МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

А.Е. Платунов, Н.П. Постников

ВЫСОКОУРОВНЕВОЕ ПРОЕКТИРОВАНИЕ ВСТРАИВАЕМЫХ СИСТЕМ

(Часть 1)

Учебное пособие

Санкт-Петербург

2011

Платунов А.Е, Постников Н.П. Высокоуровневое проектирование встраиваемых систем. – СПб.: НИУ ИТМО, 2011. – 121 с.

Учебное пособие посвящено вопросам проектирования встраиваемых систем. В 1 части пособия рассматриваются: основные проблемы, с которыми сталкиваются разработчики встариваемых систем (ВсС), состояние и перспективы высокоуровневого проектирования ВсС, архитектурное проектирование ВсС, модели процесса проектирования ВсС.

Для подготовки бакалавров и магистров по направлению 23.01.00 «Информатика и вычислительная техника»; по группе магистерских программ «Встроенные вычислительные системы» по программам подготовки «Проектирование встроенных вычислительных систем», «Системотехника интегральных вычислителей. Системы на кристалле» и «Сетевые встроенные системы».

Рекомендовано к печати Ученым советом факультета КТиУ, 20 сентября 2011 г., протокол №7

В 2009 году Университет стал победителем многоэтапного конкурса, в результате которого определены 12 ведущих университетов России, которым присвоена категория «Национальный исследовательский университет». Министерством образования и науки Российской Федерации была утверждена Программа развития государственного образовательного учреждения высшего профессионального образования «Санкт-Петербургский государственный университет информационных технологий, механики и оптики» на 2009–2018 годы. В 2011 году Университет получил наименование «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики»

© Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики, 2011

© А.Е. Платунов, Н.П. Постников, 2011.

Оглавление

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 объединенные с ним конструктивно.

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

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.

Создание четкой системы понятий архитектурного уровня позволит разработчику ВсС эффективно работать не на уровне примеров реализаций, а на уровне принципов организации ВС/вычислительного процесса. Важнейшей задачей является создание системы архитектурных абстракций, в которых не противопоставляются аппаратная и программная составляющие ВС, но при этом охватываются все уровни технических решений.

Практическая польза от подобной формализованной системы понятий состоит в возможности развития на ее основе общей теории и методологии проектирования ВсС, создания эффективных методик и САПР архитектурного и сквозного проектирования ВсС.

Следует признать, что в подавляющем большинстве коллективов проектировщиков ВсС сегодня недостаточно высоко оценивается роль и трудоемкость этапов высокоуровневого проектирования, отсутствует адекватный технический язык для общения на этом уровне. Они оперируют лишь конкретными реализациями вычислительных механизмов (то есть «ассемблерными кодами», в которых трудно или невозможно проследить концептуальные моменты и решения). Мери Шоу в статье «Мы можем обучать информатике лучше» пишет: «*Давайте организуем наши курсы вокруг идей, а не вокруг артефактов. Это сделает наши цели более ясными как для студентов, так и для преподавателей. Машиностроительные институты не преподают проектирование бойлера, они преподают термодинамику. В то же время, как минимум два из основных курсов по информатике «Создание компиляторов» и «Операционные системы» являются артефактными динозаврами программирования».* [83]. С момента опубликования статьи ситуация кардинально не изменилась, данный призыв может быть обоснованно распространен на область проектирования ВсС.

Система вычислительных архитектурных абстракций может привнести эффективный язык общения в область проектирования ВсС, повысить «прозрачность» разработок, резко ускорить развитие вычислительных архитектур. Однако для этого необходимы усилия не только со стороны действующих разработчиков вычислительной техники, но и поддержка высшей школы в части модернизации учебного процесса на профильных кафедрах университетов.

1-ая часть учебного пособия посвящена рассмотрению основных проблем, с которыми сталкиваются разработчки встраиваемых вычислительных систем, состоянием и перспективами высокоуровневого проектирования ВсС. Особое внимание 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 уделено аспектной модели процесса проектирования ВсС, аспектной технологии сквозного проектирования ВсС на основе понятия архитектурных агрегатов.

Учебное пособие предназначено для студентов, специализирующихся в области вычислительной техники и программирования. Освоение студентами вопросов высокоуровневого проектирования встроенных систем позволит им эффективно внедрять данную методологию в практическую деятельность коллективов разработчиков, в которых они будет работать после окончания учебы.

1 Проблемы проектирования встраиваемых вычислительных систем

1.1 Встраиваемые вычислительные системы

1.1.1 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 сокращенным 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.

Разнообразие задач автоматизации и способов их решения порождает огромное число вариантов ВсС. С учетом существующих технических ограничений и выделяемых финансово-временных бюджетов выбор варианта реализации может превращаться для разработчика в сложную научно-техническую задачу [32, 85]. Разработчику очень важно иметь четкое представление о предмете проектирования, доступных методах и средствах его создания, уметь подобрать или создать близкие прототипы.

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 [43, 45, 58].

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».

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

- сборка «готовых компонентов», зачастую приобретенных 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 четкого определения для этого класса вычислительных систем. Приведем некоторые примеры определений [11].

«Встроенной системой можно считать любую вычислительную систему, которая не является ПК, портативным компьютером (laptop) или большим универсальным компьютером (mainframe computer)».

«Устройство, которое включает в себя программируемый компьютер, но не является при этом компьютером общего назначения».

«Сложно определить. Практически любая вычислительная система, не являющаяся настольным компьютером».

«Система обработки информации, встроенная в какой-либо продукт».

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 жестких ограничений самого разного плана [23, 13, 22, 24].

Безусловно, часть задач в области создания ВсС удается решать шаблонными способами, особенно если речь идет о развитии или модификации уже готовой системы. Но даже в этом случае требуется использование качественной вычислительной платформы, мощного специализированного инструментария, тщательная верификация и тестирование продукта.

Задачи создания ВсС, которые не укладываются по тем или иным причинам в рамки шаблонных решений, постоянно требуют совершенствования методов и средств проектирования.

Тенденция усложнения ВсС проявляется, прежде всего, в том, что большинство систем реализуются в виде многопроцессорных распределенных ВС или контроллерных сетей. Это дополнительно усложняет задачу проектировщика. Рассмотрим основные свойства современных распределенных ВсС [7, 6].

- 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 распределенная 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).
- Распределенные 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.

Суммируя перечисленные выше особенности ВсС, необходимо отметить следующее. Это 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 [42]:

- *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.

Принципиальное отличие информационных систем (Information Technology) от СРВ (real-time) состоит в трактовке параметра «реакция вход-выход»: «The right answer late is wrong» («Правильный ответ поздно – неправильный»).

Безусловно, важно проследить эволюцию сегодняшнего понятия «встраиваемые системы» на протяжении времени развития вычислительной техники (рис. 1.1):

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.

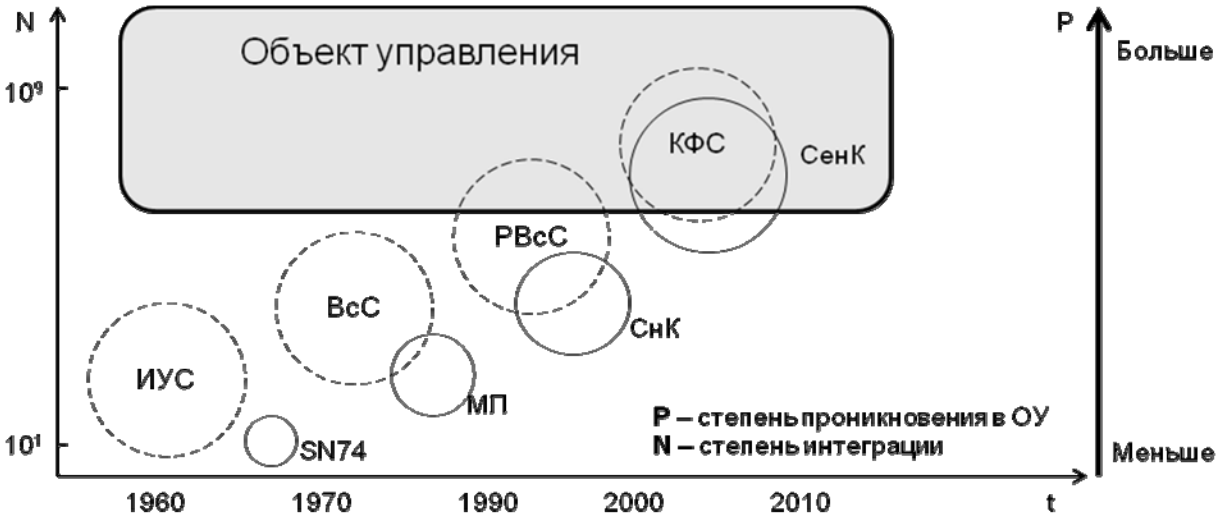
ИУС (1) последовательно «переросли» в ВсС (2) по мере пространственного объединения ВС с объектом контроля / управления благодаря, в первую очередь, технологическим достижениям в электронике.

На сегодняшний день подавляющее большинство ВсС по-факту являются распределенными (3). Представление ВсС как распределенных или включение в эту категорию, так называемых, контроллерных сетей или распределенных информационно-управляющих систем для многих отечественных специалистов является по-прежнему спорным. В зарубежной научно-технической литературе, напротив, достаточно четко декларируется, что ВсС - это не только малогабаритные или моноблочные, одномодульные устройства, но и пространственно и/или архитектурно распределенные системы, которые непосредственно сопряжены с объектами большого масштаба, пространственно распределенными и т.д. Нецелесообразность выделения РИУС в отдельный класс ВС диктуется едиными условиями, принципами и инструментами их проектирования с «традиционными» ВсС. В этом смысле определенные изменения в восприятии, в мышлении у специалистов, связанных с этой

областью, у нас в стране обязательно должны произойти. Иначе наши специалисты будут продолжать разговаривать с зарубежными специалистами в области ВсС на разных языках.

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 определенных или, как минимум, очень тесно взаимодействующих инструментальных средств. На это нацелен тезис (4), в соответствии с которым в мире реально развернуты работы по созданию технологий и инструментов для автоматизированного проектирования гетерогенных технических систем.

Область проектирования ВсС на сегодняшний день тесно смыкается с областью проектирования СБИС. В своем 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. 1.1).



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. 1.1. Эволюция понятия «Встраиваемые вычислительные системы»

Необходимо отметить еще один взгляд на определение ВсС, широко распространенный среди специалистов, которые создают, продают и комплексируют полуфабрикаты, необходимые для создания целевых ВС. В качестве таких компонентов выступают вычислительные модули, периферийное оборудование, программные компоненты, IP-ядра и другое, что составляет в

сумме специализированные вычислительные платформы или само является такой платформой. Именно эти платформы в среде данных специалистов определяются термином «встраиваемые системы» [10]. В контексте задачи проектирования конкретной специализированной ВС такое определение ВсС следует считать не продуктивным.

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.

1.1.2 Проектирование программно-реализованных встраиваемых систем

Часто считается, что ВсС представляют собой программируемые в традиционном стиле ВС, отличающиеся, например, от ПК ограниченными вычислительными ресурсами, использованием языков программирования более низкого уровня и ограниченным сервисом инструментальных средств. Реальная ситуация в этом вопросе намного более сложная и неоднозначная.

Стремительное развитие цифровой элементной базы, стилей и технологий программирования, вычислительных архитектур и парадигм проектирования определили необходимость выделения большей части современных ВС в специальную группу, для которой изначально подчеркивается доминирующее значение программных технологий на всех уровнях организации системы. Такие ВС принято называть «Software Intensive Systems» [50]. Применительно к ВС общего назначения, в которых создание целевых приложений (то есть собственно решение прикладных задач) достаточно четко отделено от создания вычислительной платформы (то есть самой универсальной ВС) данный термин означает в первую очередь то, что акцент в решении задачи смещен в область

программной разработки. В подавляющем большинстве случаев такое программирование выполняется в традиционном стиле, не вторгаясь в область аппаратно-зависимых частей и, тем более, не предполагая использование специализированной программируемой аппаратуры или методов кремниевой компиляции.

Для области ВсС ситуация выглядит иначе. Здесь, наоборот, вычислительная платформа и прикладная надстройка не просто тесно связаны между собой, а во многих случаях между ними вообще сложно провести четкую границу. Например, процессы ввода-вывода в ВсС непосредственно составляют часть прикладной задачи и требуют тщательного временно́го согласования в рамках организации всего вычислительного процесса. Создание ВсС обязательно затрагивает все уровни организации системы. На практике в равной мере используются сценарии полностью заказного и полузаказного проектирования наряду с проектированием на готовой вычислительной платформе. Используется весь арсенал современных технологий и элементной базы.

Следует отметить, что создание или модификацию универсальной вычислительной платформы (то есть самой универсальной ВС) следует рассматривать как частный случай проектирования ВсС.

Естественным следует считать появление для области ВсС уточненного варианта термина, отражающего значение и место программных технологий. Сегодня такие ВсС принято называть *системами с преимущественно программной реализацией* (SW-dominated), которые строятся на основе *вычислительных платформ с большой долей программируемых средств* (highly programmable platforms – «глубоко» программируемых платформ [45]). Можно использовать термин *«программно-центрированные вычислительные системы»*.

Смысл приведенной выше «двухзвенной» характеристики программируемости современных ВсС состоит в следующем.

Первая часть характеристики современных ВсС, а именно отнесение их к системам с преимущественно программной реализацией, проявляется в двух вариантах. Первый состоит в широком использовании на уровне аппаратуры программируемых процессоров – последовательных и параллельно-последовательных программных интерпретаторов. Второй проявляется в использовании языковых методов представления решаемой задачи (представление вычислительного процесса в виде программ того или иного стиля) с последующей реализацией средствами всего арсенала доступных технических решений (от последовательной программной интерпретации аппаратными процессорами до исполнения специально синтезированной аппаратурой). Естественным является использование иерархии аппаратно и программно реализованных интерпретаторов (виртуальных машин) в сочетании со специализированной аппаратурой.

Перечислим основные проблемы, порождаемые этой частью «программируемости» ВсС:

- противоречие между ростом желаемого разработчиком уровня абстрактности в представлении все более сложных задач ВсС и необходимой «прозрачностью» (контролируемостью) реализации;
- гетерогенность вычислителей (процессоров);
- сложность функциональной и пространственной декомпозиции задачи;
- специфический «портрет специалиста», необходимого для работы в области программирования ВсС;
- неадекватность в части надежности большинства современных технологий программирования требованиям проектирования ВсС;
- отставание темпов роста эффективности проектирования от потребностей отрасли;
- низкий коэффициент повторного использования результатов проектирования.

Вторая часть характеристики проистекает из особенностей современной элементной базы. Наряду с традиционными программируемыми процессорными ядрами (микропроцессорами) разработчик имеет дело с большим числом программируемых спецпроцессоров, с конфигурируемыми функциональными расширителями и контроллерами, со схемами программируемой логики. Современная цифровая и цифро-аналоговая элементная база конфигурируется в очень широком диапазоне: настройка временных параметров и режимов обмена элементов памяти, приемо-передатчиков интерфейсов, контроллеров интерфейсов, АЦП, ЦАП, усилителей, источников питания, супервизоров и схем контроля. Сложность конфигурирования и программирования таких элементов может быть значительной, как, например, для контроллеров и процессоров ввода-вывода в составе микропроцессорных комплектов, коммуникационных устройств (модемы беспроводной связи различных стандартов), ПЛИС.

В сумме это расширяет возможности разработчика, одновременно резко увеличивая риск ошибки и трудоемкость низкоуровневого проектирования. Попытки вообще уйти от низкоуровневого проектирования в этой части пока успехом не увенчались, так как навязывание разработчику ограниченного числа шаблонов конфигурирования резко ухудшает качество проектирования, а диапазон уровней организации системы, который вынужден представлять разработчик для качественного управления аппаратурой, чрезвычайно широк.

Таким образом, «двойственная программируемость» современных ВсС порождает целый ряд методологических, методических и инструментальных проблем в проектировании, требующих сегодня решения. Результатом таких решений должны стать качественно новые САПР, охватывающие системные

уровни проектирования ВсС и предоставляющие разработчику возможность целенаправленного поиска вариантов во всем многообразии доступных архитектур.

1.1.3 Встроенное программное обеспечение

Важным шагом в области проектирования ВсС, непосредственно вытекающим из программно-центрированной модели ВсС, явилось определение термина встроенное программное обеспечение (embedded software, ВПО) [84, 63, 71], который направлен на то, чтобы обозначить границы и особенности отрасли программирования ВсС. Характеристика современных ВсС как программно-центрированных систем определяет первоочередное значение ВПО как составляющей проектирования ВсС.

Четкое осознание серьезных отличий и проблем в области создания ПО ВсС по сравнению с ПО иных категорий произошло в 90-е годы 20-го века. До этого чаще всего считалось, что ВПО – это программирование небольших компьютеров с ограниченными вычислительными ресурсами. Некоторые важнейшие проблемы ВПО, такие как вопросы эффективного взаимодействия аппаратчиков и программистов, средства адекватного машинного представления задач реального времени, вопросы надежности ПО решались ведущими научными школами уже в этот период [63, 52, 86, 60].

Качественный анализ состояния, проблем и перспективных направлений развития ВПО был сделан в [71, 64]. Рассмотрим основные моменты.

Прежде всего, отмечается, что ВПО – это один из вариантов реализации функциональности ВсС, которая с тем же успехом может быть реализована как аппаратный компонент, и что в области ВПО нельзя абстрагироваться от жестких характеристик и ограничений ПО, как это обычно делается в традиционных областях программирования. Попытки прямого переноса методов и технологий, традиционных для индустрии ПО, в область ВПО привели сегодня к кризису в этой области. Маловероятно, что из этого кризиса можно выйти, используя традиционные способы проектирования. Авторы [71] обоснованно предлагают сфокусироваться на источнике проблем. Они отмечают, что *«для ВПО нужно радикально изменить способ проектирования, обеспечив следующие действия: 1) связывание ВПО с функциональностью системы на более высоких уровнях абстракции; 2) связывание ВПО с программируемыми платформами, которые поддерживают его. Все это обеспечит необходимые средства верификации того, что наложенные на ВсС ограничения, удовлетворяются»*.

Для реализации этого видения, с одной стороны, должны быть разработаны такие формальные методики на абстрактном уровне, чтобы верификацию можно было начать раньше, причем, с использованием корректного набора инструментов и методов. С другой стороны, проектировщику нужно понимать архитектуру ПО и аппаратуры ВсС одинаково, в одном ключе.

Перечисленное выше в значительной мере объясняет широкое использование микроконтроллеров с примитивной архитектурой и массовое низкоуровневое программирование при создании ВсС.

Существует еще ряд важных проблем в области проектирования ВсС, косвенно связанных с ВПО. Одна из них – проблема «программистского перекоса» в проектировании ВсС, суть которой поясним ниже.

Технология создания ВсС всегда предполагает разработку (в той или иной мере) и аппаратной, и программной составляющих. На этапе архитектурного проектирования должна анализироваться вся система в целом. В сегодняшних условиях выполнять такую работу должен «интегральный» специалист, владеющий абстракциями, характерными для аппаратуры и ПО, представляющий возможности реализации аппаратного и программного компонентов. С развитием и внедрением технологий и инструментальных средств сквозного проектирования архитектор ВсС может быть освобожден от необходимости держать в голове информацию уровня физической реализации системы.

Существующие в значительном количестве коллективы разработчиков ПО систем реального времени (СРВ) в рамках технологии создания ВсС должны работать в качестве одного из соисполнителей в команде проектировщиков системы. Однако в силу различных причин очень часто такие коллективы берут на себя роль всей команды создания ВсС. Взгляд на проектирование СРВ только в виде традиционного программирования (пускай и со всеми этапами, начиная от архитектуры программной надстройки), безусловно, неэффективен.

Рассмотрим более подробно проблемы создания встроенного программного обеспечения.

В [71] отмечаются следующие основные проблемы, характерные для области проектирования ВПО:

- необходимость увеличения степени повторного использования;
- Co-Design ПО и аппаратуры;
- создание средств моделирования нефункциональных свойств;
- усиление роли в проектировании уровня архитектуры систем и ПО;
- проверка достоверности и верификация;
- адаптация ПО и аппаратуры посредством использования реконфигурируемых архитектур и компонентов «plug and play»;
- разработка общей технологии и стандартов процесса проектирования в части семантики.

На первый план выдвинуты следующие перспективные тенденции развития в области ВПО:

- формальная верификация;

- развитие технологии HW/SW Co-Design;
- повторное использование и интеграция компонентов, (это было выдвинуто на первый план как главная потребность);
- Co-Design вычислительной платформы и функциональности;
- объединенные потоки HW/SW;
- Co-Design предметной области и архитектуры BcC.

В частности, отмечается, что «необходимы стандарты повторного использования компонентов, чтобы реализовать проектирование компонентов интеллектуальной собственности, что будет востребовано в области проектирования ВПО», скромное и хитрое признание того факта, что сегодня при проектировании ПО, использование IP и повторное использование являются более обычным, чем в области ВПО. Несколько ключевых проблем эффективного повторного использования компонентов в ВПО – это способность к изменению конфигурации (чтобы позволить оптимизированное повторное использование), декомпозиция и модульность, и тщательно проработанные интерфейсы.

Кроме того, в [64] отмечается следующее.

Для выхода из кризисной ситуации в создании ВПО должен использоваться целостный подход, охватывающий методологию проектирования, инструментальные средства, IP, аппаратные и программные платформы. Только используя глобальное, высокоуровневое представление проблемы, могут быть предложены решения, которые реально повлияют на проектирование BcC.

Важнейшая проблема, которую необходимо решить – связь между функциональностью системы и программируемыми платформами. Для этого необходимо начинать проектирование с абстракции высокого уровня, отражающей адекватно функциональность системы. Такие абстракции должны быть полностью независимо реализованными от элементной базы и способов традиционного программирования (кодирования) и основаны на солидных теоретических фундаментах, которые позволят выполнять формальный анализ. Необходимо научиться выбирать вычислительную платформу, которая может поддержать функциональность, отвечающую физическим ограничениям, наложенным на окончательную реализацию BcC.

В учебном пособии представляется общая цель исследований в области ВПО:

Иметь оптимизированный, полуавтоматический, прозрачный, поддающийся проверке и математически корректный проектный поток от спецификации продукта до реализации для BcC с доминирующим программным компонентом, создаваемых на «глубоко» программируемых платформах.

«Глубоко» программируемые платформы могут содержать в своем составе аналоговые компоненты, реконфигурируемую логику, специализированные

ASIC, программируемые процессоры всех типов, конфигурируемые процессоры и блоки, специализированные блоки аппаратуры и ПО, взятые из библиотек.

Очень важным является то, что большая часть перечисленных выше проблем и задач, которые являются типичными для ВПО, обычно не воспринимаются в сообществе специалистов по программному обеспечению.

Развитием и конкретизацией представленной выше точки зрения на проблемы создания ВПО является [60]. Автор работы, Э.А. Ли, предметно критикует сегодняшнюю ситуацию в области ВПО, однако в отличие от авторов [71] он понимает ВПО в большей степени как традиционное ПО различного рода последовательных интерпретаторов в составе ВсС.

