II

Ptolemy II создан и постоянно разрабатывается группой исследователей Ptolemy Project из университета Калифорнии в Беркли, возглавляемой с профессором Edward A. Lee. Комплекс разрабатывается как open-source и распространяется свободно. Ptolemy II является третьим по счету инструментом, который разрабатывается группой Ptolemy Project, после Gabriel (1986-1991)

и Ptolemy Classic (1990-1997). Большое количество идей для Ptolemy II было взято из Ptolemy Classic, который представлял собой средство моделирования и проектирования систем обработки данных и управляющих систем. Ptolemy Classic позволял создавать в графической среде гетерогенные модели систем с использованием разных моделей вычислений, исполнять эти модели на симуляторах различных аппаратных и программных платформ, а также генерировать из моделей код на ассемблере (для DSP), C и VHDL. Ptolemy II начал разрабатываться в 1996 году и был призван сместить акцент на моделирование с использованием технологий, предоставляемых платформой Java™. В новой среде были введены понятия доменного полиморфизма, «режимных» (modal) моделей, а также типового полиморфизма. Был также разработан механизм разрешения типов данных в модели, множество моделей вычислений, технология распределенного моделирования, а также

мощный язык выражений, который может быть использован, в том числе, и в исполняющейся модели.

## 5.2. Теоретическая часть

### 5.2.1. Моделирование и проектирование

Предметом исследований проекта Ptolemy II является гетерогенное (неоднородное) моделирование, симуляция, и проектирование параллельных систем. Основной фокус сделан на встроенных системах, в особенности на тех, которые используют смешанные технологии: аналоговую и цифровую электронику, программные и аппаратные составляющие, электронные и механические устройства. Еще один важный фокус сделан на комплексных системах, совмещающих множество функций, такие как функции сетевого взаимодействия, цифровая обработка сигналов, управление с обратной связью, функции принятия решений и обеспечение пользовательских интерфейсов.

Моделирование – это процесс формального представления системы или подсистемы. Модель может быть математической, в этом случае она представляется в виде набора математических утверждений о параметрах системы, таких как функциональность или физические характеристики. Модель может быть конструктивной, в этом случае она определена в виде вычислительной процедуры, имитирующей набор параметров системы. Конструктивные модели часто используются, чтобы представить поведение системы в виде ответов на внешние воздействия. Конструктивные модели также называют исполняемыми.

Проектирование – это процесс определения системы или подсистемы. Обычно проектирование включает в себя определение одной или нескольких моделей с последующей их детализацией и совершенствованием, до тех пор, пока не будет достигнута требуемая функциональность и удовлетворены все ограничения.

Очевидно, что процессы проектирования и моделирования тесно связаны. В отдельных случаях некоторые модели могут быть статичными в ходе проектирования, в том смысле, что они описывают подсистемы и ограничения, которые являются внешними по отношению к проекту. К примеру, они могут описывать механическую систему, которая не является частью проекта, но должна управляться электронной системой, разработка которой является целью проекта.

Исполняемые модели иногда называются симуляциями, этот термин особенно хорошо подходит в тех случаях, когда исполняемая модель не связана по своей природе с системой, которую она моделирует. Однако, во многих электронных системах, модель, которая создается как симуляция, впоследствии эволюционирует в программную реализацию системы. Граница между моделью и самой системой в этом случае сильно размыта. Это особенно характерно для встроенного программного обеспечения.

Встроенное программное обеспечение – это программное обеспечение, которое находится в устройствах, не являющихся типичными персональными компьютерами. Оно распространено повсюду: в автомобилях, телефонах, бытовой электронике, игрушках, авиационной технике, поездах, охранных системах, оружейных системах, принтерах, модемах, копирах, термостатах, производственной технике, научных приборах и др..

Технически активный человек регулярно взаимодействует с огромным количеством встроенного ПО. Основной особенностью встроенного ПО является взаимодействие с физическим миром, что налагает на него временные ограничения, не свойственные для обычного ПО настольных компьютеров. Исполняемые модели конструируются в рамках определенной модели вычислений, которая является набором “физических законов” управляющих взаимодействием между элементами модели. Если модель описывает механическую систему, тогда моделью вычислений могут быть

настоящие законы физики. Однако гораздо чаще свод этих правил является более абстрактным, предоставляя проектировщику адекватный и удобный каркас для конструирования моделей.

Набор правил, который управляет взаимодействием элементов, называется семантикой модели вычислений. Модель вычислений может иметь несколько семантик (более одной), описывающих различные наборы правил, но устанавливающих идентичные ограничения на поведение элементов системы. Выбор модели вычислений сильно влияет на тип конструируемой модели. К примеру, для чисто вычислительной системы, которая преобразует один конечный набор данных в другой конечный набор данных, адекватной является императивная семантика, характерная для таких языков программирования как C, C++, Java и MATLAB. Для моделирования механических систем используемая семантика должна включать в себя понятия параллельности и времени. В этом случае использование модели вычислений с непрерывным временем

в таких системах как Simulink, Saber, Hewlett-Packard‟s ADS и VHDL-AMS является наиболее адекватным.

Способность модели системы эволюционировать в еѐ реализацию сильно зависит от используемой модели вычислений. Некоторые модели вычислений подходят для реализации только в виде специальной аппаратуры, в то время как другие подходят для этого плохо из-за своей последовательной природы. Неверный выбор модели вычислений может привести проектировщика к более дорогой и менее надежной реализации. Один из принципов проекта Ptolemy заключается в том, что выбор модели вычислений сильно влияет на качество проекта системы.

Для встроенных систем наиболее полезными являются те модели вычислений, в которых существуют понятия параллельности и времени. Причина этого в том, что встроенные системы, как правило, состоят из параллельно работающих компонентов и имеют множество независимых внешних источников воздействия. В

дополнение к этому, они работают в условиях реального мира, когда время ответа на воздействие может быть не менее важным, чем корректность этого ответа.

*Цель Ptolemy II – обеспечить возможность конструирования и поддержания взаимодействия между исполняемыми моделями, построенными в широком диапазоне различных моделей вычислений.* Ptolemy II использует компонентный подход к проектированию, когда модели конструируются из набора взаимодействующих элементов. Модель вычислений управляет семантикой взаимодействия, и, таким образом, определяет дисциплину взаимодействия между компонентами. Компонентно-ориентированное проектирование в Ptolemy II подразумевает взаимодействие элементов системы по определенной дисциплине, определяемой моделью вычислений.

### 5.2.2. Модели вычислений

Существует множество различных моделей вычислений, которые по-разному рассматривают понятия параллельности и времени.

Каждая представляет свой механизм взаимодействия между элементами. Назначение модели вычислений происходит от параметров моделирования, применяемых к похожим моделям. Для многих моделей вычислений эти параметры унаследованы из формальной математики. В зависимости от модели вычислений, модель может быть детерминированной, со статичным расписанием или безопасными во времени. В зависимости от параметров моделирования, модель вычислений представляет стиль моделирования пригодный во всех возможных случаях, рассчитанных на данный набор параметров. Другими словами, модель вычислений формирует шаблон проектирования взаимодействия компонентов, в том же смысле, в котором Гамма [Gamma] описывает шаблоны проектирования для объектно-ориентированных языков.

Модель вычислений, применённая в своей специальной области, не накладывает лишних ограничений, но в то же время достаточно

ограничена для того, чтобы из этого получить полезные свойства. К примеру, ограничив пространство проектирования синхронными проектами, Scenic (основа SystemC) позволяет проводить циклическую симуляцию [cycle-driven simulation], которая сильно поднимает эффективность исполнения по сравнению с более общими моделями с дискретными событиями (такими как в VHDL). Однако для приложений с мультичастотным поведением разработка в рамках синхронного проекта может стать сильным ограничением. В таких случаях менее ограниченная модель вычислений, такая как синхронный поток данных [synchronous dataflow] или сеть процессов Кана, может оказаться более подходящей. Отрицательная сторона таких послаблений по сравнению с ограничениями синхронного проекта в том, что анализ буферизации значительно усложняется. С другой стороны, существующие в моделях с

синхронным потоком данных методики значительно упрощают совместную оптимизацию использования памяти и рабочей частоты

в мультичастотных системах. Выбор подходящей для специализированного приложения модели вычислений часто является большой проблемой, но эту проблему нужно решать, а не пытаться еѐ избежать. В этом разделе мы опишем некоторые из моделей вычислений, реализованные в доменах Ptolemy II. Мы будем фокусироваться на моделях вычислений, которые наиболее полезны для встроенных систем. Модели Ptolemy II представляются в виде (блочных или иерархичных) графов, где узлы это сущности, а дуги – отношения. Для большинства доменов, сущности это акторы (сущности с функциональностью) а отношения, связывающие их, представляют каналы связи между акторами. При этом используемая модель параллельности и механизм взаимодействия между элементами в различных моделях могут сильно различаться.

### 5.2.3. Взаимодействующие компоненты (Component Interaction – CI)

Домен с взаимодействующими компонентами (CI) моделирует системы, которые сочетают в себе управляемые данными и управляемые запросами стили вычислений. К примеру, взаимодействие между веб-сервером и браузером является управляемым запросами. Когда пользователь щелкает по ссылке в браузере, он “вытаскивает” соответствующую страницу из веб-сервера. Сервис, предоставляющий данные о биржевых котировках может использовать управляемый данными стиль вычислений. Сервер генерирует события, когда изменяется курс акций. Данные заставляют клиентов обновить отображаемую у них информацию. Такие “push/pull” (затолкнуть/вытащить) взаимодействия между производителем и потребителем данных крайне характерны для распределенных систем, и были внедрены в middleware сервисах, таких как сервис событий CORBA. Эти сервисы мотивировали

создание этого домена для изучения взаимодействий в распределенных системах, таких как сервисы биржевых котировок, транспортные системы или сервисы информации о погоде. Другие приложения этой модели вычислений включают в себя системы баз данных, файловые системы, модульные маршрутизаторы. Актор в модели CI может быть активным, это означает, что он обладает собственным потом выполнения. К примеру, источник прерывания во встроенной системе может быть смоделирован как активный актор. Такой источник генерирует события асинхронно по отношению к программному обеспечению, исполняемому на встроенном процессоре. Модели CI могут быть использованы для симуляции и изучения того, как встроенное ПО обрабатывает асинхронные события, такие как внешние прерывания и асинхронный ввод/вывод.

### 5.2.4. Модель с непрерывным временем (Continuous Time – CT)

В домене с непрерывным временем (CT) акторы являются компонентами, которые взаимодействуют через непрерывные сигналы. Акторы, как правило, определяют

алгебраические или дифференциальные отношения между входами и выходами. Задачей директора домена является поиск “фиксированной точки”, т.е. набора непрерывных функций которые удовлетворяют всем отношениям. Домен CT содержит большой набор дифференциальных решающих устройств. Таким образом, этот домен является удобным для моделирования физических систем с линейным или нелинейным поведением, таких как аналоговые цепи и механические системы. Используемая модель вычислений похожа на те, что используется в Simulink, Saber и VHDL-AMS, и тесно связана с той, что используется в симуляторе эклектических цепей Spice.

### 5.2.5. Модели со смешанными сигналами (Mixed Signal Models)

Встроенные системы часто содержат компоненты, которые лучше всего моделируются с использованием дифференциальных уравнений, такие как MEMS и другие механические компоненты, аналоговые и микроволновые цепи. Эти компоненты, однако, взаимодействуют с электронной системой, которая может играть роль управляющей системы или получателя данных с сенсоров. Эта электронная система может быть цифровой. Совместное моделирование подсистем с непрерывным временем и цифровой электроники известно как моделирование систем со смешанными сигналами. Домен CT спроектирован для взаимодействия с другими доменами Ptolemy, такими как DE или FSM, чтобы сделать возможным моделирование гибридных систем и систем со смешанными сигналами. Чтобы поддержать такое моделирование, домен CT моделирует дискретные события в виде дельта-функций Дирака. Также он позволяет обнаружить точки пересечения

пороговых значений, производя при этом дискретные события. Модальные модели и гибридные системы [Modal Models and Hybrid Systems]. Физические системы всегда представляются в виде простых моделей, корректных только в определенном режиме вычислений. Вне этого режима, другая простая модель может стать подходящей. Модальная модель эта модель, которая переключается между этими простыми моделями, когда система меняет режим вычислений. Домен CT взаимодействует с доменом FSM для создания таких модальных моделей. Часто такие модели называются гибридными системами.

### 5.2.6. Дискретные события (Discrete events – DE)

В домене с дискретными событиями (DE) акторы взаимодействуют через последовательности событий, расположенных вдоль временной шкалы реального времени. Событие состоит из значения и временной метки. Акторы могут быть либо процессами, которые реагируют на события (реализованы

в виде потоков Java), либо функциями, которые срабатывают на появление новых событий. Эта модель вычислений популярна для спецификации цифровой аппаратуры и для симуляции телекоммуникационных систем. Она была реализована во многих средах моделирования, языках моделирования и языках описания аппаратуры, включая VHDL и Verilog.

В домене DE реализован довольно сложный симулятор дискретных событий. Симуляторы дискретных событий в общем случае должны реализовывать глобальную очередь ожидающих событий, отсортированных по временной метке (это называется очередью по приоритету). Это может быть достаточно дорогим решением, т.к. добавление новых событий требует поиска нужной позиции в списке. Домен DE использует структуру данных “календарь” для глобальной очереди событий. Календарную очередь можно представить как хеш-таблицу, которая квантует время как хеш-функцию. Благодаря этому, обе операции “вставить в очередь” и

“убрать из очереди” могут быть выполнены за время, не зависящее от числа событий в очереди.

### 5.2.7. Распределенные дискретные события (Distributed discrete events – DDE)

Домен распределенных дискретных событий (DDE), может рассматриваться либо как вариант DE, либо как вариант PN (рассмотренной ниже). Он нацелен на решение ключевой проблемы моделирования с дискретными событиями, заключающейся в том, что глобальная очередь событий задает центральную точку управления моделью, крайне ограничивая возможность распределения модели через сеть. Распределение моделей через сеть может понадобиться либо для защиты интеллектуальной собственности либо чтобы загрузить параллельные вычислительные ресурсы. Домен DDE поддерживает локальное представление времени на каждом соединении между акторами, вместо одного глобального непрерывного представления времени. Каждый актор

это процесс, реализованный в виде Java потока, который может продвинуть свое локальное время на минимум из локальных времен среди его входных соединений. Домен систематизирует передачу нулевых событий [null events], это гарантирует, что ни одно событие с временной меткой меньшей, чем некоторое определенное число, не будет инициировано.

### 5.2.8. Динамический поток данных (Dynamic Data Flow – DDF)

Домен с динамическим потоком данных (DDF) это смесь доменов с синхронным потоком данных (SDF) и булевым потоком данных (BDF). В домене SDF актор потребляет и производит фиксированное число лексем [tokens] при каждом запуске. Статичность этой информации позволяет синтезировать расписание во время компиляции модели. В домене DDF актор изменяет частоту производства и потребления после каждого запуска. Планировщик не будет делать попыток простроить расписание во время компиляции и не пытается ответить на вопросы о возможном

возникновении взаимоблокировок или границ, которые фундаментально неразрешимы. Вместо этого, каждый актор имеет последовательный набор правил(шаблонов) запуска и может быть запущен, если одно из них удовлетворено, т.е. любой шаблон запуска формирует последовательность неиспользованных лексем на входных портах. Планировщик динамически формирует расписание запуска акторов согласно некоторому критерию. Каноническими акторами домена DDF являются Select и Switch, которые производят или потребляют лексемы на различных каналах, в зависимости от лексемы, полученной на управляющем порте.

### 5.2.9. Дискретное время (Discrete time – DT)

Домен с дискретным временем расширяет домен SDF (описанный ниже) понятием времени между токенами. Связь между акторами принимает форму последовательности токенов с одинаковым временем между каждыми двумя токенами. Мультичастотные модели, где различные соединения имеют

различные временные интервалы между токенами, также поддерживаются. Существует определенная тонкость в этом домене, когда используются мультичастотные компоненты. Семантика определена таким образом, что поведение компонента всегда предсказуемо в том, что вывод, значение которого зависит от входов, никогда не будет произведен раньше ввода на этих входах.

### 5.2.10. Конечные автоматы (Finite state machines – FSM)

Домен с конечными автоматами радикально отличается от остальных доменов Ptolemy II. Сущности в этом домене представляют не акторы а состояния, а соединения представляют переходы между состояниями. Исполнение модели представляет собой строго упорядоченную последовательность переходов между состояниями. Домен FSM использует встроенный в Ptolemy II язык описания выражений чтобы вычислять условия переходов [guards], которые определяют, когда будет осуществлен переход между состояниями. FSM модели идеальны для выражения управляющей

логики и для построения модальных моделей (моделей с несколькими режимами работы, где поведение различно в каждом режиме). Модели FSM подлежат глубокому формальному анализу, и поэтому могут использоваться, чтобы избежать неожиданного поведения. У FSM моделей есть один ключевой недостаток. Во-первых, на фундаментальном уровне, они не так выразительны, как другие модели вычислений описанные здесь. Они не достаточно богаты для описания всех частично рекурсивных функций. Однако, эта слабость приемлема в свете того, что формальный анализ модели становиться возможен. Многие вопросы проектирования решаются для FSM и не разрешимы для других моделей вычислений. Вторая ключевая слабость в том, что число состояний может стать огромным даже в скромных по сложности системах. Это делает такие модели громоздкими. Обе проблемы зачастую могут быть решены с помощью использования FSM в комбинации с параллельной моделью вычислений. Первый раз это было

опробовано Дэвидом Харелом [David Harel], который представил формальное описание диаграмм состояний. Диаграммы состояний сочетают свободную интерпретацию синхронно-реактивного моделирования с FSM. FSM также могут быть объединены с дифференциальными вычислениями, порождая так называемую модель вычисления гибридных систем.

Домен FSM в Ptolemy II может быть иерархично объединен с другими доменами. Результирующая формальная система называется “*charts” (произноситься “starcharts” )] где звездочка представляет “дикую” карту. Так как большинство доменов представляют параллельные вычисления, *charts принимает вид параллельных конечных автоматов с широким диапазоном параллельных семантик. FSM совмещенная с CT порождает гибридные системы и модальные модели. Совмещение с SR порождает нечто крайне близкое к диаграммам состояний. Совмещение с сетями процессов напоминает SDL.

## 5.2.11. Сети процессов (Process Networks – PN)

В домене сетей процессов (PN) процессы взаимодействуют с помощью сообщений, передаваемых через каналы с буферизацией. Отправителю сообщения не требуется ждать, пока получатель будет готов к приему сообщения. Этот стиль коммуникации называется асинхронной передачей сообщений. Существует несколько вариантов этой техники, но домен PN использует ту, которая гарантирует детерминированность вычислений. Она называется сетями процессов Кана. В модели вычислений PN дуги представляют последовательности значений (токенов), а сущности играют роль функций, преображающих входные последовательности в выходные. Несколько технических ограничений на эти функции гарантируют детерминированность, в том смысле что последовательности токенов строго определены. В частности, функции реализуемые сущностями должны быть префикс монотонными. Домен PN реализует подкласс таких функций,

описанный Каном и МакКвином [Kahn and MacQueen], где блокирующее чтение гарантирует монотонность. Модели PN слабо связаны, что дает возможность сравнительно просто распараллеливать и распределять их. Они могут быть одинаково эффективно реализованы в виде ПО и аппаратуры, что делает вопрос реализации открытым. Основной слабостью моделей PN является то, что они не подходят для описания логики управления, хотя могут быть улучшены комбинированием их с моделями FSM.

### 5.2.12. Гибридные системы

Гибридные системы это системы, которые совмещают в себе динамику непрерывного времени с дискретными переходами между состояниями. Строго говоря, гибридные системы не являются доменом в Ptolemy II, а скорее комбинацией из доменов. Гибридные системы конструируются в Ptolemy II с помощью иерархического вкладывания домена с непрерывным временем в домен FSM, DE или GR. Пример гибридной системы показан на рис 1.7. В этой модели

две массы закреплены на пружинах. Исполнение модели начинается с сжатия и разжатия пружин, так что массы совершают колебания. Когда массы сталкиваются, происходит дискретный переход и физика модели меняется. Это изменение отражено на переходе между состояниями в автомате, показанном на рис 1.8. Состояния этого автомата соответствуют физическим моделям с непрерывным временем, как показано на рис 1.9. Всего в этой модели используется четыре различных домена Ptolemy II.

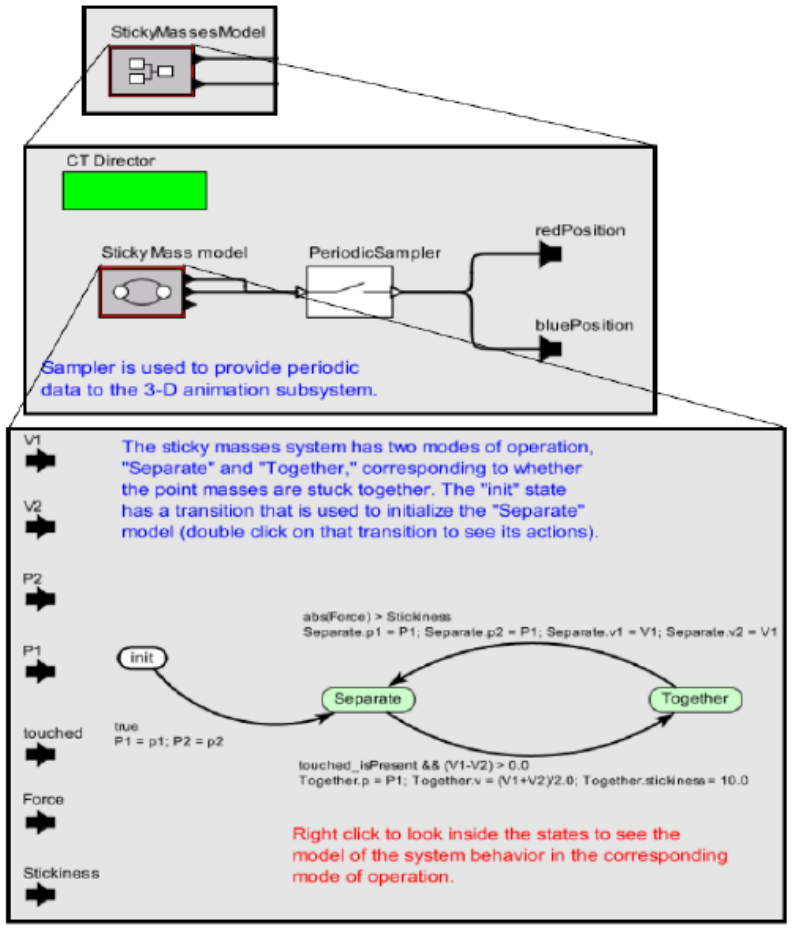



Рисунок 1. Раскрытие блоков обнаруживает модели CT и FSM внутри модели DE.

Рис. 5.1. Раскрытие блоков обнаруживает модели CT и FSM внутри модели DE.

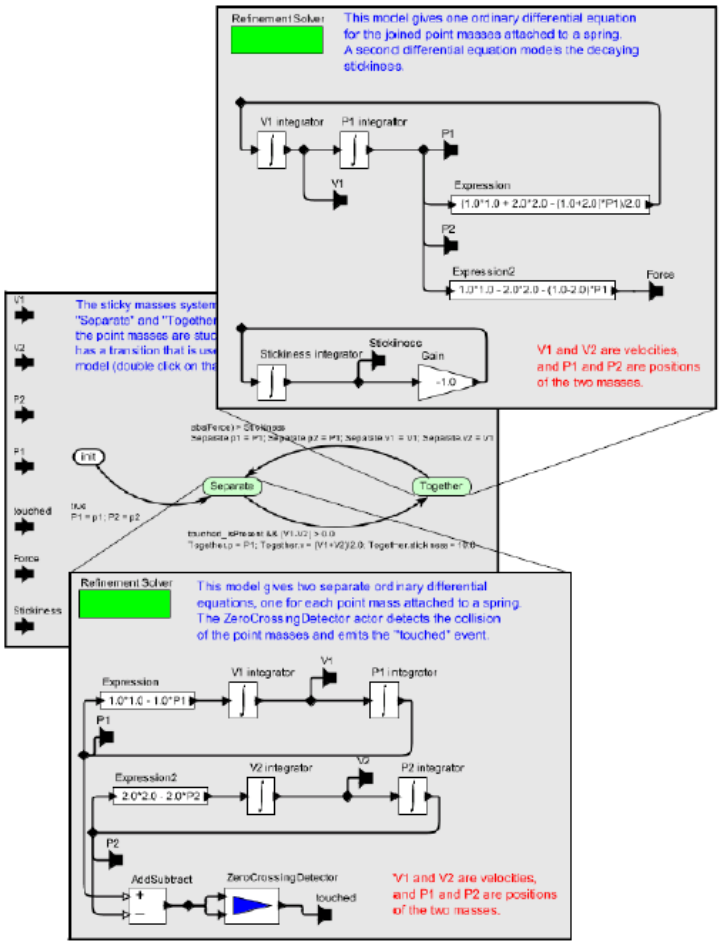



**Рисунок 2. Внутри состояний конченого автомата находятся модели описывающие дифференциальные соотношения.**

Рис. 5.2. Внутри состояний конченого автомата находятся модели описывающие дифференциальные соотношения.

## 5.3. Назначение пакета программ Ptolemy II

Пакет программ Ptolemy II – попытка создать систему автоматизированного проектирования, учитывающую особенности встроенных систем. В рамках Ptolemy II введена в САПР для встроенных вычислительных систем концепция моделей вычисления. Ptolemy II – перспективное средство проектирования.

Описание пакета программ Ptolemy II

### 5.3.1. Актор-ориентированное проектирование

Большинство из моделей вычислений в Ptolemy II поддерживают актор-ориентированное проектирование. В отличие от объектно-ориентированного проектирования, оно акцентирует внимание на параллельности происходящих в модели процессов и взаимодействии между элементами модели. Элементы модели, называемые акторами, выполняются и взаимодействуют с другими акторами в модели. Как и у объектов, у акторов есть четко определенный интерфейс. Этот интерфейс абстрагирует внутреннее

состояние актора и устанавливает правила взаимодействия актора с окружающей средой. Интерфейс включает в себя порты, которые являются точками соединения с актором, и параметры, которые используются для настройки поведения актора. Часто, но не всегда, значения параметров заранее определены и не меняются в ходе исполнения модели. “Порты/параметры”, показанные на рисунке ниже являются одновременно и портами и параметрами модели.