ВПО предназначено не для трансформации данных (как ПО «обычных» ВС), но для взаимодействия ВС с реальным миром. Рассматривая организацию ВПО на уровне архитектуры, Э.А. Ли выделяет в качестве основных абстракций программирования своевременность (timeliness), параллельность, живучесть (оригинальное liveness) и неоднородность (heterogeneity).

Анализируя основные проблемы в области ВПО, Э.А.Ли [63, 62, 65, 60] указывает на следующее: «Что следовало бы сделать для получения параллельного и сетевого ВПО, которое было бы абсолютно надежным во времени? К сожалению, все необходимо было бы изменить. *Базовые абстракции вычисления необходимо модифицировать для охвата времени. Архитектуру компьютера необходимо изменить для получения точного временно́го поведения. Сетевые методы необходимо изменить для обеспечения параллельности во времени. Необходимо изменить языки программирования для включения времени и параллелизма в их базовую семантику. ОС необходимо изменить для того, чтобы меньше полагаться на свойства для определения (косвенно) временны́х требований. Необходимо изменить методы проектирования ПО для определения и анализа временно́й динамики ПО. А традиционная граница между ОС и языком программирования должна быть переосмыслена.* То, что необходимо, является почти переизобретением компьютерной науки».

К счастью есть достаточно много того, что можно использовать. Подводя итог, Э.А. Ли призывает компьютерное сообщество собрать перечисленные методы вместе и направить усилия на построение «Встроенной Компьютерной Науки 21 века».

Важным является тезис о потенциальной неэффективности современных универсальных процессорных платформ со статистическими механизмами повышения производительности с позиций требований встроенного ПО. Применение подобных вычислителей, а также ряда технологий программирования (например, многих реализаций языка Java) вступает в прямое противоречие с основными критериями проектирования ВсС, и в первую очередь, с надежностью. Требуемая надежность, а во многих случаях и

функциональность, в этих случаях могут быть достигнуты только путем «виртуозного» программирования при условии значительной избыточности вычислительных ресурсов. Фактически это означает, что программист будет обходить на низком уровне негативные проявления механизмов повышения производительности с вероятностной составляющей, которые присутствуют в современных универсальных процессорах и реализациях языков высокого уровня. Все это ведет к резкому росту стоимости, как разработки, так и самих ВсС.

Как было показано в предыдущем разделе, современные ВсС предполагают программирование на различных (на всех) уровнях и в различном проявлении. Программирование по-прежнему рассматривается как особый род деятельности, предполагающий определенные методологии, методики и технологии. Следовательно, важнейшими вопросами в контексте формирования единого проектного цикла ВсС необходимо считать четкое определение зоны ответственности программиста в проекте, стили и средства взаимодействия программиста со специалистами других профилей, возможность и целесообразность изменения традиционных зон ответственности и технологий программирования в области создания ВсС.

Обсудим некоторые варианты определения ВПО и вытекающие из них трактовки зон ответственности и охватываемые технологии.

Если ВПО определять как совокупность конечного кода, который определяет функционирование всех категорий программно-настраиваемых и программно-управляемых физических элементов в составе ВсС, то по стилям (методикам) создания целесообразно выделять область конфигурационного обеспечения (кода) и область программного обеспечения (кода). Отдельно в этом случае рассматривается вопрос о процедурах и средствах подготовки, проверки, доставки элементов ВПО для ВсС. Эти технологии и инструментальные средства оказываются в значительной степени изолированными от самого процесса создания ВсС.

Такой взгляд на ВПО обладает рядом достоинств, из которых основным можно считать простое и понятное (упрощенное) неискушенному специалисту в области ВсС представление о месте, роли, способах создания ВПО. Приемлемое качество проектирования ВПО в такой идеологии достигается только в рамках небольших и несложных проектов ВсС, где приемлемо использование шаблонных решений.

Недостаток такой упрощенной модели состоит в искусственном отделении конечного продукта (кода ВПО) от технологий и средств его создания. Это, по сути, исключает возможность эффективного поиска решений на системном уровне, препятствует использованию технологий на основе многоуровневой организации ПО (например, технологии виртуальных машин), ограничивает использование технологий проектирования с высоким уровнем абстракций, вступает в противоречие с перспективными способами проектирования и применения многих категорий элементной базы ВсС.

Вторым («обобщающим») вариантом трактовки ВПО может выступать определение ВПО как всей совокупности алгоритмического обеспечения ВсС. Этот подход хорошо укладывается в модель единого сквозного цикла проектирования ВсС, в которой важнейшее место занимают этапы высокоуровневого проектирования, широко используются абстракции организации вычислительного процесса и унифицированный взгляд на создание HW и SW компонентов системы.

Эта позиция в вопросе создания ВПО является наиболее продуктивной и перспективной, и именно этот подход будет представлен далее в учебном пособии.

Однако, забегая вперед, отметим ряд важных моментов в понимании ВПО, следующих из второй, «обобщающей» трактовки. В состав ВПО может включаться:

- только то ПО, которое «работает» в режиме run-time внутри ВсС на последовательных интерпретаторах;
- то, что на этапе проектирования технологически представлялось, как программирование и было в дальнейшем реализовано в ВсС в любом виде (в виде аппаратных блоков, конфигурационного обеспечения, традиционного программного кода);
- вся совокупность деятельности по созданию алгоритмического наполнения ВсС, которая присутствует на этапах design-time и run-time.

Перечисленные варианты идеологической позиции проектировщика в вопросах ВПО предполагают серьезные различия в организуемой технологической цепи, удельном весе этапов проектирования, влияют на требуемый состав команды разработчиков, доступные модели процесса проектирования, следовательно, на масштаб решаемых задач и эффективность решения.

1.1.4 Классификация встраиваемых систем

Отсутствие сегодня четкого определения класса ВсС проявляется в проблемах классификации, как самих встраиваемых систем, так и архитектурных парадигм в их проектировании, вариантов организации в них вычислительного процесса, технологий их проектирования, программирования и отладки, ряда других их важных аспектов. Эффективное накопление и передача опыта в проектировании ВсС, выстраивание единого языка общения специалистов в данной области невозможно без «работающих» классификаций. В настоящем разделе обобщаются классификационные подходы в области проектирования ВсС и делается ряд новых предложений в данном вопросе.

Из известных общих классификаций ВС для сегодняшнего состояния вычислительной техники видится наиболее удачной классификация Дэвида Паттерсона [74], в соответствии с которой выделяются три категории вычислительных систем (табл. 1.1, рис. 1.2)

- настольные компьютеры (ПК – рабочая станция / интеллектуальный терминал);
- серверы (ВС коллективного пользования);
- встраиваемые системы (все прочие ВС).

Таблица 1.1. Характерные особенности вычислительных систем различных классов

	Персональный компьютер	**Сервер**	**Встраиваемая система**
Стоимость системы	$500-$5000	$5000-$5000000	$10-$100000
Стоимость микропроцессорного модуля	$50-$5000	$200-$10000	$0.01-$100
Наиболее весомые характеристики	Цена – производительность, скорость работы с графикой	Пропускная способность, доступность, расширяемость	Цена, энергопотребление, производительность для специфических приложений

Классификация разделяет ВС по характеру их использования. При всей кажущейся простоте данная классификация выделяет важнейшие свойства класса ВсС, и прежде всего, самый широкий из всех ВС диапазон изменения сложности. Это объясняет сложившуюся непростую ситуацию в вопросах классификации ВсС.

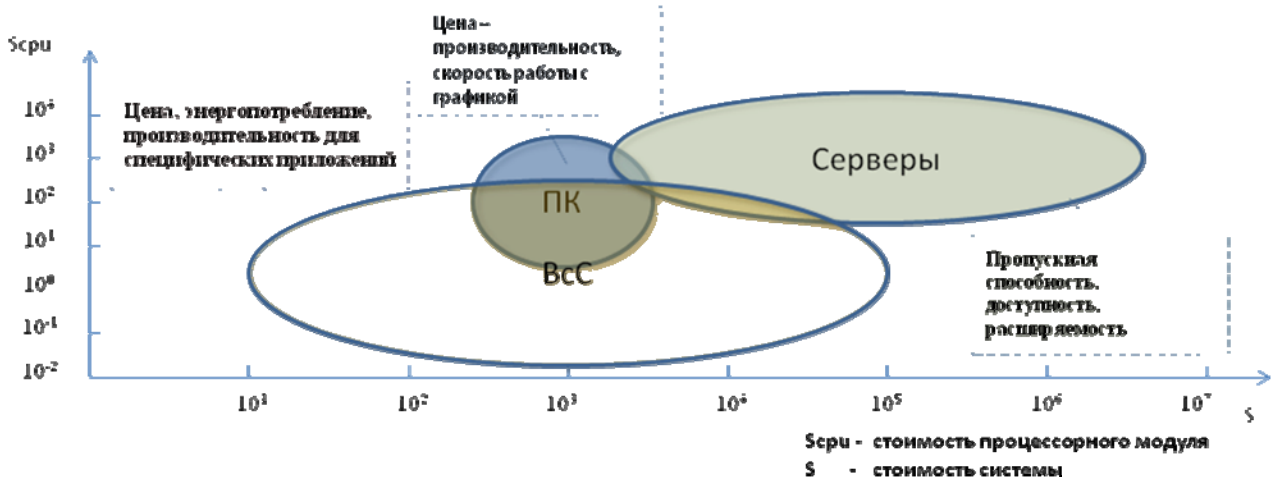


Рис. 1.2 Классификация современных ВС по Д.Паттерсону

В предыдущем разделе была продемонстрирована эволюция понятия ВсС, шаги которой можно в некотором роде также считать классификацией, которая отражает степень интеграции ВС с объектом управления:

- информационно-управляющие системы (ИУС);
- распределенные информационно-управляющие системы (РИУС);
- встраиваемые системы (ES);
- сетевые встраиваемые системы (NES);
- киберфизические системы (CPS).

Традиционно классификацию ВС начинают с функционального признака. Разделить ВсС по назначению можно только очень условно. Попытка такого деления представлена ниже:

1. Системы автоматического управления (САУ);
2. Измерительные системы и системы сбора информации с датчиков (приборные, характерны ярко выраженные измерительные функции наряду с управлением);
3. Информационные системы “запрос-ответ” реального времени (платежные системы, резервирование билетов и т.д.) Они занимают пограничное положение с информационными системами общего назначения;
4. Цифровые системы передачи данных (телекоммуникационные системы);
5. Сложные иерархические системы реального времени (обеспечивают контроль и управление сложными, в том числе, пространственно-распределенными объектами);
6. Системы управления подвижными объектами;
7. Подсистемы ВС общего назначения;
8. Мультимедийные системы.

Функциональная классификация позволяет косвенным образом формировать характеристику каждой группы ВсС. Однако разброс свойств проектных шаблонов оказывается в этом случае очень широк, что снижает значимость такой классификации для разработчика.

Кроме классификации по назначению возможно разделение ВсС по таким признакам, как:

- сложность системы (большие, средние, малые);
- топология системы (сосредоточенные, распределенные);
- тип ВС, являющейся основой ВсС (одно- и многопроцессорные, гомогенные и гетерогенные, сильно и слабосвязанные, использующие ОС и не использующие);
- особенность реализации реального масштаба времени (мягкое и жесткое реальное время);

- конструкция (моноблочные, модульные, встроенные, расширяемые и нерасширяемые, обслуживаемые и необслуживаемые);
- реализуемая для ВсС надежность, безопасность, информационная защищенность и т.д.;
- другие функциональные и нефункциональные характеристики.

В рамках фаз жизненного цикла ВсС на этапе исполнения (RunTime) важно выделять:

- инструментальные возможности системы;
- тип платформы с точки зрения возможностей (частично скрытые возможности; платформа, возможности которой раскрываются по мере выхода новых экземпляров встраиваемой системы).

На этапе разработки (Design Time) у ВсС появляется множество критериев для классификации на архитектурном уровне. Можно выделять следующие основные структурные характеристики:

- количество уровней иерархии;
- количество вычислителей в системе;
- степень разнородности вычислителей в системе;
- количество процессоров в вычислителе;
- тип связи между процессорами в вычислителях;
- степень однотипности процессоров в вычислителе;
- степень равноправия процессоров.

По принципу комплексирования, ВсС можно разделить на четыре категории: полностью готовые, блочные, заказные, полузаказные или смешанные.

Полностью готовая система – это система, выполняющая целевую функцию, либо требующая незначительной по трудоемкости доводки для выполнения своей целевой функции. Как правило, ВсС на основе готовых систем обладают максимальной избыточностью и соответственно стоимостью.

Следующий по сложности вариант – система из готовых блоков. Чтобы такая система выполняла свою целевую функцию, необходимо комплексирование или интеграция покупных программных и аппаратных блоков. Избыточность системы существенно ниже, чем в первом случае. Стоимость разработки также невысока.

В третьем и четвертом вариантах присутствует заказной компонент, требующий зачастую проведения научно-исследовательских работ, подготовки производства, разработки новых технологий.

Важными классификационными критериями для разработчика являются:

- соотношение обработка данных – управление;
- степень программируемости системы;
- способ проектирования;
- способ реализации платформы;
- сила воздействия на проект тех или иных аспектов проектирования.

Далее в пособии указанные проектные критерии будут обсуждаться долее подробно.

1.2 Состояние и перспективы высокоуровневого проектирования ВсС

1.2.1 Проектирование заказных микропроцессорных систем

Сложившаяся практика проектирования вычислительных систем вообще и ВсС, в частности, как было отмечено выше, состоит в выборе одной из канонических вычислительных платформ (ВП), на которой за счет программной надстройки решается прикладная задача. Задача делится на две части: выбирается база (платформа), база достраивается вверх (за счет программирования в широком смысле) до получения требуемой ВС. Для такого способа проектирования существуют технологические приемы и инструментальные средства. Примерами являются языки программирования, на которых описывается конечная задача, исполнительные устройства (готовые вычислительные машины) и трансляторы.

Применяется и второй вариант: выбирается ВП и наряду с достройкой вверх выполняется модификация вниз. В этом случае базовая платформа выступает и в роли прототипа. Такой способ используется реже из-за высокой трудоемкости.

Первая проблема, с которой сталкиваются разработчики ВсС, состоит в следующем:

- существующие языки программирования предполагают описание задачи для идеализированной виртуальной (языковой) машины;
- транслятор отображает эту языковую машину на реальную машину, внося определенные ограничения и не учитывая многих важных технических особенностей исполнительской машины (например, особенностей системы ввода-вывода, защитных механизмов и др.);
- для того чтобы учесть эти ограничения и особенности, программисту необходимо помимо знаний о языковой машине иметь знания о трансляторе и об исполнительской машине.

На сегодня отсутствует единая система описания этих трех составляющих, они описываются в различных предметных пространствах и языковых стилях.

Вторая проблема – большое число задач, особенно в области систем управления физическими объектами, которые плохо укладываются в схему реализации на основе канонических ВП с языковой программно – реализуемой надстройкой. При таком подходе решения оказываются экономически неоптимальными, либо задача вообще не решается в рамках современных технических средств. Необходимо проектировать специализированные ВП, например, с высокой степенью параллелизма и специализацией операционных блоков. Попытка проектировать целевую ВсС на такой архитектурной основе по описанной выше массовой традиционной технологии катастрофическим образом обостряет проблему нестыковки специфических требований языковой

машины, трансляторов и ВП. Если для определенного класса задач традиционная схема проектирования приемлема, то для специализированных систем эффективность проектирования может сводиться к нулю (огромные проблемы с параллелизмом, с защитными механизмами, с ограничением реального времени, с тестированием и отладкой и т.д.).

Третья проблема связана с постоянно растущим объемом необходимого проектирования ВсС. Разобщенность описаний, отмеченная выше, препятствует повторному использованию разработанных компонентов (аппаратных блоков, программ, реализаций алгоритмов и т.д.). Магистральным направлением в повышении эффективности проектирования, как отмечают ведущие специалисты, является закрепление результатов разработок в виде абстрактных технических решений, инвариантных к способу реализации [71]. Переходу в проектировании ВсС на такой уровень препятствует отсутствие соответствующей методологии, языковой базы и инструментальных средств.

Сегодня на рынке востребованы все варианты проектирования ВсС по шкале «глубины погружения» в аппаратно-программную организацию системы:

- Прикладное программное обеспечение (ПО);
- Прикладное и системное ПО;
- Устройства сопряжения с объектом (УСО), прикладное, системное ПО;
- Аппаратура «центра», коммуникации, УСО, прикладное и системное ПО.

Как отмечается в [41], проектирование ведется в рамках трех типовых сценариев (рис. 1.3):

- целевая ВсС (a);
- платформа ВсС (b);
- модификация существующей ВсС (c).

Анализ традиционного процесса проектирования ВсС позволяет выделить следующие недостатки:

- неформальное разбиение на аппаратную и программную части на начальном шаге;
- последовательное проектирование аппаратуры и программы;
- раздельное моделирование аппаратуры и программы;
- ручная интеграция аппаратной и программной частей проекта;
- компенсация выявившихся в процессе отладки ошибок за счет изменения программы.

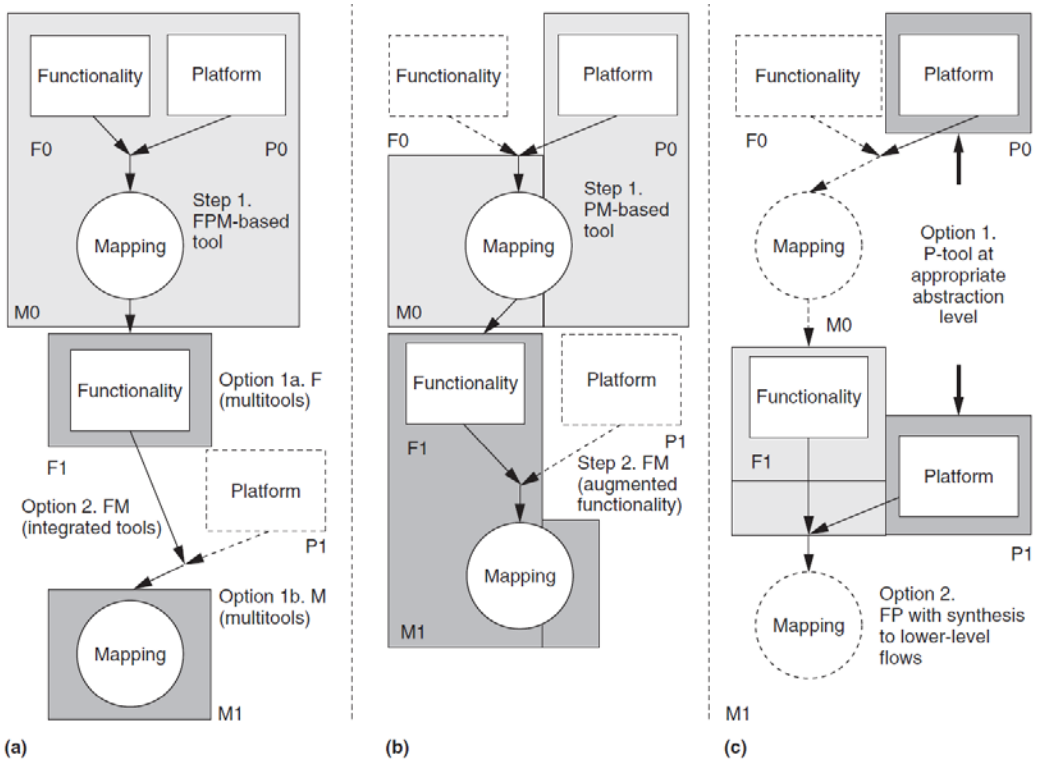


Рис. 1.3. Типовые сценарии проектирования ВсС [41]

Основные тенденции в развитии процессов и средств проектирования ВсС непосредственно связаны с увеличением удельного веса и усилением значения этапов архитектурного, функционального, логического проектирования:

- Повышение уровня абстракции проектирования;
- Широкое применение моделирования, методов формального анализа и верификации моделей;
- Выделение уровня абстрактного представления вычислительного процесса;
- Выделение технологий создания встраиваемого ПО (ВПО).

Перечисленные этапы проектирования сегодня составляют до 70% общего объема разработки ВсС (рис. 1.44) и относятся к области высокоуровневого проектирования (HLD), центральным понятием которого следует считать *абстрагирование – отвлечение в процессе анализа и синтеза ВсС от несущественных сторон, свойств, связей элемента или процесса с целью выделения их существенных, закономерных признаков*.

На рис. 1.5 представлен упрощенный маршрут проектирования, характерный для ВсС. Видно, что от того, насколько проектировщики могут расширять зону работы с абстрактными моделями вычислительного процесса и ВС в рамках всего маршрута проектирования кардинально зависит качество проектирования встраиваемых систем.

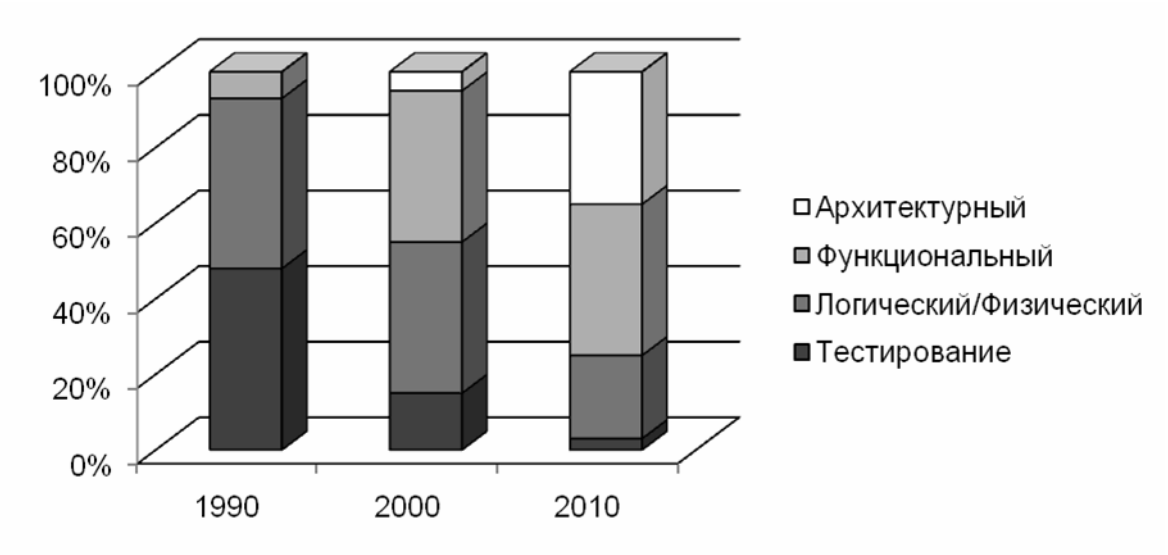


Рис. 1.4. Относительная трудоемкость этапов проектирования ВсС [2]

Рис. 1.5. Маршрут проектирования ВсС

На сегодняшний день задачи этапа высокоуровневого проектирования ВсС группируются в шесть крупных блоков, покрывая фазу проектирования, и не затрагивая фазу реализации:

1. Концепция решения целевой задачи, исходные спецификации.
2. Организация вычислительного процесса (модели вычислений – МоС).
3. Генерация архитектуры и микроархитектуры.
4. Оценка и выбор архитектурных решений.
5. Верификация архитектурных решений.

Выходные спецификации для этапа реализации. Эти этапы принадлежат системному уровню традиционной Y-диаграммы проектирования ВС, разделяя его следующим образом (рис. 1.6а):

- *Уровень архитектуры (architecture) (или макроархитектуры (macroarchitecture), или исполнения (performance))* – формирование архитектуры системы, безотносительно способа реализации: анализ архитектуры на предмет соответствия функциональным и нефункциональным требованиям/ограничениям.
- Уровень микроархитектуры или функциональный(functional) или (особенно в разработке цифровых СБИС и СнК) – Electronic system Level (ESL) – осуществляется выбор способа реализации компонентов архитектурной модели, разрабатываются алгоритмы, интерфейсы, подготавливаются спецификации и среда верификации для этапа реализации системы.

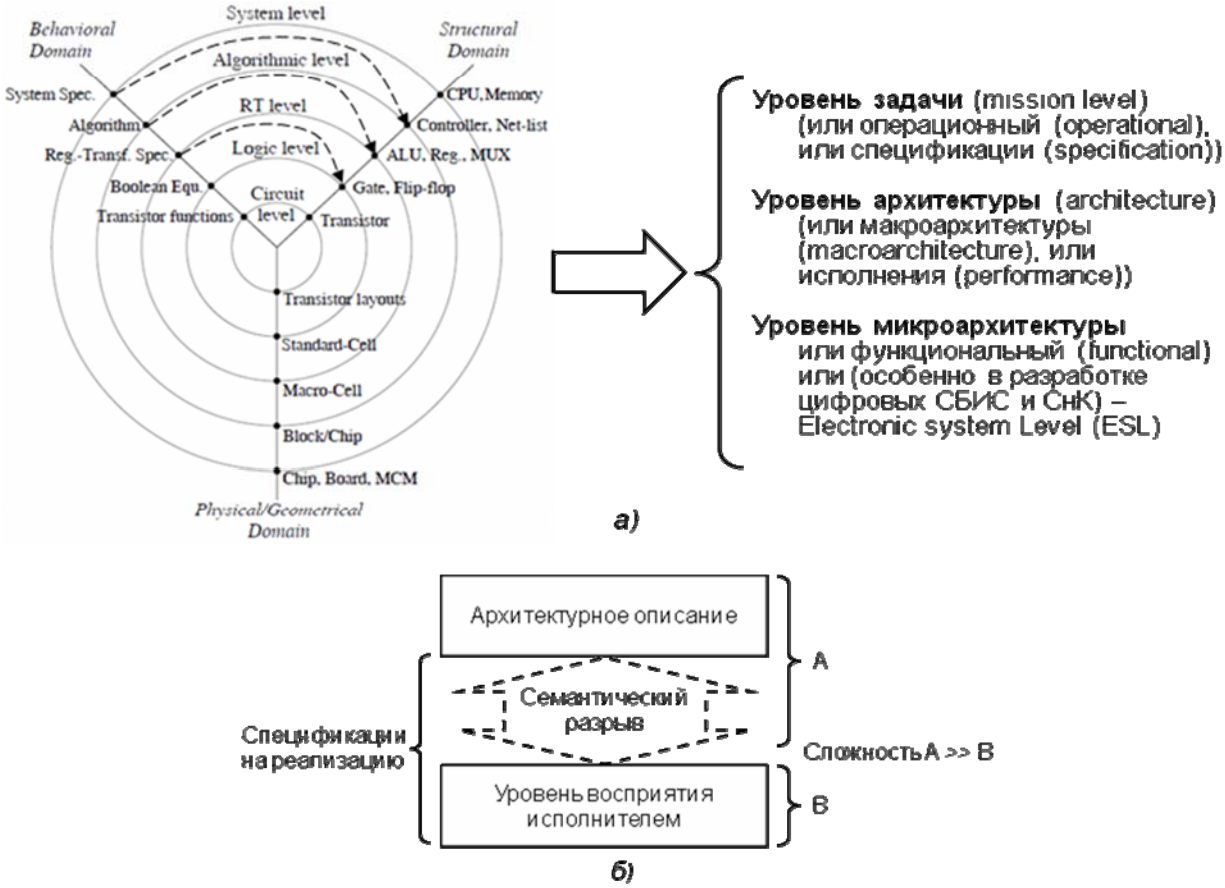


Рис. 1.6. Уровни проектирования (а), семантический разрыв (б)

Важнейшей проблемой HLD остается преодоление семантического разрыва между HLD-описаниями и специафикациями, как на реализацию, так и на создание частных архитектур (микроархитектур) (рис. 1.6, б).

Исследования, направленные на создание эффективных технологий проектирования ВсС, ведутся по многим направлениям [33, 1, 4]. Преодоление

указанных в предыдущих разделах проблем возможно в результате решения ряда задач. Основными из них следует считать [14, 17]:

- реальную интеграцию ветвей проектирования аппаратной и программной составляющих проектов;
- продвижение в вопросе формализации этапа архитектурного проектирования;
- изменение технологий верификации и тестирования аппаратных и программных продуктов;
- создание средств эффективного проектирования программного продукта параллельно с созданием аппаратуры;
- создание и внедрение технологий повторного использования для продуктов проектирования всех уровней иерархии ВсС;
- развитие алгоритмических моделей систем реального времени;
- совершенствование механизмов надежности ВсС.

Важнейшее место в высокоуровневом проектировании ВсС занимают инструментальные средства (САПР), включая методики их использования. В параграфе 1.2.1.1 представлен краткий анализ ситуации с инструментарием в области HLD ВсС. Картина исследований в настоящее время носит во многом фрагментарный характер, и различные группы исследователей пытаются решать частные задачи и предлагать собственные подходы к решению описанной проблемы. Подходы обладают разной степенью проработанности, формализации и распространенности. Очевидно, что сегодня необходимо активное развитие HLD – инструментов.

1.2.1.1 Состояние и проблемы инструментария HLD ВсС

В работе [41] в качестве основных рассмотрены две категории инструментальных средств (табл. 1.2):

- Индустриальные продукты (от ведущих производителей САПР системного уровня, покрывают частично средний и нижний уровни проектирования и программирования; используют «промышленные» языки и ограниченное число МоС).
- Академические продукты (Университеты Беркли, Канзас, Гренобль и др.; используют различные МоС, специальные языки).

Таблица 1.2. Инструментальные средства HLD

(F – Functionality, P – Platform, M – Mapping)

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

Ниже зафиксированы общие проблемы доступных методологий, инструментов и сред проектирования (frameworks), побуждающие к развитию исследовательских работ в области системного проектирования для BcC:

- Отдельные (особенно нефункциональные) аспекты проектирования рассматриваются изолировано, в рамках специализированных общесистемных моделей: энергопотребления, надежности, информационной защиты и т.п. Следует разработать такую структуру и форму описания архитектурных компонентов, которая будет в явном виде предлагать аспектные оценки (веса), что отражено в понятии архитектурного агрегата, которое поясняется далее в пособии.
- Сохраняется явный приоритет функционального аспекта на архитектурном уровне. Нефункциональные аспекты либо играют роль вспомогательных критериев проекта, либо многие из них не учтены вообще. Но в общем случае для BcC функциональность не всегда оказывается наиболее приоритетным требованием. Данные вопросы должны быть учтены при разработке инструмента архитектурного описания системы.
- Множество вариантов реализации микроархитектур ограничивается рамками HW/SW фазы runtime. Поиск проектных решений при варьировании соотношения Design/Run Time, рассмотрении различных уровней виртуализации архитектуры, интеграции инструментальной составляющей в большинстве случаев в рамках системного проектирования не проводится.

Картина исследований в настоящее время носит во многом фрагментарный характер, и различные группы исследователей пытаются решать частные задачи и предлагать собственные подходы к решению описанной проблемы. Подходы

обладают разной степенью проработанности, формализации и распространенности.

Далее кратко охарактеризованы основные крупные тенденции и важные направления исследований.

1.2.2 Методики проектирования встраиваемых систем

Методики высокоуровневого проектирования ВсС активно развиваются на протяжении последних 15-ти лет. Их основные направления:

- Объектно-ориентированное проектирование (вытекает из ООП).
- Параллельное аппаратно-программное проектирование (Hardware/Software CoDesign):
- Компонентное и платформно-ориентированное проектирование.
- Акторно-ориентированное проектирование.
- Многоязыковое проектирование.
- Аспектное проектирование.

Специалистами предлагаются и анализируются различные математические модели, формальные описания и алгоритмы [25, 48, 80, 82]. Рассматриваются автоматные подходы (сети Петри [68, 75], асинхронные и синхронные ко-автоматы [27, 37]), многоязыковые описания [44, 52], алгоритмы совместного аппаратно-программного моделирования (co-simulation) и верификации (co-verification) [30, 35, 43, 52, 57], технологии повторного использования результатов проектирования [81], аспектное программирование [55, 77].

1.2.2.1 Совместное проектирование аппаратуры и программного обеспечения

Широкое распространение традиционной микропроцессорной техники и стремительное развитие интегральной программируемой логики, привели к формированию новой философии совместного проектирования аппаратуры и программного обеспечения, именуемой в литературе "Hardware–Software CoDesign" [47]. Комплексный характер проектирования ВсС отражается в стремлении разработчиков интегрировать языковую базу проектирования, создавать иерархию симуляционных моделей системы для предварительной оценки (профилирования) вычислительной сложности и последующей отладки (процессы верификации и тестирования).

Hardware-Software CoDesign рассматривается сегодня как один из перспективных подходов в проектировании ВсС. Эта технология является сравнительно молодой, она появилась в первой половине девяностых годов. В настоящее время существует несколько некоммерческих САПР (Polis [30], Ptolemy [34, 35], Chinook и др.) разрабатываемых в ряде университетов мира.

Тезис распараллеливания и распределенности вычислений применительно к современным ВсС рассматривается большинством разработчиков как основа

их архитектурной организации независимо от потенциально возрастающей при этом сложности проектирования [45]. Это объясняется существенным улучшением характеристик конечного продукта по сравнению с альтернативными вариантами или на фоне принципиального отсутствия других технических решений. Увеличение сложности проектирования таких систем является объективным процессом и определяется резким увеличением размерности задачи в результате изначальной асинхронности функционирования элементов распределенных и параллельных ВсС.

Как было показано выше, используемые сегодня на практике методы и средства проектирования позволяют формализовать и автоматизировать нижние уровни проектирования – схемотехнику, конструирование, программирование (кодирование). В определенной мере можно считать формализованными верхние уровни – создание спецификаций системы и архитектурное проектирование (в меньшей степени). В средней части цепи по-прежнему прослеживается разрыв. Не формализован переход от высокоуровневого описания к реализации аппаратно-программными средствами. CoDesign, претендующий на автоматическое разделение проекта на аппаратную и программную составляющие, в известных реализациях позволяет решать эту задачу для ряда канонических структур и ограниченной элементной базы.

Краткий обзор технологии CoDesign и соответствующего инструментария представлен в следующем параграфе.

1.2.2.1.1 Технология CoDesign

Предлагаемый в рамках CoDesign процесс проектирования схематично представлен на рис. 1.7.

Рис. 1.7. Процесс проектирования в рамках Codesign [47]

CoDesign предлагает следующие этапы технологии проектирования:

- Разработка требований;
- Разработка модели поведения системы на базе формального подхода (сети Петри, CFSM и т.д.);
- Формальный анализ модели, верификация, симуляция, уточнение требований;
- Декомпозиция на аппаратную и программную составляющую;
- Выдача ТЗ для разработчиков программной и аппаратной составляющих или автоматическая генерация кода.

Переход к разделению на программную и аппаратную компоненту может происходить по целому ряду методик:

- Simulated annealing algorithm (Henkel, Ernst);
- All hardware solution (Gupta, De Micheli);
- Алгоритм динамического программирования, основанный на алгоритме Ранца (Jantsch);
- Алгоритм динамического программирования PACE (LYCOS) [56].

В целом эти методики представляют собой последовательность синтеза и анализа вариантов разделения. В некоторых методиках происходит переход от чисто программной к аппаратной реализации (например, Simulated annealing algorithm), в некоторых от аппаратной к программной (например, в All hardware solutions).

После разбиения на аппаратные и программные компоненты в системах CoDesign осуществляется автоматическая генерация кода из FSM в язык C и VHDL (или аналогичные). В ряде систем, в качестве примитивов программной системы используют классические механизмы, используемые в ОСРВ.

Примерами инструментальных средств сопряженного проектирования являются:

- Cierto VCC – система комплексного проектирования в технологии CO-DESIGN от Cadence Design System;
- COSYMA – универсальная система для оценки и синтеза аппаратно–программных компонент, Баруншвейского университета в Германии;
- LOCOS – система синтеза и декомпозиции графовых потоков данных для аппаратно-программных системных компонент, Датского технического университета;
- POLIS – универсальная система для проектирования гетерогенных аппаратно-программных компонент, совместный проект калифорнийского

университета Беркли (США) и Туринского политехнического университета (Италия);

- PTOLEMY– универсальная система моделирования гетерогенных систем калифорнийского университета Беркли (США);
- CoWare – система проектирования гетерогенных однокристальных компонент на основе программируемых схем с перестраиваемой структурой. Леувен, Бельгия.

1.2.2.2 Концепция платформно-ориентированного проектирования

Методология PBD (Platform Based Design) [36, 69] появилась несколько лет назад в контексте комплексного системного проектирования аппаратно-программных ВсС и занимает особое место среди перспективных технологий решения задач проектирования. Основные положения данной концепции отражены в работах [78, 70, 54, 31]. Авторы концепции утверждают, что она позволит повысить производительность проектируемых систем, снизить их стоимость, стоимость разработки, а главное – позволит повторно использовать как аппаратные, так и программные компоненты разрабатываемых систем.

Данная концепция проектирования, продвигаемая интернациональным сообществом, возглавляемым группой исследователей из Калифорнийского университета в Беркли (США) развивалась в значительной мере со стороны «проектирования аппаратуры»: руководители группы являются техническими идеологами ведущих фирм-разработчиков САПР электроники и микросхем, таких как Cadence, Synopsys, Mentor Graphics. Достоинства PBD – унификация взгляда на программные и аппаратные компоненты системы, явный приоритет архитектурного этапа проектирования, ориентация на математическую формализацию процесса создания и анализа моделей системного уровня (в виде «моделей вычисления» (model of computation, MoC)).

Основной упор в своей работе авторы делают на качественное повышение коэффициента повторного использования на всех этапах проектирования ВсС. В жестких условиях рынка решением является повторное использование компонентов системы (плат или их частей, драйверов, интерфейсов и т.п.). Для лучшей проработки альтернативных решений, что важно для повторного использования, предлагается рассматривать систему с двух ракурсов и с различных точек зрения:

- функциональность (что система делает) и архитектуру (как система это делает);
- взаимодействие (обмен данными) и вычисления (выполнение целевой функции системы).

Функциональность может определяться либо разработчиком, либо заказчиком. В первом случае в процесс разработки включается этап функционального проектирования. Архитектура определяет интерфейс и описывает функциональность реализации. При этом она должна не зависеть от

реализации. Кроме архитектуры авторы вводят понятие μ-архитеткуры или микроархитектуры. μ-архитектура — функционально завершенный набор вычислительных компонент (микропроцессор, периферия, программируемая логика, память). При проектировании разработчик должен руководствоваться следующими принципами [78]:

- время и цена разработки определяют процесс принятия решений;
- проектирование должно вестись на высоком уровне абстракции;
- нужно делать надежные, устойчивые системы;
- в системе должно быть несколько сложных и много простых аппаратных компонентов;
- программирование этих компонентов будет производиться на разном уровне.

В параграфе 1.2.2.2.1 представлен краткий обзор элементов методики PBD и инструментальных средств, поддерживающих это направление.

Как и большинство подобных концепций, концепция PBD не обладает достаточной завершенностью, чтобы предложить разработчику относительно конкретные рецепты проектирования. Тезисы предлагаемой методики верны и известны многим разработчикам. Проблема в том, что методика говорит как проектировать, то есть как нужно делать, а не что нужно делать.

1.2.2.1 Платформно-ориентированное проектирование

В концепции платформно-ориентированного системного проектирования основным понятием является понятие платформы. Авторы выделяют аппаратную и программную платформу, они присутствуют в каждом проекте, и их объединение составляет системную платформу.

Аппаратная платформа – это множество архитектур вычислительных машин, позволяющее решать поставленную задачу. При проектировании ограничения архитектуры обычно определяются в терминах производительности и размеров. Как правило, 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. То есть 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.

Примерами программных платформ являются продукты фирм ISI, WindRiver, Microsoft (Windows CE), QSSL (QNX), POSIX [49].

Стратегия платформно-ориентированного системного проектирования представлена на рис. 1.8.

Системная платформа, как уже говорилось, объединяет аппаратную и программную платформы. В начале проектирования обе платформы являются абстракциями. Причем чем выше эта абстракция, тем лучше, тем больше свободы в выборе решений по конкретизации системы. Нежелательно преждевременное разделение функций между аппаратурой и программой. 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 определенная программистская модель.

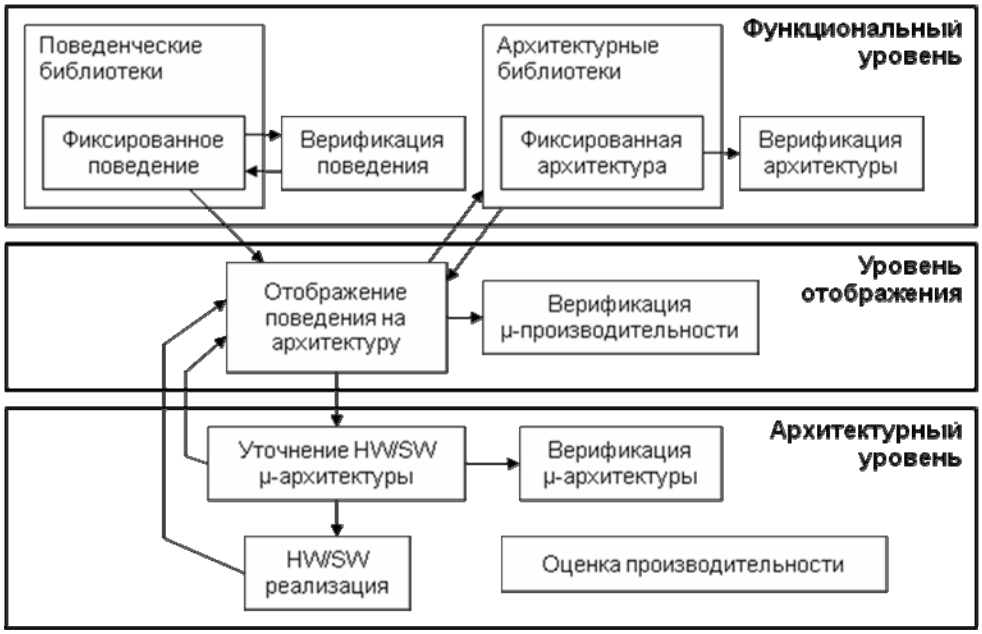


Рис. 1.8. Процесс платформно-ориентированного проектирования

Среди примеров применения своей методики авторами указываются как разработки сторонних фирм, так и результаты собственного проекта MESCAL. В первом случае рассмотрены система передачи видеоинформации (Philips), контроллер управления двигателем (Magneti-Marelli) и беспроводная сеть (Berkley Wireless Research Center). В каждом из примеров приводится архитектура системы, этапы выбора программной платформы, объем кода (строки кода) и т.п. Проект MESCAL (Modern Embedded Systems, Compilers, Architectures and Languages) реализуется для определения эффективности использования методики платформно-ориентированного проектирования. По

результатам анализа проектов складывается ощущение, что в данном исследовании чрезмерное влияние уделяется программной платформе.

Наиболее мощным и комплексным инструментальным средством PBD на сегодня видится Metropolis Framework (Center for electronic system design, University of California at Berkeley), объединяющий методологию PBD, встроенный механизм представления моделей системы (Metropolis Meta Model, MMM), инструментальные средства разработки и отладки проектов [79]. Metropolis Framework поддерживает модели: функциональные, которые применяются в том числе и на (под)уровне миссии, архитектурные, относящиеся к (под)уровню (макро)архитектуры и исполняемые TLM-модели (Transaction Level Modelling на базе языка SystemC), покрывающие уровень микроархитектуры с выходом на генерацию программной или аппаратной реализации компонентов архитектуры. Предусмотрен механизм поддержки нефункциональных требований в виде специальных объектов «quantity manager», управляющих ресурсами, ассоциированными с процессами. Metropolis Framework имеет достаточно длительную историю: она базируется на системе Polis, созданной в 1990 году и ориентированной на специфику автомобилестроения, и как было декларировано, предполагается к развитию в рамках проекта METRO II [40] (однако следует заметить, что начиная с января 2008 года сведения о развитии проекта Metropolis отсутствуют).

Другим известным в разряде PBD проектом того же университета является система Ptolemy II. Ptolemy II ориентирован не на разработку конечных систем, но на исследование различных моделей вычисления, которые должны быть положены в основу архитектуры встраиваемых систем различного рода. Инструмент открытый, предлагает большую библиотеку готовых моделей и архитектурных компонентов, но также предоставляет возможность добавлять собственные. Таким образом, Ptolemy II – это инструмент исследований, но не проектирования.

Из коммерческих инструментальных средств наиболее известно MLDesigner (Mission Level Designer, www.mldesigner.com). MLDesigner в большей степени поддерживает этапы (под)уровней миссии и архитектуры, имеет средства контроля нефункциональных требований (в терминологии MLDesigner – ресурсы), однако явно прослеживается ориентация пакета на разработку встроенного программного обеспечения, нет развитых средств программно/аппаратного деления и реализации компонентов архитектуры.

1.2.2.3 Методология MDD

Методология MDD (Model Driven Design, модельно-ориентированный подход) для проектирования систем реального времени и встраиваемых систем (MDD RTES) является развитием MDA/MDD/MDSD предложенных OMG для создания программного обеспечения. Данное направление в большей степени развивается европейскими исследовательскими и коммерческими организациями, в частности ассоциацией ARTEMISIA (Advanced Research & Technology for EMbedded Intelligence and Systems Industrial Association) [29]

разработчиков европейской технологической платформы GENESYS (GENeric Embedded SYStem platform). Достоинством данного направления является опора на многочисленные и широко апробированные языковые (UML) и инструментальные средства MDD, применение инструмента метамоделей для описания различных (в том числе нефункциональных!) аспектов системной архитектуры для различных прикладных и технологических доменов. Недостатки MDD RTES являются прямым следствием ее истоков: ориентация на создание программного обеспечения, слабая или искусственная адаптация к проблематике проектирования аппаратуры, ко-дизайна.

В следующем параграфе 1.2.2.3.1 дан краткий обзор инструментария MDD BcC.

1.2.2.3.1 Инструментарий методологии проектирования BcC MDD

В части инфраструктуры (framework) MDD RTES наиболее развитым видится методологическая и инструментальная инфраструктура в рамках упомянутой выше технологической платформы GENESYS. В качестве основного языкового средства здесь используются профили OMG UML: MARTE (UML profile for Modeling and Analysis of Real Time and Embedded systems) и SysML (System Modeling Language). Кроме того предложен субпрофиль UML MARTE NFP (Non Functional Properties). Особый и явный акцент на анализ нефункциональных требований и свойств на системном уровне выгодно отличает платформу GENESYS от иных существующих технологий.