Важным элементом актор-ориентированного проектирования являются коммуникационные каналы, которые передают данные из одного порта в другой, в соответствии с некоторой схемой передачи сообщений. В то время как в объектно-ориентированном проектировании компоненты взаимодействуют в основном с помощью передачи управления через вызовы методов, в актор-ориентированном проектировании они взаимодействуют с помощью передачи сообщений по каналам. Использование каналов означает,

что акторы взаимодействуют только лишь с подключенными к ним каналами, а не напрямую с другими акторами.

Как и в акторах, в моделях также может быть определен внешний интерфейс; этот интерфейс называется иерархической абстракцией. Интерфейс состоит из внешних портов и внешних параметров модели, не являющихся портами и параметрами отдельных акторов в модели. Внешние порты модели могут быть соединены с другими внешними портами модели или с портами акторов из которых состоит модель. Внешние параметры модели могут быть использованы, чтобы определить значения параметров акторов внутри модели.

Встроенное программное обеспечение – это программное обеспечение, которое находится в устройствах, не являющихся типичными персональными компьютерами. Оно распространено повсюду: в автомобилях, телефонах, бытовой электронике, игрушках, авиационной технике, поездах, охранных системах,

оружейных системах, принтерах, модемах, копирах, термостатах, производственной технике, научных приборах и др..

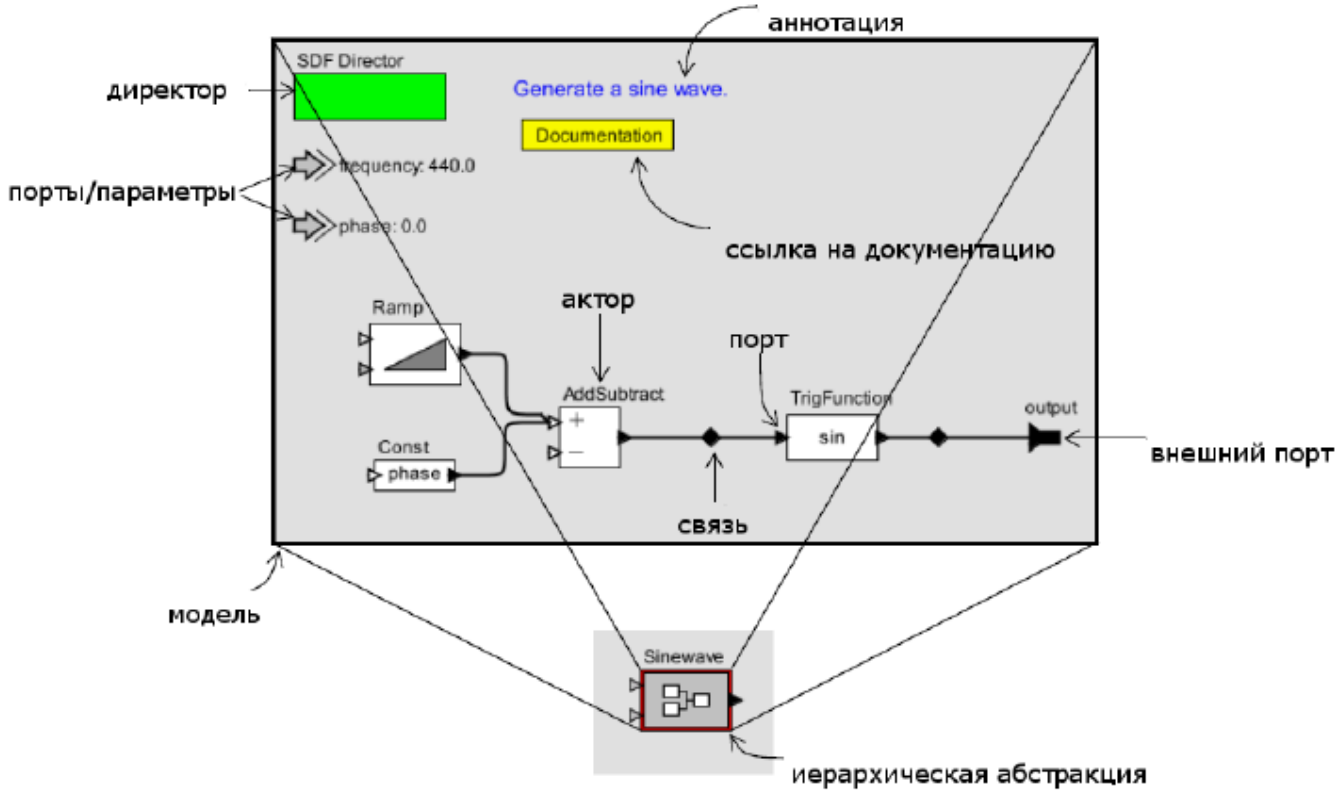



Рисунок 3. Иллюстрация актор-ориентированной модели и принципа иерархической абстракции.

Рис. 5.3. Иллюстрация актор-ориентированной модели и принципа иерархической абстракции

Взятые вместе, концепции моделей, акторов, портов, параметров и каналов описывают абстрактный синтаксис актор-ориентированного проекта. Абстрактный синтаксис определяет структуру моделей, не говоря ничего о том, как они исполняются. Этот синтаксис может быть представлен несколькими способами: в графическом виде, в виде XML или в виде программы использующей специальный API (как в SystemC). Ptolemy II предлагает все три альтернативы. Важно понимать, что синтаксическая структура актор-ориентированного проекта мало говорит о семантике. Семантика во многом перпендикулярна синтаксису, и определяется моделью вычислений. Модель вычислений должна описывать правила взаимодействия, по которым исполняется модель. Эти правила определяют, когда акторы производят свои внутренние вычисления, обновляют свое внутреннее состояние и производят внешние взаимодействия.

Модель вычислений также определяет природу взаимодействий между элементами.

Термин актор-ориентированное моделирование был позаимствован из работ Гула Агха (Gul Agha) и др. Термин актор был впервые употреблен в 1970‘‘х Карлом Хьюитом (Carl Hewit) из MIT чтобы описать концепцию автономных "мыслящих" исполнительных устройств [autonomous reasoning agents]. Термин был развит в работах Агха чтобы описать формализованную модель параллельности. Каждый из акторов Агха имел независимый поток управления и взаимодействовал с другими акторами через асинхронную передачу сообщений. В проекте Ptolemy проведено дальнейшее развитие этого термина, чтобы применить его к широкому набору различных моделей параллельности, которые, как правило, являются более строгими, чем простая передача сообщений. Акторы в Ptolemy II концептуально также параллельны, но, в отличие от акторов Агха, им не требуется собственный поток управления.

Более того, хотя взаимодействие между акторами и производится с помощью передачи сообщений, вовсе не обязательно, чтобы оно было строго асинхронным. Актор-ориентированное моделирование существует уже достаточно давно и широко используется в таких системах как Simulink от The Mathworks, LabVIEW от National Instruments и множестве других. Оно получило еще большее признание благодаря появлению OMG в UML-2, корни которого лежат в актор-ориентированной методологии ROOM. Множество исследовательских проектов базируются на различных формах актор-ориентированного моделирования, но проект Ptolemy уникален по широте исследуемых семантик и проработанности в рамках конкретной модели вычислений в каждом домене. Так как домены могут иерархично смешиваться, каждый домен может быть узко специализирован, не теряя при этом своей практичности.

## 5.4. Состав пакета программ Ptolemy II

Ptolemy II включает в себя графическую среду разработки Vergil, позволяющую схематично представлять модели, аннотировать и конфигурировать их, проверять их корректность и запускать на исполнение.

Vergil включает редактор моделей в акторном представлении (Graph editor), редактор конечных автоматов, текстовый редактор, редактор иконок акторов, позволяет просматривать HTML-страницы и редактировать текст файлов.

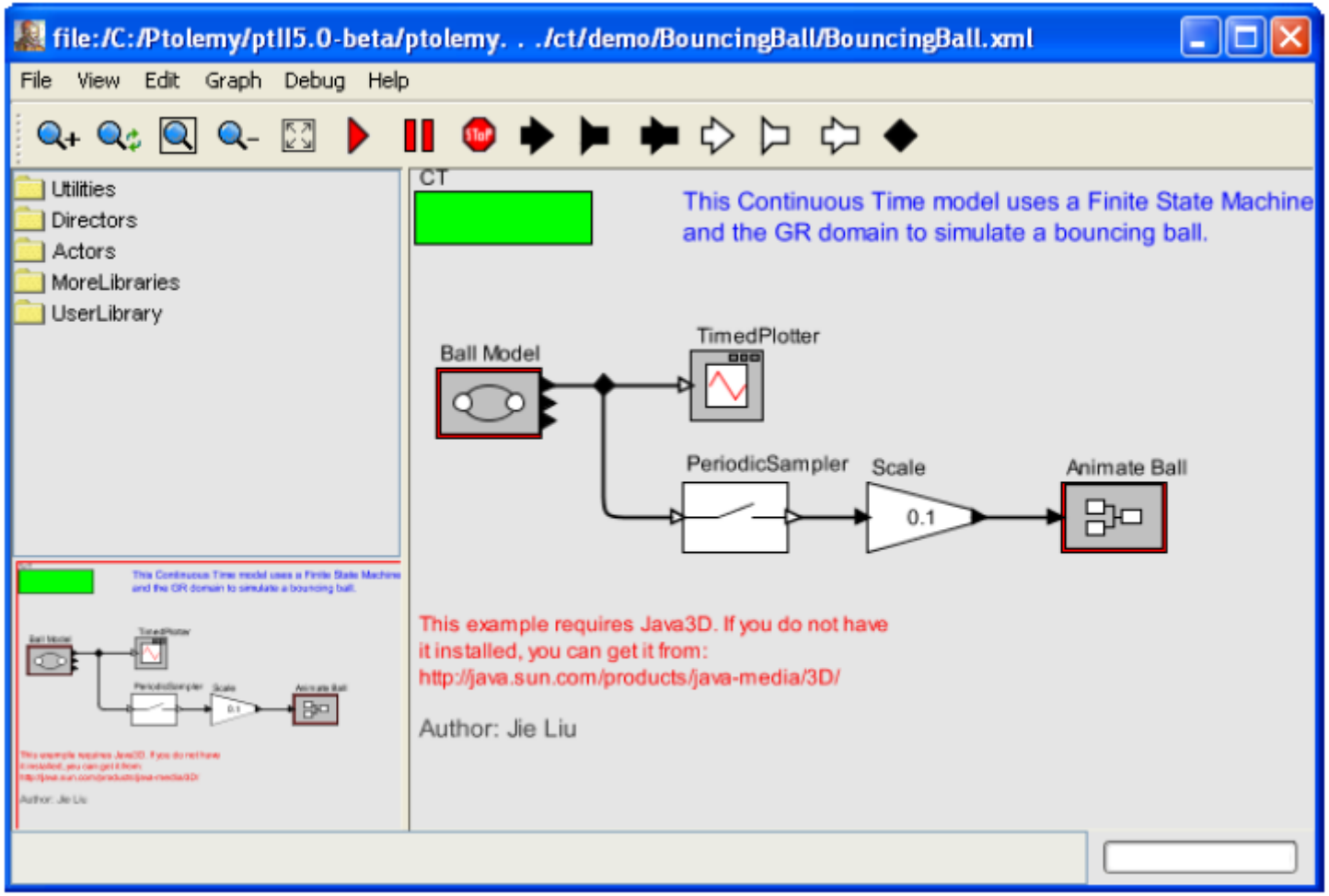



Рисунок 4. Редактор моделей Vergil

Рис. 5.4. Редактор моделей Vergil

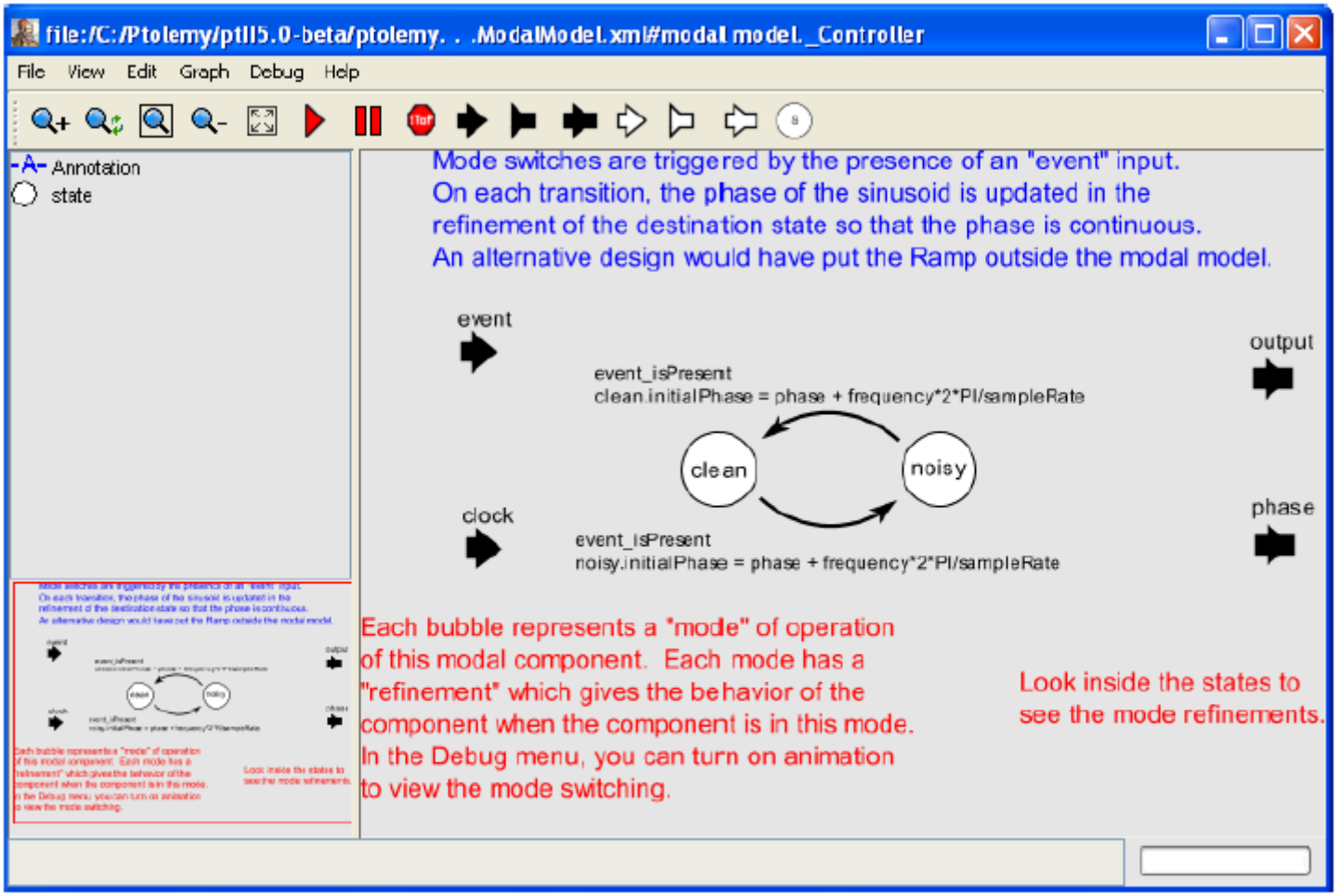



Рисунок 5. Редактор конечных автоматов Vergil

Рис. 5.5. Редактор конечных автоматов Vergil

### 5.4.1. Моделирование моделей вычислений

Ptolemy II позволяет моделирование вычислительных систем различной природы. На основе комплекса созданы специализированные подсистемы для моделирования: *беспроводных сетей и распределенных систем*, объединенных радио-, оптическими или акустическими каналами связи (пакет VisualSense); *гибридных систем*, объединяющих в себе дискретную логику управления и модели с непрерывным временем. Пакет HyVisual включает в себя модели вычислений FSM (finite state machines) и CT (continuous-time), библиотеку акторов, поддерживающих эти модели вычислений и средства визуализации моделей. Каждая подсистема представляет собой подмножество классов, механизмов и акторов, объединенное одной управляющей программой.

Ptolemy II создан для моделирования гетерогенных вычислительных систем, то есть систем, объединяющих компоненты, описываемые в разных моделях вычислений:

дискретную и аналоговую технику, сети и телекоммуникационные протоколы, стохастические и криптографические системы, радиокомпоненты, компоненты, реализующие обработку сигналов и изображений и т.д. Для этого комплекс поддерживает моделирование в рамках различных моделей вычислений, и объединение отдельных моделей в гетерогенную иерархию с последующим комплексным моделированием. В настоящее время комплекс поддерживает следующие модели вычислений:

1. CT (continuous time) – модели с непрерывным временем;
2. DE (discrete-event) – модели с дискретными событиями;
3. DDE (distributed discrete events) – системы с частично упорядоченным множеством дискретных событий;
4. FSM (finite-state machines) – конечные автоматы, в модели отсутствует понятие времени;
5. PN (process networks) – сети процессов Кана (Kahn), в модели отсутствует понятие времени;

6. SDF (synchronous dataflow) – модели для представления систем обработки сигналов, в моделях отсутствует понятие времени;

7. DDF (dynamic dataflow) – расширение SDF, позволяющее компонентам изменять количество потребляемых и генерируемых элементов данных за одну итерацию в процессе исполнения модели;

8. HDF (heterogeneous dataflow) – расширение SDF, позволяющее сохранить статическое планирование и другие свойства моделей в ее рамках, и избегать ограничений на неизменность количества потребляемых и генерируемых компонентами элементов данных;

9. PSDF (parameterized synchronous dataflow) – расширение SDF с параметризируемой дисциплиной планирования вычислений;

10. DT (discrete time) – периодически запускаемые статически планируемые процессы с дискретным временем. Аналог SDF, в котором присутствует понятие времени;

11. SR (synchronous-reactive) – синхронно-реактивная модель вычислений;

12. CI (component interaction) – модель вычислений со стилем взаимодействия компонент «push/pull»;

13. CSP (communicating sequential processes) – взаимодействующие последовательно выполняемые процессы;

14. Giotto – поддержка одноименного языка и его семантики. Один из вариантов синхронной модели, в котором компоненты могут активироваться с разной частотой;

15. TM (timed multitasking) – вариант взаимодействия задач ОСРВ;

16. Wireless – модель вычислений для исследования систем, объединенных беспроводными каналами связи;

17. GR – поддержка 3-D графики.

Компонентами моделей Ptolemy II по большей части являются акторы, атомарные функциональные преобразователи сигналов, поведение которых описано на Java. Каждый такой актор представляет собой Java-класс, который может быть в процессе

моделирования расширен и дополнен различными механизмами Ptolemy II.

Акторы могут быть составными, то есть являться контейнерами других моделей, обеспечивая таким образом их иерархичность. Комплекс включает обширную библиотеку акторов, систематизированную по группам выполняемых ими преобразований. Большинство акторов стандартной библиотеки Ptolemy II обладает полиморфизмом функциональности относительно модели вычислений, в которой они используются. Этот механизм является одним из базовых механизмов Ptolemy II и обеспечивает широкие возможности поведенческого описания в рамках почти любой модели вычислений пользователем в графическом редакторе Vergil, не прибегая к кодированию на Java или существенной модификации стандартных акторов.

Еще одним базовым механизмом является полиморфизм функциональности акторов относительно типа преобразуемых

данных. Большинство акторов могут реализовывать свое поведение над сигналами различных типов. Этот механизм в значительной степени опирается на существующую в Ptolemy II систему типов и механизм их разрешения при инициализации модели. Он реализован за счет построения иерархии классов, которые представляют элементы данных различных типов, и реализации множества операций, смысл которых инвариантен к типу данных.

Типы данных, с которыми может работать модель в Ptolemy II, организованы в систему, которая сравнима по сложности с системой типов C++ или Java. Помимо простых типов данных (int, double, long, unsignedByte, complex, fixedpoint, boolean, string, scalar, matrix и т.д.), существуют составные типы (массивы, матрицы, записи и объекты). Компоненты модели, то есть акторы, обмениваются друг с другом в процессе ее исполнения элементами данных, имеющими тот или иной тип. С помощью встроенного в Ptolemy II языка

выражений и механизма разрешения типов данные, имеющие один тип, могут быть явно или неявно преобразованы к другому типу.

Массивы в Ptolemy II определяются как упорядоченные наборы данных одного типа, имеющие одно измерение. Матрицы представляют собой двумерные упорядоченные наборы данных одного типа. Ptolemy II поддерживает матрицы, состоящие из элементов следующих типов: *boolean*, *complex*, *double*, *fixedpoint*, *int* и *long*. Другие типы элементов в матрицах не разрешены. Для матриц предусмотрено множество операций векторно-матричной алгебры и функций, практически все из которых присутствуют в MATLAB (MathWorks Ltd.). Записи в Ptolemy II состоят из именованных полей, которые могут иметь разные типы. Доступ к отдельным полям осуществляется с помощью операции „.“ (как в C++ или Java). Интересно отметить, что для данных, имеющих тип «запись», предусмотрены бинарные операции пересечения и объединения по именам полей.

Механизм разрешения типов Ptolemy II позволяет автоматически проверять корректность преобразований данных в модели в целом, не указывая вручную конкретные типы или ограничения на порты отдельных акторов. Разрешение типов происходит в процессе

инициализации модели и устанавливает для каждого порта актора конкретный тип данных, которые он воспринимает, исходя из информации о зависимостях между типами данных портов, декларируемой акторами. Ограничения, накладываемые на порт могут быть менее строгими, чем указание его конкретного типа, благодаря возможности преобразования данных одних типов к другим. Например, ограничение на порт может заключаться в том, что тип данных, которые он воспринимает, может быть больше или равен конкретному типу. В случае несоответствия ограничениям, то есть невозможности разрешения типов, инициализация модели завершается с ошибкой. Ошибка может быть устранена вручную.

В Ptolemy II разработан довольно мощный язык выражений, позволяющий не только выполнять математические операции над данными, получая значение результата, но и создавать переменные различных типов, в том числе и произвольных составных, выполнять преобразования типов, выполнять сложные встроенные процедуры обработки данных и создавать собственные, производить ввод/вывод и управлять вычислительным процессом.

В языке выражений предусмотрен набор стандартных именованных констант, которые могут быть использованы в выражениях. Это числовые константы: *PI*, *pi*, *E*, *e*, *true*, *false*, *i*, *j*, *NaN*, *Infinity*, *PositiveInfinity*, *NegativeInfinity*, *MaxUnsignedByte*, *MinUnsignedByte*, *MaxInt*, *MinInt*, *MaxLong*, *MinLong*, *MaxDouble*, *MinDouble*; и строковые константы *PTII*, *HOME* и *CWD*. Эти константы имеют заранее определенное значение. Константы с другими значениями могут быть определены с помощью литералов, задающих как значение, так и (либо явно, либо неявно) тип

константы. Например, *2.0* даст константу типа *double*, *2i* (или *2j*) даст константу, имеющую тип *complex*. Константы составных типов также задаются с помощью литералов:

1. массивы с использованием фигурных скобок с элементами, разделяемыми запятыми, например: {2, 3, 4, 5};

2. матрицы задаются с помощью квадратных скобок и разделения столбцов запятыми, а строк знаками „;“, например: [1,2,3; 4,5,6] – матрица из двух строк и трех столбцов. В языке выражений также поддержан способ задания матриц в стиле MATLAB ([*p*:*q*:*r*]);

3. записи задаются с помощью фигурных скобок, в которых перечислены имена полей и их значения, например: {first_name = ”Edward”, data = {0, 1, 2}, associated_record = {a = 1, b = 2 + PI * j}}.

В языке выражений предусмотрено задание переменных и функций с произвольными именами. Операции, предусмотренные в языке выражений, включают как алгебраические (над числами, строками, массивами, записями и матрицами), так и логические,

сравнения, побитовые (&,|,~,#,>>,<<), преобразования типов и условного вычисления („ ? : “). Также в языке предусмотрены комментарии в стиле C++ (/* */). Достаточно объемная библиотека стандартных функций, включающая также и статические методы стандартных классов

Java (java.lang.Math, java.lang.Double и т.п.), позволяет реализовать достаточно сложные вычисления. Выражения могут быть использованы в редакторе выражений Vergil, при задании значений параметров, в отдельных акторах, а также в качестве элементов данных.

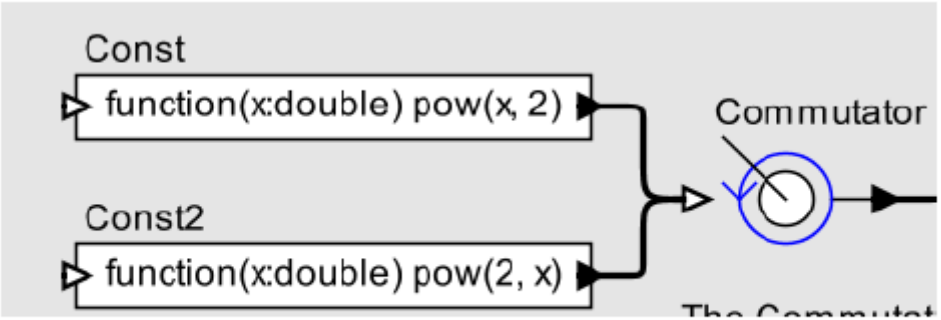



Рисунок 6. Использование языка выражений в акторах

Рис. 5.6. Использование языка выражений в акторах

В Ptolemy II существует группа акторов, называемая «компонентами высшего порядка» (Higher-Order Components, HOC). Эти акторы выполняют действия над структурой и функциональной организацией модели в процессе ее исполнения. Примеры акторов этого типа:

1. ModelReference – актор, представляющий собой ссылку на модель, заданную с помощью URL. Активация этого актора представляет собой команду на исполнение модели, на которую он ссылается;

2. MultiInstanceComposite – составной актор, создающий заданное количество копий самого себя в процессе инициализации модели;

3. ModalModel – актор, реализующий модель, которая может работать в нескольких «режимах». Каждый такой «режим» представляет собой произвольную модель с теми же портами, что у ModalModel, а переход между ними осуществляется с помощью

конечного автомата. Актор может быть использован для моделирования гибридных систем (дискретная логика в сочетании с непрерывной динамикой).