С точки зрения инструментария GENESYS предлагает очень широкий выбор: от пакетов UML-дизайна от Rational Software до свободно распространяемых средств типа Papirus (Grafical UML2 modelling), Times (Tool for Modeling and Implementation of Embedded Systems), MAST (Modeling and Analysis Suite for Real Time Application) и других. Обратной стороной такого разнообразия инструментальных пакетов в методологии GENESYS являются значительные трудозатраты и вероятность низкой эффективности сопряжения данных средств в рамках одной инструментальной инфраструктуры. В настоящее время ассоциация ARTEMISIA декларировала начало работ по созданию согласованного, свободно распространяемого пакета программных средств.

Ощутимым недостатком GENESYS (упомянутым ранее при описании MDD RTES в целом) является явно видимая второстепенная роль средств SW/HW Codesign и проектирования уровня микроархитектуры/ESL, то есть всего, что касается аппаратных средств.

Также в рамках MDD RTES предлагаются коммерческие инструментальные средства. Можно назвать CoFluent Studio [38] и Mentor Graphics BridgePointUMLSuite [72]. CoFluent Studio опирается на собственную методологию, названную MCSE или CoMES (Codesign Methodology for Electronic Systems), которая декларируется как расширение

MDD (Enhance Model-Driven Development). Особенностью CoMES является четкая ориентация на создание SystemC модели ESL-уровня и последующий выход на проектирование аппаратных средств. Недостатком видятся не очень мощные средства архитектурного анализа на верхних уровнях. Пакет BridgePointUML Suite, наоборот, ориентирован на создание (автоматическую генерацию) встроенного программного обеспечения, хотя декларируются работы по развитию его в сторону генерации RTL-кода.

1.2.2.4 Тенденции развития методик проектирования ВсС

Выше упомянуты только средства, наиболее интересные и значимые для развития области проектирования системного уровня. Вне обсуждения осталось большое количество инструментов на базе SIMULINK (MathWorks), так как данное направление имеет более коммерческо-прикладную направленность, нежели развивает методы проектирования. Более полный обзор существующих инструментальных методологий и пакетов в области проектирования встраиваемых систем, прежде всего на системном уровне, можно найти в [41, 52, 79, 67, 53].

Примечательно, что ведущие разработчики САПР в области электроники и встраиваемых систем имеют очень ограниченное предложение методик и инструментов в этой области – концентрируются в основном на средствах (под)уровня архитектуры: TLM/RTL-design, SystemC HW/SW co-verification (например, System Studio (Synopsys), Palladium Series (Cadence), Vista/Visual Elita Series (Mentor Graphics)). Причины этого лежат, скорее всего, в области коммерческой эффективности: указанные фирмы продолжают поддерживать исследовательские работы в ожидании законченного продукта.

В завершении попытаемся зафиксировать общие недостатки доступных методологий, инструментов и сред проектирования (frameworks), побуждающие к определенному выбору и развитию исследовательских работ в области системного проектирования для встраиваемых систем:

Отдельные (особенно нефункциональные) аспекты проектирования рассматриваются для системы в целом, в рамках соответствующих общесистемных моделей: энергопотребления, надежности, информационной защиты и т.п. Это, естественно, объясняется внутренней сложностью и сильной взаимозависимостью отдельных архитектурных компонентов системы. С другой стороны, ограничены возможности оценки влияния конкретной реализации конкретного компонента на аспектные показатели (характеристики). Следует разработать такую структуру и форму описания архитектурных компонентов, которая будет в явном виде предлагать аспектные оценки (веса). Данный подход отразился в понятии архитектурного агрегата (АА), обсуждаемого в главе 2.

Во всех рассмотренных методологиях проявляется явный приоритет функционального аспекта на архитектурном уровне. Нефункциональные аспекты либо играют роль вспомогательных (второстепенных) критериев

проекта, либо, многие из них, не учтены вообще. Так выпали из рассмотрения инструментальный и конструктивно-технологический аспект. Что касается приоритетов аспектов, то функциональность не всегда (не для всех приложений) может быть основной. Например, требования реального времени могут отвергать функциональные требования обязательной доставки данных по сети; или набирающая популярность концепция вероятностной логики может повышать приоритет аспекта энергопотребления над точностью вычисления функций. Данные вопросы должны быть учтены при разработке инструмента архитектурного описания системы.

Пространство вариантов реализации архитектурных компонентов остается в основном в рамках HW/SW фазы runtime. Варианты Design/Run Time, варианты различных уровней виртуализации архитектуры не учитываются в рамках системного проектирования.

1.2.3 Языки описания архитектуры встраиваемых систем

В вычислительной технике под процессором понимается устройство обработки информации. Несмотря на то, что в основе современных процессоров могут лежать различные модели вычислений, исторически сложилось так, что чаще всего используется модель на базе машины Фон-Неймана.

1.2.3.1 Методология представления гетерогенных систем IRSYD

Одной из основополагающих задач технологий сквозного проектирования BcC является создание эффективного системного описания, удовлетворяющего ряду противоречивых требований. Один из предлагаемых подходов к этой проблеме – язык внутреннего представления разнородных систем IRSYD (IRSYD – An Internal Representation for System Description Version 0.1) [44].

Авторы подхода определяют следующие понятия:

Функциональность – идеологические параметры модели, отвечающие за выполнение целевых функций системы;

Структура – элементы модели системы, выражаемые осязаемыми или виртуальными объектами, являющимися носителями определенной части функциональности. Делается допущение о том, что могут существовать модели, не обладающие структурной составляющей.

Поведение – динамическое представление и динамические законы изменения всех параметров модели. Можно говорить как о *структурном*, так и *функциональном* поведении модели.

Авторы предлагают способ (методологию) внутреннего представления сложных разнородных систем. В качестве критериев такого представления декларируются следующие требования: статическая и динамическая структурная и функциональная модульность и иерархия, коммуникация между различными подсистемами, представление абстрактных типов данных,

механизмы представления “нефункциональной” информации, повторное использование шаблонов разработки.

В рамках языка IRSYD предлагаются средства описания структурного, функционального и временно́го (temporal) поведения сложной системы. Основной идеей такого описания является представление системы в виде обобщенного графа, выражающего потоки управления и потоки данных. При этом вводится функциональная модульность и иерархия, которая является структурой системы.

В методологии IRSYD основной упор делается на функциональную составляющую. Именно эта составляющая ставится во главу угла при моделировании и проектировании системы. Но нужно сказать, что разработчики IRSYD оставляют возможность внесения некоторых “нефункциональных” элементов в модель в виде некоторых атрибутов.

Право рассматривать атрибуты в процессе проектирования и моделирования системы предоставлено некоторым инструментальным средствам, разработанным для каждого конкретного процесса проектирования в зависимости от нужд разработчика. Никаких общих рецептов построения таких средств и критериев определения атрибутов в IRSYD не предлагается.

Задача декомпозиции hardware/software также затрагивается в IRSYD. Эта задача должна быть решена, и, как утверждают авторы, относительно легко, на уровне отражения процессов на структурные элементы. К сожалению формальных методов и критериев принятия решений при декомпозиции HW/SW в IRSYD не предлагается.

1.2.3.2 Язык описания архитектуры встраиваемых систем AADL

Язык AADL начал разрабатываться как стандарт международной ассоциации инженеров автомобилестроения в 2001 году [76]. Первоначально он позиционировался как язык описания архитектуры систем авиационной электроники (Avionics Architecture Description Language), однако на практике им покрываются все встраиваемые (и некоторые другие) системы, а не только авиационные. Поэтому сейчас AADL – это, дословно, язык архитектурного анализа и проектирования (Architecture Analysis and Design Language). Стандарт определяет текстовую нотацию AADL, в приложении к нему также описывается графический способ представления архитектуры.

Наличие такого стандарта дает разработчикам следующие преимущества:

- Четко определена общая форма записи, что исключает неоднозначность трактовки;
- Используется единая архитектурная модель, дополненная анализируемыми свойствами;
- Возможно повторное использование и интеграция архитектурных моделей;

- Обеспечиваются возможности взаимодействия и расширения инструментальных средств архитектурного моделирования;
- Возможности языка соответствуют широко распространенным способам проектирования и анализа архитектуры вычислительных систем.

Подробно характеристика AADL представлена в параграфе 1.2.3.2.1.

1.2.3.2.1 Обзор языка описания архитектуры BcC AADL

AADL в первую очередь ориентирован на моделирование и анализ встраиваемых систем с ограничениями реального времени. Для успешного функционирования к таким системам обычно предъявляются специфические требования по отказоустойчивости, защищенности, безопасности, производительности и т.д. Четко определенная семантика объектов и конструкций AADL позволяет анализировать вычислительные системы по всем вышеприведенным аспектам. Типичные области приложения данного языка – проектирование информационно-управляющих систем, от систем бытовой электроники, медицинских приборов до автотранспортных и аэрокосмических систем управления. Это, однако, не исключает применения AADL в проектировании любых других вычислительных систем.

Любое описание архитектуры на языке AADL практически полностью состоит из описаний отдельных ее компонентов. Каждый компонент принадлежит к одной из следующих категорий (описания приводятся в контексте AADL):

Память, запоминающее устройство (memory) – это аппаратный компонент, служащий для хранения двоичных данных и программного кода;

Прибор, устройство (device) – некоторый активный аппаратный узел с определенной функциональностью, внутренняя структура которого не раскрывается, в соответствии с принципом “черного ящика”;

Процессор (processor) – основной вычислитель, занимающийся диспетчеризацией и исполнением программного кода;

Шина (bus) – физический интерфейс (обычно вместе с коммуникационным протоколом), связывающий процессоры, запоминающие устройства и приборы;

Данные (data) – абстракция типа данных в программном коде, возможно, имеющего сложную внутреннюю структуру;

Подпрограмма (subprogram) представляет собой входную точку в программном коде, которой может быть передано управление в ходе исполнения потока или подпрограммы;

Поток, нить управления (thread) – это исполняемая последовательность команд, которая с точки зрения процессора рассматривается как единый объект диспетчеризации;

Группа потоков (thread group) служит для логического объединения одного или более потоков;

Процесс, задача (process) соответствует отдельному виртуальному адресному пространству и является законченной программной единицей, если содержит, хотя бы один поток управления;

Система (system) – совокупность программных и аппаратных компонентов, а также других систем, возможно, связанных между собой, с полностью или частично известной внутренней структурой.

Первые четыре категории соответствуют аппаратной части вычислительной системы, следующие пять – программной части. Таким образом, AADL позволяет описывать программные системы и их аппаратные платформы как взаимодействующие компоненты со строго определенной семантикой и связями.

Система, как компонент особой категории, представляет собой композицию аппаратных и программных элементов, может быть иерархически вложенной. Система, в противоположность абстрактному прибору, выражает собой принцип “белого ящика” и дает возможность моделировать сколь угодно сложные архитектурные элементы.

Поддерживаются три типа взаимодействий между компонентами:

Направленные потоки данных и управления, моделируемые посредством соединений (connection) между портами различных типов (data port, event port, event data port);

Вызовы (call) подпрограмм с последующим возвратом;

Доступ нескольких компонентов к разделяемым данным (data access).

Для большинства компонентов разрешено задавать несколько режимов работы (mode), в каждом из которых поведение компонента различно. Задаются также правила переходов из одного режима в другой.

Каждый компонент определяется на AADL в два этапа. Первый, называемый описанием типа, задает функциональный интерфейс компонента, видимый извне и доступный для использования. На втором этапе описывается реализация компонента (implementation), то есть его внутреннее устройство, включающее подкомпоненты, свойства, связи и т.д. Каждому типу компонента может соответствовать несколько реализаций. Таким образом, четко прослеживается поддержка одного из базовых принципов объектно-ориентированного подхода (ООП): инкапсуляции. Это оставляет большую свободу при проектировании системы: например, описания типа и реализации одного и того же компонента могут производиться на разных стадиях проектирования, возможно, разными инженерами.

Еще один фундаментальный принцип ООП – наследование – также находит отражение в языке AADL. Наследоваться может как тип компонента (от другого типа), так и его реализация (соответственно, от другой реализации). Наличие наследования избавляет от необходимости многократно повторять совпадающие части описания схожих компонентов.

Для каждого компонента AADL, а также для составных частей компонентов, связей, потоков данных и управления, режимов и вызовов может быть определено любое количество свойств (properties). Каждое свойство имеет уникальное имя, тип данных и, собственно, значение. Свойства в AADL – универсальный механизм семантического расширения конструкций языка, так как могут вводиться пользователем в соответствии с абстракциями создаваемой модели практически без ограничений. Кроме того, стандарт определяет базовый набор свойств, доступных с самого начала. Вот некоторые примеры предопределенных свойств, иллюстрирующие диапазон возможных применений этого механизма:

- Latency – задержка передачи потока данных;
- Memory_Protocol – права доступа к памяти (чтение и/или запись);
- Queue_Size – размер входного буфера порта;
- Source_Code_Size – размер памяти, занимаемой статическим исполняемым кодом и константными данными;
- Period – период (интервал диспетчеризации) задачи или нити;
- Allowed_Message_Size – допустимый размер пакета данных, передаваемого по шине;
- Base_Address – адрес ячейки памяти, содержащей первое слово структуры данных.

Способ использования информации, содержащейся в свойствах, определяется пользователем и зависит от индивидуальных особенностей моделируемой системы. Это позволяет целенаправленно исследовать большинство системных характеристик, критичных для архитектурного проектирования, даже если они не входят в базовый набор языковых конструкций.

Не менее сильной стороной AADL, чем его расширяемость, можно также считать возможность описывать системы на любом уровне абстракции. Например, описание вида

```
system My_System
end My_System;
```

является полностью корректным с точки зрения синтаксиса языка, то есть не вызывает ошибок при формальной верификации. Это означает, что можно строить архитектуру системы поэтапно, начиная с самого общего представления и постепенно понижая уровень абстракции, уточняя внутреннюю организацию компонентов, вводя дополнительные свойства и ограничения. При таком стиле проектирования на каждом этапе мы получаем полноценную архитектуру как с точки зрения AADL, так и в соответствии с определением архитектуры, приведенным в учебном пособии. Ее можно подвергать анализу, насколько это позволяет тот или иной уровень

рассмотрения, и таким образом, выявлять потенциальные проблемы и выбирать лучшие высокоуровневые решения на ранних этапах проектирования.

1.3 Предпосылки повышения уровня абстракции в методиках проектирования встраиваемых систем

1.3.1 Кризис методик проектирования встраиваемых систем

Специалисты отмечают, что использование традиционных методов проектирования в сегодняшних условиях привело к системному кризису в области создания ВсС [63, 16]. Как показано на рис. 1.9 существует растущий разрыв между потенциальными возможностями элементной базы, эффективностью схемного и программного проектирования, требуемыми объемами верификации и тестирования, архитектурным проектированием. Использование распространенных технологий проектирования заставляет разработчиков постоянно повышать уровень "строительных кирпичей", что при отсутствии средств комплексной оптимизации и верификации ведет к эффекту "насыщения" в достижимой сложности и качестве конечного продукта [16].

Проблема состоит в том, что существующий потенциальный объем разработок ВсС не может быть выполнен коллективами, работающими в данной области, в рамках традиционных технологий проектирования либо в требуемые сроки, либо с достаточным качеством. Кризис влияет на качество разработок (массовое появление "сырых" продуктов на рынке, в первую очередь это относится к микросхемам и программному продукту), ограничивается доступная сложность проектируемых ВсС, практически не применяются технологии повторного использования элементов разработок.

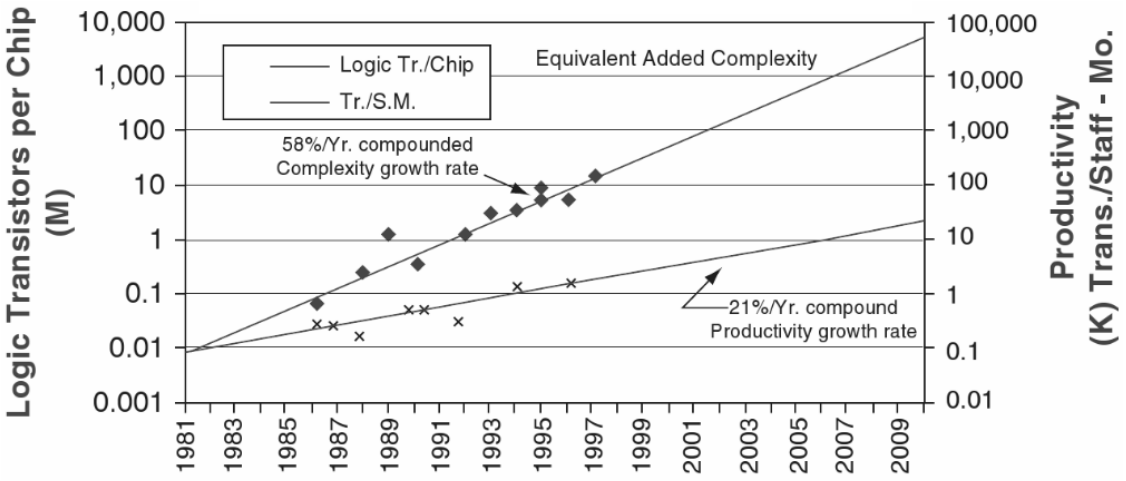


Рис. 1.9. Кризис методик проектирования ВС
(Bryan Preas, Xerox PARC, 35th DAC)

Уместным будет привести слова Э. Дейкстры из статьи "The end of Computing Science?" [62]: "…я хочу сказать, что самая главная проблема вычислительной техники, а именно «как избежать неразберихи», НЕ была решена. Напротив, сложность большинства наших систем значительно превышает разумную, и они слишком беспорядочны и хаотичны для того, чтобы использовать их с удобством и уверенностью".

Все это в первую очередь влияет на сферу разработки ответственных ВсС, связанных с промышленностью, энергетикой, транспортом, медициной, обороной, где недопустимо внедрение некачественных разработок, присутствует требование обеспечения длительной эксплуатации. В результате во многих случаях сдерживается применение новейших технологических достижений.

Традиционно раздельное проектирование программных и аппаратных компонентов приводит также к искусственному завышению требований к аппаратной части проекта для того, чтобы гарантированно "вписаться" в полученные ресурсы. Еще одна проблема – это трудности оценки надежности и тестирования систем, созданных по раздельной технологии. Задача разделения системы на программно/аппаратную реализацию и выбор стандартных компонентов в настоящий момент не имеет формального способа решения, и каждый раз разработчик принимает такие решения, основываясь на собственном опыте и номенклатуре доступных электронных компонентов.

Мощность вычислительных машин растет, рынок ВсС развивается и требует все более сложных систем за все более короткие сроки. Понятно, что чем проще система, тем меньше времени уходит на ее разработку. При этом различные авторы прогнозируют, что в недалеком будущем на разработку какой-либо ВсС будет отводиться фиксированное время независимо от ее сложности. В таком ключе разработка систем “с нуля” будет не просто неэффективной, а провальной. Проекты будут неминуемо растягиваться по времени. В противном случае будет страдать качество создаваемых систем.

Таким образом, для вычислительной индустрии жизненно важно создание качественно новых методик, технологий и средств проектирования ВсС.

1.3.2 Перспективы развитися методик проектирования встраиваемых систем

Многообразие методик и технологий, которые используются сегодня при создании ВсС, демонстрирует сложность и неоднозначность реализации предмета проектирования. Предыдущий материал убедительно демонстрирует актуальность и эффективность перемещения акцентов в область высокоуровневых этапов проектирования (часто рассматриваемых в качестве начальных), что ассоциируется с усилением роли общего видения системы, активным использованием различных абстракций, явным разделением этапов проектирования и реализации. Естественное участие в создании ВсС специалистов различных категорий и направлений определяет присутствие нескольких "доминирующих" идеологий со своими ролевыми приоритетами, оценками значимости выполняемых работ, базовыми моделями проектируемых систем, глоссариями и приемами формализации деятельности. В зависимости от складывающейся ситуации в проекте ВсС, которая обычно определяется не имеющими прямого отношения к существу решаемой задачи факторами, руководящую роль чаще всего могут выполнять специалисты "прикладники"

(например, в области теории автоматического управления, АСУТП, связи), "аппаратчики" (специалисты по цифровой технике) или "программисты" (специалисты в области системного или прикладного программирования). Любой из приведенных примеров будет приводить к "перекосу" в организации проектирования, что скажется на всех ключевых характеристиках как создаваемой системы, так и проекта в целом.

Стремительное увеличение сложности и объема вопросов, которые относятся к сфере деятельности специалиста по информатике и вычислительной технике, привело к ситуации, которую можно охарактеризовать следующим образом.

Первоначальное развитие вычислительной техники как интегрального предмета с естественным общим видением заменилось определенной изоляцией и противопоставлением ветвей создания аппаратуры (с ее "аппаратным " понятием архитектуры) и создания программного обеспечения (с одной стороны, идущее от "чистых" математических абстракций, а с другой стороны, вынужденное приспосабливаться к продукту "аппаратчиков", оказывая на него слабое влияние). Такой разрыв определил дефицит специалистов с единым видением данной предметной области. *Область деятельности таких специалистов-системотехников – архитектура информационной инфраструктуры любой создаваемой информационно-вычислительной системы*.

Следует отметить, что на сегодняшний день существует значительное число областей применения ВС, в которых вполне допустимо в качестве идеолога проекта использовать специалиста в области программного обеспечения. Запас вычислительных ресурсов, предоставляемый современными стандартными аппаратными средствами, и уровень развития системного ПО позволяют в этих областях успешно решать прикладные задачи с приемлемым качеством и надежностью.

Иначе выглядит ситуация, когда роль идеолога проекта выполняет специалист по вычислительной аппаратуре. Обычно, в его поле зрения попадает лишь незначительная низкоуровневая часть организации вычислительного процесса в системе. Это приводит к невозможности адекватно оценить как требуемые аппаратные ресурсы, так и общие трудозатраты на реализацию проекта. Очень часто базовая парадигма проектирования такого специалиста формулируется следующим образом: "Главное создать программируемую аппаратуру, а запрограммировать можно будет все". Положительный результат такого проектирования возможен лишь в случае простейших систем с незначительной "инфовычислительной" составляющей.

Область проектирования ВсС накладывает жесткие ограничения на избыточность технических ресурсов системы, является критической в смысле надежностных показателей, сроков и бюджетов проектов. Таким образом, здесь необходимо наличие истинно "интегральных" специалистов – вычислителей, способных эффективно исполнять роль архитекторов системы. Условиями для

формирования и эффективной деятельности таких специалистов – системотехников могут быть соответствующие методики и технологии проектирования ВсС, основанные на "интегральном" взгляде на предмет проектирования.

Современная вычислительная техника представляет собой некоторое число базовых принципов организации вычислительного процесса при решении прикладной задачи и огромное количество технических решений различного уровня, качества и степени общности, которые охватывают как сферу реализации ВС, так и все части процесса проектирования. Многообразие доступных реализаций ВС, особенно в области ВсС, давно доказало неэффективность использования ограниченного числа типовых решений. Явно прослеживаются проблемы, возникающие уже на уровне классификаций даже базовых вычислительных архитектур [26]. Можно приводить многочисленные примеры дублирования одних и тех же принципов в технических решениях из разных областей вычислительной техники, которые имеют различные названия и подаются (в том числе, студентам) совершенно с разных позиций. Существующая степень классификации, унификации, степени абстрагирования в технических решениях из области вычислительной техники неудовлетворительна. Кроме того, значительный вред (в техническом плане) развитию информатики и вычислительной техники как науке наносит осознанное искажение и сокрытие сути технических решений (начиная от терминологии), которые происходят в угоду нетехническим соображениям.