В Ptolemy II реализован механизм объявления и наследования классов, реализующий механизмы из объектно-ориентированного подхода для составных акторов. Составной актор может реализовываться в графическом редакторе в рамках какой-либо модели вычислений, а затем может быть преобразован в описание «класса». Экземпляры этого «класса» затем могут быть использованы в модели как обычные составные акторы. Любые изменения в описании класса ведут к изменениям в его экземплярах. Этот подход удобен, когда в модели верхнего уровня иерархии используется несколько однотипных компонент, преобразующих различные потоки данных.

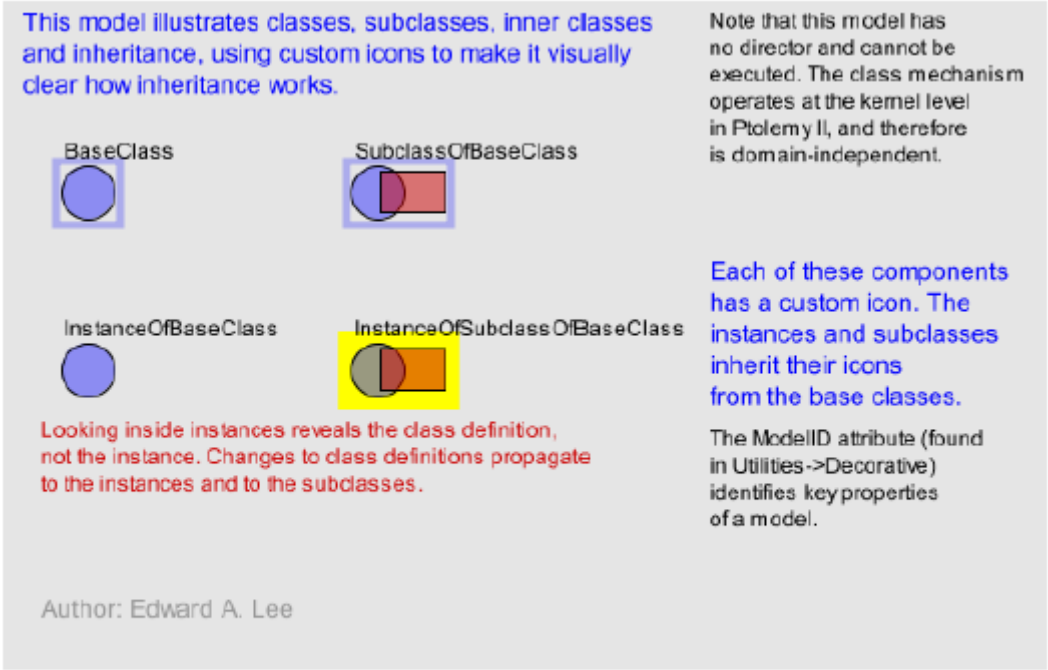



**Рисунок 7. Классы составных акторов**

Рис. 5.7. Классы составных акторов

Кроме того, механизм позволяет строить иерархию «классов», «наследуя» от базовых «классов» все элементы модели нижнего уровня, добавляя в них новые компоненты и «перегружая» часть наследуемых. Реализация Ptolemy II на базе Java™ позволяет пользоваться всеми возможностями, которые предоставляет эта

платформа. В частности, стандартная библиотека Ptolemy II включает акторы, позволяющие:

1. пользовательский ввод и интерактивное управление моделированием и структурой модели в процессе ее исполнения;
2. ввод, вывод и обработку сигналов и изображений, включая преобразование видеосигналов в реальном масштабе времени;
3. файловый ввод-вывод и доступ к устройствам инструментального ПК;
4. доступ к ресурсам сети Internet и распределенное моделирование;
5. интеграцию с другими инструментами и языками.

Модели в Ptolemy II представляют собой Java-приложения, выполнением которых управляет один из специально созданных классов. В зависимости от используемых моделей вычислений, приложения могут быть как однопотоковыми, так и

многопотоковыми (например, при использовании модели вычислений PN). Такой подход позволяет гибко использовать как сами модели, так и предоставляемые ими результаты. Например, можно получить готовую отлаженную модель, включающую средства отображения результатов моделирования, и оформить ее в виде самостоятельной программы (отдельного исполняемого модуля). Или можно создать Java-апплет, и опубликовать в Internet, обеспечив возможность удаленной загрузки готовой модели, ввода исходных данных и получения результатов моделирования на любом ПК, в котором реализована поддержка Java™.

Наконец, можно исключить весь графический ввод-вывод модели, существенно снизив вычислительную нагрузку на операционную систему, и запустить модель в виде

консольной программы Java, обеспечив вывод результатов моделирования в файл. Этот подход удобен, если объем вычислений

и время моделирования существенны, в отличие от анимации и диалога с пользователем.

Комплекс Ptolemy II является open-source-проектом, что способствует распространению и широкому использованию подходов и механизмов, реализованных в нем. Комплекс является масштабируемым практически в любом аспекте, включая поддержку новых моделей вычислений. Исходный текст комплекса оформлен аккуратно и в соответствии с принятым в проекте стилем кодирования, что позволяет легко разобраться в исходном коде любого Java-класса. Кроме того, весь исходный код документирован с помощью утилиты Javadoc, что позволяет, избегая тщательного анализа исходных текстов, быстро перемещаться по систематизированной документации в поиске требуемой информации о том или классе. Комплекс снабжен тремя томами подробного описания всех возможностей, реализованных механизмов и моделей вычислений, а также техники моделирования

и рекомендациями по применению и расширению возможностей комплекса Ptolemy II. Лицензия Ptolemy II предоставляет право свободного применения всего или частей комплекса в любых разработках, включая коммерческие продукты.

### 5.4.2. Актор-ориентированные классы, подклассы и наследование

Начиная с версии 4.0 Ptolemy II технологии актор-ориентированного проектирования были расширены модульными механизмами, аналоги которых можно найти в объектно-ориентированных языках. Рассмотрим простой пример, показанный на Рис. 5.8. Модель в левой нижней части рисунка является тем же генератором синусоидального сигнала что и на Рис. 5.3. На Рис. 5.3 блок, обозначенный “Sinewave” на самом деле является экземпляром класса, определение которого дано на блок-схеме выше. На Рисунок 8 определение этого класса расширено, чтобы создать определение подкласса названного “NoisySinewave”, показанное в правой нижней части рисунка. Этот подкласс “наследует” элементы (акторы) и

соединения от базового класса. Унаследованные компоненты обведены розовой линией. Также он перегружает определение базового класса, добавляя второй выходной порт, два дополнительных актора и соединения между ними. Эти добавочные элементы не имеют обводки.

Определение класса NoisySinewave является локальным для модели на Рис. 5.8. Это означает, что определение класса является частью определения модели и доступно для создания экземпляров и наследования только внутри модели. Определение класса обведено голубой линией, чтобы графически отделить его от экземпляров этого класса.

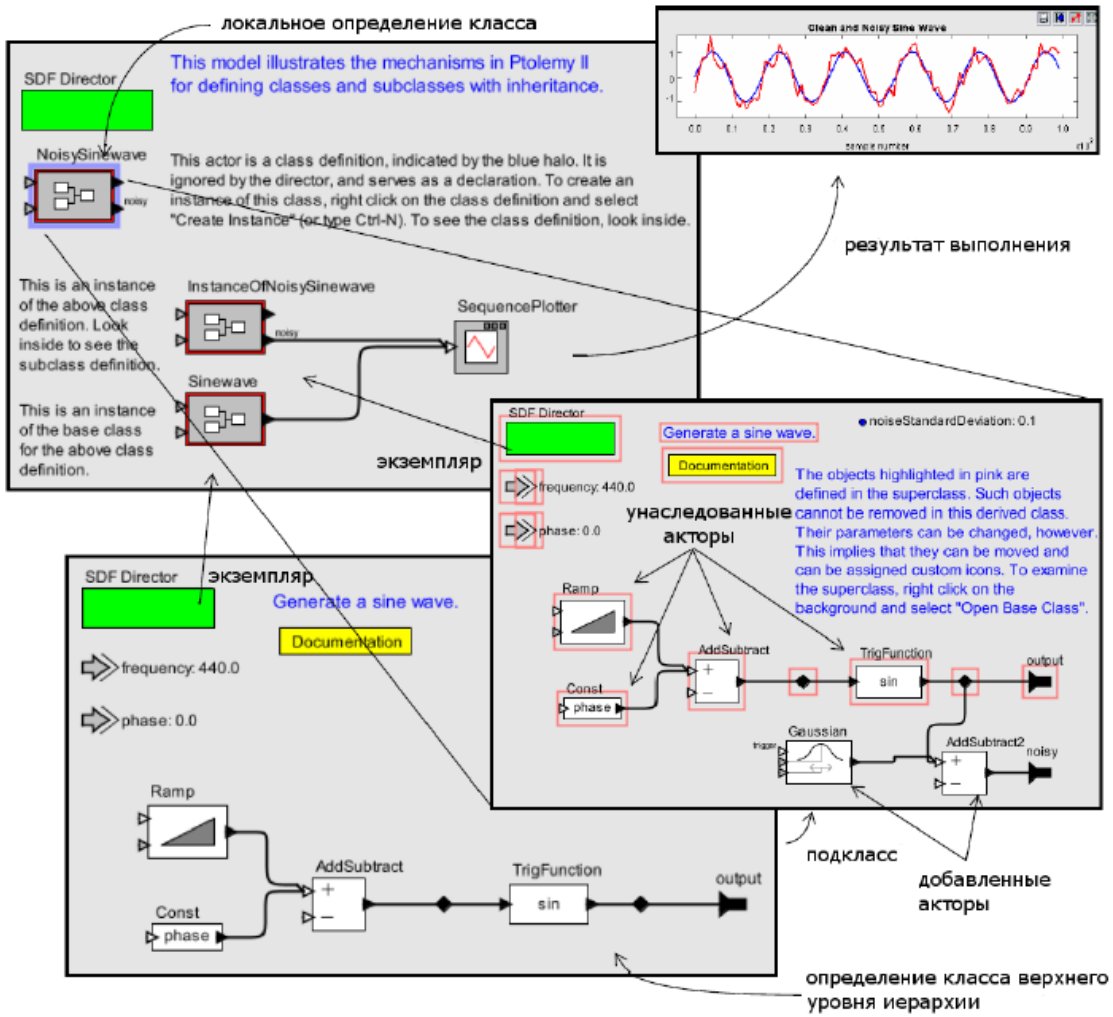



Рисунок 8. Иллюстрация актор-ориентированных классов, подклассов и наследования

Рис. 5.8. Иллюстрация актор-ориентированных классов, подклассов и наследования

Класс Sinewave, в отличие от NoisySinewave, определен в отдельном файле и доступен в любой модели. Таким образом, Ptolemy II предоставляет области видимости для определений классов. Более того, определения классов могут сами содержать определения других классов, предоставляя, таким образом, некоторое подобие “вложенных классов”.

Класс, определенный в своем собственном файле, называется классом верхнего уровня. Любая модель может содержать экземпляры и подклассы класса верхнего уровня, если файл, содержащий этот класс, находится в CLASSPATH (переменной окружения, которая указывает путь к классам языка Java). Класс, являющийся элементом модели, называется локальным классом. Класс Sinewave на Рисунок 8 является классом верхнего уровня, в то время как NoisySinewave – локальный класс. Когда класс определен внутри модели, он доступен внутри самой модели и на всех уровнях иерархии, низших по отношению к тому, в котором он определен.

Поэтому, модель в верхней части Рисунок 8 содержит одновременно и определение класса NoisySinewave и его экземпляр InstanceOfNoisySinewave.

Модель на Рис. 5.8 во время выполнения строит графики двух сигналов, как показано на диаграмме в правой верхней части рисунка. Первый – простой синусоидальный сигнал, а второй – синусоидальный сигнал с помехами. Простой синусоидальный сигнал генерируется актором Sinewave, который является экземпляром класса Sinewave (определенного в отдельном файле), а сигнал с помехами генерируется актором InstanceOfNosisySinewave, который является экземпляром класса NosisySinewave, подкласса Sinewave. Встраивая эти механизмы в Ptolemy II, разработчикам пришлось сделать несколько решений, которые по своей значимости могут считаться решениями в области дизайна языка.

Во-первых, в Ptolemy II модель – это набор акторов, портов, параметров и соединений.

Во-вторых, модель может быть представлена в виде программы с визуальным синтаксисом.

Каждая из трех блок-схем, представленных на рис. 3 является моделью, содержащей акторы, каналы связи и аннотации. В Ptolemy II любая модель может быть либо классом, либо экземпляром. Класс служит прототипом для экземпляров. Этот механизм очень похож на тот, что используется в основанных на прототипах языках [prototype-based languages], но со своими особенностями. Чтобы гарантировать, что механизм классов будет работать только на абстрактном синтаксическом уровне, классы в Ptolemy II являются чисто синтаксическими объектами и не влияют на то, как будет выполняться модель. Они не видимы для директора [director], задающего механизм исполнения модели. Как следствие, Ptolemy II запрещает соединять порты класса с другими портами, и попытка соединить прямоугольник, обозначенный NoisySinewave на рисунке 3приведет к возникновению ошибки.

Подкласс наследует структуру базового класса. Таким образом, каждый объект (актор, параметр, порт или канал связи) содержащиеся в базовом классе будет иметь соответствующий ему объект в подклассе. Мы называем это инвариантностью наследования. Такие “унаследованные объекты” на рисунке 3 обведены розовой линией. Эта обводка означает, что соответствующие объекты не могут быть удалены, т.к. это нарушило бы инвариантность наследования. Тем не менее, подклассы могут иметь новые объекты, а также изменять (перегружать) значения параметров базового класса. Так как модель может содержать определения классов и сама может быть определением класса, то

мы можем получить вложенные классы. Это создает некоторую форму множественного наследования.

### 5.4.3. Создание моделей в Ptolemy II

Модели в Ptolemy II могут быть созданы одним из трех способов. Визуальная нотация, как на Рис. 3, является наиболее

распространенной. Альтернативой является XML, как на Рис. 11, хотя этот формат не так удобен для ручного редактирования и создания программ. Третей альтернативой является использование API ядра Ptolemy II для написания исполняемых моделей на языке Java. Пример показан на Рис. 9. Этот метод является, безусловно, самым гибким, но большинство пользователей предпочитают визуальный синтаксис, т.к. модели, созданные графически, являются наиболее наглядными. Графические изображения систем хорошо воспринимаются людьми, что делает их незаменимыми для передачи информации о структуре системы. Многие из специализированных доменов в Ptolemy II используют такие изображения, чтобы формально и полностью описать модель.

*Один из принципов проекта Ptolemy в том, что графическое представление системы позволяет компенсировать все возрастающую сложность, появляющуюся в гетерогенном моделировании.*

```
public static void main(String[] args) {
  try {
    TypedCompositeActor top = new TypedCompositeActor();
    top.setName( "DiningPhilosophers");
    Manager manager = new Manager("Manager");
    top.setManager(manager);
    new CSPDirector(top, "CSPDirector");

    Parameter thinkingRate = new Parameter(top, "thinkingRate");
    thinkingRate.setToken("1.0");

    Parameter eatingRate = new Parameter(top, "eatingRate");
    eatingRate.setToken("1.0");


    Philosopher p1 = new Philosopher(top, "Aristotle");
    Philosopher p2 = new Philosopher(top, "Plato");
    Philosopher p4 = new Philosopher(top, "Descartes");
    Philosopher p3 = new Philosopher(top, "Sartre");
    Philosopher p5 = new Philosopher(top, "Socrates");

    Chopstick f1 = new Chopstick(top, "Chopstick1");
```

Рис. 5.9. Программа на Java, которая создает и исполняет модель (классическая проблема обедающих философов).

Графический синтаксис может быть таким же точным и полным, как и текстовый, особенно если два этих синтаксиса дополняют друг друга. На рисунках показаны две графические интерпретации моделей Ptolemy II. Они были построены с помощью Vergil, графической рабочей среды для Ptolemy II.

На рисунке модель Ptolemy II показана в виде блок схемы, которая является удачной интерпретацией для большинства моделей с дискретными событиями. В этом примере, в левой части схемы собираются сообщения, составленные из строк и чисел, представляющих порядковые номера сообщений. Затем сообщения отправляются в сеть, представленную в виде случайной задержки. Сообщения могут приходить из сети не по порядку, но актор Sequence в правой части модели будет использоваться для того, чтобы упорядочить их по порядковому номеру.

На рисунке также показано графическое представление модели Ptolemy II, но на этот раз элементы представлены в виде овалов, а

соединения между ними в виде маркированных дуг. Этот синтаксис является широко распространенным способом представления конечных автоматов (FSM, finite state machines). Каждый овал представляет состояние модели, а дуги отображают переходы между состояниями. Пример на рисунке пришел из моделирования гибридных систем, а два состояния: „Separate“ и „Together“ представляют два различных способа функционирования системы с непрерывным временем. Дуги обозначены двумя строками, первая из которых является условием перехода [guard], а вторая представляет действие [action]. Условие перехода это булево выражение которое определяет, когда должен быть совершен переход, а действие представляет последовательность команд, выполняемых при переходе от одного состояния к другому.

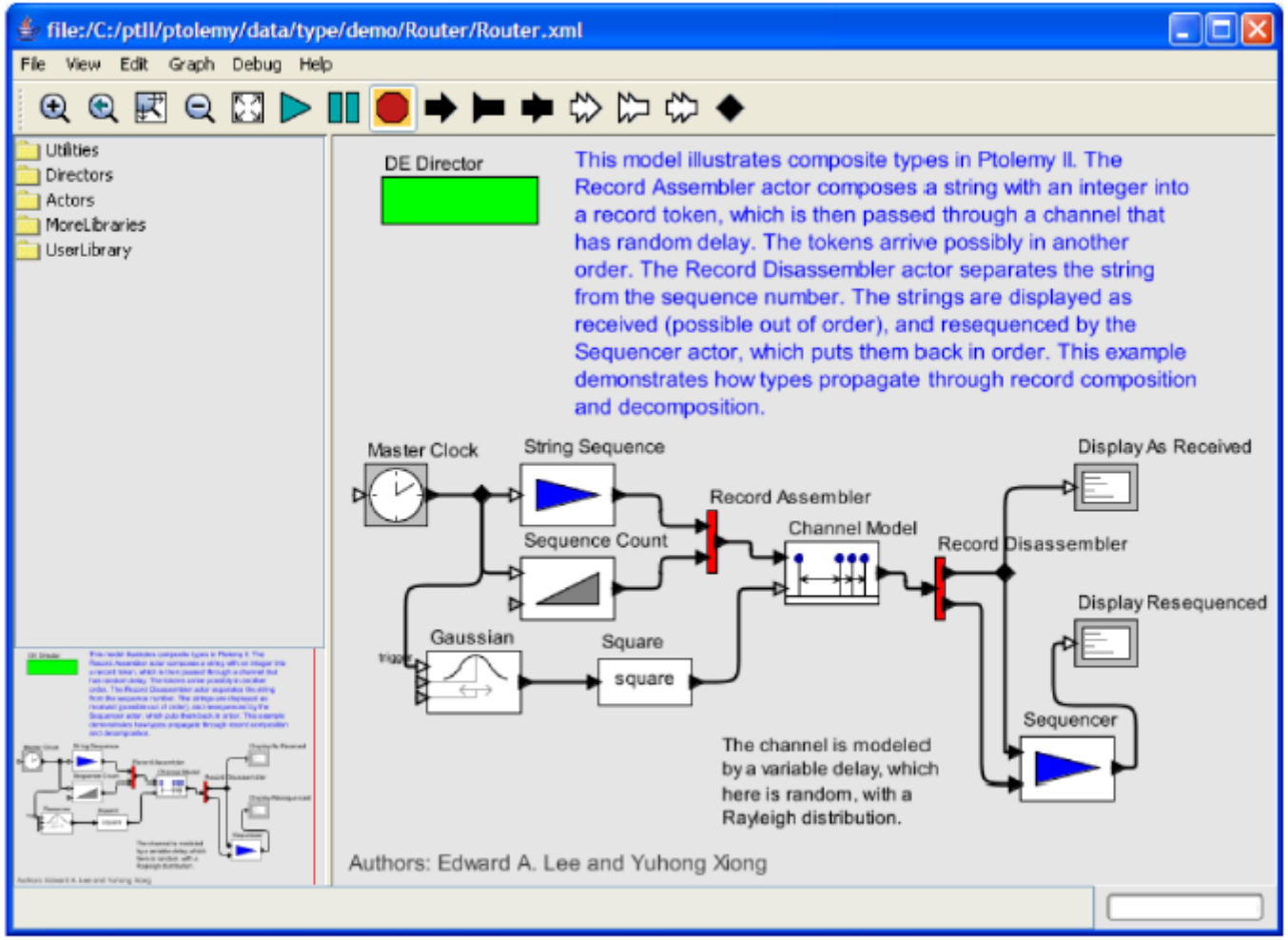



Рисунок 10. Графическое представление модели Ptolemy II в виде блок-схемы (домен DE)

Рис. 5.10. Графическое представление модели Ptolemy II в виде блок-схемы (домен DE)

Графические представления моделей имеют противоречивую историю. В схемотехнике принципиальные схемы повседневно используются для описания всей необходимой для реализации систем информации. Хотя в последнее время схемы часто заменяются текстом на языках описания аппаратуры, таких как VHDL, Verilog или SystemC. В других случаях использование графических представлений не стало столь популярным; так, к примеру, случилось с блок-схемами, описывающими алгоритмы программ. Подкласс визуальных языков, часто называемый “блок-схемами” [block diagrams] используется для представления параллельных систем. Существует множество возможных параллельных семантик (и множество моделей вычислений), связанных с такими схемами. Формализация этих семантик является задачей, которую необходимо решать при использовании таких схем в проектировании и спецификации реальных систем.

Ptolemy II позволяет исследовать множество параллельных семантик. Принцип проекта в том, что сильные стороны этих семантик должны дополнять друг друга. Поэтому обеспечение функциональной совместимости различных моделей является важной задачей.

### 5.4.4. Описание моделей в Ptolemy II (MoML)

Для описания моделей в Ptolemy II используется язык MoML. MoML представляет собой расширение XML, поддерживаемое Ptolemy II.

```
<class name="Sinewave">
   <property name="samplingFrequency" value="8000.0"/>
   <property name="frequency" value="440.0"/>
   <property name="phase" value="0.0"/>
   <property name="SDF Director"
class="ptolemy.domains.sdf.kernel.SDFDirector"/>
   <port name="output">
   <property name="output"/>
   <entity name="Ramp" class="ptolemy.actor.lib.Ramp">
      <property name="init" value="phase"/>
      <property name="step"
value="frequency*2*PI/samplingFrequency"/>
```

Рис. 5.11. XML-описание модели

## 5.5. Описание основных методик работы

### 5.5.1. Пример модели в Ptolemy II

Для демонстрации возможностей комплекса Ptolemy II подготовлена простая модель D-триггера, представляющего собой функциональный блок ОСМВ. Модель уже является гетерогенной, так как ОСМВ не допускает источников событий, а, следовательно, не может находиться на самом верхнем уровне иерархии. Верхний

уровень модели реализован в модели вычислений с дискретными событиями, в которой один из акторов реализован на нижнем уровне в рамках OCMB. На рисунке представлен верхний уровень модели в графическом редакторе моделей Ptolemy II Vergil. (Рис. 5.12).

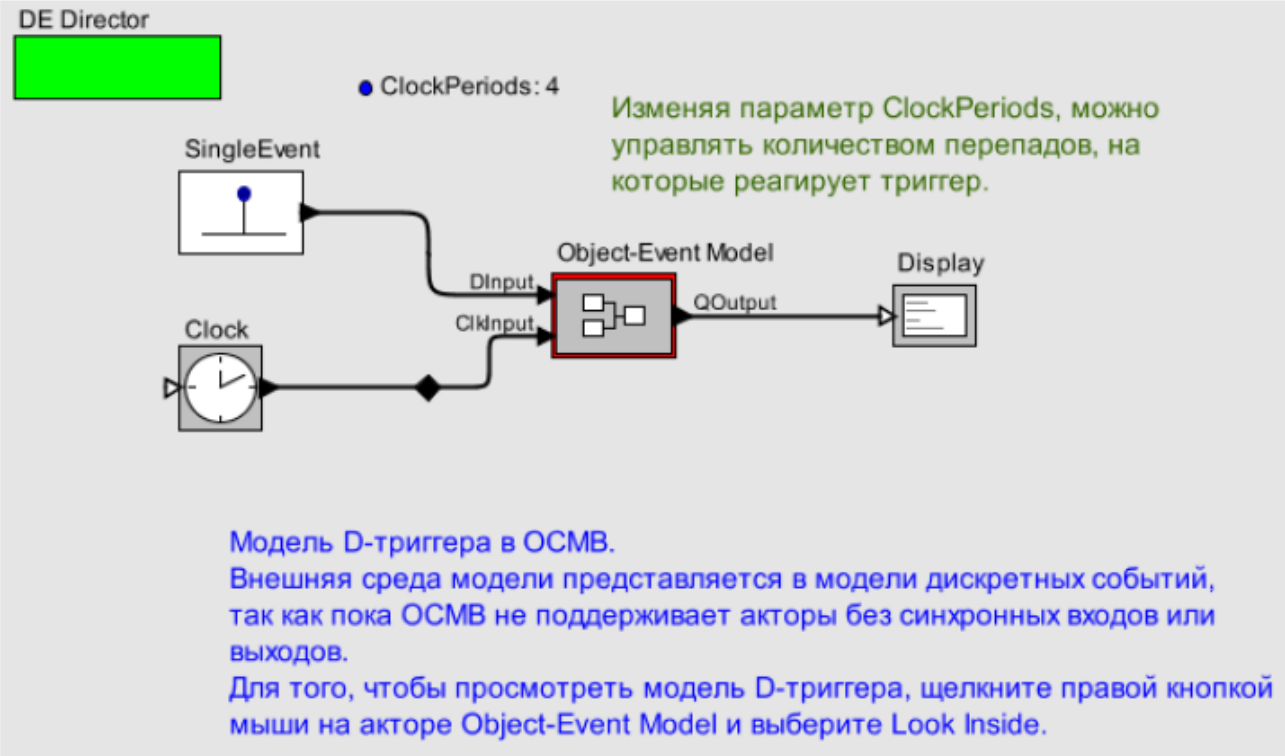



Рисунок 12. Верхний уровень гетерогенной модели

Рис. 5.12. Верхний уровень гетерогенной модели

Как видно из рисунка, модели можно снабжать различными декорационными элементами: аннотациями, для каждого актора задавать стандартную или собственную иконку (Vergil предоставляет простой редактор иконок). На рисунке не показаны другие декорационные элементы, однако Ptolemy II предоставляет несколько разнотипных элементов, которые по сути (то есть в смысле абстрактного синтаксиса модели) представляют собой атрибуты модели (кроме иконок акторов, которые являются атрибутами самих акторов).