Необходима единая система координат, в которой разработчики ВС смогут представлять свои решения, накапливать элементы повторного использования, сосредоточиться на организации прикладного вычислительного процесса, подбирая для этого эффективную реализацию. К сожалению, прямые упоминания о такой постановке задачи в литературе находят ограниченное отражение [71, 39, 50, 12].

В области ВсС можно отметить еще ряд факторов, которые необходимо учитывать при создании единого проектного пространства. Такими факторами являются жесткая специализация создаваемой системы, комплексный характер "невычислительных" ограничений, разнообразие элементной базы (в том числе, предполагающей специфические технологии программирования), постоянно растущий спрос на выполнение новых разработок.

Выводы

1. Класс ВсС постоянно развивается, о чем свидетельствует техническая и терминологическая эволюция категорий ВС, которые ассоциируются с системами контроля и/или управления объектами физического мира. Однако отсутствует четкое определение этой категории ВС, эффективные варианты

классификации ВсС, что препятствует систематизации технических решений и методов проектирования в данной области.

2. В главе предложено определение ВсС, основанное на характере взаимодействия ВС и контролируемого и/или управляемого объекта физического мира, на общих ключевых принципах проектирования, не ограничивающее сложность, топологию и принципы организации системы.
3. Показано, что ВсС предполагают всегда комплексное аппаратно-программное проектирование в рамках решения конечной (прикладной) задачи, что на практике проявляется в ряде канонических сценариев проектирования. Сценарии различаются «глубиной погружения» в аппаратно-программную организацию проектируемой ВС с точки зрения фиксируемой разработчиком проектной платформы и определяют потенциальную эффективность проектирования.
4. Анализ традиционных подходов к проектированию ВсС демонстрирует отсутствие в настоящее время эффективных средств разработки. Это подтверждается огромным количеством “сырых” продуктов на рынке ВсС, которые содержат значительное количество дефектов, не устраненных на этапе проектирования и опытной эксплуатации. В основном проблема заключается в несостоятельности традиционных подходов к проектированию ВсС на фоне современных требований к таким системам. Также в значительной мере остаются нереализованными потенциальные возможности современной элементной базы. В ближайшее время существующий значительный разрыв между требованиями к ВсС и эффективностью схемного и программного проектирования, требуемыми объемами верификации и тестирования устройств будет только возрастать.
5. Последние годы различные научные группы ведут активные исследования в области разработки новых парадигм проектирования ВС, призванных решить назревшие проблемы и вывести отрасль из сложившегося глубокого кризиса. Нужно отметить, что картина исследований в настоящее время носит фрагментарный характер, различные подходы обладают разной степенью проработанности, формализации и распространенности. Однако на основании сложившейся картины уже можно делать выводы об основных направлениях, признанных перспективными с точки зрения построения методик проектирования ВС:

   - увеличение удельного веса этапов проектирования, на которых разработчик оперирует с абстрактным представлением системы и ее элементов;
   - разработка системы на уровне абстрактного представления вычислительных механизмов, широкое использование моделей вычислений, средств моделирования;
   - накопление технических решений на уровне абстрактного представления с целью их повторного использования;

- унификация возможно большей части процесса проектирования аппаратных и программных элементов, возможность принятия решений о реализации элементов системы на более поздних этапах проектирования;
- комплексный учет традиционно "второстепенных" требований технического задания и "не вычислительных" ограничений в рамках всего проекта.

6. Анализ процессов, происходящих в области проектирования СнК, показывает стремительно растущие возможности в формировании различных архитектур и микроархитектур для разработчика СнК (ASIP, сети на кристалле, «прямая аппаратная реализация» сложных функциональных блоков, динамическая реконфигурируемость и др.). Формально оставаясь категорией электронной компонентной базы, СнК фактически становятся одной из категорий ВсС (проектной платформой «глубокого погружения»), для которой также важнейшим фактором является развитие методов и средств высокоуровневого архитектурного проектирования.

2 Архитектурное проектирование встраиваемых вычислительных систем

2.1 Система архитектурных абстракций

2.1.1 Архитектурное проектирование и группы архитектурных абстраций

2.1.1.1 Проблемы архитектурного проектирования ВсС

В настоящее время большинство ВсС являются гетерогенными многопроцессорными ВС, где помимо традиционных микропроцессорных элементов присутствуют интегральные программируемые контроллеры промышленных интерфейсов, ПЛИС, блоки памяти с различной организацией, другие интегральные компоненты.

Процесс проектирования подобных систем представляет собой сложную комплексную научно-техническую задачу, в рамках решения которой коллектив разработчиков определяет архитектуру, соотношение и функциональное наполнение аппаратной и программной составляющих системы. Проблема заключается в существовании огромного количества потенциально пригодных вариантов реализации, порождаемых по одному техническому заданию. Эти варианты могут отличаться друг от друга коренным образом, а предварительная оценка вариантов реализации затруднена. Выбору подлежат средства и технологии, направленные на выполнение требований реального масштаба времени, надежности и безопасности функционирования, эффективной отладки и тестирования на этапах проектирования, производства и эксплуатации. На практике, число анализируемых разработчиком вариантов, включая прототипы, ограничивается единицами. Это определяется сжатыми сроками и бюджетами разработок на фоне высокой сложности проектируемой системы, отсутствием эффективных технологий и инструментальных средств, в том числе САПР.

Объективная сложность структурно-функциональной организации проектируемой системы во многом определяется количеством и степенью неоднородности компонентов и подсистем, параметрами интерфейсов и протоколов, требованиями по надежности и безопасности функционирования. Неоднородность вычислительных механизмов и компонентов современных микропроцессорных систем существенно усложняет процесс проектирования и не способствует гарантированному получению качественного результата.

Действительно, архитектуру ВсС можно представить как совокупность решений, удовлетворяющих критичным характеристикам проекта. Тогда переход к реализации будет происходить через уточнение организации системы с сохранением критичных ограничений и привнесением элементов реализации, для которых ограничения считаются некритичными. Можно считать, *что для одной архитектуры с зафиксированными критичными ограничениями существует множество реализаций, которые различаются в части некритичных ограничений*. Это означает, что *реализация однозначно вытекает*

из архитектуры, а из реализации исходную архитектуру не восстановить в силу привнесенных некритичных ограничений.

В данной главе описываются HLD методы и средства, призванные повысить эффективность проектирования, в первую очередь, сложных ВсС. Архитектурное рассмотрение разрабатываемой системы предлагается в качестве основного в процессе проектирования. Проводится исследование проблем формализации архитектурного представления ВсС, формулируются основополагающие понятия архитектурной модели. Даются формальные определения основных положений аспектной технологии сквозного проектирования ВсС на основе понятия архитектурных агрегатов.

Важно подчеркнуть, что взгляд на вычислительную технику и инфо-коммуникационные технологии (ВТ&ИКТ) с «абстрактных» высот необходим даже в условиях слабой формализации методологии. В инженерный язык и практику проектирования необходимо срочно вносить единые абстракции вычислительной и инфо-коммуникационной отрасли, которые:

- собственно определяют (ограничивают снизу и сверху) область деятельности – зону ответственности ВТ&ИКТ – специалиста;
- делают «прозрачным» представление того, что называют «технологиями» во всех областях ВТ&ИКТ;
- взвешенно распределяют влияние реализации на принцип решения задачи в ВТ&ИКТ – областях;
- позволяют аккумулировать и свободно (осознанно и без «шор») развивать концептуальные технические решения, менять [расширять] уровень reuse-технологий;
- выявляют (проявляют) «болевые точки» в организации и проектировании ВТ&ИКТ – систем, следовательно, обеспечивают грамотную постановку задач.

2.1.1.2 Группы архитектурных абстракций

В рамках задачи высокоуровневого проектирования ВсС в работе рассматривается единое проектное пространство, которое характеризуется следующим рядом положений:

- множеством базовых абстракций процесса проектирования ВсС;
- рассмотрением архитектуры ВсС как всестороннего (разнопланового) непротиворечивого представления критичных для результата аспектов;
- расширением задачи проектирования ВсС до задачи непосредственной организации целевого вычислительного процесса;
- созданием условий для повторного использования абстрактных объектов процесса проектирования ВсС.

Множество базовых архитектурных абстракций сведены в следующие четыре группы:

- базовые элементы ВС (вычислительный механизм, вычислительная платформа, архитектурный агрегат);
- абстракции представления ВС на системном уровне (архитектура, архитектурная платформа, архитектурная модель, аспект);
- абстракции процесса проектирования (проектирование ВсС, инфраструктура проекта, проектное пространство, аспектное пространство);
- понятия для анализа и оценки существующих архитектурных решений (вычислительный процесс, виртуальная машина, модель вычислений).

Часть из предлагаемых 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.

В представленный список вошли основные (базовые) абстракции HLD ВсС, в пособии также используются производные и уточняющие абстракции как вычислительного, так и «смешанного» характера.

2.1.2 Вычислительные и невычислительные абстракции

2.1.2.1 Вычислительные абстракции

Объединяющим началом предлагаемой модели проектного пространства ВсС следует считать тезис о возможности представления вычислительного процесса в терминах вычислительных абстракций, не привязанных непосредственно к конкретным реализациям. Будем считать, что такие виртуальные вычислительные процессы должны строиться на основе вычислительных механизмов различного назначения. Следует отметить, что на практике виртуальные вычислительные процессы активно реализуются в современных ВС, при этом разработчики такие процессы связывают с понятием виртуальной машины, трактуя ее как программно-реализованную модель реальной вычислительной машины. Данный технический прием является удобной архитектурной абстракцией, направленной на структуризацию вычислительного процесса и, тем самым, на локальное понижение степени его сложности.

Существует известный тезис о "семантическом разрыве" между элементами (операциями, структурами данных) представления вычислительных процессов в языках программирования высокого уровня и возможностями физических (аппаратных, микропрограммных) средств реализации на уровне микроопераций, микрокоманд и команд с соответствующими форматами данных, уровнем и степенью параллелизма, ограничениями памяти и т.д. Развитие вычислительной техники в области методов и технологий

проектирования идет по пути развития вычислительных парадигм (таких, как модели вычислений, парадигмы программирования), предлагая качественно новые решения. Это движение стремительно увеличивает семантический разрыв с уровнем физической реализации. 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.

Любая задача в рамках ВсС может быть подготовлена к исполнению (решению) с использованием той или иной степени вложенности уровней вычислительной иерархии. Это будет в первую очередь определяться размерностью задачи, квалификацией и личными способностями и пристрастиями разработчиков, допустимыми сроками проектирования. В дальнейшем такое решение может быть реализовано с различной степенью оптимизации, от использования "как есть", до глобально оптимизированного "плоского" одноуровневого представления нижнего уровня вычислительной иерархии. Естественно, ресурсоемкость целевого варианта системы в зависимости от числа оставшихся в результате оптимизации уровней, будет меняться в широких пределах.

Проектное пространство ВсС представляется посредством совокупности базовых абстракций процесса проектирования. Сразу оговоримся, что проектное пространство ВсС не является чисто вычислительной абстракцией, что будет обсуждаться далее.

Прежде всего, в пространство входят 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.

Как было отмечено выше, необходима система вычислительных абстракций, позволяющих разработчику оперировать в рамках целевого и проектного пространств при создании ВсС. Традиционно перечисленные выше элементы рассматриваются в качестве составляющих целевой системы. Частью проектного пространства, которую в этом случае целесообразно представлять в подобных понятиях, будет выступать внутренняя организация инструментария. Однако, как будет показано ниже, полезно рассмотрение целевой системы с

позиций организации вычислительного процесса с его off- и online фазами. В этом случае, принадлежность и область действия вычислительных абстракций расширяется, захватывая оба пространства проектирования. Примером такого перехода, в значительной мере привычного и понятного большинству разработчиков ВсС, является трактовка термина "вычислительная платформа" как совокупности целевых аппаратно-программных средств (например, аппаратура и ОС ПК) и транслятора ЯВУ.

Существующие сегодня вычислительные абстракции в терминологическом и смысловом плане нечетко определены, часто имеют несколько различных трактовок, иногда пересекаются по области действия или противоречат друг другу. Кроме того, существует потребность в расширении области действия, унификации трактовки существующих понятий и введении ряда новых абстракций при проектировании ВсС.

Одним из центральных в вычислительной технике является понятие вычислительного процесса, определим его следующим образом.

Вычислительный процесс – процесс преобразования данных вычислительным устройством (вычислителем, вычислительной машиной) в соответствии с заданной функциональностью. (Может выполняться однократно или циклически).

С позиций данного определения введем ряд понятий, которые далее будут рассматриваться подробнее.

Вычислитель – функционально завершенное устройство, позволяющее осуществлять [выполнять] один или более вычислительных процессов [не менее одного раза]. Процесс преобразования данных описывается алгоритмом на алгоритмическом языке.

ВМх (вычислительный механизм) – техническое решение, реализующее часть (фрагмент, элемент, «аспектный срез») вычислительного процесса. Будем считать, что ВМх в отличие от функционального элемента (например, регистра) сочетает в себе некоторую функциональность и внутреннее устройство (что делает и как делает).

ВВМ (виртуальная вычислительная машина) – техническое решение, реализующее вычислительный процесс. Разделение ВС (вычислительной системы) на механизмы и машины возможно относительно выделенных (в качестве самостоятельных) вычислительных процессов.

2.1.2.2 Абстрактная элементная база ВсС

Переход в область абстрактного проектирования и расширение этой области при создании ВсС возможны только при серьезном изменении способа представления проектируемой системы и создании новых проектных методов и средств. Важнейшим тезисом, который обеспечивает такой переход, является *принципиальная свобода разработчика в выборе реализации построенной архитектуры ВсС* (и ее частей). При этом элементная база (в традиционном

понимании) должна выступать не более чем частным ограничением, даже в том случае, если она фиксируется на уровне технического задания.

Традиционное понимание термина "элементная (компонентная) база" в современной вычислительной технике по-прежнему составляет совокупность электронных, оптических, механических и иных физических компонентов (элементов, модулей, блоков), из которых складывается физическая реализация вычислительной системы. На сегодня в перечень таких компонентов входят сложные микросхемы процессоров, контроллеров, акселераторов, системные платы вычислителей. Во многих случаях в состав таких элементов входят программные средства, размещаемые во встроенных ПЗУ. Таким образом, даже традиционное представление вычислительной элементной базы выходит далеко за границы описания только конструкции и схемотехники, затрагивая все больше вопросы системотехники, программирования, архитектуры.

В области программирования магистральным направлением является создание повторно-используемых компонентов различного уровня и сложности. В области проектирования кристаллов развивается технология виртуальных компонентов, фиксирующих на разных уровнях (от системного до топологического) функциональность технического решения для дальнейшего использования в процессах создания микросхем.

Таким образом, совершенно естественным следует считать появление *расширенной трактовки вычислительной элементной базы, как всей совокупности зафиксированных для масштабного повторного использования решений, как физических, так и иных уровней абстракции*. Это могут быть схемные решения, объекты логического уровня (протоколы, алгоритмы), программные компоненты с различным уровнем представления (исходные тексты, объектные модули, параметризуемые библиотеки и др.), стандарты на интерфейсы, возможно, объекты более высокого уровня абстракции (вычислительные модели).

Будем называть *традиционную вычислительную элементную базу физической, а элементную базу в расширенной трактовке абстрактной*. При этом абстрактная элементная база включает в себя элементы различного уровня и степени реализации, в том числе и физические.

Понимание абстрактной элементной базы хорошо согласуется с тезисом расширения этапа абстрактного проектирования ВсС, с аспектным методом проектирования, понятием проектного пространства технических решений, вычислительными механизмами как элементами проектирования.

На свободу проектировщика от ограничений элементной базы на этапе абстрактного (или архитектурного) проектирования в значительной мере влияет вопрос о проведении границы в проекте между фазами проектирования и реализации. Тезис такой свободы при определении архитектуры и, возможно, других значимых проектных решений требует выбора глубины архитектурной проработки. С этим будет связано определение круга абстрактных элементов,

выбор которых будет относиться, соответственно, к фазам проектирования и реализации.

2.1.2.3 Аспектный подход к проектированию ВсС

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. Мы будем называть такие локализованные стороны проекта или целевой системы *аспектами*. Другими словами, *аспект – это некоторая частная проблема проектирования в рамках задачи создания ВсС*. Подчеркнем еще раз, что 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*.

Такие стороны, или аспекты, могут обладать различной степенью пересечения. Кроме того, полезным является рассмотрение проектного пространства целевой системы, как части пространства всего проекта в целом, что повышает эффективность проектирования ВсС. В этом случае в поле зрения разработчиков попадают дополнительные аспекты. Очевидным является стремление выделять ортогональные аспекты, что позволяет выполнять их условно независимую и параллельную разработку в рамках проекта. 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 и другие аспекты.

Проработка аспекта в рамках проекта идет последовательно на всех стадиях и выражается в его специфицировании, проектировании, верификации, реализации и т.д. Другими словами, работа в рамках аспекта представляет собой мини-проект, который направлен на реализацию одного из свойств создаваемой системы. Причем это свойство может, как непосредственно обеспечивать требуемую функциональность целевой системы, так и быть направленным на достижение иных целей, например, на поддержание заданного процента повторного использования объектов некоторой категории в проекте.

Важнейшим фактором успеха проектирования ВсС является учет требований и ограничений, которые накладываются в явном и неявном виде ТЗ, а также вносятся самим коллективом проектировщиков, отражая его возможности, предпочтения, вторичные задачи и т.д. Традиционное на сегодня выделение только "чисто вычислительных" сторон (аспектов) ведения проекта в распространенных методиках и инструментальных средствах проектирования ВсС существенно ограничивает возможности совершенствования всех основных показателей качества проектов.

В дальнейшем мы будем рассматривать аспектную модель представления процесса проектирования и самой целевой системы в качестве основного инструмента совершенствования процесса проектирования ВсС. Распространенное мнение о том, что внутренние факторы проектного коллектива не должны оказывать влияние на результаты разработки, если они не отражены явным образом в ТЗ, не является конструктивным, так как на практике такие факторы всегда работают. Примером такого фактора может служить вектор коллектива в области освоения определенных семейств аппаратных или программных продуктов (микроконтроллеров, языковых средств и т.д.), что будет проявляться в создании целевых систем. Аспектная модель проектной инфраструктуры с вычислительными и невычислительными элементами позволяет легализовать многие важные процессы, сделать проект прозрачным и управляемым.

Основополагающими понятиями в рамках аспектной модели следует считать понятия архитектуры ВС, архитектурного агрегата, архитектурной модели целевой системы и архитектурной платформы проекта. Поясним кратко их наполнение и взаимосвязь.

Архитектурой ВС будем называть совокупность концептуальных аспектов ВС некоторого уровня детализации, адекватно отображающих проектируемую систему для данного уровня рассмотрения. Следует отметить, что конкретный перечень аспектов может меняться в зависимости от уровня рассмотрения, образуя ряд архитектурных представлений системы, адресуемых различным специалистам для использования. Удобно и полезно иметь возможность формировать представление системы архитектурного (концептуального, абстрактного) уровня, управляя приоритетами отображения различных аспектов ее организации. Однако такие частные архитектурные представления ВсС должны существовать лишь как вспомогательные наряду с полным архитектурным представлением, существующим у авторов системы в процессе ее создания.

Архитектурный агрегат выступает базовым элементом процесса проектирования ВсС, объединяя в себе различные точки зрения на целевую систему, агрегируя соответствующие элементы выбранных аспектов пространства проектирования.

Архитектурная модель целевой системы определяется как модель системы, выраженная в архитектурных агрегатах. В основе формирования множества

архитектурных агрегатов лежит понятие [вычислительного] механизма как некоторого технического, организационного или иного решения, не привязанного к конкретной реализации. Очевидно, что при создании ВсС, формирование полного архитектурного представления должно вестись от вычислительных механизмов (определяющих основную функциональность в рамках группы вычислительных аспектов) агрегированием их с необходимыми свойствами прочих аспектов. Успешность работы на таком уровне представления определяется как эффективностью содержательного наполнения архитектурных агрегатов, так и развитием формальных методов их описания и оперирования с ними.

2.1.3 Элементы архитектурного проектирования

2.1.3.1 Вычислительные механизмы

Выше мы отметили важность понятия вычислительный механизм в контексте процессов проектирования ВсС. Стремление расширить область абстрактного представления и проектирования ВсС предполагает замену и расширение списка объектов, которыми оперирует разработчик.

Для абстрактного стиля проектирования ВсС, использования понятий абстрактной элементной базы и расширенного толкования виртуальной машины, аспектного подхода к организации процесса проектирования и расширения значимости невычислительных ограничений и аспектов необходимо универсальное понятие – строительный элемент, не привязанное к реализации. Таким понятием предлагается считать *механизм (Мх)*, как *некоторое конкретное решение из любой области деятельности, фиксирующее и демонстрирующее для повторного использования принцип решения некоторой задачи или проблемы*.

Оставим на время анализ механизмов, принадлежащих к невычислительным аспектам, и сосредоточимся на определении и анализе вычислительного механизма. Мы определили понятие механизма как некоторого технического, организационного или иного решения, не привязанного к конкретной реализации. Будем называть *вычислительным механизмом (ВМх) техническое решение, направленное на организацию или непосредственное обеспечение вычислительного процесса*.

В предыдущем разделе мы отметили, что в рамках качественных переходов в организации вычислительной системы говорить об организации вычислительного процесса эффективно можно, начиная с регистрового уровня представления. Начиная с этого уровня, в логике работы системы появляются структуры данных и управления, которые реализуют алгоритмы в стиле, близком к программному представлению.

Конечно, такая трактовка нижней границы вычислительного механизма очень условна, однако мы воспользуемся именно таким определением. Тогда, все объекты, расположенные в иерархии представления вычислительного процесса выше невычислительного базиса, будем относить к вычислительным

механизмам. Так как верхняя граница этой категории объектов не фиксируется, выделение среди них дополнительных категорий элементов (например, виртуальных машин) должно выполняться в рамках специальных определений, по сути, представляя собой вариант переименования или макроподстановки.

Можно говорить о способах реализации вычислительного механизма, разделяя их на аппаратные и программные. Аппаратная реализация будет состоять в наличии физических структур, которые непосредственно соответствуют логике организации и функционирования вычислительного механизма и реализуют часть или весь вычислительный процесс. Программная реализация предполагает наличие виртуальных, то есть программно смоделированных, структур, которые также соответствуют логике организации и функционирования вычислительного механизма и реализуют часть или весь вычислительный процесс, и логически являются надстройкой или пристройкой к вычислительным структурам физического уровня.

Полезно ввести различные уровни определенности вычислительного механизма: аппаратную, программную, смешанную, виртуальную, абстрактную. Аппаратное определение предполагает вариант реализации механизма непосредственно только физическими структурами. Программное определение указывает на то, что механизм представлен только элементами, программно надстроенными над элементами физического уровня и функционирующими непосредственно так, как предписано описанием механизма. Смешанный вариант предполагает использование одновременно и аппаратных и программных элементов. Виртуальное определение предполагает подмену принципа работы механизма с сохранением его внешнего представления (функциональности, интерфейсов и др.). Абстрактное определение не фиксирует реализацию механизма.

Важным является вопрос выбора сложности механизма и детальности его описания. Проанализировав современные канонические вычислительные архитектуры и их реализации, можно разложить их на ограниченное число принципов – вычислительных механизмов, из которых они составлены. Результаты зафиксировать в виде библиотеки механизмов. Может возникнуть вопрос о том, что при таком подходе появится множество вариантов разбиения вычислительных систем на механизмы. Однако такая проблема является надуманной, так как, если двигаться в анализе сверху вниз, предполагая наличие исходного архитектурного описания, не создавая его методом "обратной раскрутки", результат будет однозначный или с вариантами по числу вариантов архитектур.

Описание ВсС на уровне вычислительных механизмов целесообразно рассматривать в качестве одной из форм представления проектируемой системы, которая отражает в первую очередь внутреннее устройство системы на уровне подсистем и контуров определенной функциональности. Такое представление дополняет, например, совокупность потоков данных и управления, раскрывая принципы устройства системы.

Представление вычислительных механизмов возможно, например, в виде структур данных и алгоритма, оперирующего с ними, в виде математического выражения, в виде регистровой модели и диаграммы ее функционирования т.д. Выбор формы представления и способа композиции вычислительных механизмов представляет собой сложную проблему, и будет обсуждаться далее.

Вычислительные и иные категории механизмов, широко используемые в ВсС, требуют классификации, системы метрик и подробного описания. Можно говорить о выделении подмножества базовых вычислительных механизмов, характерных для современного состояния области ВсС. В целом множество механизмов является открытым для расширения, в нем могут выделяться *универсальные* или *специализированные системы механизмов*, поддерживающие конкретные технологии синтеза целевых ВсС.

Важной является возможность поддержки аспектной модели проектирования ВсС за счет использования наряду с вычислительными механизмами механизмов иной направленности, например, в части энергосбережения, тестового обеспечения, конструктивного исполнения и другого. В зависимости от контекста, один и тот же механизм может быть включен в состав различных аспектных срезов ВсС (например, в поведенческий и инструментальный).

Отметим, что качественный переход в представлении проектируемой ВсС от уровня функционального описания (отвечает на вопрос, что надо сделать) к уровню исполняющей инфраструктуры (отвечает на вопрос, как надо сделать) и состоит в покрытии функций механизмами или, наоборот, в отображении функций на композицию механизмов.

Предлагается делить вычислительные механизмы по уровню сложности на 4 категории:

- атомарные (стандартные) вычислительные функциональные элементы (SN74, операторы языка…);
- составные (композитные вычислительные функциональные элементы);
- подсистемы организации вычислительного процесса (прерывания, кэш-память, очереди …);
- комплексный вычислительный процесс – *виртуальная вычислительная машина* (ВВМ) (*значимая вычислительная функциональность, самодостаточная для решения завершенной вычислительной задачи*).

2.1.3.2 Виртуальные вычислительные машины

В вычислительной технике крайне популярен термин *virtualization* (*виртуализация*) и, в частности, *virtual machine* (*виртуальная машина*) [51, 20]. Определенная проблема заключается в том, что значения понятий *“virtual”* и *“виртуальный”* заметно отличаются.

Понятие *“virtual”* в контексте вычислительной техники определяется как *“simulated”* (моделировать или воспроизводить поведение или свойства чего-либо) или *“imitated”* (следовать модели или шаблону). Для русского понятия *“виртуальный”* аналогичного специфического широко распространенного определения нет. В общем случае *“виртуальный”* трактуется как “условный, воображаемый, несуществующий, возможный, кажущийся; такой, который может или должен проявиться при определённых условиях”. Реже встречается конкретизация для информатики в виде “условный, кажущийся; не существующий в действительности, не имеющий физического воплощения”, что еще хуже. В результате понятие *“виртуальная машина”*, воспринимаемое дословно, заметно искажается относительно исходного *“virtual machine”*.

В сложившейся ситуации на помощь может придти понятие *“абстрактный”*. Абстрагированием называют мысленное отвлечение от каких-либо признаков, свойств, связей объекта или явления с целью выделения, обособления его существенных сторон. В процессе абстрагирования зачастую происходит игнорирование или сокрытие отдельных деталей исходного объекта или явления, в результате чего получается *“модель”*. Абстрактный объект как таковой не существует в природе, он существует только в виде абстракции или идеи (модели). Абстрактные объекты часто используются для снижения сложности описания вычислительных систем, а также для предоставления широкого спектра различных реализаций, за счет свободы реализаций в рамках несущественных для модели признаков или свойств.

Виртуализация – абстрактное (модельное) представление вычислительных ресурсов программно-аппаратной платформы.

Виртуальная машина – вычислитель (computer), полученный в результате виртуализации базовой вычислительной платформы.

Сама по себе модель того или иного вычислителя не подразумевает реализации как таковой, однако каждая виртуальная машина имеет конкретную реализацию, как способ виртуализации базовой платформы. При этом в общем случае не делается принципиальных различий между “программной” и “аппаратной” реализациями.

Ниже приводятся несколько трактовок термина *виртуальная вычислительная машина* (виртуальный вычислитель, ВВМ), которые раскрывают сложность и значимость данной абстракции в проектировании ВсС.

Под *виртуальной машиной можно понимать вычислитель, для которого определены правила поведения* (например, система команд, условия ввода команд, данных, получения результата, правила синхронизации процесса), *позволяющие однозначно описать алгоритм решения задачи*. Описание виртуальной машины демонстрирует лишь внешние свойства вычислителя и правила его использования, не касаясь его устройства.

ВВМ наиболее часто представляют собой сегодня последовательные интерпретаторы команд. Распространение термина *виртуальная машина* на

широкий класс так называемых *самодостаточных вычислительных механизмов* предоставляет ряд серьезных возможностей в повышении важности и удельного веса этапов абстрактного проектирования ВсС.

Разделение вычислительной системы на механизмы и вычислительные виртуальные машины – важная и сложная задача, которая плохо поддается формализации. Такое разделение ВсС возможно относительно выделенных (в качестве самостоятельных) вычислительных процессов. Степень инвариантности к аппаратному/программному способу реализации для абстрактных ВМх и ВВМ определяется в первую очередь сопряженными с ними МоС.

Итак, важнейшие абстракции для представления ВсС – вычислительные механизмы и виртуальные машины. Будем рассматривать виртуализацию в качестве базовой абстракции вычислительной техники, которую попробуем распространить на процесс высокоуровневого проектирования в целом.

- *Виртуализация* – абстрактное (модельное) представление вычислительных ресурсов программно-аппаратной платформы.
- *Виртуальная машина* – вычислитель (computer), полученный в результате виртуализации базовой вычислительной платформы.
- *Виртуальная вычислительная машина* (ВВМ) – техническое решение, реализующее семантику модели вычислений (МоС).
- *Вычислительный механизм* (ВМх) – техническое решение, реализующее субъект (актор) МоС. Другими словами вычислительный механизм – носитель модели вычислений.

2.1.3.3 Архитектура встраиваемых систем как иерархия виртуальных машин

В качестве инструмента представления ВсС предлагается использовать создание иерархии ВВМ. Известно, что использование виртуальных машин позволяет эффективно бороться с семантическим разрывом в вычислительной технике, существующим между средствами формального, абстрактного представления вычислительного процесса и возможностями аппаратных средств. Развитие вычислительной техники убедительно демонстрирует эффективность покрытия «семантического разрыва» посредством организации иерархии виртуальных машин в архитектуре ВС.

Принцип выделения ВВМ – мощный инструмент, позволяющий:

- структурировать вычислительный процесс и саму ВС;
- масштабировать проектные решения;
- обеспечивать программную совместимость и переносимость;
- абстрагироваться от способа реализации вычислителя;
- повышать надежность вычислительного процесса;

- управлять сложностью и другими характеристиками технологии программирования ВС для различных категорий пользователей;
- *совмещать («по вертикали») различные МоС, обеспечивать переходы от одной модели вычислений к другой*.

Распространенная трактовка виртуальной машины как программно-реализованной модели аппаратного вычислителя во многих случаях становится узкой и неэффективной. Это понятие должно быть расширено до обозначения вычислителя вообще без ограничения его функциональности и способа реализации. Таким образом, *виртуальный вычислитель* или *виртуальная машина могут определяться как абстрактное вычислительное устройство, обладающее значимой и завершенной функциональностью в контексте организации вычислительного процесса*. Поясним этот тезис.

Вычислительный процесс всегда рассматривается как существующий в пространстве и времени. В силу различной его сложности и разнородности его частей он может быть разделен на специфические функциональные блоки, на уровни вложенности, на параллельные ветви. *Можно говорить о нижней границе сложности и функциональности процессов, которые являются самодостаточными (полноценными) вычислительными процессами*. При рассмотрении системы на уровне ниже этой границы теряется видение вычислительного процесса, мы переходим на качественно иной уровень представления и понимания работы системы. Выделение таких качественных переходов в представлении ВсС является важнейшим моментом. Качественно различающиеся уровни представления ВсС позволяют разделить области действия и сферы ответственности специалистов различного профиля.

Важнейшими качественными уровнями, которые традиционно определяются для ВС, являются (снизу вверх) транзисторный, вентильный, регистровый и системный уровни [46]. Сегодня системный уровень из традиционной классификации, в свою очередь, нуждается в делении. Абстрактное проектирование захватывает регистровый и системный уровни, и нуждается в определении базовых категорий элементов, с которыми разработчик будет на этих уровнях работать.

В данном разделе мы говорим о двух основных категориях таких "кирпичиков" и "блоков", не привязывая их непосредственно к способу реализации. Это вычислительные механизмы и виртуальные машины.

Развитие вычислительной техники убедительно демонстрирует эффективность покрытия семантического разрыва посредством организации иерархии виртуальных машин в архитектуре вычислительной системы.

Поясняя приведенное выше определение виртуальной машины, рассмотрим ряд примеров. Пускай существует ПК с ОС, на котором может исполняться программа, написанная на языке BASIC. Такая программа может транслироваться разными способами и в конструкции различного уровня для нашего ПК. Транслятор может быть компилирующего или интерпретирующего

типа. Уровень выходных конструкций трансляции может соответствовать командам аппаратного процессора ПК, композиции команд процессора и сопроцессоров (математического, графического и др.), композиции команд процессоров и вызовов ОС. Если же наш ПК оснащен процессором с открытым уровнем микропрограммирования, то можно добавить и вызовы микроподпрограмм или микропрограммные вставки (макросы микропрограммного уровня).

Традиционно из перечисленных вариантов выполнения BASIC-программы с виртуальной машиной будет ассоциироваться запущенный на ПК программный интерпретатор. Он и будет рассматриваться в качестве виртуальной машины. В этом случае остальные варианты исполнения нашей BASIC-программы, то есть организации вычислительного процесса, будут распадаться на безымянные варианты по их количеству.

Распространив термин *"виртуальная машина"* на любой вычислитель, который обеспечивает выполнение некоторого вычислительного процесса (исполнения программы, решения прикладной или системной задачи и т.д.), мы получим удобное и универсальное определение, пригодное для обозначения устройств различной организации и степени абстракции, самодостаточных для решения вычислительной задачи. Тогда в нашем примере все перечисленные варианты исполнения BASIC-программы будут реализованы виртуальными машинами различной организации. Важно заметить, что вариант с компилятором может рассматриваться как композиция работы двух виртуальных машин или как одна сложная машина с явно выделенными фазами предобработки и исполнения.

Вопросы представления виртуальных машин в предлагаемой трактовке имеют принципиальный характер. Будем придерживаться следующего деления. *Описание виртуальной машины как объекта с известными функциональностью, интерфейсами и свойствами, но без раскрытия внутренней организации, будем называть собственно представлением виртуальной машины*. Такое *описание, дополненное внутренней организацией, будем называть архитектурным описанием (или архитектурой) виртуальной машины*. В этом смысле мы стремимся сохранить принятые в вычислительной технике правила для процессоров, вычислительных машин, операционных систем, коммуникационных протоколов и других значимых объектов организации вычислительного процесса.

Рассмотрим вычислительную систему как сложную иерархию виртуальных машин. Тогда мы можем говорить о привычной, в плане распределенности вычислительного процесса, иерархии подчинения уровней и композиции в пределах уровня, а также о вложенности, которая направлена на изменение или повышение уровня обрабатывающих конструкций. Представление архитектуры ВсС как иерархии виртуальных машин должно трактоваться как средство декомпозиции, структурирования, унификации, распараллеливания элементов самой системы и процесса ее создания.

Вопросы стратегии в представлении ВсС как композиции виртуальных машин требуют исследования. Действительно, можно управлять числом, вложенностью, однородностью, сложностью и другими параметрами виртуальных машин в рамках представления архитектуры ВсС.

Из предложенного выше обобщенного определения виртуальной машины следует, что в пределе виртуальная машина может совпадать с физической реализацией оговоренной функциональности. Важно уметь определять в качестве виртуальных машин в проекте не полуфабрикаты вычислительного процесса (каковыми выступают вычислительные механизмы), а устройства, самостоятельно обеспечивающие выполнение вычислительного процесса некоторой функциональности. Действительно, соотношение понятий *вычислительный механизм* и *виртуальная машина* может определяться как *включение или подчинение* (виртуальная машина является частным случаем вычислительного механизма) или как *перечисление* (две категории абстракций в пространстве проектирования). Второй вариант привлекательнее, однако, он требует более жестких критериев определения этих категорий элементов. Кроме того, необходимо исследование, насколько выделение двух категорий абстракций скажется на сужении пространства поиска решений.

Если придерживаться второго варианта трактовки виртуальной машины, то проявляется важная и полезная связь с понятием *вычислительной платформы, как зафиксированного для повторного использования набора спецификаций*. Во многих случаях возможен и полезен прямой переход от виртуальной машины к платформе, в результате которого спецификация машины становится платформой. В этом случае вычислительная платформа становится частным случаем виртуальной машины.

Рассматривая архитектуру ВсС через призму целевой функциональности, можно говорить о различных уровнях детализации представления, о различной направленности представления (для категорий специалистов), о различной степени оптимальности реализации в соотношении с технологической сложностью. Более высокоуровневым, доступным для восприятия (прозрачным) и простым технологически представляется способ подачи архитектуры ВсС как простой иерархии виртуальных машин. Такое представление с последующей прямой реализацией удобно и понятно в контексте ряда параллельно работающих команд исполнителей и при условии действия проектной модели "неограниченных вычислительных ресурсов". Требование минимизации ресурсов вступает в противоречие с подобной моделью, заставляя сокращать число уровней иерархии и переходить к так называемым "плоским" моделям реализации.

Иерархическое представление ВсС в терминах виртуальных машин является очень важным и мощным инструментом проектирования. Важнейшее свойство такого представления состоит в возможности достигать сокращения трудоемкости проектирования и повышения степени повторного использования при условии выполнения других ресурсных ограничений проекта. Кроме того,

такое представление полезно при обучении и формировании специалистов в области ВсС, так как оно позволяет демонстрировать воспринимаемый человеком образ системы, заставляет оперировать в явном виде различными вычислительными моделями, согласовывать их друг с другом.

2.1.3.4 Платформы в проектировании ВсС

Понятие платформы в вычислительной технике на интуитивном уровне последние 10 лет применяется исключительно широко. Например, в рамках одного из ведущих направлений проектирования ВсС и СнК «Platform-Based Design» (PBD) вычислительная платформа определяется как множество проектов, удовлетворяющее некоторому общему условию [66].

Важнейшее свойство вычислительной платформы, как зафиксированного для повторного использования набора спецификаций, – *возможность предоставлять заданный уровень абстрагирования от конкретики реализации*. В системе абстракций *вычислительная платформа (ВПл)* рассматривается как единство «внешнего» и «внутреннего» представления функционально-завершенного и функционально-значимого объекта в составе ВсС. *Платформа в рамках проекта ВсС – техническое решение, фиксируемое в проекте для повторного использования*. ВПл выступает основным инструментом повторного использования на архитектурном уровне.

Сегодняшняя практика состоит в проектировании на базе готовых вычислительных платформ (аппаратных, программных, инструментальных, конструктивных …), которые выбираются, к сожалению, «по вторичным признакам». Рассматриваемая в пособии методология проектирования позволяет усилить / добавить грамотный выбор или создание совокупности вычислительных платформ проекта.

В рамках аспектной модели центральным понятием этапа архитектурного проектирования выступает *архитектурная платформа*. Это понятие фиксирует инфраструктуру проекта, в которой будет осуществляться генерация, конкретизация, верификация архитектурной модели целевой системы, а также формирование спецификаций для этапа реализации. Таким образом, архитектурная платформа может рассматриваться как объединение следующих элементов проектирования:

- аспектное пространство процесса проектирования (перечень аспектов проектирования);
- модель (модели) вычислений;
- внешние факторы, задающие допустимые соотношения между отдельными аспектами (критерии проектирования);
- перечень зафиксированных шаблонов повторного использования;
- элементная база (в расширенной трактовке).

Фактически, архитектурная платформа выступает в качестве глобального набора решений, ограничений и приоритетов, регламентирующих проектную деятельность как на этапе проектирования, так в значительной степени и на этапе реализации. Она фиксирует концептуальные решения проекта.

Важнейшей задачей современного этапа развития методик и средств проектирования ВсС является повышение степени повторного использования результатов, получаемых на ранних (высокоуровневых) этапах проектного процесса. В качестве инструмента, обеспечивающего повторное использование концептуальных решений в области ВсС в рамках аспектной модели проектирования, предлагается использовать понятие *архитектурной платформы*.

Как было отмечено в главе 1, вопрос создания эффективной классификации объектов в сегменте ВсС остается открытым. Понятие платформы представляется исключительно удобным классификационным параметром в данной области проектирования. Можно строить дерево классификаций ВсС на основе понятия *проектной платформы (вычислительной или иной платформы, взятой за базу в рамках комплексного или частного проекта ВсС)*, что позволяет акцентироваться на общих свойствах системы или ее части во всем пространстве свойств, технологий и реализаций ВсС. Пример классификации на базе проектной платформы представлен ниже:

- Вычислительные платформы (процессоры, ОС, МоС и т.д.):
  - Платформы промышленных ПК и ПЛК;
  - Полуфабрикаты от мультимедиа-индустрии;
  - Микроконтроллеры и DSP;
  - ПЛИС, ПСнК;
  - ASIC, ASSP, ASIP, SOC, NOC;
  - «Свободная» кремниевая компиляция.
- Платформы системного ПО (ОС), сетевые, интерфейсные, конструктивные, инструментальные и другие.

2.1.4 Проектное пространство ВсС и фазы организации вычислительного процесса «Design-Time / Run-Time»

2.1.4.1 Единое проектное пространство технических решений встраиваемых систем

Уже на этапе начального формирования архитектуры создаваемой ВсС проектировщик сталкивается с проблемой анализа множества возможных вариантов организации, как самой целевой системы, так и инфраструктуры проекта (например, в части инструментального аспекта, стиля и способов

повторного использования элементов, тестирования, документирования и т.д.). Последующее развитие проекта вглубь также предполагает решение значительного числа задач выбора технических решений при создании частных архитектур подсистем, платформ, при проработке аспектов. В сегодняшнем арсенале методов и средств разработчика ВсС инструменты решения такой проблемы сильно ограничены и предлагают либо шаблонные решения в рамках конкретных Framework – систем, либо свободное неформализованное проектирование.

Одним из направлений в решении данной задачи следует считать формализацию проектного пространства технических решений архитектурного уровня ВсС (пространство поиска технических решений в рамках проекта создания ВсС). Такое *n-мерное пространство можно представлять в виде совокупности координат изменения (параметризации) ключевых принципов (свойств) архитектуры*. Примерами осей координат, существенных для частных и общих архитектур ВсС, выступают соотношение HW/SW реализации, степень реконфигурируемости, распределенности, параллелизма, on/off-board размещения [инструментальных] функций, on/off-line фазы вычислительного процесса. Элементами этого пространства выступают вычислительные механизмы, которые, как было отмечено выше, составляют основу архитектурных агрегатов аспектной модели системы и проекта.

Ближайшими шагами по формализации этого пространства являются выделение и классификация параметров архитектуры, и определение (фиксация) наиболее значимых точек этого пространства (де-факто существующих технических решений, доказавших свою эффективность) с разнесением их по координатам. Вторым очевидным шагом должен быть анализ заполнения пространства координат известными решениями и попытка синтеза решений, заполняющих выявленные пустоты. Третьим шагом может быть систематическое описание (с единых позиций) свойств и характеристик решений по каждой из осей пространства.

Конечно, укладывать такие сложные объекты, как подсистемы вычислительных архитектур, в многомерное пространство проектных решений, возможно только при условии высокой степени абстрагирования. Разработчику необходимо приложить значительные усилия, направленные на представление его проектных решений в терминах организации вычислительного процесса. Дополнительную проблему может составлять необходимость отражать содержательную сторону функционирования готовых сторонних компонентов, которые разработчик планирует использовать в системе или в проекте, так как необходимая информация по организации таких компонентов может быть недоступна по вторичным причинам.

Важной проблемой является соотношение координат проектного пространства технических решений с аспектным пространством проекта. Если аспектная модель направлена на развитие и отслеживание состояния локализованных частных проблем проектирования ВсС на всем протяжении

проекта, то проектное пространство технических решений выступает своеобразным пулом известных и потенциально-существующих решений, которые разработчик использует на конкретных фазах и шагах проекта. Можно считать, что эти пространства ортогональны: аспектное пространство расположено вдоль оси потока проектирования, а пространство технических решений – поперек, пересекая ось проектирования многократно по мере необходимости поиска решения при проработке проекта вглубь.

Устанавливающим связь понятием между этими пространствами в проекте, по существу, является архитектурная платформа. Она фиксирует шаблоны реализации и элементную базу, выбранные разработчиком на основе анализа всего единого пространства технических решений архитектурного уровня. Анализ единого проектного пространства технических решений позволяет в значительной мере оценить степень сбалансированности полученной архитектуры и существующие резервы альтернативных технических решений.

2.1.4.2 Координаты проектного пространства ВсС

Важнейшим понятием проектного пространства ВсС следует считать *систему проектных координат*, число которых не ограничено и явный учет которых разработчиком выполняется на основе вычислительных и невычислительных ограничений проекта, с учетом приоритетов этих ограничений. *Система проектных координат включает как вычислительные* (непосредственно связанные с целевой функциональностью), *так и невычислительные плоскости*. Примерами *вычислительных осей координат* могут служить соотношение HW/SW реализации, степени реконфигурируемости, распределенности, параллелизма, on/off-board размещение различных категорий функций (в первую очередь, инструментальных), реализация design/run-time (on/off-line) фаз вычислительного процесса и другие. *Невычислительные оси* могут представлять технические решения по конструкции узлов, способам документирования частей проекта, системе энергопитания, вопросам обеспечения тепловых режимов, электромагнитной совместимости, технологии производства, метрологии и другим.