Модель вычислений указывает специальный атрибут модели «*директор*», являющийся экземпляром одного из подклассов ptolemy.actor.Director. Экземпляры этого класса реализуют конкретную модель вычислений, управляя процессом имитационного моделирования в ее рамках, то есть инициализацией и активацией акторов, транспортом данных, продвижением модельного времени и синхронизацией с моделями вычислений

более высокого уровня. Модели нижних уровней иерархии, реализованные в других моделях вычислений, рассматриваются директором на данном уровне иерархии как непрозрачные составные акторы (opaque composite actor). (Рис. 5.13).

Параметр *offsets* соответственно задает смещения моментов от начала периода, когда генерируются события со значениями массива *values*. В представленном примере генерируются события с двумя значениями: *false* в самом начале периода и *true* спустя 1.0 единиц времени от начала периода. При том, что период составляет 2.0 единиц времени, расстояние между событиями (то есть разность временных меток) в генерируемой на выходе последовательности составляет 1.0 единиц времени. Время моделирования задается с помощью параметра модели ClockPeriods, который, в свою очередь, влияет на фактически управляющий временем моделирования параметр директора stopTime. В представленном примере время моделирования задается таким образом, чтобы за это время

произошло ClockPeriods «перепадов» false-true на выходе времязадающего актора.

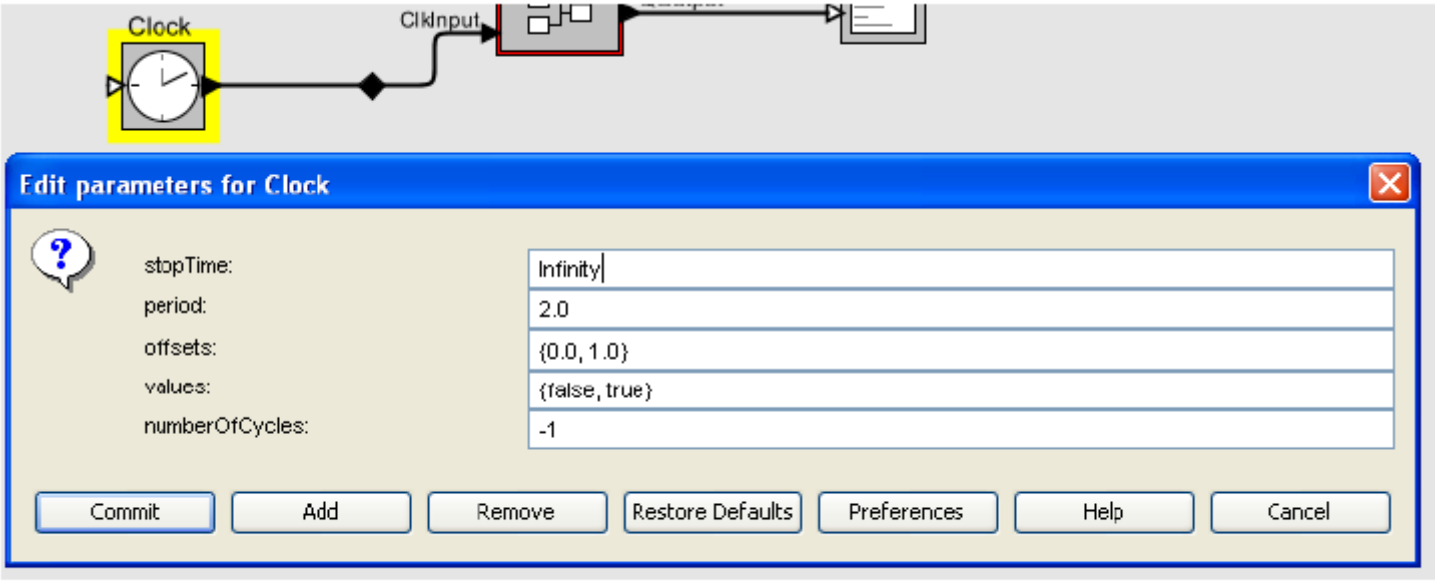



Рисунок 13. Конфигурирование акторов

Рис. 5.12. Конфигурирование акторов

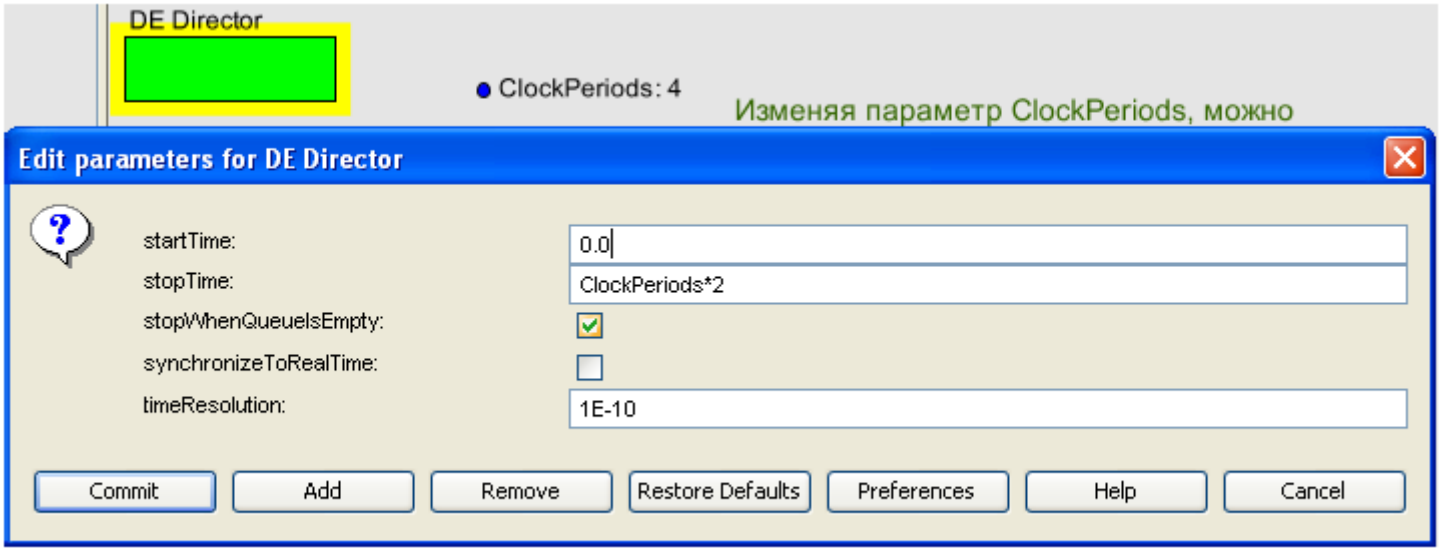



Рисунок 14. Параметры имитационного моделирования верхнего уровня

Рис. 5.14. Параметры имитационного моделирования верхнего уровня

DE Director имеет также другие параметры для управления моделированием, в частности, параметр *synchronizeToRealTime*, который управляет выполнением модели в реальном масштабе времени. Если он равен *true* (в Vergil это соответствует установленному флажку), то симулятор будет пытаться продвигать модельное время с учетом реального времени, предполагая по

умолчанию, что одна единица модельного времени соответствует одной секунде.

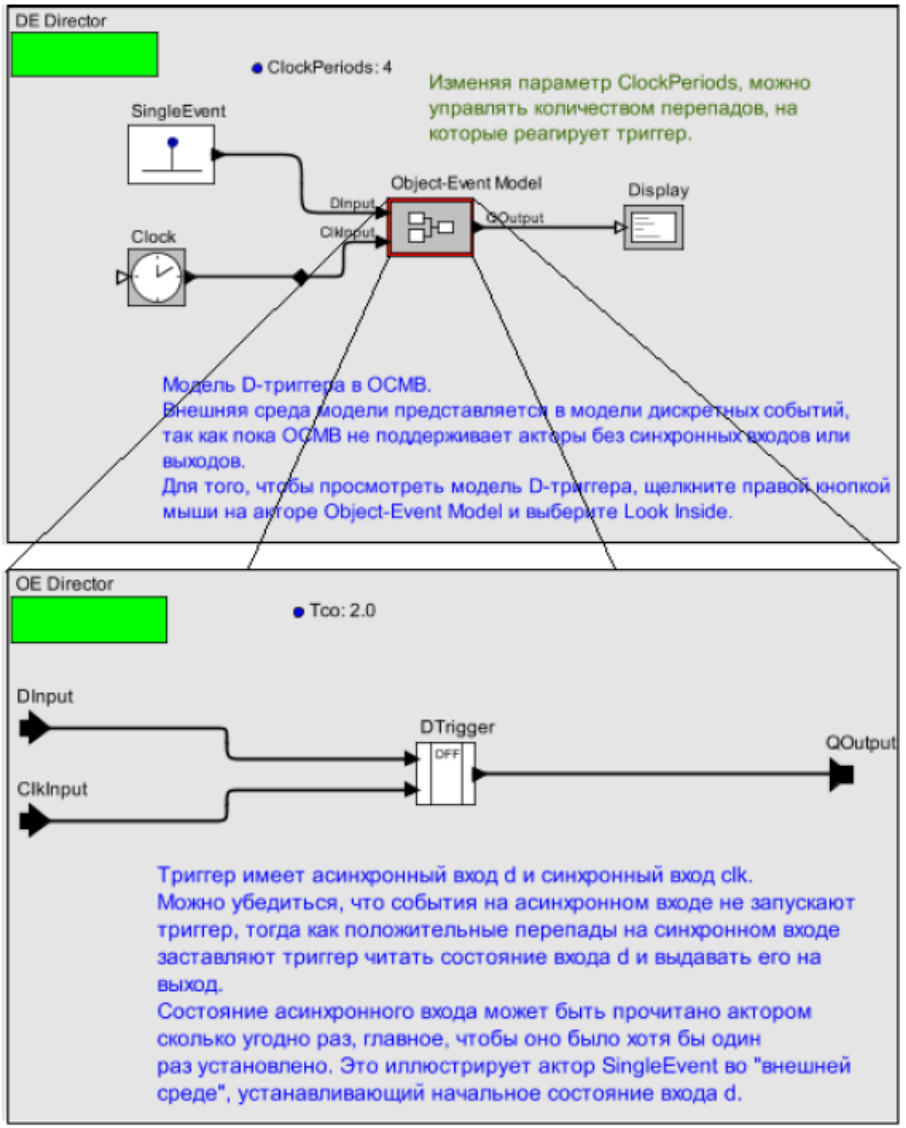



Рис. 5.15. Иерархия модели

В Vergil можно посмотреть документацию на любой элемент модели, представляющий собой Java-класс, выбрав в соответствующем контекстном меню пункт «Get Documentation». Параметры моделирования в модели низкого уровня настраиваются с помощью параметров *OE Director*. Поскольку ptolemy.domains.oe.kernel.OEDirector имеет в качестве базового класса ptolemy.domains.de.kernel.DEDirector, для настройки доступны все параметры последнего, а также единственный добавленный в OEDirector параметр *checkDanglingSynchronousInputs*, разрешающий одну из проверок модели на корректность.

Модель в рамках OCMB содержит единственный функциональный блок, *DTrigger*, реализованный в виде несоставного актора (atomic actor), то есть описанного на Java. Тем

не менее, в Vergil можно не только посмотреть реализацию его поведения, но и редактировать ее, выбрав в соответствующем

контекстном меню пункт «Look Inside». Актор снабжен собственной иконкой, нарисованной в редакторе иконок Vergil.

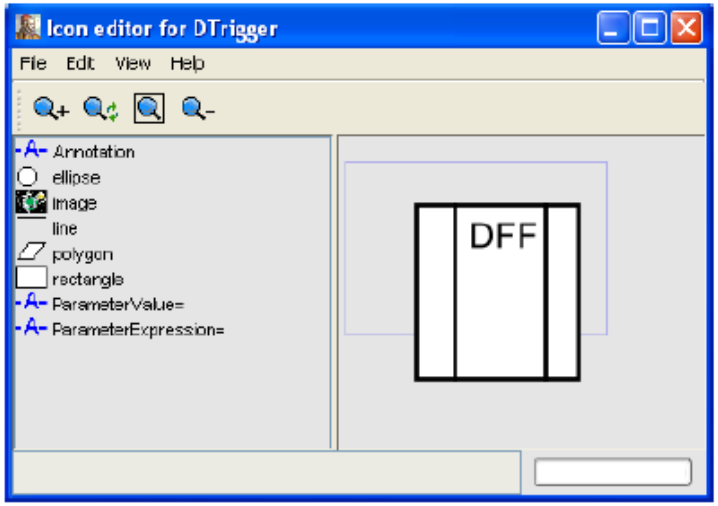



Рисунок 16. Редактор иконок Vergil

Рис. 5.16. Редактор иконок Vergil

Функциональный блок *DTrigger* имеет два входа, один из которых является асинхронным и имеет имя *d*. Асинхронным его делает наличие параметра *asynchronous* со значением *true*. Триггер запускается по «положительному» перепаду на синхронном входе *clk* (*false-true*), считывая состояние асинхронного входа и выдавая элемент данных на выход. Состояние асинхронного входа может

быть прочитано сколько угодно раз, однако перед первым чтением оно должно быть инициализировано элементом данных. При попытке чтения состояния неинициализированного асинхронного входа вырабатывается исключение. Для инициализации асинхронного входа триггера в модели верхнего уровня присутствует актор *SingleEvent*, который в начальный момент времени генерирует событие, данные которого поступают на вход *d* функционального блока *DTrigger*, таким образом устанавливая его (входа) начальное состояние.

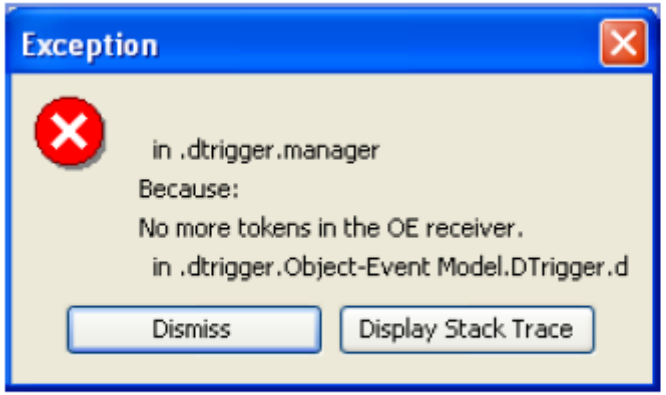



Рисунок 17. Чтение неинициализированного асинхронного входа

Рис. 5.17. Чтение неинициированного асинхронного кода

Временные характеристики функционального блока рассчитываются по его параметрам *Gamma*, *Tau*, *G* и *Omega*, каждый из которых является матрицей соответствующей

размерности. Последний параметр представляет собой проектные ограничения, накладываемые на минимальные интервалы генерации выходных событий. Фактические интервалы рассчитываются в процессе инициализации модели в создаваемом параметре *Omega_* и проверяются на соответствие значениям в *Omega*. При инициализации также проверяется наличие и корректность типов всех параметров.

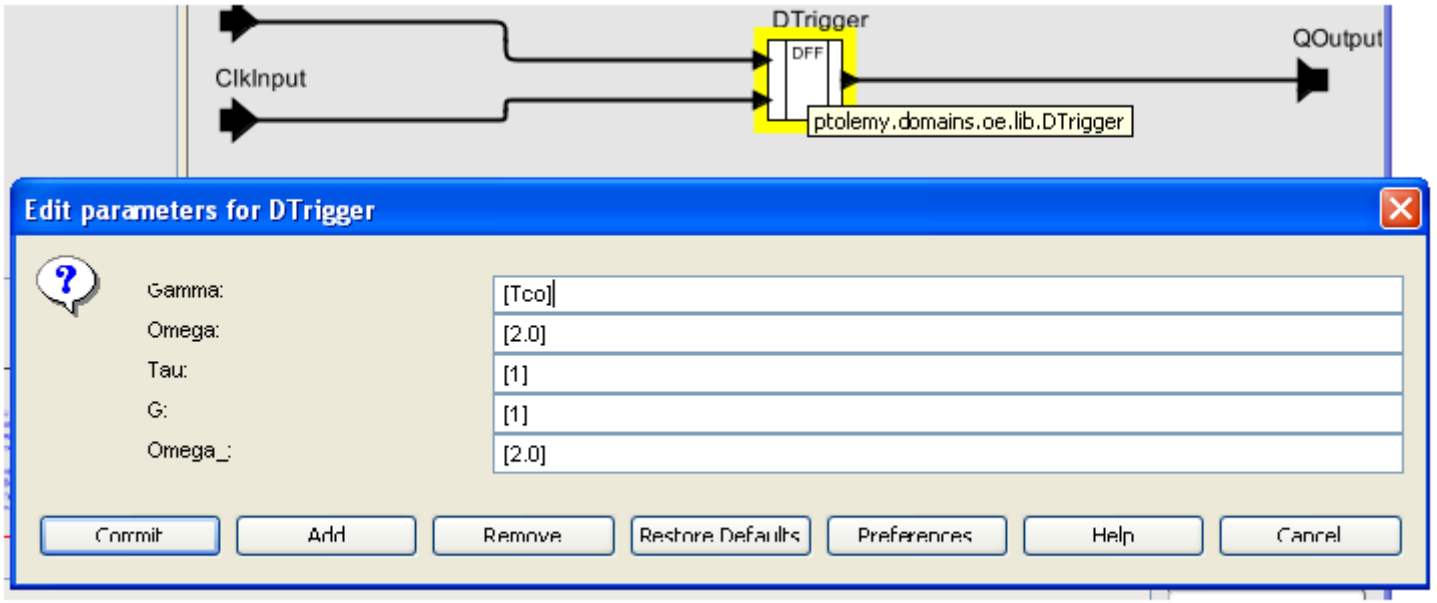



Рисунок 18. Временные параметры функционального блока

Рис. 5.18. Временные параметры функционального блока

## 5.6. Критерии отбора персонала для работы с Ptolemy II

Для эффективной работы с пакетом программ Ptolemy II ***необходимо представлять процесс проектирования вычислительных систем, понимать концепцию высокоуровневого проектирования и иметь знания в предметной области, в которой будет работать проектируемая система.***

## 5.7. Рекомендации по использованию в образовательном и научно-исследовательском процессах

Комплекс программ Ptolemy II может быть использован для обучения курсам проектирования встроенных и распределенных информационно-управляющий систем. Также Ptolemy II может быть использован в качестве системы моделирования при проведении лабораторных работ и научных экспериментов. В научно-исследовательском процессе программный комплекс Ptolemy II можно использовать для проведения научно-исследовательской работы по проектированию вычислительных систем и изучению различных моделей вычислений. Также Ptolemy II может быть платформой для создания собственных САПР.

Программа
https://ptolemy.berkeley.edu/ptolemyII/ptII11.0/index.htm

## 5.8. Подготовка специалистов в области ВсС

На кафедре вычислительной техники СПбГУ ИТМО наш коллектив создал две магистерские учебные программы, которые успешно развиваются:

- ***«Проектирование встраиваемых вычислительных систем»*** — организация встраиваемых систем и комплексов с различной архитектурой, технологии и инструментарий высокоуровневого проектирования, встроенное программное обеспечение, схемотехническое проектирование.

- ***«Системотехника интегральных вычислителей. Системы на кристалле»*** — создание вычислительных компонентов и встраиваемых систем в интегральном исполнении, технологии высокоуровневого проектирования СБИС/СнК, тестопригодное проектирование, энергосберегающие технологии, средства моделирования и верификации СнК.

Основным отличительным моментом этих специализаций следует считать то, что мы готовим специалистов системного уровня, условно называя их «архитекторами».

Это специалисты, которые могут эффективно работать на высокоуровневых этапах проектирования сложных ВсС, выполняя следующие функции:

- «архитектор» — сбалансированный взгляд и знания в системотехнике, архитектуре ВС, ПО, аппаратуре, инструментарии;
- специалисты по направлениям, но с базовым кругозором — встраиваемое ПО, ПЛИС/ПСнК, РИУС, HLD (грамотное моделирование с использованием современных инструментов), схемотехника микроконтроллеров + ПЛИС + аналого-цифровых узлов, инструментарий.

# Глава 6. Сопряжённое проектирование аппаратуры и ПО

## 6.1 Введение

Высокоуровневое проектирование специализированных, в частности – встраиваемых, вычислительных систем, использует технологию, носящую название в англоязычной литературе Hardware/Software Co-Design.

В отечественной литературе нет общепринятого названия данного направления, для его обозначения используют такие термины, как «аппаратно-программный кодизайн», «кодизайн», «сопряжённое проектирование аппаратуры и программного обеспечения», «совместное проектирование аппаратуры и программного обеспечения».

Требования, предъявляемые к встраиваемым системам, по определению жёстче, чем к ЭВМ общего назначения. В общем случае встраиваемые системы должны быть надёжными, малогабаритными, дешёвыми, с низким энергопотреблением и с

предсказуемым временем реакции на внешние воздействия (должны удовлетворять требованиям реального времени).

Ввиду этого, зачастую использование существующих аппаратных решений оказывается неэффективным, и помимо написания программного обеспечения – как делается обычно в системах общего назначения – встаёт задача создания также и специализированной аппаратуры.

При проектировании встраиваемых систем разработчик вынужден проектировать аппаратуру и программное обеспечение с учётом множества разноплановых аспектов, и зачастую вынужден работать одновременно в нескольких областях инженерной деятельности.

Элементная база и потенциальные возможности постоянно совершенствуются, предоставляя всё более совершенные и эффективные средства для создания специализированных вычислительных систем.

Направление кодизайна объединяет многочисленные исследования в области параллельного и скоординированного проектирования программного обеспечения и аппаратуры.

В контексте проектирования специализированных вычислителей термин «кодизайн» фактически используется как синоним прогрессивных направлений системного, высокоуровневого или архитектурного проектирования, и представляет собой набор подходов и инструментальных средств для решения актуальных проблемпроектирования специализированных вычислительных систем.

## 6.2. Направление «кодизайн»

Сопряжённое, или совместное проектирование аппаратуры и ПО,

называемое в англоязычной литературе «hardware-software co-design», «software-hardware co-design», или просто «co-design» – представляет собой процесс параллельного и скоординированного проектирования электронных программно-аппаратных систем,

который основывается на не зависящем от конечной реализации описании, и использует средства автоматизации проектирования.

Кодизайн помогает находить компромиссы между гибкостью вычислительной системы и её производительностью, комбинируя при проектировании два радикально отличающихся подхода:

- последовательную декомпозицию во времени, создаваемую при помощи ПО, и
- параллельную декомпозицию в пространстве, реализуемую на основе аппаратного обеспечения.

Благодаря одновременному анализу, исследованию и проектированию аппаратного и программного обеспечения сокращаются сроки разработки и уменьшаются риски задержек выхода на рынок.

Основные цели и смысл кодизайна могут быть выведены из различных трактовок слога «ко» в слове кодизайн:

- Координация: методы кодизайна используются для для координации этапов проектирования, выполняемых междисциплинарными группами. В эти группы со стороны ПО входят разработчики встроенного микропрограммного обеспечения, системные и прикладные программисты, а со стороны аппаратного обеспечения – разработчики аппаратных средств и микросхем. Это является исходной интерпретацией слога «ко» в слове кодизайн. Другие, описанные далее интерпретации, являются скорее дополнениями.
- Параллелизм (Concurrency): Сжатые рамки времени выхода на рынок вынуждают разработчиков ВС работать одновременно, вместо того, чтобы начать работу с ПО лишь после созданияаппаратной платформы. Кодизайн как-раз достиг значительного прогресса в обходе данного узкого места процесса разработки,избавляя от

необходимости иметь готовую аппаратуру. Это достигается благодаря использованию исполняемых спецификаций и/или благодаря применению концепции виртуальных платформ и виртуального прототипирования – для того, чтобы запускать разрабатываемое ПО на симуляторе целевой платформы, на очень раннем этапе.

- Корректность: Требования к отсутствию ошибок во встраиваемых системах требуют, чтобы не только проверялись корректность ПО и аппаратуры по отдельности, но и ко-верифицировалась правильность их взаимодействия после интеграции.
- Сложность (Complexity): Методы ко-дизайна решают проблему роста сложности проектирования сегодняшних электронных систем, они являются средством (или, по крайней мере, пытаются быть таким средством) для решения известных проблем проектирования, для

создания корректно работающих и оптимизированных системных реализаций.

Обычно аппаратные проектные решения принимаются на основе опыта команды разработчиков, а успех и эффективность создания программного обеспечения зависит от команды разработчиков программного обеспечения.

Недостатки классической цепочки проектирования многочисленны:

- длинный критический путь и обычно непредсказуемое время выхода на рынок:
- риски очень позднего обнаружения потенциальных ошибок в каждой части проектной цепочки;
- риск избыточности или неполноты конечного продукта из-за отсутствия ранней оценки проектных альтернатив.

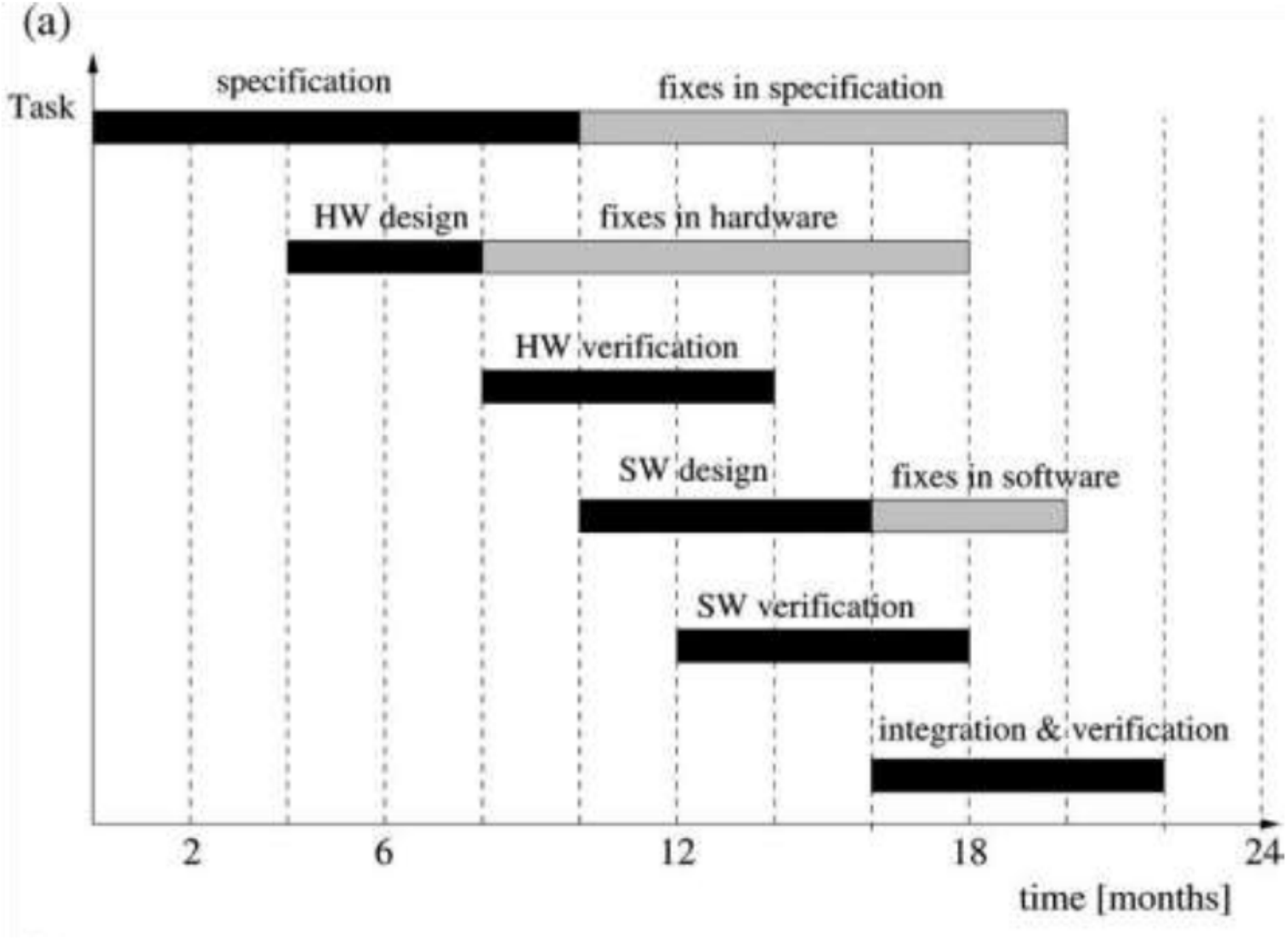



Рис. 6.1. Классический маршрут проектирования

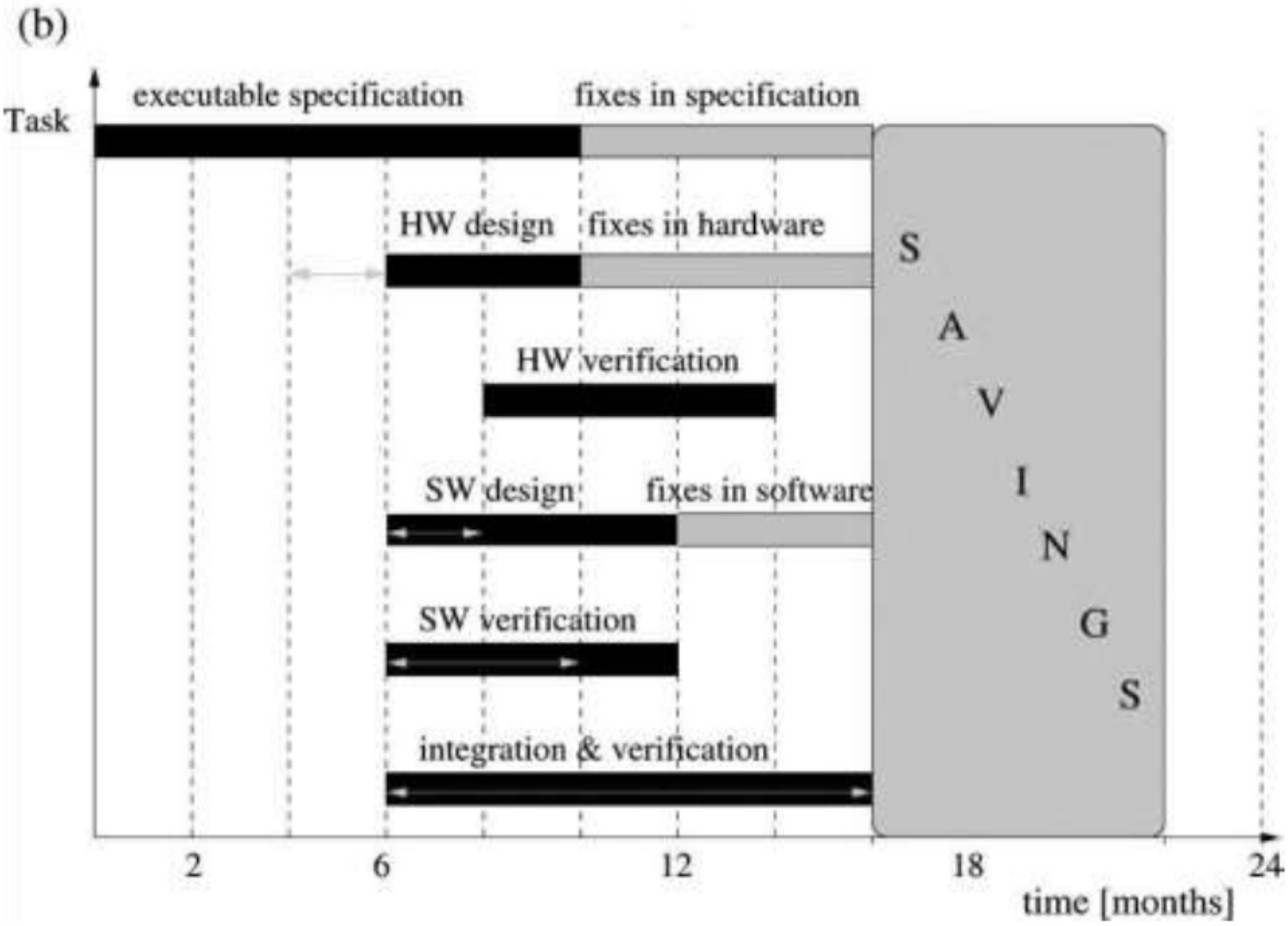



Рис. 6.2. Маршрут проектирования ESL

Использование современных, основанных на методологии ESL (Electronic System Level), инструментов кодизайна, позволяет получить значительный выигрыш по времени выхода на рынок.

*Процесс разработки с использованием методов кодизайна начинается с создания на ранних этапах проектирования функциональной, но уже исполняемой, спецификации системы (например, на языках C, C++ или SystemC). Кроме того, в зависимости от существующей платформы или аппаратных IP, создаётся модель платформы, включающая в себя модели стоимости.*

Благодаря этим начальным накладным расходам на моделирование, становится возможным раннее исследование пространства потенциальных решений при выборе системных компонентов, их соединений, компоновки памяти, распределения программных функций.

При этом, конечно, моделирование архитектуры системы, параметризация моделей для исследования пространства проектных решений, калибровка моделей соответствующим образом может занять значительное количество времени.

По окончании исследования пространства проектных решений и нахождения некоторого оптимального варианта производится разделение на программные и аппаратные части, создаются спецификации на их разработку, и запускается скоординированное проектирование этих частей в параллели, с последующей взаимной интеграцией. *Одновременная разработка ПО и аппаратуры как-раз и является ключевой, отличительной особенностью кодизайна.*

Применяют технологии : косимуляция (Co-симуляция) и коверификация (Co-верификация).

Косимуляция позволяет производить запуск проектируемого ПО на модели проектируемой, но ещё не существующей аппаратуры, благодаря чему возможны как ранняя отладка ПО на целевой

аппаратуре до её физической реализации, так и оценка параметров будущей системы.

Коверификация позволяет осуществлять раннюю верификацию ПО (формальную оценку корректности его выполнения), также используя некоторую работающую модель аппаратуры, что позволяет наряду с проверкой ПО производить обнаружение и исправление возможных аппаратных ошибок.

В SystemC – программа SCV (SystemC Verification Library)

Важной составляющий процесса проектирования с применением кодизайна является синтез. Синтез на всех уровнях абстракции является очень схожим процессом – по определённым правилам, элементам более высокого уровня абстракции в соответствие ставятся реализующие их элементы более низкого уровня. Выбор элементов, правила их компоновки могут зависеть от выбранной оптимизации.

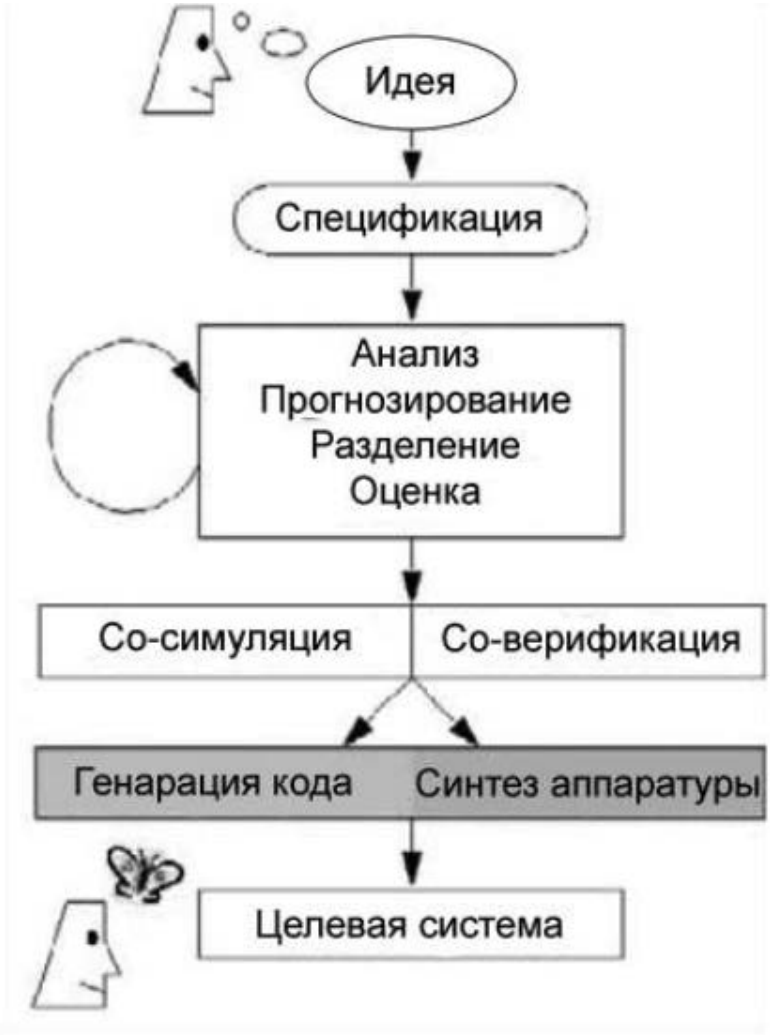



*Рис. 2. Современный маршрут проектирования на основе кодизайна.*

Рис. 6.3. Современный маршрут проектирования на основе кодизайна

Кодизайн должен позволять объединять существующие полу- и автоматизированные этапы проектирования и предоставлять интерфейсы для соединения различных уровней абстракции.

Идеальным является случай, когда кодизайн выполняет всю необходимую детализацию проекта автоматически, позволяя экономить время и обеспечивая быструю верификацию этапов проектирования.

### 6.2.1 История развития кодизайна

Кодизайн появился в начале 90-х годов XX века как новая дисциплина для проектирования сложных интегральных схем (ИС), включающих программируемые процессоры.

В первые годы много внимания было уделено проблеме разделения заданной функциональной спецификации на две части – на аппаратное и программное обеспечение. Спецификация определялась на C подобном языке и отображалась на заданную аппаратную платформу одного центрального процессора (ЦП) и

определяемый специализированный аппаратный блок или интегральную схему (ASIC). Обе части связывались по шине и использовали разделяемую память или регистры для реализации буфера – т. е., за исключением вопроса, какие части реализовать на ASIC, целевая платформа была уже фиксированной.

До начала 2000-х была хорошо проработана проблема разделения аппаратных средств и программного обеспечения, значительно расширившись для более сложных типов архитектур. Ограничение на однопоточное выполнение программ было убрано, появилась возможность использования мультипрограммирования и многопроцессорной обработки.

Кроме того, ко-симуляция стала всё более важной областью исследования, предоставляя возможность ранней проверки допустимости проектных решений.

При ко-симуляции выполнение программного обеспечения моделируется на ЦП с использованием виртуальной модели

аппаратных средств процессора или синтезируемой аппаратной спецификации разрабатываемой системы. На вентильном уровне или RTL уровне такие системы не моделируются из-за большой сложности и, следовательно, слишком медленной симуляции.

## 6.3. Современный кодизайн на основе ESL

Сегодня мы уже живём в третьем поколении технологии ко-дизайна, который обусловлен постепенным появлением инструментов для сквозного синтеза сложных электронных систем. В течение первого десятилетия XXI века имел место быть значительный прогресс, в основном вызванный решением следующих проблем:

- Технология гетерогенных СнК стала реальностью благодаря достижениям в микро- и нано-электронике. Сложные системы, сочетающие несколько микропроцессоров различных типов (DSP, ASIP, микроконтроллеров), могут быть интегрированы в одну СнК. вместе с IP-блоками аппаратных ускорителей, аналоговых

устройств и блоками памяти. Сегодня востребованы методы её эффективного использования.

- Сложность аппаратуры и ПО постоянно растут, и необходимы средства для борьбы с нею. Сложность аппаратуры проявляется не только в технологии СнК, но и на уровне распределенных систем, состоящих из взаимодействующих электронных устройств – таких, как электронные блоки управления (ЭБУ) современных автомобилей. В них десятки взаимосвязанных ЭБУ, предоставляющих специальные функции.
- Требуется панацея интеграции. С точки зрения разработчикаВсС важнейшим препятствием было и до сих пор остаётся отсутствие стандартов на интеграцию подсистем, разработанных даже другими компаниями, для обеспечения приемлемых сроков выведения продуктов на рынок. Это, опять же, в особенности верно для области автомобилестроения. В ней интеграция блоков начинается на тестовом стенде, где соединяются различные

подсистемы, и трудоёмкие сценарии тестирования применяются для анализа функциональной корректности и выявления потенциальных временных ошибок до того, как происходит интеграция в автомобиль. Причём эти тесты производятся очень часто без знания и наличия ПО каждой подсистемы, так как они разрабатываются поставщиками.

Перечисленное привело к пониманию того, что:

• необходимо повышать уровень абстракции, на котором разработчики представляют свои системы при проектировании,

• необходим способ комбинирования и повторного использования проектов сквозь различные уровни абстракции.

***Это дало рождение идее проектирования электроники на системном уровне – ESL.***

Фундаментальный принцип ESL-проектирования – детализация и усложнение абстракций при некоторой фиксированной проектной концепции. В течение процесса проектирования проект

представляется на различных уровнях абстракции, отличающихся степенью детализации.

В качестве уровней абстракции выделяются:

• Требования рынка товаров. Диктуют функционал, предоставляемый конечному пользователю, и физические требования к продукту.

• Функциональная спецификация. Описывает функциональные требования к изделию как к чёрному ящику.

• Архитектурная модель. Показывает разделение аппаратной и программной частей.

• Спецификации проектных решений аппаратной и программной частей. Описывают требования к реализации аппаратной и программной частей.

• Функциональные и поведенческие модели аппаратной и программной частей. Реализуют алгоритмы, требуемые последующими моделями с приблизительной точностью

моделирования времени выполнения или вообще без учёта временного аспекта.

• Модели RTL и программного обеспечения. Синхронизированные поведенческие модели аппаратуры и ПО.

• Модель на уровне вентилей и встраиваемое ПО. Модель на уровне вентилей синтезируется из RTL, встраиваемое ПО транслируется машинным образом из модели ПО, или, что чаще встречается сейчас – пишется вручную.

• Базы данных связей аппаратных компонентов. Задаёт структурную геометрию и соединения всех аппаратных элементов.

С каждой итерацией эта информация, представленная в виде модели, детализируется и дополняется, становится более точной, в конце концов превращаясь в базу данных связей аппаратных компонентов и низкоуровневое ПО.

Принципы ESL наглядно демонстрирует так называемая модель «двойной крыши» (Рис. 6.4). Данная модель отображает проблему

синтеза электронных систем, которая включает в себя три больших задачи так называемого отображения.

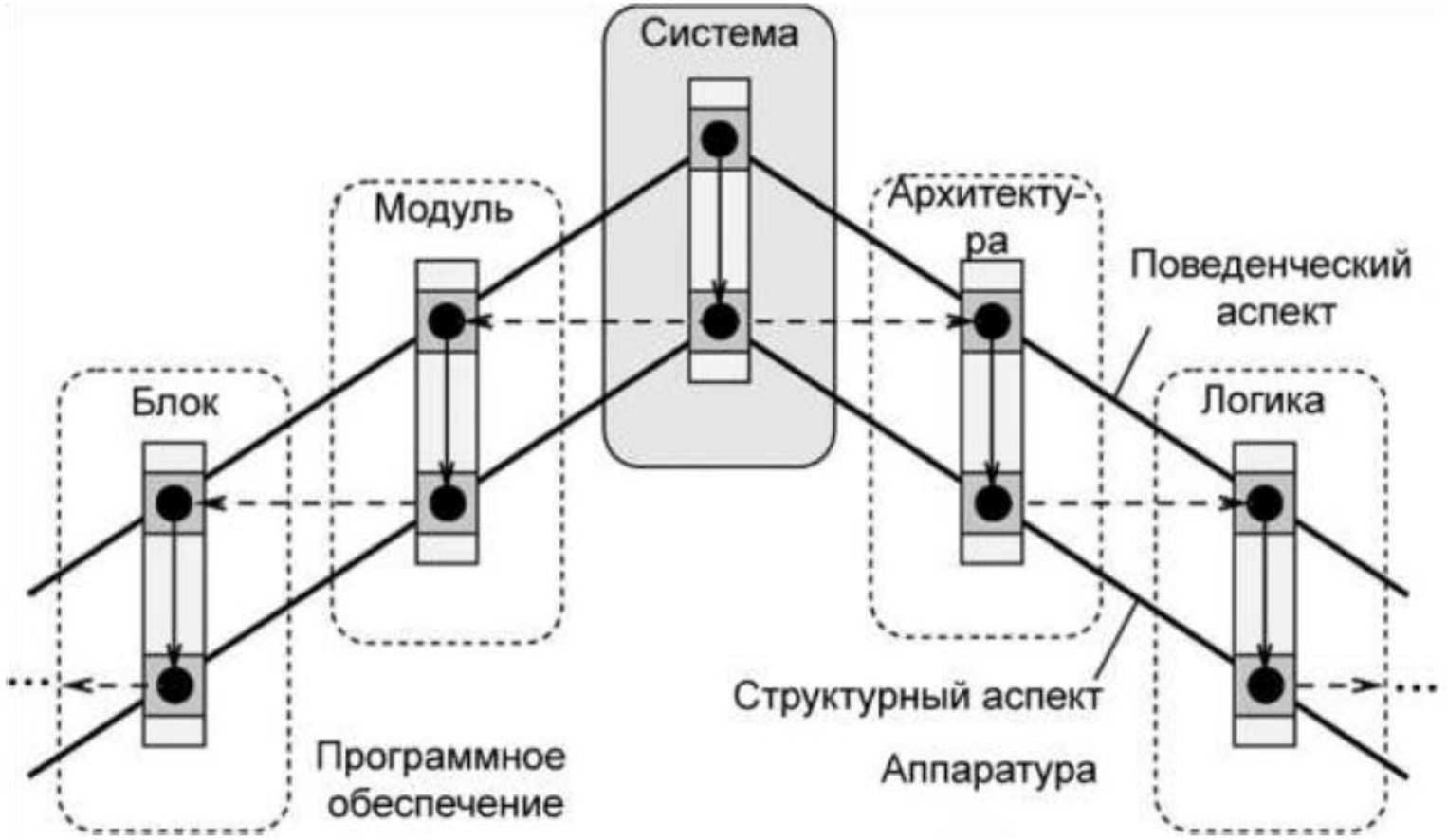



*Рис. 3. Модель «двойной крыши»* [1].

Рис. 6.4. Модель двойной крыши

• ***Выделение***: выделение и отображение системных ресурсов, включая процессоры, аппаратные IP-блоки (интерфейсы, запоминающие устройства и т.д.), и их сети соединений, и таким образом формирование архитектуры системы с точки зрения ресурсов. Эти ресурсы могут существовать как библиотечные шаблоны, но, с другой стороны, маршрут проектирования должен иметь возможность синтезировать их.

• ***Привязка***: отображение: функциональности (т.е. задач, процессов, функций или базовых блоков) – на конкретные вычислительные ресурсы, переменных и структур данных – на запоминающие устройства, коммуникаций – на маршруты между соответствующими ресурсами. В современных системах привязка не только рассматривает взаимно разделяемую память и простые шинные коммуникации, но также должна определять маршруты от источников до места назначения, отражая очень сложные сети коммуникационных соединений, включая также и сети на кристаллах.

• ***Планирование***: определение временных аспектов выполнения конкретных функций на выделенных им ресурсах, включая само выполнение функции, обращения к памяти и коммуникацию с другими блоками. Может включить определение частичного порядка выполнения или спецификацию планировщиков для каждого ЦП, ресурсов коммуникации и памяти, наряду с приоритетами задач и т.д.

В модели с двойной крышей системный уровень представления ВС, на котором еще нельзя различить аппаратуру и ПО, изображён коньком крыши. Системный уровень соединяет процессы разработки программного обеспечения (скат слева) и аппаратуры (скат справа) путем последовательного детализирующего синтеза (вертикальные стрелки, идущие вниз). Верхний свод описывает функциональный вид или вид спецификации системы на соответствующем уровне абстракции, тогда как нижний свод описывает свою структурную реализацию, включая назначенные ресурсы, решения планирования, привязки и соответствующий код. Каждый шаг синтеза отображает

функциональную спецификацию на структурную реализацию на следующем, более низком уровне абстракции.

Посредством каждого этапа синтеза спецификация преобразуется в реализацию на следующем, более низком уровне абстракции.

Горизонтальные стрелки указывают шаги передачи информации о реализации на определенном уровне к следующему, более низкому уровню абстракции. Каждый шаг синтеза добавляет дополнительную информацию о спецификации или ограничениях.

Типичный маршрут проектирования, представленный моделью двойной крыши, обычно начинается со спецификации ESL. Это функциональная (поведенческая) спецификация всей системы, которая либо представляет собой модель, либо языковое описание. Дополнительно, задается ряд ограничений (максимальная занимаемая на кристалле площадь, минимальная пропускная способность и т.д.). Модель платформы в ESL обычно – структурная модель, состоящая из архитектурных компонентов, таких как

процессоры, шины, сети на кристалле (NoC), запоминающие устройства и аппаратные IP-блоки, использующиеся в качестве ускорителей или внешних коммуникационных интерфейсов. Задача синтеза ESL – процесс выбора соответствующей архитектуры платформы из этого разнообразия, определение отображения поведенческой модели на эту архитектуру и генерация соответствующей реализации поведения для данной платформы.

Результатом является детализированная модель, содержащая все проектные решения и обычно множество нефункциональных метрик качества, таких как пропускная способность, задержка на обработку данных, количество элементарных ячеек и энергопотребление.

## 6.4. Типовой маршрут ESL-проектирования

Идеализированный процесс ESL-проектирования можно разбить на шесть шагов, которые параллельны вышеупомянутым уровням абстракций:

• создание спецификации и моделирование;

• предварительный анализ перед разделением на аппаратные и программные части;

• разделение на аппаратные и программные части;

• анализ, отладка и устранение обнаруженных ошибок после разделения;

• верификация после разделения;

• реализация аппаратуры;

• реализация ПО;

• верификация реализации.

Первый шаг – ***написание спецификаций и моделирование***. Он представляет собой разработку документов, описывающих систему или концепцию проекта и его ограничения, а также их трансляцию в различные исполняемые или декларативные модели.

В дополнение к спецификации на обычном языке, существует класс исполняемых спецификаций, которые представляют собой функциональные или поведенческие описания системы, являются

моделью будущего изделия как чёрного ящика. В этот класс попадают исполняемые архитектурные и проектные спецификации. Первые являются средством для демонстрации альтернатив реализаций проекта и концепции выбранного решения. Вторые – отражают ключевые структурные решения реализации микроархитектуры, представляют систему прозрачной.

Типичные языки для создания ESL-спецификаций: MATLAB M-Code, **SystemC,** SystemVerilog, и т.д.

***Анализ перед разделением на аппаратные и программные части*** (pre-partitioning analysis) – процесс исследования спектра алгоритмических компромиссов.

Для их оценки можно использовать различные сценарии и методы – статический анализ, платформно-ориентированное проектирование, динамический анализ, алгоритмический анализ и т.д.