Явное выделение координат направлено на использование разработчиком потенциала существующих степеней свободы проекта, что в значительной степени базируется на важнейшем тезисе – свободе в выборе реализации. Выстраивание и заполнение координат в техническом мировоззрении проектировщика само по себе является мощным инструментом формализации знаний и опыта, способствует более глубокому анализу возможных решений, позволяет эффективнее выполнять декомпозицию задачи.

Пространство проектных координат в методиках и технологиях проектирования ВсС может занимать различное место и иметь разное значение. Предлагаемая в работе аспектная модель проектирования в значительной мере основывается на данном пространстве, трактуя его как пространство возможных технических решений, используемое последовательно на разных

фазах детализации проекта ВсС. При этом координаты технических решений могут явно включаться в орбиту процесса проектирования в соответствии с важностью того или иного аспекта разработки. Решения могут выбираться согласованно в пространстве подмножества координат, как для целевой, так и для обеспечивающей части проекта.

2.1.4.3 Расширение задачи проектирования ВсС до задачи непосредственной организации целевого вычислительного процесса

На эффективность поиска архитектурного решения ВсС определяющим образом влияет взгляд на организацию вычислительного процесса в целом.

Важнейшим тезисом, направленным на расширение и унификацию проектного пространства создания ВсС следует считать расширенную постановку задачи проектирования ВсС как *задачи организации конкретного вычислительного процесса в соответствии с ТЗ, которая должна быть решена с помощью конкретных проектных мероприятий*. В процессе таких мероприятий должны быть реализованы технические средства (инструментальные и целевые), которые в рамках *подготовительных и исполнительных этапов* будут решать поставленную прикладную задачу. Важно отметить, что при такой постановке задачи понятие прикладного (целевого) вычислительного процесса рассматривается как единое целое, захватывая как этапы исполнения (традиционное решение задачи экземпляром вычислительной системы при эксплуатации), так и этапы подготовки (например, компиляция, загрузка в ПЗУ, формирование таблиц коэффициентов, традиционно рассматриваемые изолированно в рамках этапов проектирования, производства, конфигурирования системы).

Взгляд на проектирование ВсС как на единый процесс организации целевых вычислений позволяет в едином ключе анализировать варианты организации различных фаз вычислительного процесса. Данный тезис предлагает разработчику нетрадиционную модель проектируемой системы, обладающую значительно более высокой общностью, высоким потенциалом оптимизации решений, возможностью абстрагироваться от конкретных реализаций, навязанных традициями или ситуацией без должного анализа.

Действительно, во многом шаблонный характер сегодняшнего проектирования ВС использует в большинстве своем эмпирические решения. Цепочки реализации вычислительного процесса включают такие элементы, как компиляторы, интерпретаторы, виртуальные машины, аппаратные программируемые процессоры, специальные функциональные аппаратные блоки и многое другое. Разработчик, часто не осознанно, распределяет элементы вычислительного процесса внутри инструментальной (Design-Time) и исполнительской (Run-Time) фаз проекта. Анализ и осознанный выбор решений в обеих фазах существования ВсС позволяет резко повысить качество проектирования.

Рассматриваемый взгляд на процесс проектирования требует использования понятий, представляющих ВсС именно с позиций организации вычислительного процесса. Таким базовым понятием должен выступать вычислительный механизм, как комбинированное (в стиле понимания вычислительной архитектуры) структурно-функциональное представление части вычислительного процесса в комплексе со средствами, его обеспечивающими.

В проектировании ВсС известна высокая значимость и сложность инструментального обеспечения. Выбор, как архитектуры создаваемой системы, так и технологии ее проектирования и отладки теснейшим образом переплетаются в деятельности разработчика, оказывая сильное взаимное влияние. Важным является вопрос разделения инструментальных компонент на резидентные и внешние составляющие. Также большое значение имеет выделение элементов подготовительной и исполнительной частей вычислительного процесса. Все это непосредственно влияет на архитектуру и реализацию целевой вычислительной системы. Объединение потоков проектирования целевого продукта и его инструментального обеспечения на ранней стадии работ позволяет значительно повысить качество создаваемой ВсС и улучшить все основные характеристики процесса проектирования.

Таким образом, *процесс проектирования ВсС* как объекта, решающего конечную задачу, предлагается рассматривать как *организацию вычислительного процесса в пространстве и времени в оговоренных техническим заданием ограничениях*.

2.1.4.4 Принцип актуализации вычислительного процесса ВсС

Данный взгляд позволяет сформулировать *принцип актуализации вычислительного процесса* как совокупность преобразований исходного представления целевого алгоритма в конечный (по ТЗ) вид. Это, в свою очередь, позволяет дополнить понятие «решение целевой задачи в фазе Run-Time» совокупностью «вычислительных» подготовительных операций этапа «Design-Time». Предлагается искать архитектурное решение в едином пространстве *Design-Time и Run-Time фаз существования ВсС*, устраняя априори часто не обоснованно навязанные ключевые для проекта решения.

Следует отметить, что данная модель пространства потенциальных решений эффективна в первую очередь для ВсС значительной сложности.

Принцип актуализации вычислительного процесса ВсС положен в основу модели актуализации вычислительного процесса [8]. Более подробно организация и использование процесса актуализации в проектировании ВсС будет рассмотрено в части 2 учебного пособия.

Платформы ВсС с программируемой архитектурой

2.1.4.5 Принцип актуализации вычислительного процесса позволил выделить перспективный *класс платформ ВсС с программируемой архитектурой* как прямой результат унификации проектирования по осям «Design/Run Time» и «HW/SW реализация» [16, 15]. Основными свойствами таких платформ являются:

- возможности широкого конфигурирования/программирования пользователем для ВсС со специализированными архитектурами;
- САПР-ориентированность (необходимость инструментальной поддержки);
- масштабирование реализаций;
- необходимость использования А-платформ в качестве шаблонов.

Примерами таких платформ является система MiniLab.

2.2 Проектирование архитектуры ВсС

2.2.1 Архитектурная платформа и критерии проектирования архитектуры

В рамках архитектурного проектирования возникают вопросы: как формируется элементная база системы, как разработчик выбирает ведущие аспекты проекта и откуда он берет внешние ограничения проекта (критерии проектирования). Очевидно, что разработка каждого проекта “с нуля” будет крайне неэффективна и практически невозможна. Различные разработчики используют некоторые готовые элементы, компоненты, решения и т.д. Откуда они берутся?

Ответом на эти вопросы является понятие платформы или *архитектурной платформы*. Можно пытаться рассматривать архитектурную платформу как некоторое обобщение виртуальной машины, зафиксированной для создания очередной функциональной надстройки. Однако это не вполне удачно, так как в этом случае упущены многие существующие платформы, которые определяют не только функциональность. Как было отмечено выше, *архитектурная платформа* является объединением таких элементов процесса проектирования как:

- аспектное пространство процесса проектирования (перечень аспектов проектирования);
- модель (модели) вычислений;
- внешние факторы, задающие допустимые соотношения между отдельными аспектами (критерии проектирования);
- перечень зафиксированных шаблонов повторного использования;
- элементная база.

В общем случае при выборе той или иной А-платформы разработчик сразу же оказывается в определенных рамках, диктуемых выбранной платформой. А-платформа определяет состав и взаимную важность аспектов процесса проектирования. Она диктует определенные характеристики элементной базы, определяя тем самым допустимые соотношения между отдельными аспектами. А-платформа, в частности, являясь реализуемой или виртуальной А-моделью, предлагает или подразумевает некоторые способы реализации шаблонов повторного использования. А-платформа определяет одну или несколько МоС, которые она “воплотит” при реализации. Соотношение этих параметров и дает разработчику возможность выбирать те или иные А-платформы для решения конкретной задачи.

Если вернуться к рассмотрению поведенческого аспекта проектирования, то А-платформа играет тут огромную роль, определяя (фиксируя) МоС. Поведенческий аспект очень важный при проектировании ВсС, но далеко не единственный, поэтому могут сложиться ситуации, когда выбранная А-платформа удовлетворяет разработчика практически по всем пунктам, но

предоставляемая ею МоС, абсолютно не подходит для решения задачи. Именно из-за распространенности таких случаев для А-платформы определяется такая характеристика, как реконфигурируемость.

Реконфигурируемость А-платформы определяется как способность к изменению “воплощаемой” при реализации МоС. Конечно же, наличие такого свойства “разумно усложняет жизнь разработчику”, так как требует дополнительных усилий при работе с А-платформой, но зачастую дает ощутимый выигрыш. В общем случае реконфигурируемость может иметь разный характер (или способ реализации):

- на этапе проектирования, когда конфигурированию подлежат те или иные методики;
- на этапе реализации, когда устанавливаются те или иные параметры компиляции или используются различные компоненты элементной базы;
- на этапе исполнения, когда система позволяет гибко адаптировать функциональность.

Реконфигурируемость А-платформы может быть обеспечена введением широкого перечня электронных компонентов элементной базы, различных по функциональным характеристикам и физическим параметрам, но совпадающим по выводам, использованием языков высокого уровня и условной компиляции для описания шаблонов реализации, введением в состав платформы элементов программируемой логики и многим другим.

В некоторых случаях, когда А-платформа не имеет внутренних возможностей изменить МоС, внесением таких изменений должен заниматься сам разработчик. Обычно для этого делается некоторая реализация выбранной А-платформы, после чего она достраивается “сверху” до уровня “воплощения” желаемой МоС, адекватной решаемой задаче. При дальнейшем проектировании созданная надстройка вместе с базовой А-платформой рассматриваются как новая платформа, удовлетворяющая критериям выбора. Надстройка над А-платформой, созданная с целью изменить или скорректировать МоС А-платформы, называется *операционной средой*. Особо следует отметить, что МоС А-платформы может нуждаться в изменении не только вследствие несовпадения с выбранной согласно специфике задачи, но и из-за ошибок элементов самой А-платформы. Таким образом, именно операционная среда призвана исправлять все обнаруженные ошибки БИС и ПО, входящих в базовую вычислительную платформу.

В качестве общего критерия выбора той или иной модели вычислений в рамках аспектного проектирования рассматривается отношение трудоемкости решения задачи в выбранной модели вычислений к трудоемкости реализации операционной среды архитектурной платформы.

Обобщенная схема процесса проектирования ВсС с использованием А-платформы приведена на рис. 2.1.

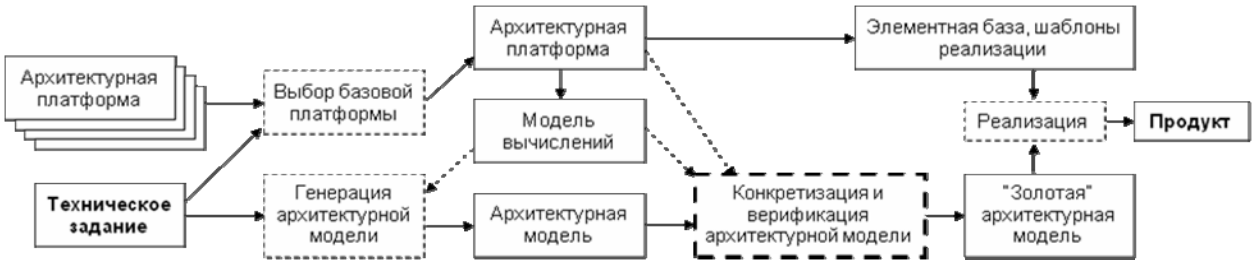


Рис. 2.1. Место и значение архитектурной платформы в процессе проектирования ВС

На основании опыта разработок, используя экспертные оценки, на начальных этапах проектирования происходит выбор базовой А-платформы. В дальнейшем выбранная А-платформа будет играть важнейшую роль практически на всех этапах проектирования целевой системы. Опираясь на воплощаемую А-платформой МоС, учитывая требования ТЗ, разработчик создает А-модель системы. Критерии проектирования, определяемые А-платформой, используются при конкретизации модели, в результате чего формируется “золотая” модель системы. Реализация “золотой” модели опирается на элементную базу и шаблоны повторного использования, предлагаемые А-платформой. Из всего сказанного можно сделать вывод, что удачно сформированная А-платформа является мощным инструментом повторного использования концептуальных решений в процессе проектирования ВсС.

В рамках архитектурного проектирования необходимо сформулировать критерии, которые позволят формальным образом оценивать качество принимаемых решений и эффективность проектирования в целом.

Для архитектурного проектирования критерием эффективности может быть себестоимость конечного изделия, срок разработки, равномерность загрузки коллектива разработчиков, качество и надежность проектируемой системы, степень повторного использования наработок и т.д. В качестве критерия может выступать требование оптимизации частных “вычислительных” характеристик системы при зафиксированных остальных параметрах системы. В каждой конкретной ситуации (для каждого конкретного проекта) ведущую роль играет конкретной критерий (система критериев). В общем случае необходимо проводить многокритериальную оптимизацию решения задачи проектирования ВсС.

2.2.1.1 Критерии проектирования архитектуры ВсС

Из теории многокритериальной оптимизации [5, 18, 21, 28] следует, что [19]:

- существуют эффективные решения, если множество значений допустимых параметров замкнуто и критерии качества – непрерывные функции от параметров;

- решение эффективно, если среди сравнимых с ним нет безусловно лучшего, чем оно;
- два решения сравнимы, если все критерии одного не хуже (или не лучше) соответствующих другого;
- из двух сравнимых решений, безусловно, лучше то, у которого хотя бы один критерий строго лучше соответствующего критерия другого;
- только для эффективных решений между альтернативными критериями имеется такая взаимосвязь, при которой улучшение одного критерия влечет ухудшение хотя бы одного из других;
- для получения такой взаимозависимости необходимо определить предельное улучшение любого критерия при фиксированных значениях остальных, которые, в свою очередь, меняются в заданных пределах;
- какой бы критерий не выбрать в качестве улучшаемого при заданном значении остальных, результатом оптимизации будет одна и та же взаимосвязь между критериями.

Математически задача оптимального проектирования ВсС означает нахождение условного экстремума (максимума или минимума) некоторого функционала, который рассматривается как критерий оптимальности (целевая функция) проведения процесса архитектурного проектирования. Как было сказано выше, критерии проектирования определяются А-платформой и их частный вид может изменяться от проекта к проекту. Общий вид критерия архитектурного проектирования приведен в формуле (1.1):

$$C = \{c : \bigcup_i f_i \to R, N, Z\}, \qquad (1.1)$$

где C – множество критериев архитектурного проектирования,

c – частный критерий,

i – номер аспекта проектирования (не больше чем аспектная полнота),

f_i – множество характеристических функций соответствующего аспекта [см. формулу (2.5) и (2.6)], а R, N и Z множества действительных, натуральных и целых чисел соответственно.

Невозможно перечислить абсолютно все критерии аспектного проектирования, ввиду их большого количества и зависимости от условий проведения процесса проектирования. Кроме того, аспектная модель процесса проектирования в настоящий момент формализована не в достаточной мере, чтобы предложить устоявшийся шаблон проектирования для различных классов задач и целей проектирования. В связи с этим вместо указания конкретных функционалов можно перечислить широко распространенные частные виды критериев проектирования:

$$\frac{\chi_i}{\chi_j}, \qquad (1.2)$$

$$\sum_i \alpha_i \chi_i \,, \tag{1.3}$$

$$\sum_i \alpha_i \cdot (\chi_i - \overline{\chi}_i)^2 \,, \tag{1.4}$$

где χ_i – характеристическая функция некоторого аспекта,

α_i – эмпирический весовой коэффициент.

Критерий-отношение [см. формулу (1.2)], аддитивный критерий [см. формулу (1.3)] и критерий рассогласование [см. формулу (1.4)] могут быть использованы при оценке эффективности проекта в целом и при решении частных задач.

2.2.2 Шаблоны процессов проектирования ВсС

2.2.2.1 Процесс проектирования типовой ВсС

Анализ типовых сценариев высокоуровневого проектирования ВсС демонстрирует необходимость формирования для них ряда шаблонов процесса проектирования. Шаблоны формируются в терминах разработанной системы абстракций и с учетом аспектной модели проектирования по составному априорному критерию сложности проектирования, учитывающему масштаб (сложность) задачи и допустимую «глубину погружения», которая определяет доступные для проектирования уровни иерархии ВсС.

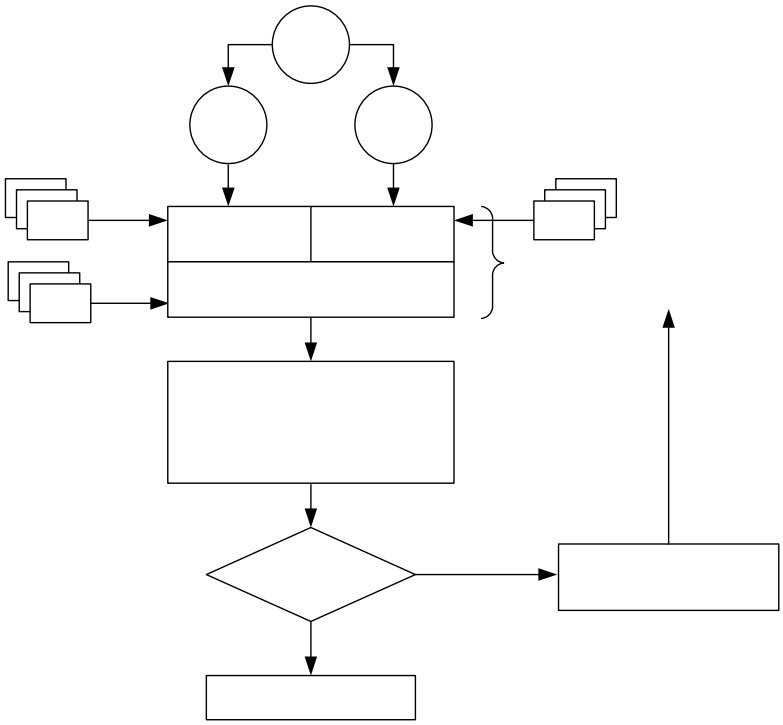


Рис. 2.2. Шаблон процесса проектирования ВсС на основе традиционной для ВС схемы проектирования (ЯОФ/ТЯОФ – языки описания функциональности/трансляторы ЯОФ; ВПл – вычислительная платформа)

На рис. 2.2 представлен шаблон процесса проектирования ВсС на основе традиционной для ВС схемы проектирования, который имеет следующие особенности:

- этот процесс инвариантен к уровню реализации. Он может сводиться к любому уровню абстракции, требуя уточнения блоков оценки функциональных требований (ФТ) и нефункциональных требований (НФТ);
- ТЗ уже подготовлено для реализации ВС, то есть для проектируемой системы в целом уже решены вопросы выделения функций «навешиваемых» на вычислительную систему (это во многих случаях неэффективно);
- процесс отражает в явном виде каноническую архитектуру ВС и приоритеты как между ФТ и НФТ, так и в рамках процесса реализации ФТ (эффективность потенциально низкая).

Перечислим основные недостатки процесса:

- присутствует жесткая привязка ВПл – ЯОФ, следовательно, уровень ВПл будет стремиться вверх, минимизируя свободу выбора для архитектуры целевой ВсС в целом. Это является предпосылками избыточности, неэффективности реализации;
- много задач, которые при такой гранулярности ВПл окажутся практически не поддержанными в плане методики проектирования (сложно выполнить адекватный выбор исходной архитектуры, которая должна быть многоуровневой гетерогенной);
- в процессе доминирует функциональность (ФТ), что ведет к серьезной (катастрофической во многих проектах ВсС) потере эффективности проектирования за счет обнаружения проблем на поздних этапах проекта.

Суммарно процесс по данному шаблону характеризуется низкой сбалансированностью проектирования сложных, критичных к ресурсам, ВсС:

- по времени проектирования – в единицы раз;
- по избыточности реализации – до десятков раз.

2.2.2.2 Перспективный процесс проектирования сложных ВсС

Процесс проектирования сложных ВсС должен предполагать иерархическое представление ВПл, ЯОФ, ТЯОФ и прикладной надстройки в единой системе абстракций, которое допускает представление, контроль и сопровождение аспектов проекта (функциональных и нефункциональных), унифицирующее проектирование всех компонент ВсС (по возможности).

Центральной идеей выступает последовательное уточнение/проработка целевой системы через иерархию проектов с понижением степени абстракции.

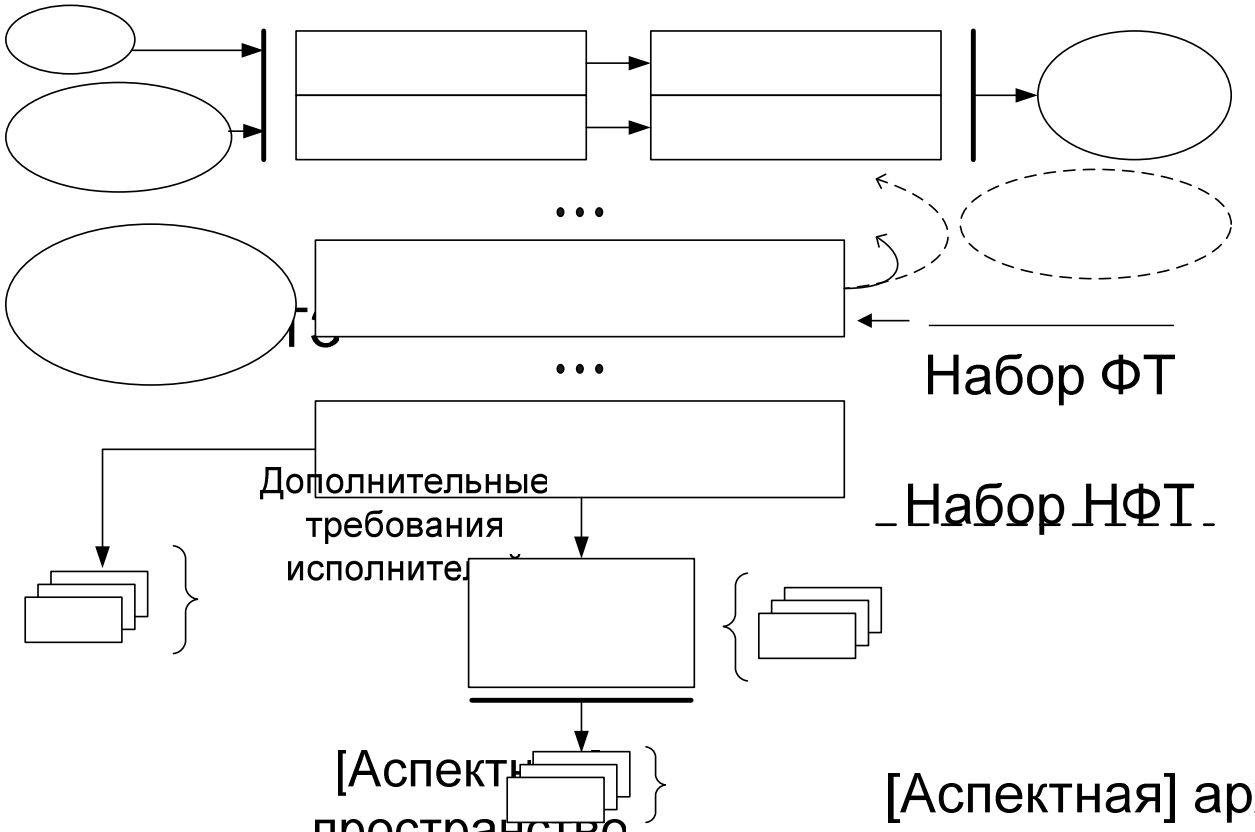


Рис. 2.3. Шаблон процесса проектирования ВсС на базе аспектной модели и композиции ВПл (ФТ, НФТ – функциональные/нефункциональные требования; ЯОФ/ТЯОФ – языки описания функциональности/трансляторы ЯОФ; ВПл – вычислительная платформа)

На рис. 2.3 представлен шаблон процесса проектирования ВсС на базе аспектной модели и композиции ВПл. Его преимуществами являются:

- параллельная сбалансированная работа с ФТ и НФТ;
- генерация архитектуры с позиций предложенной системы абстракций;
- объединение фаз проектирования и исполнения вычислительного процесса в единое пространство технических решений;
- отложенная фаза аппаратно-программного разделения.