***Разделение*** – процесс, в ходе которого производится выбор, какие алгоритмы (или их части), заданные в спецификации, реализовывать в виде запускаемого на процессорах программного обеспечения (также, на этом этапе выбираются типы процессоров), какие – в виде чисто аппаратных компонентов. Происходит разделение алгоритмов между аппаратными и программными частями. Процесс разделения можно рассматривать в следующем порядке: *функциональная декомпозиция*, *архитектурная декомпозиция*, *разделение*, *разделение аппаратной части*, *разделение программной части*, *определение реконфигурируемых частей*, и *реализация коммуникаций*.

В *функциональной декомпозиции* за отправную точку берётся спецификация, настолько свободная от артефактов реализации, насколько это возможно, и позволяет получить представление о возможностях распараллеливания на уровне приложений.

В *архитектурном описании* принято использовать элементарные блоки, каждый из которых реализует какую-то функцию: хранение данных, вычисления, связи с другими блоками.

После уточнения или отображения абстрактных моделей на архитектуру требуется *разделить аппаратные и программные компоненты*. Аппаратное разделение может быть реализовано в виде одного или нескольких процессоров распределённой системы. Системное проектирование аппаратуры всё больше становится ничем иным, как просто конфигурированием платформы, когда основная масса работы в проектировании снизу-вверх перемещается в область программного обеспечения.

Программные модули состоят не только из операционной системы, библиотек и промежуточного ПО (middleware), но и из приложений, которые зачастую берут на себя основную функциональную нагрузку.

*Реконфигурируемые вычислители* призваны заполнить пропасть между аппаратурой и программным обеспечением, позволяя добиться компромиссов в производительности, стоимости и энергопотреблении.

Проблемой реконфигурируемых вычислителей является сложность их программирования. Подобные вычислители представлены двумя основными категориями: реконфигурируемый массив, действующий как функциональный модуль управляющего процессора, или реконфигурируемый массив, работающий как сопроцессор, подключённый к основному процессору.

Одной из самых ответственных задач при разделении является *реализация коммуникаций*, так как от выбранных протоколов зависит множество аспектов системы, таких, например, как производительность и надёжность.

Существуют два подхода к реализации коммуникационных архитектур: использование шаблонных решений (template instantiation) и синтез интерфейсов.

После разделения следует обязательный и важный этап *анализа и верификации*. Он состоит из двух шагов.

Первый заключается в реализации алгоритмов, которые предназначаются для выполнения на процессорах и написаны на одном из языков программирования, т.е. в их компиляции для конкретных процессорных архитектур.

Второй – в моделировании на поведенческом уровне предназначенных для аппаратной реализации алгоритмов, при помощи одного из языков описания аппаратуры.

Моделирование аппаратуры и ПО всей системы ограничивается целями моделирования (исследование параметров проекта, валидация, верификация), которые должны быть не слишком всеобъемлющими, для возможности создания модели.

Модели для аппаратуры и ПО должны иметь возможность исполняться параллельно. Существует три варианта моделирования программных элементов:

- общая модель аппаратуры и ПО,
- модель только ПО с адаптерами для аппаратных коммуникаций (hardware communication adapters),
- модели аппаратуры и ПО отдельно.

*Функциональный анализ*, обычно относимый к функциональной верификации, требуется на этом шаге проектирования для понимания требований к размеру различных элементов хранения. Начальный *анализ производительности* производится для валидации разделения и определения размеров аппаратных и программных элементов. *Анализ интерфейсов* позволяет обнаружить рано, задолго до реализации, подходит или нет конкретный вариант коммуникационных интерфейсов модуля. *Анализ энергопотребления* выгоден на этом этапе в случае, когда модель имеет свойства скорее

физической реализации, чем программной. *Анализ занимаемого места* использует метрики сложности для оценки требований к аппаратуре по занимаемому на кристалле месту. *Анализ стоимости* при помощи эвристик рассчитывает стоимость проектирования, производства, поддержки и полного жизненного цикла изделия. Наконец, *анализ пригодности для отладки* проверяет сложность и риск функциональных ошибок предполагаемой реализации аппаратуры, чтобы выяснить, какие возможности контролируемости и наблюдаемости требуются от внешних интерфейсов физической реализации чипа.

***Верификация*** после этапа разделения должна выявить, сохранилось ли ожидаемое поведение компонентов системы разрабатываемого изделия в уточнённой модели, полученной после разделения.

Это первый шаг верификации, следующий – это уже верификация реализации.

Предпринимаются три шага: *планирование верификации, реализация окружения верификации и анализ результатов верификации.*

Когда область интереса для верификации определяется, а план верификации и функциональная спецификация для верификационного окружения написана, создаётся само *верификационное окружение*, обычно с использованием высокоуровневых языков верификации (High-Level Verification Languages), возможно даже – аспектно-ориентированных.

***Реализация аппаратуры*** – процесс создания моделей, которые могут быть синтезированы в модели вентильного уровня. Аппаратные модели обычно описываются на уровне передач сигналов между регистрами (RTL- модели), или даже на более высоком поведенческом уровне.

Существует пять основных вариантов реализации аппаратуры, которые доступны на данном этапе: расширяемые (extensible)

процессоры, DSP сопроцессоры, специализированные VLIW сопроцессоры, ASIP и ASIC/FPGA.

Процесс ESL-проектирования строится поверх традиционного RTL- проектирования, состоящего из следующих шагов:

создания RTL,

верификации RTL,

синтеза RTL в вентили (элементарные логические элементы),

верификации временных задержек, размещения и трассировки вентилей,

проверки правил проектирования,

генерации битового потока.

RTL-описание, как правило, создаётся с компонентами тракта обработки данных ("data path") и тракта управления ("control flow"), где последний представляет собой реализацию взаимодействующих конечных автоматов, которые управляют прохождением потока информации через тракт передачи данных.

***ESL-проектирование***, состоящее из создания системной спецификации, разделения на аппаратные и программные компоненты, создания виртуального прототипа, проектирования на уровне транзакций, верификации на уровне транзакций – заканчивается, в том числе, и синтезом в RTL для аппаратных компонентов. Таким образом, RTL- описание находится на стыке, используется для обмена данными между RTL и ESL процессами проектирования.

ESL-синтез не решает автоматически все проблемы – он всего-навсего делает более продуктивным проектирование аппаратуры трансляцией высокоуровневых моделей в RTL.

Одним из основных преимуществ ESL-синтеза является возможность раннего исследования вариантов реализаций. Варьируя ограничения при фиксированном поведенческом описании, инженер может ознакомиться с пространством решений и их чувствительностью к конкретным параметрам.

***Реализация ПО*** предоставляет человеко-машинный интерфейс, позволяющий использовать аппаратную часть, и представляя собой самые высокоуровневые элементы проекта. Обычно, процесс разработки ПО соответствует классической водопадной модели ("waterfall" model). ПО пишется для уже существующей аппаратуры с целью удовлетворения всех требований к продукту, включая компенсацию аппаратных багов и попытки исправить возможный недостаток производительности.

Альтернативный подход – использование ESL-моделей для прототипирования программных компонентов системы со спиральным процессом разработки. В ходе подобного процесса создаются серии реализаций, каждая оценивается со связанной с ней аппаратурой в системном контексте, и затем уточняется для исследования ограничений, обнаруженных в последней реинкарнации. Подобная последовательность позволяет аппаратуре

и ПО последовательно преобразовываться в решение, которое будет соответствовать требованиям к разработке.

Последний шаг ESL-проектирования – ***верификация реализации***. В ходе этого этапа нужно в очередной раз убедиться, что полученная система соответствует основной концепции и целям проекта. На данном этапе детализация RTL-описания и встраиваемого ПО должна быть произведена полностью, без оставшихся неясностей.

## 6.5. Пример: проектирование средствами Mentor Graphics

Mentor Graphics предоставляет инструменты для ESL-ориентированного проектирования электронных систем, дающие возможность работать над проектом, начиная с создания архитектурного описания и фиксации формальных требований и вплоть до конечной реализации в виде СнК или ПЛИС.

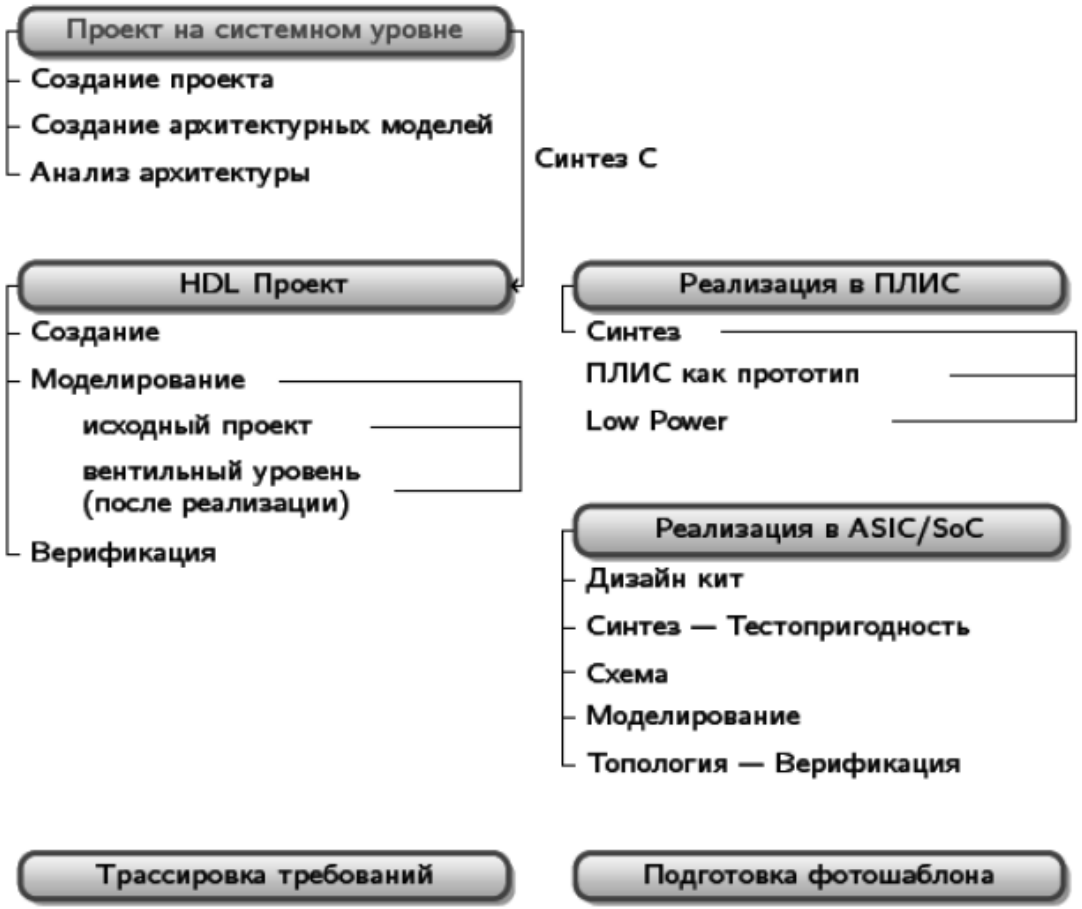



*Рис. 4. Маршрут проектирования FPGA/CPLD/ASIC/SoC* [16].

Рис. 6.5. Маршрут проектирования Mentor Grafics

На самом верхнем уровне предполагается исследование пространства проектных решений и выбор оптимальной архитектуры. Для этого создаётся виртуальный прототип, реализованный на уровне транзакций, который используется для примерных оценок характеристик будущей системы и применяется в качестве симулятора целевой аппаратуры для создания ПО. При этом среда разработки ПО позволяет одинаково работать как с моделями, созданными на уровне транзакций, так и с более детализированными, в том числе с эмуляторами на основе ПЛИС и с реальными, уже реализованными в ПЛИС, СнК или ИС системами.

Следующим шагом идеализированного нисходящего проектирования, который идёт после создания архитектурного описания и определения требуемого функционала изделия, является генерация из высокоуровневых языков программирования (C, C++, SystemC), на которых создаётся высокоуровневая спецификация будущего изделия, в языки описания аппаратуры (Verilog, VHDL),

для дальнейшей доработки аппаратными инженерами. Генерацию осуществляет специальный инструмент синтеза, который позволяет получать на выходе HDL-проект, представляющий собой описание на уровне регистровых передач (RTL).

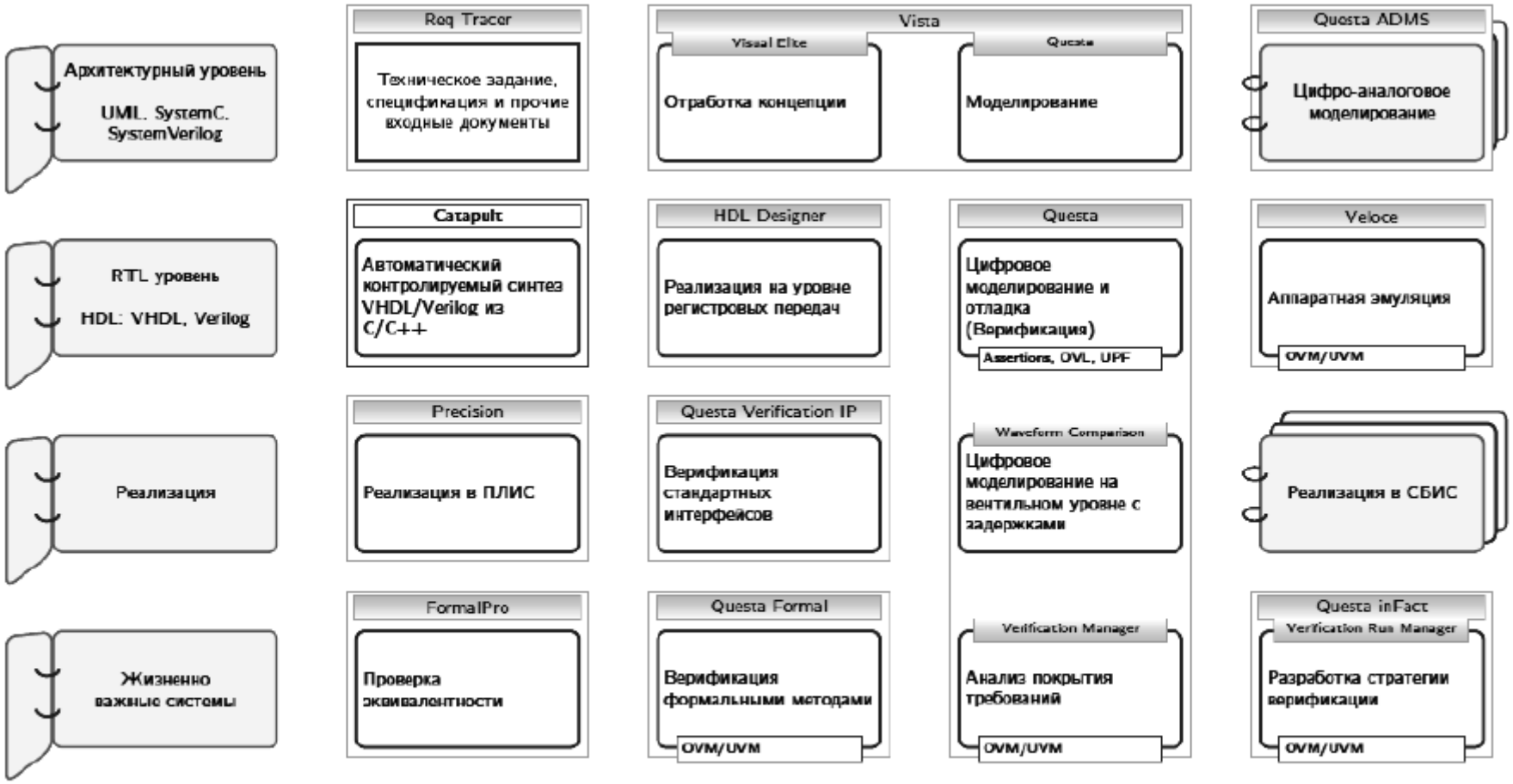



*Рис. 5. Инструменты для функционального проектирования* [16].

Рис. 6.6. Инструменты для функционального проектирования

Для получения окончательной реализации применяются программы-синтезаторы, которые генерируют представления системы на вентильном уровне. Кроме того, различные вспомогательные пакеты позволяют добиться высокого качества и работоспособности продукта, что особенно важно при создании ИС и СнК. Доступны инструменты, облегчающие верификацию и анализ полученного решения.

Работа с проектом на разных этапах и на разных уровнях абстракции производится несколькими взаимодополняющими программными пакетами (Рис. 6.6).

Начальная проработка проекта на архитектурном уровне, ранние оценки работоспособности принимаемых решений, оптимизация и поиск компромиссов производится при помощи пакета Vista. Его средствами формируется рабочая модель аппаратной части на уровне транзакций (используется так называемая TLM 2.0 Scalable Modeling Methodology).

При этом модель создаётся интуитивно понятными графическими интерфейсами, генерирующими компактный SystemC код.

Моделирование предполагает несколько разделённых уровней, позволяющих отделить коммуникации, функциональность, а также энергопотребление и временные задержки, не смешивая их друг с другом.

Так, функциональный уровень совсем не учитывает временной аспект, а отражает только поведение модели с точки зрения того, «что» она делает.

Задержки, ассоциируемые с конкретными реализациями микроархитектурных функций, учитываются во временном уровне, отображая примерную продолжительность вычислений. Функциональность и коммуникации отделяются друг от друга естественным образом.

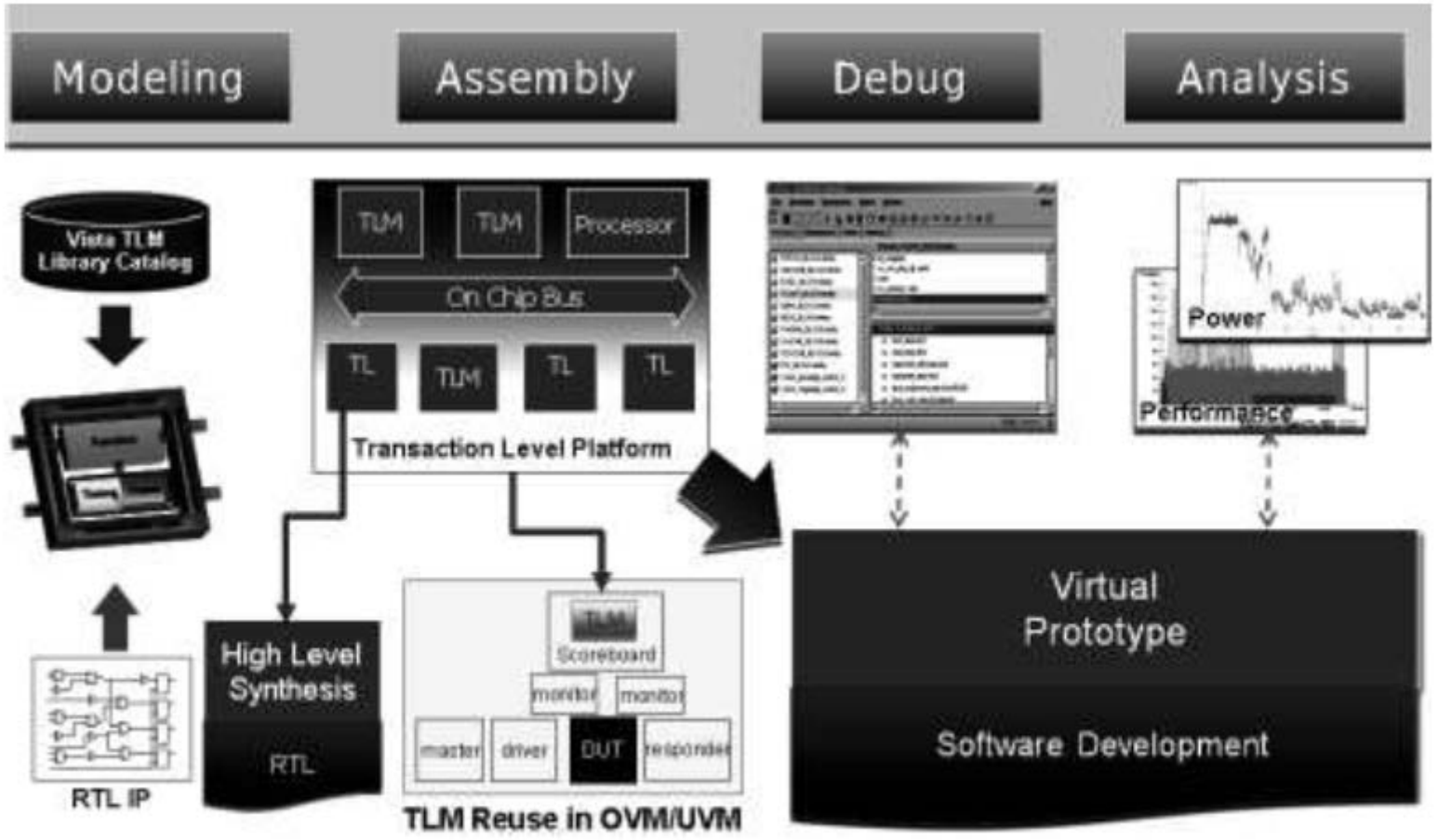



*Рис. 6. Vista* [17].

Рис. 6.7. Структура программы Vista

Полученную в итоге исследования пространства решений оптимизированную и верифицированную модель аппаратуры можно использовать в последствии в качестве виртуального прототипа для дальнейшей разработки программного обеспечения, а также для верификации RTL.

*Pu*

Для интерактивного анализа и трассировки требований используется программа ReqTracer. ReqTracer связывает между собой процесс задания начальной спецификации, реализации и верификации, структурирует все имеющиеся по проекту данные в виде требований и зависимостей между ними.

***Генерацию из высокоуровневых описаний архитектуры на языках C/C++/SystemC в Verilog/VHDL осуществляет пакет Catapult***, обеспечивающий настраиваемый и контролируемый синтез – получаемый RTL может оптимизироваться по энергопотреблению, производительности, объёму или удобству верификации. На вход программе подаются файлы, написанные на **SystemC/C++**, т.е. без

добавления каких-либо дополнительных конструкций. Генерируется управляющая логика и тракт данных, возможен выбор из микроархитектурных альтернатив, возвращение на уровень наверх.

Для интеграции и верификации проектов на смешанном уровне TLM/RTL применяется входящий в состав пакета Vista инструмент Visual Elite. Используя его, системные архитекторы и инженеры могут в интуитивно понятном виде объединять SystemC и HDL, производить верификацию, сравнивая результаты, полученные в ходе симуляции, полученные моделей разной степени детализации, создавать смешанные модели для ускорения симуляции и проверки определённых областей, представляющих интерес для разработчиков.

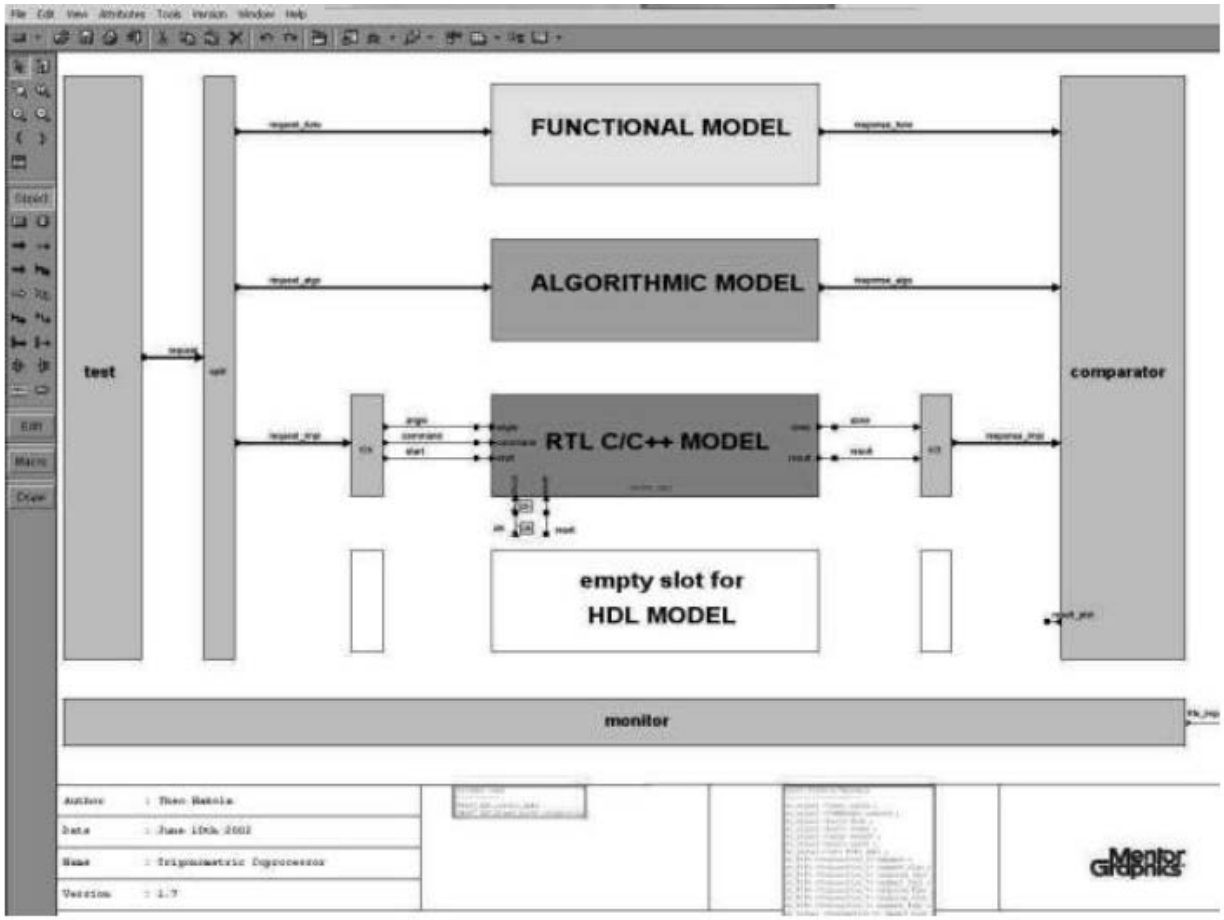



*Рис. **11**. Visual Elite [55].*

Рис. 6.8. Моделирование в Visual Elite

Для ручной работы с HDL-описанием аппаратной части проекта и доведения её до работоспособного состояния применяется HDL

Designer, в котором можно дорабатывать и уточнять описание аппаратуры на RTL уровне.

Симулятор Questa используется для отладки и верификации вплоть до вентильного уровня с реальными задержками сигналов. В случае необходимости доступа к аналоговым объектам может применяться также Questa ADMS, средство для цифро-аналогового моделирования.

Параллельно с проектированием аппаратной части системы должна производиться разработка ПО, для чего используется интегрированная среда разработки (IDE) Sourcery CodeBench Virtual Edition. Sourcery CodeBench является надстройкой для широко используемой и мощной платформы Eclipse и включает в себя улучшенный GNU-компилятор (GCC) с библиотеками, позволяющий компилировать надёжный и производительный код, а также некоторые дополнительные возможности, облегчающие процесс проектирования ПО.

В контексте ESL-ориентированного проектирования одной из наиболее интересных возможностей Sourcery CodeBench Virtual Edition является возможность использования виртуальных платформ, создаваемых пакетом Vista. Это позволяет отлаживать аппаратуру и ПО совместно ещё на ранних стадиях разработки, давая примерные оценки производительности и энергопотребления, и начинать разработку ПО задолго до появления работающей аппаратной части. В течение всего процесса проектирования ПО используется одна и та же среда разработки – как на начальных этапах, когда имеется лишь виртуальный прототип, так и в конце, когда уже существует реальная, работающая аппаратура.

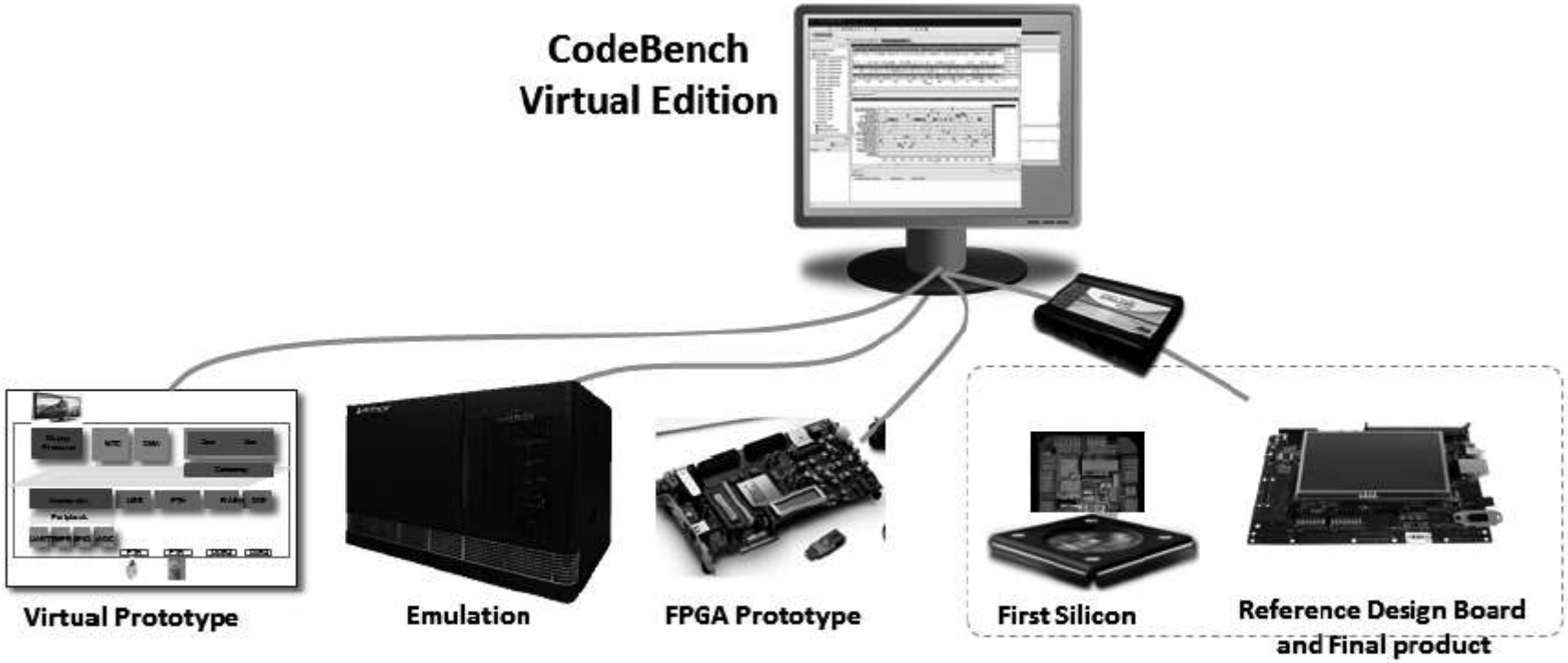



*Рис. 12. Sourcery CodeBench Virtual Edition* [18].

Рис. 6.9. Состав пакета Sourcery CodeBench Virtual Edition

## 6.6. Создание технического задания на проектируемое изделие

Любой процесс проектирования начинается с составления спецификации, или технического задания, в том или ином виде.

Спецификация определяет намеченный функционал и область действия продукта, платформу, при необходимости – архитектурные требования, а также многие операционные и не функциональные аспекты (размер, потребляемая энергия и т.д.).

Гетерогенная электронная система может состоять из большого количества подсистем, создаваемых с помощью различных технологий, самые очевидные из которых – аппаратные средства, программное обеспечение и механические части.

Практическое применение ESL в основном сконцентрировано на описании и моделировании поведенческих аспектов и технических требований систем, что естественно, поскольку поведение легко формализуется и моделируется. Моделирование поведения также позволяет решить много возникающих в процессе проектирования проблем за счёт проверки функциональности системы.

Помимо поведенческих и архитектурных особенностей, другие атрибуты системы, такие как потребляемая мощность, размер, цвет и

т.д., оказывают значительное влияние на коммерческий успех конечного продукта, но их труднее специфицировать в инфраструктуре ESL. Подобные аспекты обычно описываются словесно, на естественном языке.

*Желательно иметь все технические требования в форме, доступной для автоматического анализа*. У *исполняемых и формальных спецификаций* есть три основных потенциальных применения:

- автоматизация отслеживания требований в процессе проектирования и по подсистемам;
- улучшение понимания интеграции гетерогенных систем;
- лучшая интеграция процессов реализации и верификации.

***Процесс управления требованиями.*** Управление требованиями должно делать эти требования видимыми, трассируемыми и доступными в течение всего процесса проектирования.

Изначально требование представляет собой некую потребность заказчика, как он себе её представляет. Оно имеет жизненный цикл и статус с желаемыми сроками его изменения. Это требование самого высокого уровня связывается с одним или более требованиями, отражающими понимание проблемы с точки зрения подрядчика (“features”), которые, в свою очередь, привязываются к одному или нескольким требованиям самого низкого уровня(“sub-features”), каждое из которых отвечает за отдельный компонент системы (рис. 6.10).

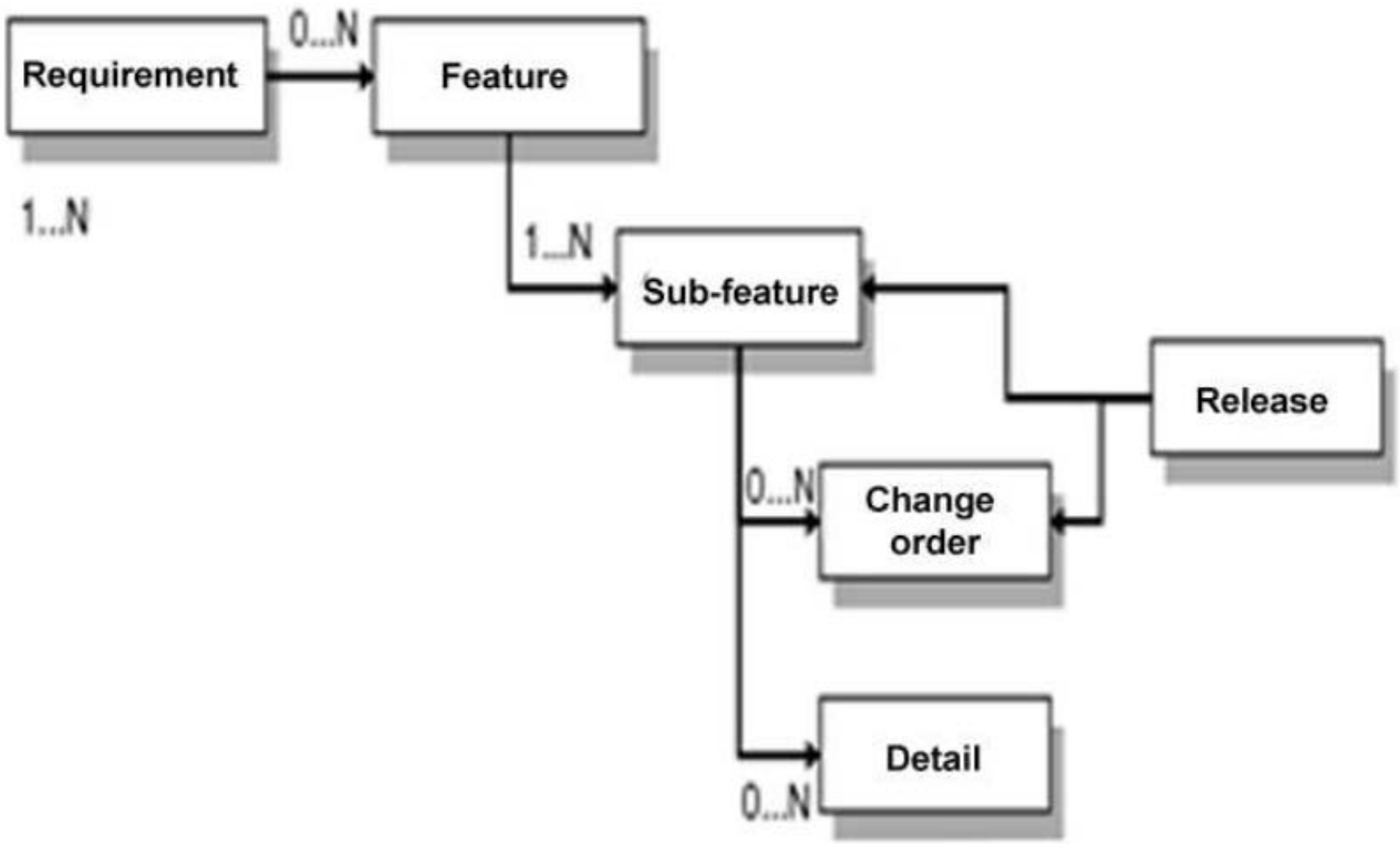



*Рис. 13. Пример реализации процесса управления требованиями* [15].

Рис. 6.10. Пример реализации процесса управления требованиями

***Предметные области проектирования.*** Существует несколько областей проектирования, имеющих между собой некоторые отличия с точки зрения ESL-проектирования, отражающиеся на создании спецификаций.

*В области проектирования трактов потоковой обработки данных*, несмотря на то что они всегда требуют также и некоторый механизм управления, ранняя разработка и анализ алгоритмов, которые определяют преобразования данных ввода-вывода, обычно игнорируются или абстрагируются от потока управления.

*Другая область проектирования – стеки протоколов,* представляющие собой субсистемы коммуникационных систем, которые управляют потоком данных в некотором передающем канале в соответствии с заданным протоколом. Стеки обычно разрабатываются как иерархия уровней и, как правило, имеют управляющий и пользовательский сегменты. Обычно стеки представляют собой низкоуровневое ПО, хотя нижние уровни стеков

могут реализовываться аппаратно, а верхние – на процессоре персонального компьютера. С точки зрения ESL- моделирования, общими чертами стеков протоколов являются очень жёсткие требования реального времени, очень модульные архитектуры, коммуникации путём передачи сообщений, ограниченные потребности параллелизма.

*В отдельную область выделяются BcC.* Их отличительными чертами являются встраивание в некоторый объект, требования реального времени, взаимодействие с окружающим миром не через традиционный пользовательский интерфейс, а через низкоуровневые команды и сигналы, и нефункциональные ограничения, вроде энергопотребления и размера.

Для встраиваемых систем, как правило, платформа известна заранее, и проектирование с точки зрения неограниченности пространства проектирования контрпродуктивно. Кроме того,

требуется очень детальное моделирование платформы с целью проверки, удовлетворяет ли она ограничениям.

***Исполняемая спецификация*** – поведенческое описание проектируемого компонента или объекта системы, которое отражает определенный функционал с учётом временных задержек. Основная цель исполнимой спецификации заключается в проверке того, что поведение разрабатываемой системы удовлетворяет системным требованиям после интеграции с другими компонентами системы и того, является ли реализация объекта непротиворечивой указанному поведению.

На раннем этапе проектирования исполняемая спецификация, как правило, является симуляцией некоторой системной функции (или функций). В процессе детализации и анализа различных архитектурных вариантов исполнения эта спецификация превращается в исполняемую архитектурную спецификацию, а затем – в исполняемую спецификацию реализации.

В качестве языков для спецификирования могут использоваться:

MCode от MATLAB, который обычно применяется на ранних фазах разработки алгоритмов ЦОС и управления; Rosetta, созданная для облегчения декларативного спецификирования;

SystemC, позволяющий реализовывать параллельные процессы, события, сигналы во многом подобно языкам описания аппаратуры, но с объектно-ориентированным подходом, шаблонами и типами данных из C++;

SDL (Specification and Description Language), созданный для спецификирования комплексных приложений с ограничениями по времени, интерактивных и с управляемыми событиями, пригодный, в том числе, для анализа нефункциональных и статических свойств;

UML – язык графического описания для объектного моделирования, разработанный для ПО, но пригодный и для аппаратуры, и применявшийся как основной язык для моделирования аппаратных систем;

XML – язык общего назначения для создания на своей основе произвольных специализированных языков, пригодный для описания разных типов информации;

Bluespec (Bluespec SystemVerilog и Bluespec SystemC) – декларативный язык для создания аппаратных спецификаций, с существующим компилятором, позволяющим синтезировать SystemVerilog или SystemC код.

Для функциональной верификации также хорошо подходят *аспектно-ориентированные языки*, позволяющие выделять сквозные аспекты, влияющие сразу на некоторое количество объектов при объектно-ориентированном подходе. Самые широко используемые аспектно-ориентированные языки – Aspect J (вариант для Java), AspectL (расширение Common Lisp) и много других аспектных модификаций не аспектных языков.

## 6.7. Исследование пространства проектных решений

Разработка аппаратуры и программного обеспечения отдельно друг отдруга может привести к реализациям системы, спроектированной с недостаточным запасом, или систем, которые не отвечают всем нефункциональным требованиям, таким как надёжность, энергопотребление и т.д. Проектные решения по назначению вычислительных ресурсов для задач могут быть излишне строгими, что приводит к слишком дорогостоящим реализациям. Бороться с этим призвано исследование пространства проектных решений, или же анализ перед разделением и после в ESL-маршруте проектирования.

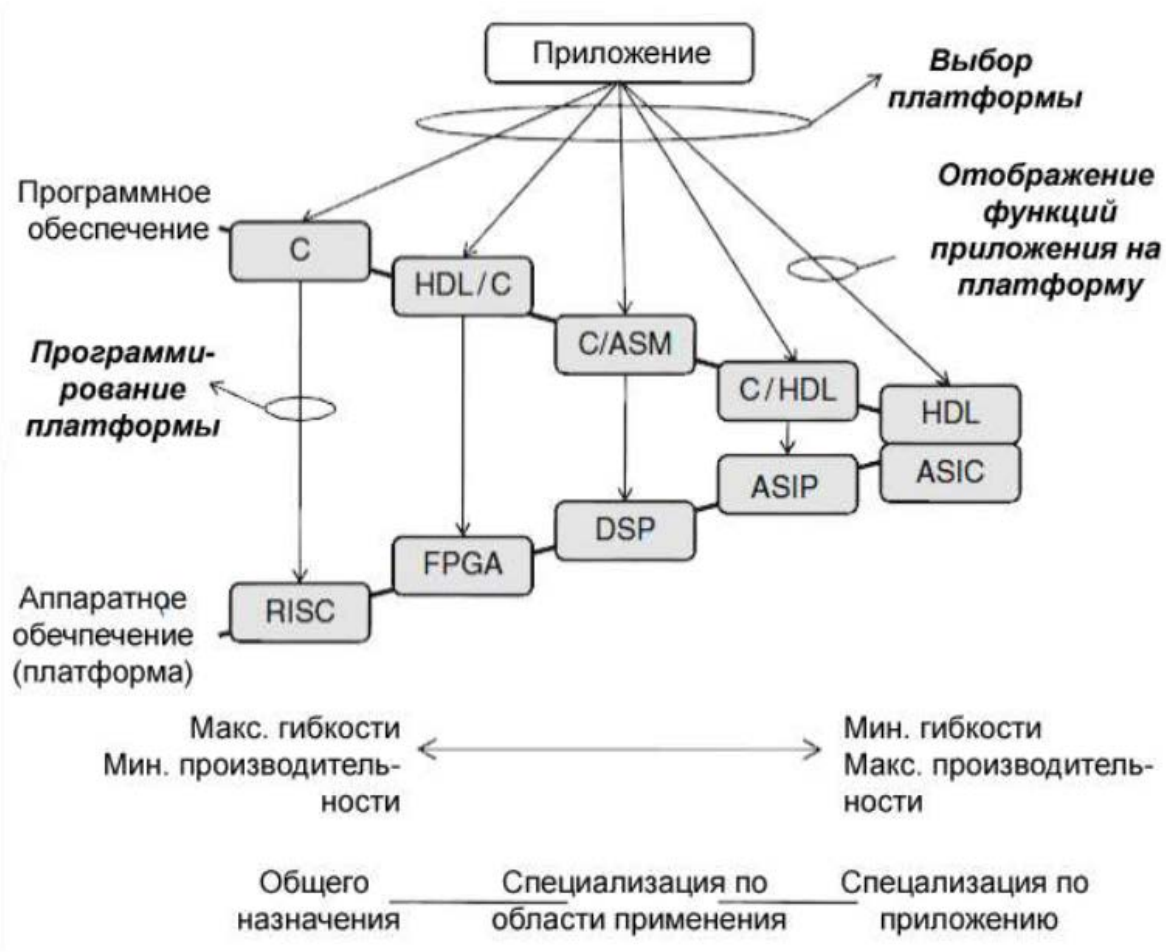



*Рис. 14. Пример выбора отображений функциональности на архитектуру* [2].

Рис. 6.11. Пример выбора отображения функциональности на архитектуру

Системный синтез включают три этапа – выделение ресурсов архитектурной модели, привязку объектов функциональной спецификации к выделенным ресурсам и планирование задач должным образом. Поэтому пространство проектирования задано набором всех возможных *перестановок выделений, привязок и планирования*.

Любую такую тройку, удовлетворяющую определенному числу дополнительных нефункциональных ограничений, таких как ограничения стоимости, производительности, электропитания, температуры и т.д., называют допустимым решением. Из допустимого решения может быть легко получена соответствующая структурная реализация.

***Статический анализ.*** Статические методы используются при отсутствии исполняемых спецификаций и моделей, основываются на анализе спецификаций и обычно неформальны.

Несмотря на предпочтительность использования динамических моделей, эти методы также могут использоваться при проектировании. Анализ сложности системы, заимствующий принципы «анализа функциональных точек» из программной инженерии, может быть применен для оценки многих характеристик, включающих стоимость, производительность, затраты на разработку и т.д.

***Анализ функциональных точек***. Хорошо зарекомендовавшим себя методом оценки трудозатрат на создание ПО является анализ функциональных точек, позволяющий оценивать затраты на разработку и сопровождение некоторого программного продукта. Кратко суть метода заключается в оценке трудоёмкости на основе функционала программного продукта, информация о котором получается из имеющихся требований, руководств, предписаний и других документов, описывающих будущую (или уже

существующую) систему, так как оцениваться может разработка, доработка или готовый продукт.

***Модель издержек разработки.*** Для оценки стоимости разработки ПО, наряду с методом функциональных точек, также часто применяется модель издержек разработки (COnstructive COst MOdel, COCOMO) – алгоритмическая модель оценки стоимости разработки ПО. Модель предоставляет три уровня оценки – базовый, для быстрых оценок стоимости разработки и не дающий большой точности, средний, рассчитывающий трудоёмкость разработки также исходя из множества факторов, влияющих на стоимость, и детальный, включающий в себя множество характеристик с оценкой их влияния на различные этапы процесса разработки.

За единицу размера программы берутся тысячи строк кода (KLOC – kilo lines of code). Выделяются три типа проектов:

- органический, представляющий собой маленькие команды с хорошим опытом работы и довольно мягкими требованиями к разработке,
- промежуточный, для средних по размеру команд со смешанным опытом разработки и смешанными требованиями, и
- встроенный, когда разработка ведётся при наличии множества жёстких ограничений.

***Анализ спецификаций аппаратных систем или систем с преобладанием аппаратуры***. Некоторые виды методов статического анализа из области программного обеспечения были также применены для анализа систем со значительными аппаратными частями или для программно-аппаратных систем.

***Ility анализ.*** Исследование "ility" (или "ilities") – исследование атрибутов, таких как надежность, пригодность для обслуживания,

удобство использования, важность объекта, анализ отказа и т.д. (в английском языке эти слова оканчиваются на "-ility").

***Анализ требований.*** Требования могут быть проанализированы статически для оценки «функционального веса» определенных модулей проекта, и, таким образом, усилий и времени на разработку.

***Динамический анализ***. В случае анализа динамического исполнения или симуляции спецификаций посредством моделей исследуются такие характеристики, как более точные оценки производительности (задержки, пропускная способность, время ожидания), факторы, которые влияют на характеристики с временными критериями, такие как причины перегрузки, арбитражные политики и приоритеты, алгоритмы планирования для исполнения моделей (которые могут использоваться после разделения, например, как политики планирования программного обеспечения). Из динамической симуляции исполняемых моделей на этапе перед разделением требуется получить важные характеристики

базовой функциональности, которые впоследствии используются при разделении. Они включают: затраты вычислительных ресурсов, затраты на коммуникации, затраты ресурсов энергопотребления.

***Алгоритмический анализ*** позволяет получать такие оценочные критерии, как ожидаемое выполнение или вычислительные затраты, объем передаваемых данных, требуемый для того, чтобы обработать кадры, пакеты или токены посредством алгоритма, частота возникновения ошибок, частота передачи ошибочных битов. Этот анализ особенно важен для алгоритмов обработки потоков данных, используемых в сигнальных приложениях и приложениях для обработки изображений.

Известны и широко используются также такие инструменты, как MATLAB от MathWorks, Simulink и родственные инструменты (Stateflow и различные библиотеки реализации). В инструменте Mirabilis используются возможности проекта Ptolemy университета Berkeley. Инструменты  нализа конечных автоматов предоставляются

поставщиками инструментария встроенного программного обеспечения, часто использующими UML – например, IBM Rational Rose, iLogix Rhapsody, Artisan Software Tools, Esterel Technologies, MathWorks (Stateflow), CoWare, Prosilog, CoFluent и другие. Иногда подобные пакеты интегрируются в основанную на HDL среду проектирования и верификации (Cadence, Mentor и Synopsys).

***Сценарии анализа и моделирование***. Набор методик анализа может зависеть от того, в которой области приложение используется. Например, в проводной и беспроводной связи используются модели окружающей среды для каналов коммуникации – "модели канала" (которые могут быть смоделированы в программном обеспечении или использованием специальных аппаратных устройств). Введение реальных и искусственных механизмов отказа (например, помехи, затухание сигнала, фазовое искажение, отражение и другие эффекты в приложениях беспроводной связи) в таких моделях окружающей среды чрезвычайно важны для того, чтобы определить устойчивость,

скорость динамического отклика и адаптируемость основных базовых алгоритмов и функций, которые должны быть реализованы, и анализа новых алгоритмов в стрессовых условиях.

## 6.8. Разделение

В основе кодизайна лежит разделение ПО и аппаратуры на системном уровне, что позволяет сполна использовать специфику этих составляющих для лучшего удовлетворения требованиям производительности, гибкости, энергопотребления и т.д. Разделение ускоряет общий процесс проектирования, благодаря вычленению отдельных работ. При введении спецификаций на межмодульные интерфейсы эти модули вообще могут разрабатываться отдельными командами разработчиков, при этом также упрощается верификация.

Поскольку отображение на системном уровне оперируетгетерогенными объектами, оно позволяет разделить различающиеся между собой и ортогональные аспекты:

1) Вычисления и коммуникации. Вычисления обычно оптимизируются вручную, тогда как коммуникации – при помощи шаблонов.

2) Функциональность и архитектура. Эти аспекты обычно задаются и проектируются раздельно.

3) Поведенческий аспект и производительность.

Производительность может представлять и нефункциональные требования.

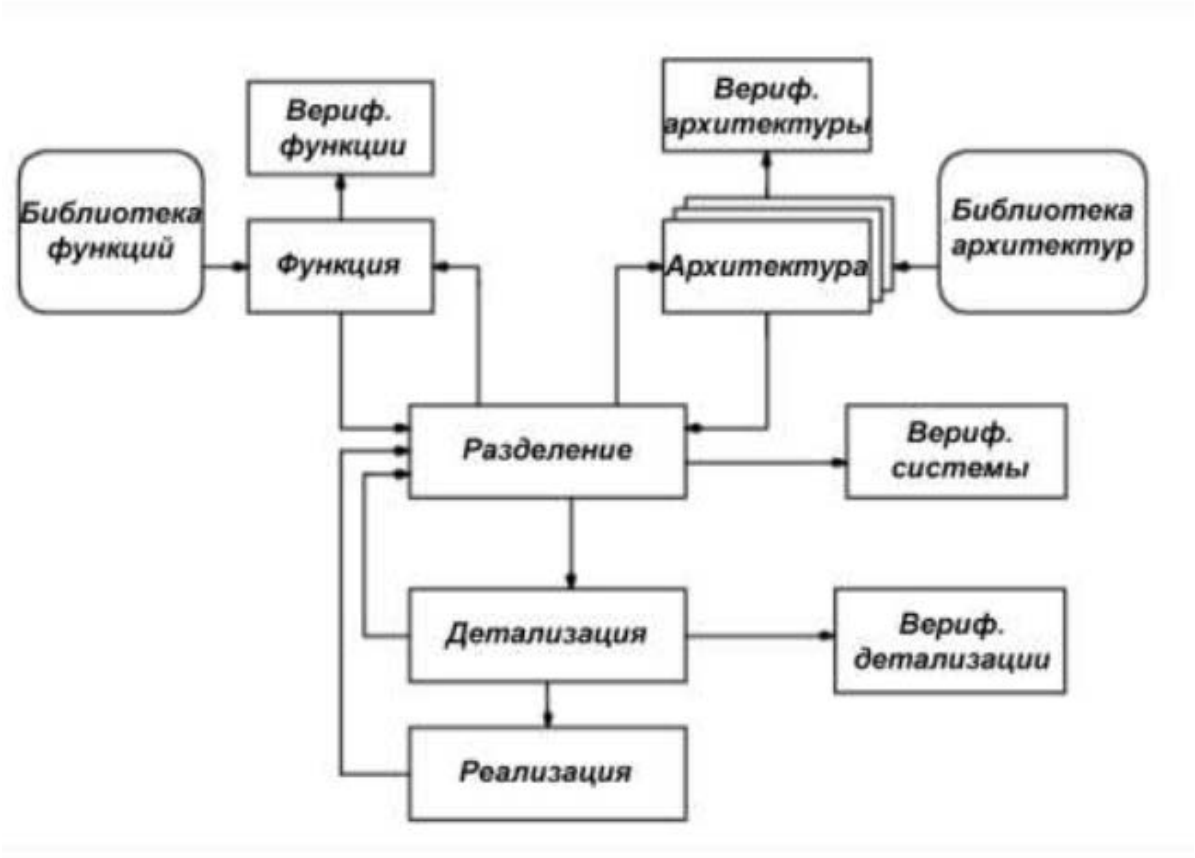



*Рис. 15. Представление возможного процесса разделения* [15].

Рис. 6.12. Представление возможного процесса разделения

В процессе разделения можно выделить несколько основных шагов. Первый, вероятно, самый сложный – выделение

индивидуальных функциональных компонентов из начальной, "монолитной" спецификации.

Ключевой задачей на данном этапе является выделение адекватных по размеру и сложности фрагментов, которые могут быть поручены разным командам, работающим более-менее независимо, и выделение параллелизма в масштабе отдельных процессов или задач.

Как правило, редкая команда разработчиков может хорошо разрабатывать системы одновременно и на RTL уровне для аппаратуры, и на уровнях C/Assembler для ПО, поэтому чёткое разделение и разграничение ответственностей необходимы.

Второй шаг при разделении – определение архитектуры системы с учётом требований, заданных в спецификации, как правило – стоимости, энергопотребления, производительности и многих других, зачастую явно неуказанных – надёжности, сопровождения,

простоты использования, повторным использованием компонентов, опытом команды разработчиков и так далее.

Третий шаг включает в себя назначение функциональных блоков элементам архитектуры с учётом ограничений по занимаемому объёму, энергопотреблению и производительности.

Четвёртый шаг – детальное описание интерфейсов между различными частями проекта; коммуникации внутри них создаются инструментами синтеза или вручную. Выбор данных интерфейсов важен для верификации и обеспечения взаимодействия между командами разработчиков.

***Функциональная декомпозиция***. Для эффективного разделения системы желательно начать с модели, максимально удалённой от деталей реализации. Предпочтительно, чтобы это была чисто функциональная модель с возможностью представления параллелизма на уровне процессов, дополненная комментариями об ограничениях. Этим требованиям удовлетворяют два подхода:

описание модели на изначально параллельном функциональном языке или автоматическое извлечение параллелизма из последовательного языка.

Примерами первого подхода являются Simulink от MathWorks, LabVIEW/MatriX (National Instruments), SPW (CoWare), Lustre (Esterel). Ни один язык не может эффективно и лаконично покрывать одновременно аспекты систем с доминированием управляющей части, в которых более важны задержки и ветвления алгоритмов, и аспекты систем с доминированием данных, в которых важнее арифметические операции и пропускная способность.

Второй подход является гораздо более сложной и многоуровневой задачей, однако при этом сулит немалые выгоды. Как правило, данные вопросы исследуются в контексте проектирования компиляторов для векторных и параллельных машин.

***Архитектурное описание***. Архитектура обычно рассматривается как набор компонентов, которые предоставляют доступ для реализации некоторого поведения. Эти компоненты обычно соединены в виде некоторого графа или списка соединений для задания способа обмена информацией между ними.

Архитектурное описание наиболее близко к аппаратной реализации системы и может иногда быть рассмотрено как абстрактная схема аппаратной части проекта

В другом, более комплексном случае архитектурное описание также включает элементы, которые не могут быть напрямую ассоциированы с физическими устройствами. Примеры таких элементов – сетевые протоколы, набор инструкций ЦП, политика доступа к разделяемым ресурсам.

Цель разделения – нахождение оптимальных соответствий функциональных компонентов архитектурным для обеспечения требований к проектируемому изделию. Это обычно означает

нахождение лучших компромиссов между различными требованиями.

Одно из представлений разных уровней абстракции и актуальный метод реализации системы представлен на Рис. 6.13. Наиболее абстрактный уровень описывается функциональной несинхронизированной моделью A, также часто называемой моделью спецификации. Добавлением архитектурной информации и приблизительного функционального временного поведения получается модель B, называемая моделью компонентов сборки. Выбор арбитража шин позволяет получить некоторое представление о реальном «тайминге» коммуникаций, давая модель арбитража шин C. Из этой модели можно пойти двумя путями для достижения конечной цели – через поведенческую шинно- функциональную модель D или через потактовую модель вычислений можно получить модель RTL описания F. Данный подход можно реализовывать через SystemC.

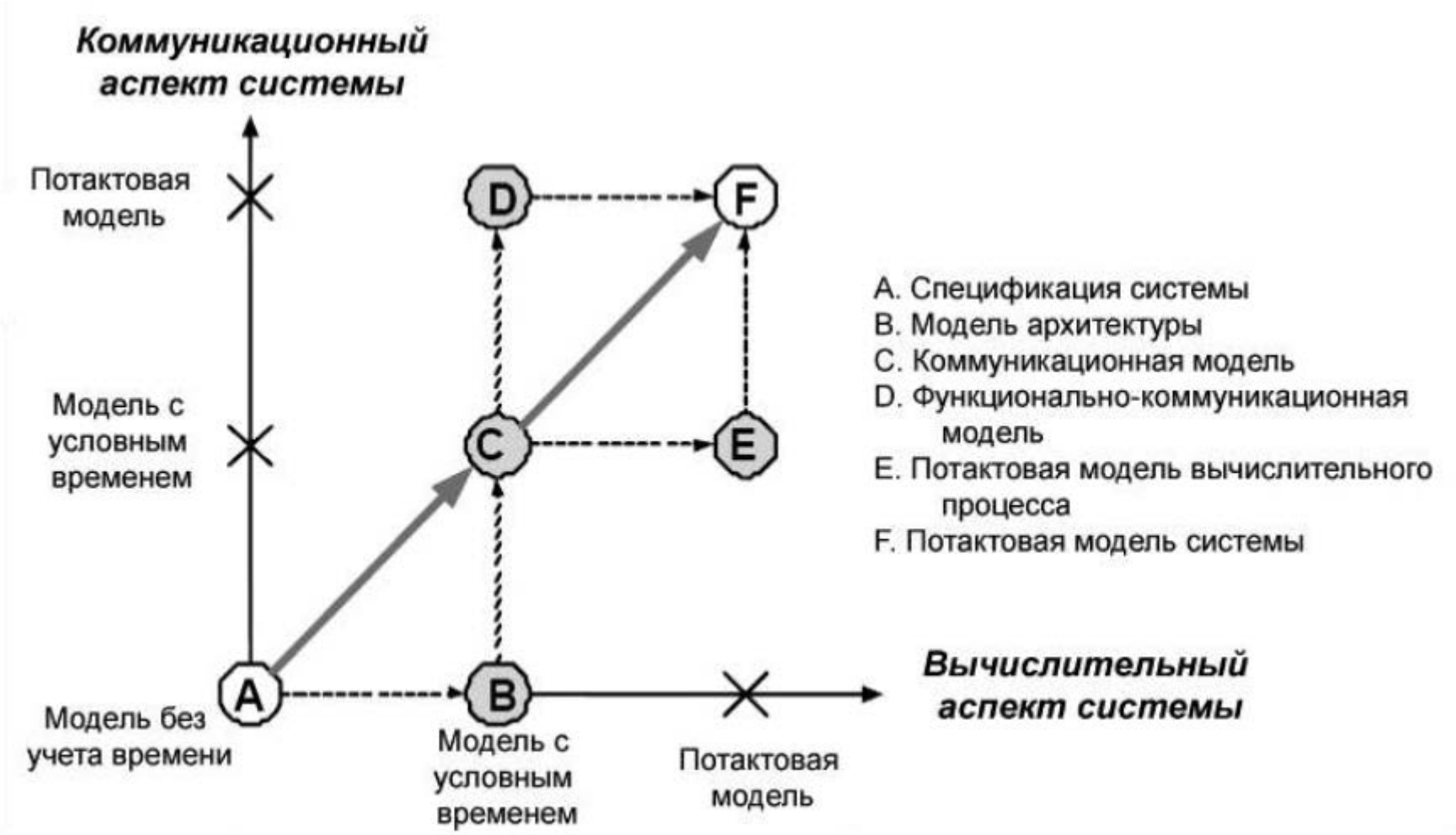



*Рис. 16. Абстрактные модели* [20].

Рис. 6.13. Абстрактные модели

***Аппаратное разделение***. Существование большого количества вариантов реализации аппаратуры, их растущая сложность требуют добавления аспектов сопровождаемости и расширяемости в процесс разделения, причём, даже для относительно жёсткого аппаратного сектора.

Проектирование аппаратуры выглядит всё больше как конфигурирование уже существующей платформы, когда основная работа сдвигается в программную область, где более дешёвые инструменты и меньше стоимость исправления ошибок. Это также приводит к новым требованиям в плане надежности и детерминизма для процесса внедрения программного обеспечения – верификация не может больше считаться роскошью и выполняться задним числом. Формальная проверка на отсутствие ошибок в конечном продукте становится обязательным требованием.

***Программное разделение***. Так как сейчас только критические к задержкам и потребляющие много энергии части систем реализуются в аппаратуре, основная часть функционала остаётся на долю программного обеспечения процессоров. Ввиду того, что может требоваться выполнение сразу нескольких задач с разными требованиями к ресурсам и времени, для управления часто применяются операционные системы, с многими уровнями абстрагирования аппаратуры, как у сетевых стеков. То есть получается, что программная архитектура не менее, а в некоторых случаях и более важна, чем аппаратная и имеет существенное воздействие на производительность всей системы. В то же время хорошо продуманная программная архитектура позволяет эффективно использовать аппаратные платформы, добавлять и убирать функции, тем самым генерируя производные продукты от общей базы исходных кодов.

Разделение между несколькими процессорами позволяет использовать крупногранулярный параллелизм, однако, накладные расходы на коммуникации и синхронизацию могут превышать получаемую пользу.

Для компенсации негативного воздействия необходимости обмениваться данными могут использоваться специализированные механизмы, вроде выделенных каналов между процессорами. Типичный случай – процессор общего назначение и DSP-сопроцессор.

Программное разделение должно учитывать наличие и характеристики операционной системы, так как многие решения могут зависеть от этого. Кроме того, может быть случай, когда отдельные свойства и функции могут быть отображены на компоненты ОС (например, драйверы).

***Разделение памяти.*** Также влияет на характеристики системы, и потому ему должно уделяться отдельное внимание. Обычно память

проектируется в виде иерархии, с виртуализацией больших, медленных и дешёвых блоков быстрыми и дорогими кэшами и сверхоперативными запоминающими устройствами. Организация памяти и способ хранения данных вместе с доступом к ним, должны проектироваться одновременно для достижения максимальной возможной производительности. На задание характеристик блоков памяти влияют также и ограничения по энергопотреблению и величине задержек.

***Реконфигурирование***. Использование реконфигурируемых вычислителей призвано бороться с растущей стоимостью проектирования и создания ASIC. Добавление реконфигурируемого блока к жёстко заданной аппаратуре позволяет поддерживать больше различных приложений, чем в случае традиционных случаях, когда вся аппаратная часть жёстко задана.

***Реализация коммуникаций***. Часто коммуникации могут быть узким местом системы и практически всегда сильно влияют на её

свойства. Существует два основных способа реализации интерфейсов: синтез и использование шаблонов.

При использовании шаблонов применяется набор готовых решений, которые настраиваются исходя из функциональных требований и требований производительности.

Подходы к синтезу интерфейсов, как правило, предполагают проектирование совместимых протоколов, синтез преобразователей протоколов, а также выявление невозможности совмещения разных протоколов с существующими ограничениями. Для определения, совместимы протоколы или нет, обычно проверяется, согласуются ли между собой интерфейсы при том потоке данных, который планируется передавать. Размеры блоков, передаваемых данных, могут различаться, но средние пропускные способности приёмника и передатчика должны соответствовать друг другу. Протокол на сигнальном уровне, как правило, описывается в семантике конечных

автоматов в расчёте на использование различных техник синтеза протоколов.

## Список обозначений

| | |
|---|---|
| BPM | Business Process Modeler - модуль построения моделей бизнес-процессов в форме диаграмм потоков данных |
| BPwin | CASE-средство , реализующее в качестве методологии IDEF0 |
| CAD | Computer Aided Design – автоматизированное проектирование |
| CAE | Computer Aided Engineering– поддержка инженерных расчетов |
| CALS | Continuous Acquisition and Life cycle Support - непрерывная информационная поддержка поставок и жизненного цикла |
| CAM | Computer Aided Manufacturing - компьютерная поддержка изготовления |

| | |
|---|---|
| CAPP | Computer Aided Process Planning - средства планирования технологических процессов |
| CASE | Computer-Aided Software/System Engineering – разработка программного обеспечения/программных систем с использованием компьютерной поддержки |
| CMMI | |
| COM | Component Object Model – компонентная модель объектов |
| CORBA | Common Object Request Broker Architecture – общая архитектура с посредником обработки запросов объектов |
| DCOM | Distributed COM – распределенная COM |
| DFD | data flow diagrams — диаграммы потоков данных |
| ECAD | Electronic CAD- системы автоматизированного проектирования (САПР) для радиоэлектроники |

| | |
|---|---|
| EDA | Electronic Design Automation – автоматизированное проектирование электроники |
| EDM | engineering data management – управление инженерными данными |
| ERD | entity-relationship diagrams |
| ERwin | CASE-средство, которое в качестве методологии использует IDEF1X |
| ERX | Entity-Relationship eXpert - модуль концептуального моделирования данных |
| ER-модель | entity-relationship model, модель «сущность — связь» |
| ICAM | Integrated Computer-Aided Manufacturing - интегрированная компьютеризация производства |
| IDEF | Integrated DEFinition |
| IDEF0 | |
| IDEF0 | метод функционального моделирования |

| | |
|---|---|
| IDEF1 | метод моделирования информационных потоков |
| IDEF1X | |
| IDEF1X | метод моделирования данных и проектирования реляционных баз данных |
| IDEF2 | метод динамического моделирования систем |
| IDEF3 | метод получения описания функционирования системы и моделирования как причинно-следственных связей |
| IDEF4 | метод объектно-ориентированного проектирования. |
| IDEF5 | метод получения онтологического описанияи исследования сложных систем |
| ISO | |
| JTAG | Сокращение от англ. Joint Test Action Group; произносится «джей-та́г») — название рабочей группы по разработке стандарта IEEE 1149 |

| | |
|---|---|
| IT | Информационные технологии |
| MSF | |
| OLE I | Object Linking and Embedding – связывание и внедрение объектов |
| OSTD | (ObjectStateTransitionDescription)– «внешнее описание» |
| PDM | Product Document Management – система управления данными об изделии |
| PFD | Process Flow Description - «внутреннее описание» и описания переходов из одного состояния в другое |
| PIM | Product information management – управление информацией об изделии |
| PLM | Product Lifecycle Management – термин используется для обозначения процесса управления полным циклом изделия |

| | |
|---|---|
| RAD | |
| RDM | Relational Data Modeler - модуль реляционного моделирования |
| RUP | |
| SADT | Structured Analisys and Design Technique - технология структурного анализа и проектирования |
| SWEBOK | Software Engineering Body of Knowledge - Сфера знаний в области разработки программного обеспечения |
| TDM | technical data management - управление техническими данными |
| TIM | technical information management - управление технической информацией |
| UML | Universal Modeling Language |
| WRM | Workgroup Repository Manager - менеджер |

| | |
|---|---|
| | репозитория рабочей группы |
| АПК | Аппаратно-программный комплекс |
| АС | Автоматизированная система |
| ИПЭ | Информационная поддержка эксплуатации |
| ПАК | Программно-аппаратный комплекс |
| ПО | Программное обеспечение |
| ПС | Программные средства |

## Приложение 2. Основная литература

1. Быковский С.В., Горбачев Я.Г., Ключев А.О., Пенской А.В., Платунов А.Е. Сопряжённое проектирование встраиваемых систем. (Hardware/Software Co-Design). Часть 1. Учебное пособие. – СПб.: Университет ИТМО, 2016. – 108 с.

2. Быковский С.В., Горбачев Я.Г., Ключев А.О., Пенской А.В., Платунов А.Е. Сопряжённое проектирование встраиваемых систем. (Hardware/Software Co-Design). Часть 2. Учебное пособие. – СПб.: Университет ИТМО, 2016. – 105 с.

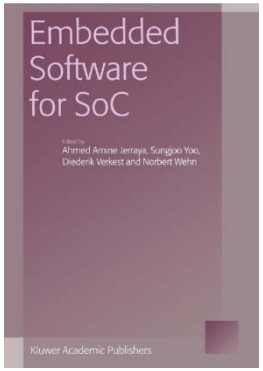
3. Платунов А.Е, Постников Н.П. Высокоуровневое проектирование встраиваемых систем. – СПб.: НИУ ИТМО, 2011. – 121 с.

4. Платунов А.Е, Постников Н.П. Высокоуровневое проектирование встраиваемых систем. – СПб.: НИУ ИТМО, 2013. – 172 с.

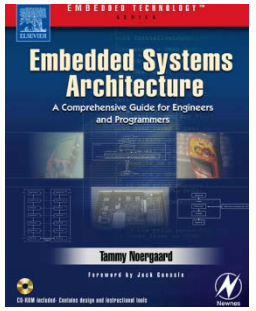
5. Гончаровский О.В. Проектирование встроенных управляющих систем реального времени: учеб. пособие / О.В. Гончаровский, Н.Н.Матушкин, А.А. Южаков – Пермь: Изд-во Перм. нац. исслед. политехн. ун-та, 2013. – 1xx с

6. Cagkan Erbas. System-Level Modeling and Design Space Exploration for Multiprocessor Embedded System-on-Chip Architectures. -Vossiuspers UvA . Amsterdam University Press, 2006.

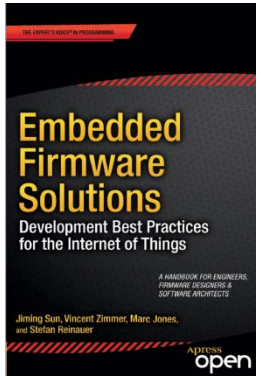

7. Embedded Software for SoC. Edited by Ahmed Amine Jerraya TIMA Laboratory, Sungjoo Yoo France TIMA Laboratory, Diederik Verkest France IMEC, Belgium and Norbert Wehn University of Kaiserlautern, Germany. - ©2004 Springer Science + Business Media, Inc.



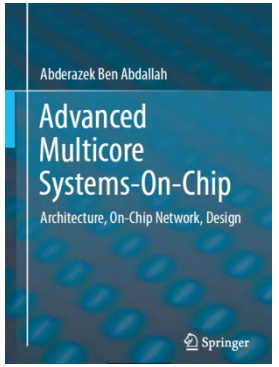
8. Embedded Systems Architecture. A Comprehensive Guide for Engineers and Programmers. By Tammy Noergaard.- Copyright © 2005, Elsevier Inc. All rights reserved.



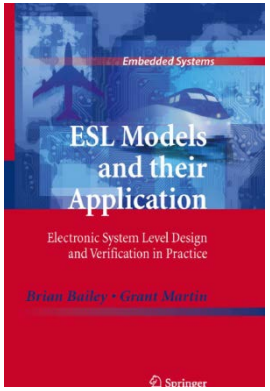

9. Embedded Firmware Solutions. Development Best Practices for the Internet of Things. Jiming Sun, Vincent Zimmer, Mars Jones, and Stefan Reinauer. Apress Open.



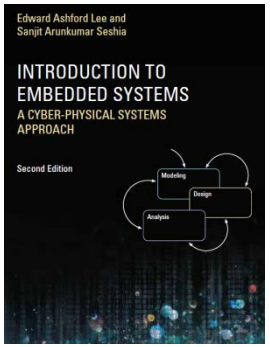

10. Abderazek Ben Abdallah. Advanced Multicore Systems-On-Chip. Architecture, On-Chip Network, Design. ISBN 978-981-10-6091-5 ISBN 978-981-10-6092-2 (eBook). DOI 10.1007/978-981-10-6092-2.



11. Brian Bailey, Grant Martin. ESL Models and theirApplication. Electronic System Level Design and Verification in Practice. - ISBN 978-1-4419-0964-0 e-ISBN 978-1-4419-0965-7. DOI 10.1007/978-1-4419-0965-7. Springer New York Dordrecht Heidelberg London




12. E. A. Lee and S. A. Seshia. Introduction to Embedded Systems - A Cyber-Physical Systems Approach, Second Edition, MIT Press, 2017.




13. Барретт С. Ф., Пак Д. Дж. Встраиваемые системы. Проектирование приложений на микроконтроллерах семейства 68HC12 / HCS12 с применением языка C. — М.: Издательский дом «ДМК_пресс», 2007. — 640 с.

## Литература, предлагаемая ИТМО

## Литература, предлагаемая студентам направления «Встроенные вычислительные системы»

### Проектирование встроенных систем

Peter Marwedel, Embedded System Design, Embedded Systems Foundations of CyberPhysical Systems, 2nd Edition, 2011

Alexandru Andrei, Dissertation, Energy Efficient and Predictable Design of Real-Time Embedded Systems, 2007, Department of Computer and Information Science, Linköping University, Sweden

Masahiro Fujita, Indradeep Ghosh, and Mukul Prasad, VERIFICATION TECHNIQUES FOR SYSTEM-LEVEL DESIGN, 2008

Gabriela Nicolescu.. Pieter J. Mosterman, Model-Based Design for Embedded Systems, 2010

Gregory J. Pottie and William J. Kaiser, Principles of Embedded Networked Systems Design, 2005

Brian Bailey, Grant Martin, Andrew Piziali, ESL DESIGN AND VERIFICATION A PRESCRIPTION FOR ELECTRONIC SYSTEM-LEVEL METHODOLOGY, 2007

Архитектура встроенных систем

John L. Hennessy, Stanford University, David A. Patterson, University of California at Berkeley, Computer Architecture, A Quantitative Approach, Fourth Edition, 2006

Wayne Wolf, High-Performance Embedded Computing. Architectures, Applications, and Methodologies, 2007

Sudeep Pasricha, Nikil Dutt, On-Chip Communication Architectures. System on Chip Interconnect, 2008

G. Nocolescu, A.A. Jerraya, GLOBAL SPECIFICATION AND VALIDATION OF EMBEDDED SYSTEMS. Integrating Heterogeneous Components, 2007 Лекции - слайды Sudeep Pasricha and Nikil Dutt On-Chip Communication Architectures, 2008, в формате Power Point

Phaophak Sirisuk Fearghal Morgan, Tarek El-Ghazawi Hideharu Amano (Eds.) Reconfigurable Computing: Architectures, Tools and Applications 6th International Symposium, ARC 2010 Bangkok, Thailand, March 17-19, 2010 Proceedings

Редакторы Scott Hauck и Andrтe DeHon RECONFIGURABLE COMPUTING. THE THEORY AND PRACTICE OF FPGA-BASED COMPUTATION. 2008

Моделирование встроенных систем

Axel Jantsch Modeling Embedded Systems and SoCs. Concurrency and Time in Models of Computation, 2004

Диссертация: Cagkan Erbas, System-Level Modeling and Design Space Exploration for Multiprocessor Embedded System-on-Chip Architectures, 2006

Richard Munden, ASIC AND FPGA VERIFICATION: A GUIDE TO COMPONENT MODELING, 2005

Практика проектирования систем Philip Simpson, FPGA Design. Best Practices for Team-based Design, 2010

Gary Stringham, Hardware/Firmware Interface Design. Best Practices for Improving Embedded Systems Development, 2010

Jivan S. Parab, Santosh A. Shinde, Vinod G. Shelake, Rajanish K. Kamat, Gourish M. Naik, Practical Aspects of Embedded System Design using Microcontrollers, 2008

Интеллектуальные электрические сети

Редакторы Zofia Lukszo, Geert Deconinck, Margot P.C.Weijne Securing Electricity Supply in the Cyber Age. Exploring the Risks of Information and Communication Technology in Tomorrow’s Electricity Infrastructure TOPICS IN SAFETY, RISK, RELIABILITY AND QUALITY, 2010

Clark W. Gellings, P.E. The Smart Grid: Enabling Energy Efficiency and Demand Response, 2009

Tony Flick, Justin Morehouse, Securing the Smart Grid Next Generation Power Grid Security, 2011

Языки проектирования

David Robinson Aspect-Oriented Programming with the e Verification Language. A Pragmatic Guide for Testbench Developers, 2007

Prabhat Mishra, Nikil Dutt, Processor Description Languages, 2008

Основы цифровой техники

Жан М.Рабаи и др. Цифровые интегральные схемы. Методология проектирования, 2007 г., перевод книги «Digital Circuit Design …», которая то же есть в оригинале.

Frank O’Brein, The Appolo Guidance Computer. Architecture and Operation, 2010