Недостатками процесса на сегодня следует считать:

- то, что процесс не знаком большинству разработчиков;
- процесс предполагает наличие компетенций в области вычислительных абстракций.

Процесс эффективен не только для проектирования ВсС, он может быть расширен до уровня CPS (КФС).

В табл. 2.1 приведена сравнительная характеристика шаблонов проектирования ВсС.

Таблица 2.1. Характеристики шаблонов проектирования

Шаблон на основе традиционной для ВС схемы проектирования	**Шаблон на базе аспектной модели и композиции ВПл**
Достоинства	
• Инвариантен к уровню реализации. • Процесс интуитивно понятен сегодняшнему «массовому» разработчику ВсС.	• Параллельная сбалансированная работа с ФТ и НФТ. • Генерация архитектуры с позиций предложенной системы абстракций. • Объединение фаз проектирования и исполнения вычислительного процесса в единое пространство технических решений. • Отложенная фаза аппаратно-программного разделения.
Недостатки	
• ТЗ уже подготовлено для реализации ВС, т.е. для проектируемой системы в целом уже решены вопросы выделения функций, «навешиваемых» на ВС. • Процесс отражает в явном виде каноническую архитектуру ВС и приоритеты как между ФТ и НФТ, так и в рамках процесса реализации ФТ. • Так как присутствует жесткая привязка ВПл – ЯОФ, уровень ВПл будет стремиться вверх, минимизируя свободу выбора для архитектуры целевой ВсС в целом. • Много задач, которые при такой гранулярности ВПл окажутся практически не поддержанными в плане методики проектирования (сложно выполнить адекватный выбор исходной архитектуры, которая должна быть многоуровневой и гетерогенной). • Доминирующая функциональность (ФТ) ведет к обнаружению проблем на поздних этапах проекта.	• Процесс не знаком большинству разработчиков • Процесс предполагает наличие компетенций в области вычислительных абстракций.

Рекомендации по использованию	
• Создание простых ВсС с низкой потенциальной вариативностью архитектуры.	• Создание сложных ВсС с высокой потенциальной вариативностью архитектуры.

2.2.3 Реализация архитектурных моделей встраиваемых систем

В сознании массового разработчика должно произойти изменение приоритетов и акцентов в таких вопросах проектирования встраиваемых систем, как оценка значимости и ресурсоемкости этапов проектирования, набор канонических архитектур, концепция надежности и безопасности функционирования, соотношение аппаратных и программных составляющих, фиксируемый уровень вычислительной платформы.

Важной предпосылкой к проектированию встраиваемых систем в базисе унифицированных вычислительных механизмов следует считать ограниченную номенклатуру последних, которая используется на практике разработчиками.

Центральным моментом перспективной методики является требование выполнять максимальный объем проектных работ инвариантно к аппаратному, программному или смешанному способу реализации. Это предполагает существование обобщенной модели встраиваемой вычислительной системы, которая исключает противопоставление различных способов реализации друг другу и позволяет с минимальной трудоемкостью отобразить (в пределе автоматически оттранслировать) абстрактную целевую систему на совокупность конкретных вычислительных платформ.

Указанная архитектурная модель базируется на ряде понятий, которые требуют всестороннего обсуждения в связи с радикальными изменениями процесса проектирования цифровых ВсС. К ним, прежде всего, относятся:

- реализация вычислительного устройства аппаратным и программным способом;
- процессор и варианты его организации;
- вычислительная платформа;
- свойства управляемости и программируемости;
- виртуальная машина;
- модель вычислений (вычислительного процесса).

Понятные и очевидные на первый взгляд термины "аппаратура" (hardware) и "программа" (software) нуждаются в более четком и, возможно,

нетрадиционном определении. Также важным является осмысление вносимого качества в процесс или изделие за счет использования принципа программируемости на этапах проектирования, реализации, эксплуатации.

Возможна более широкая трактовка терминов *hardware и software – соответственно, фиксированная и изменяемая части ВС*. Но и в этом случае проявляются терминологические неудобства: что понимать под возможностью и процессом изменения? Менее распространенный термин *firmware* означает изменяемую часть ВС, но не определяет при этом способ ее реализации. Использование терминов *hardware* и *software* в расширенном смысле удобно в рамках рассматриваемой методики архитектурного проектирования.

Важнейшее значение в контексте рассматриваемой методики приобретает проблема классификации той или иной части ВС по способу реализации. И в первую очередь это проявляется в определении процессора.

По способу организации вычислитель можно реализовать, например, в виде "клубка" нерегулярной логики, или структурированного устройства с операционной и управляющей частями. В свою очередь, управляющая часть может быть структурирована в различной степени. Необходимо отметить, что уровень входного языка для указанных выше реализаций будет, скорее всего, различным, а "вилка" вариантов может оказаться значительной.

В вычислительной технике под *процессором* понимается *устройство обработки информации*. Процессоры целесообразно классифицировать по функциональной направленности, функциональной гибкости, способу реализации.

В первом случае говорят об универсальных, математических, сигнальных, ввода-вывода и других группах процессоров. Процессоры с ярко выраженной функциональной направленностью называют специализированными (dedicated).

Степень функциональной гибкости или возможность настройки процессора на выполнение конкретной функции из допустимого множества в каждый момент времени определяется возможностью и оперативностью его программирования. Если в процессе эксплуатации функция может быть перенастроена, то такой процессор называется *программируемым (programmable processor)*, в противном случае мы имеем дело с *непрограммируемым, "жестким"* устройством (*dedicated processor* – близкий, но не точный термин, другой возможный вариант – *hardware accelerator*). Сложность механизма программного управления может изменяться в очень широких пределах.

По способу реализации процессоры, прежде всего, следует делить на *аппаратно-реализованные* (или аппаратные, *hardware*) и *программно-реализованные* (соответственно, программные, *software*). Каждая из указанных групп реализаций, в свою очередь, имеет множество вариантов. Вполне пригодным для практического использования критерием программной

реализации процессора следует считать наличие в его составе хотя бы одной программно-реализованной части.

Как на практике классифицировать способ реализации устройства по аппаратно-программному признаку?

Довольно часто программную реализацию связывают с использованием принципа программного управления при проектировании устройства. Также широко распространено мнение об обязательном свойстве последовательной интерпретации управляющей информации устройством для отнесения его к разряду программно-реализованных.

Общим критерием аппаратной или программной реализации устройства (функции) предлагается считать степень избыточности присутствующей в устройстве регулярной структуры (блока постоянной или оперативной памяти, логической матрицы, операционных элементов и т.д.). Из данного определения следует, что не только организация управляющей части устройства определяет способ его реализации. Например, табличный функциональный преобразователь также может быть отнесен к категории программных устройств.

Явным случаем аппаратной реализации является отсутствие в устройстве регулярных структур вообще (нерегулярная "жесткая" логика, то, что выше было названо "клубком схем"). На практике сегодня, в силу ряда причин, устройство создают на основе избыточных регулярных структур, а затем фиксируют его в получившемся виде, либо выполняют еще один шаг – удаляют неиспользованные элементы регулярной структуры (целевая компиляция, кремниевая компиляция и так далее в соответствии с контекстом). В первом случае мы будем иметь дело с программно-реализованным устройством, во втором случае программно-реализованная версия на этапе проектирования заменяется аппаратной реализацией ("жесткая логика").

Таким образом, можно говорить о четырех базовых вариантах процессоров:

- "жесткий" процессор с аппаратной реализацией (dedicated hardware processor);
- программируемый процессор с аппаратной реализацией (programmable hardware processor);
- "жесткий" процессор с программной реализацией (dedicated software processor);
- программируемый процессор с программной реализацией (programmable software processor).

Отдельного пояснения требует термин "микропроцессор". Традиционно он трактуется как программируемый процессор в интегральном исполнении. Однако в связи с развитием технологий ASIC, ASSP, PLD (ПЛИС), ASIP

логичнее микропроцессором называть все четыре группы процессорных элементов в случае их интегрального исполнения.

Вычислительные системы на верхнем уровне рассмотрения могут быть представлены тремя группами элементов: обработчики, устройства памяти, интерфейсы. С группой обработчиков связаны термины "*процессор*" и "*контроллер*". Предлагается процессором называть обрабатывающий элемент, функции которого в рамках прикладной задачи еще не зафиксированы. В зависимости от закрепленной прикладной (по отношению к данному элементу, а не к системе в целом) функции процессор будет играть роль контроллера (т.е. устройства управления), умножителя, супервизора, диспетчера и т.д. В свою очередь контроллеры могут быть самого различного назначения: памяти, принтера, последовательного интерфейса, технологического процесса и другие. Еще один важный термин – "*микроконтроллер*", следует понимать как контроллер, построенный на основе микропроцессорной элементной базы. Микроконтроллеры могут быть однокристальными, одноплатными, программируемыми, логическими, промышленными, универсальными и т.д.

Важную роль в проектировании ВсС играет единый взгляд на алгоритмическую организацию систем [87, 3]. Выбранный разработчиком или навязанный элементной базой способ интерпретации алгоритма определяет методы и средства проектирования и ключевые характеристики системы. Можно говорить о следующих вариантах интерпретации:

- параллельная аппаратная интерпретация (аппаратное исполнение);
- программная микропроцессорная интерпретация (процессор с набором команд RISC/CISC, включая микропрограммную реализацию, VLIW и EPIC архитектуры);
- интерпретация виртуальной программно-реализованной машиной (операционная система, выделенная сетевая коммуникационная система, СУБД, языковая машина и т.д.).

В качестве *классификационного признака реализации предлагается использовать формальное определение команды как управляющей структуры с фиксированным форматом*. Тогда *под аппаратной реализацией можно понимать прямое исполнение алгоритма, когда нет "форматных" управляющих структур* – микрокоманд, команд. *Под программной реализацией можно понимать многоуровневую интерпретацию, когда есть "форматные" управляющие структуры – примитивы, и их последовательности – программы*. Представленное деление на аппаратную и программную составляющие системы позволяет создать эффективную основу для методов и средств перехода от архитектурной модели к физической реализации ВсС.

2.2.4 Проектирование микроархитектуры ВсС

2.2.4.1 Микроархитектура компонентов встраиваемых систем.

Восприятие виртуальной машины как реализации модели определенного вычислителя позволяет использовать в проектировании такие общеизвестные понятия как ISA (Instruction Set Architecture) и микроархитектура. ISA определяет инструкции, регистры, режимы адресации, типы данных и другое, которые поддерживает определенный вычислитель. Микроархитектура в свою очередь определяет принципы внутренней организации вычислителя, позволяющие реализовать заданную ISA. Одна и та же ISA может быть реализована с помощью различных микроархитектур.

Одним из важных свойств ISA, накладывающих определенные ограничения на микроархитектуру, является аспект времени (конвейеры, параллелизм на уровне инструкций и т.д.).

При построении компонентов BcC можно оперировать следующими понятиями:

- Базовая вычислительная платформа, в свою очередь является виртуальной машиной;
- Виртуализация;
- ISA;
- Микроархитектура.

Виртуальной машиной можно считать конкретную реализацию заданной ISA в рамках выбранной микроархитектуры. В таком случае основной задачей высокоуровневого проектирования становится выбор ISA с последующим выбором микроархитектуры реализации. При всем этом не следует понимать реализацию как чисто программную или аппаратную, возможны смешанные варианты, приводящие к многоуровнемым реализациям микроархитектуры для определенного типа базовой вычислительной платформы.

Для зафиксированной ISA эффективная микроархитектура жестко привязана к особенностям базовой вычислительной платформы (нижележащей виртуальной машины), которые сложно зафиксировать в случае встраиваемых систем, ввиду частой смены элементной базы.

2.2.4.2 Проектирование микроархитектуры BcC на основе композиции «шаблон – конфигурация»

Инструкции (функциональные примитивы, методы) можно рассматривать как управляющие воздействия, призванные изменить или получить состояние отдельных компонентов системы.

Каждая ISA описывает типы данных, логику вычислительного процесса и отдельные инструкции. При этом действия, выполняемые вычислителем, также описываются в терминах поведения некоторого абстрактного нижележащего вычислителя (стек, адресные пространства, передача пакетов и т.д.).

Данная технология проектирования микроархитектуры включает следующие основные шаги:

1. Анализ нижележащей ISA и выделение в явном виде набора примитивных инструкций.
2. Создание множества *кортежей* примитивных инструкций, реализующих инструкции вышележащей ISA. В общем случае кортеж - это не просто шаблонная подстановка множества инструкций. Кортеж предполагает определенный *алгоритм* исполнения, можно сказать *микрокод*, конкретизация исполнения которого возможна только в режиме run-time, так как определенные действия будут зависеть от входных данных инструкции и состояния нижележащей виртуальной машины.
3. Минимизация количества уникальных кортежей за счет создания шаблонов с помощью обобщения кортежей и введения определенных *конфигураций*. При этом каждый изначальный кортеж может быть получен путем конфигурирования одного из шаблонов, при этом допускается, что не все конфигурации имеют смысл.

Основная задача: *рациональная минимизация количества шаблонов за счет усложнения конфигурации*. Ограничениями в процессе минимизации могут стать отношение временных масштабов, ресурсоемкость реализаций, “регулярность” вычислительных структур и алгоритмов. При рассмотрении шаблонов можно искусственно выделить: структурный шаблон, функциональный шаблон и смешанный шаблон. Шаблон можно рассматривать как определенный вид *метаинструкции* в рамках ISA.

Инструкцией вышележащей ISA является конкретный *кортеж*, задаваемый парой *шаблон* и *конфигурация*. При этом в процессе реализации есть возможность предоставить два типа вызовов инструкций вышележащей виртуальной машины:

- высокоуровневый вызов – активация заранее заданной пары по индексу из таблицы; пользователю недоступны шаблоны и конфигурации;
- низкоуровневый вызов – активизация конкретного кортежа с заданной конфигурацией; при это кодирование допускает существование инструкций, отсутствующих в исходной ISA или вовсе не имеющих смысла.

Основной областью применения описанной технологии является разработка драйверов ввода/вывода и реализация протоколов обмена. При зафиксированной вышележащей ISA имеется великое множество вариантов базовых вычислительных платформ (различная архитектура аппаратных контроллеров ввода/вывода, система прерываний и т.д.), что создает проблемы портирования программного кода и повторного использования каких бы то ни было артефактов проектирования. Введение уровня абстракции аппаратуры не всегда возможно, ввиду крайне неэффективных алгоритмов такого рода виртуализации и недостаточного запаса в соотношении временных масштабов.

Кроме того остро может стоять проблема эффективности реализации (виртуализации базовой вычислительной платформы).

В рамках такого подхода к проектированию микроархитектур были, например, реализованы драйверы ввода-вывода для платформы аналитических приборов «МиниЛаб» [9]:

- Подсистема ввода/вывода keX;
- Драйвер I2C: шаблон i2cExchange();
- Драйвер DataFlash: 3 структурных шаблона реализуют 28 инструкций;
- Драйвер 1-Wire;
- Драйвер IEEE 802.15.4 (в процессе разработки).

2.2.5 Роль моделирования в архитектурном проектировании ВсС

Видно, что предложенный выше подход к проектированию ВсС базируется на широком использовании моделей разного уровня абстракции и направленности. Собственно сам процесс проектирования превращается в управляемый и контролируемый процесс эволюции множества моделей, подчиненный внешним ограничениям.

Моделирование системы в процессе проектирования - достаточно сложный и разноплановый процесс, тесно связанный с верификацией. Моделировать проектируемую ВсС можно на разных уровнях абстракции и применять при этом различные средства и языки моделирования. Процесс проектирования ВсС имеет определенные отличия от проектирования программных комплексов или приложений баз данных. Эти отличия в основном заключаются в том, что при проектировании ВсС почти всегда необходимо разрабатывать аппаратную составляющую системы. Сложность и ответственность разработки этих компонентов не меньше, а во многих случаях и больше, чем сложность программного компонента ВсС.

При проектировании ВсС моделирование призвано решать следующие задачи: верификация (целевая и эквивалентная), виртуальное прототипирование, реализация компонентов системы (hardware и software), специфицирование и документирование разрабатываемых компонентов.

В настоящее время существует несколько способов или уровней описания модели ВсС. Для проведения успешной разработки необходимо иметь модели системы различных уровней абстракции и подробности. Модели могут описывать систему в целом или отдельные вычислительные узлы системы. В табл. 2.22 приведена оценка применимости некоторых “собирательных” языковых средств для описания моделей ВсС различных уровней абстракции.

Таблица 2.2. Применимость языковых средств для описания моделей BcC

	Система в целом	**Отдельный узел**	**Программная реализация**	**Аппаратная реализация**
UML	✓	?	✓	
HLL	?	✓	✓	?
HDL	?	✓	?	✓
RTL	?	?		✓

✓ – язык может быть использован

? – использование языка затруднено и неэффективно

UML – Unified Modeling Language [73]. Язык может быть успешно применен для моделирования всей системы целиком на высоких уровнях абстракции или отдельных программно реализуемых вычислительных узлов. Абстрактное описание вычислительного узла с помощью UML не очень удобно, так как собственно UML для этого не очень подходит по своим возможностям. Недостатком языка следует считать слаборазвитый инструментарий для поддержки проектирования.

HLL – High Level (Programming) Language. Традиционные высокоуровневые языки программирования. Отлично подходят для описания и иногда могут быть использованы для моделирования программно реализованных узлов. Несколько хуже язык подходит для описания аппаратно реализованных узлов.

HDL – Hardware Description Language. Группа языков, специально разработанная для описания аппаратуры. Постепенно по мере развития этих языков, они стали использоваться для моделирования аппаратной системы и даже для синтеза аппаратно реализованных узлов.

RTL – особая группа HDL, предназначенная для описания именно аппаратных узлов системы на уровне регистровых передач. В отдельных случаях формально можно использовать эти языки для описания и более крупных модулей.

В рамках проектирования BcC процесс моделирования выглядит следующим образом. На начальных этапах проектирования модель представляет собой высокоуровневое поведенческое описание системы. Данное описание необходимо верифицировать на адекватность решаемой целевой задаче. Собственно получение такой модели сам по себе достаточно сложный процесс, идущий параллельно с целевой верификацией и системным проектированием. Однако, после создания полной модели системы, ее необходимо “реализовать”. Реализация модели заключается в следующем:

- Принять решение о способе реализации каждого механизма построенной модели;
- Получить перечень стандартных компонентов (микропроцессоров, микроконтроллеров, интерфейсов, протоколов и т.д.), используемых в системе;
- Получить объектный код для программно реализованных компонент (SW реализация);
- Получить конфигурацию программируемой логики для аппаратно реализованных компонент (HW реализация).

В параграфе 2.2.5.1 показана роль эталонной А-модели, которая определяется как *верифицированная и зафиксированная архитектурная модель системы, не ограничивающая способов реализации*.

2.2.5.1 Эталонная А-модель BcC

В процессе эволюции моделей системы можно выделить значимый момент, к которому у разработчика формируется так называемая “золотая” модель системы. *“Золотая” (эталонная) модель – верифицированная и зафиксированная архитектурная модель системы, не ограничивающая способов реализации*. “Золотая” модель может быть реализуемой А-моделью или как максимум виртуальной (см. раздел 2.3.3). Абстрактная А-модель требует дальнейшей проработки и не пригодна для реализации. Наряду с “золотой” А-моделью можно говорить и о “золотых” аспектных моделях (АСМ).

Моделирование на начальных этапах связано с доказательством адекватности разработанной архитектуры начальным требованиям. Разработчик на этих этапах вынужден применять достаточно сложные методы “функциональной” верификации. Зачастую такие методы носят полуформальный характер и относительно слабо поддержаны инструментальными средствами. Данный тип верификации призван доказать соответствие полученных характеристик А-модели (или частных АСМ) и сформулированных в требованиях к системе характеристик.

При окончательном формировании “золотой” модели процесс моделирования переходит в фазу реализации. На этом этапе моделирование направлено на верификацию отдельных узлов, а не всей системы в целом, и призвано доказать эквивалентность реализаций. В конечном итоге задача разработчика доказать эквивалентность полученных при реализации характеристик и характеристик, заданных “золотой” моделью. На этом этапе преобладают сравнительно простые, но трудоемкие методы эквивалентной верификации и для большинства преобразований существуют специальные САПР.

Еще одной важной задачей “золотой” модели становится создание исходных спецификаций для разработчиков, которые занимаются конечной

реализацией компонентов и узлов системы. При этом, являясь АМ, “золотая” модель специфицирует не только непосредственно конечную функциональность, но и такие моменты как параметры обеспечения параллельности, допустимые пределы энергопотребления, необходимые механизмы надежности, перечень документов, необходимый уровень инструментальной поддержки и многое другое. Нужно отметить, что правильно организованное моделирование обеспечивает специфицирование и документирование проекта на всех этапах и различных уровнях абстракции.

Поведенческий аспект “золотой” модели, являясь отправной точкой реализации hardware и software, представляет собой формальное поведенческое описание системы в терминах некоторой модели вычислений. В процессе реализации этой “золотой” модели постепенно получаются “синтезируемые” software и hardware описания, методы получения которых могут существенно различаться для разных узлов системы. Инструментальный аспект, обладая высокой сложностью и всесторонним влиянием на другие аспекты проектирования, определяет потенциальный уровень сложности разработок коллектива. Исследованные в работе поведенческий и инструментальный аспекты являются необходимыми для формирования аспектной модели проектирования, применимой на практике [16].

2.3 Аспектная модель процесса проектирования ВсС

2.3.1 Аспектные подход к проектированию ВсС

Суть аспектного взгляда на процесс проектирования ВсС вытекает, в первую очередь, из возможности представления ВС посредством совокупности проектных платформ (рис. 2.4) и из целесообразности выделения в проекте отдельных частных проблем (аспектов), решение которых необходимо обеспечивать по ходу разработки на всех основных этапах (рис. 2.5).

Аспектную модель проектирования составляют набор базовых элементов, организация потока (маршрута) проектирования, метрики элементов, критерии проектирования. В основе этого подхода лежит идея выделение аспекта как сегмента проектного пространства, в рамках которого отражается частная проблема в проекте.

Перечислим основные понятия аспектной модели проектирования.

- Аспект – искусственно выделяемый сегмент проектного пространства, отражающий частную проблему проекта по ходу его выполнения (концептуальный, локальный). Проектировщик сам формирует список аспектов, которым потом пользуется, выделяет концептуальные аспекты, которые существуют по всему ходу проекта, может выделять по необходимости локальные аспекты на отдельных шагах проекта. Примеры аспектов:

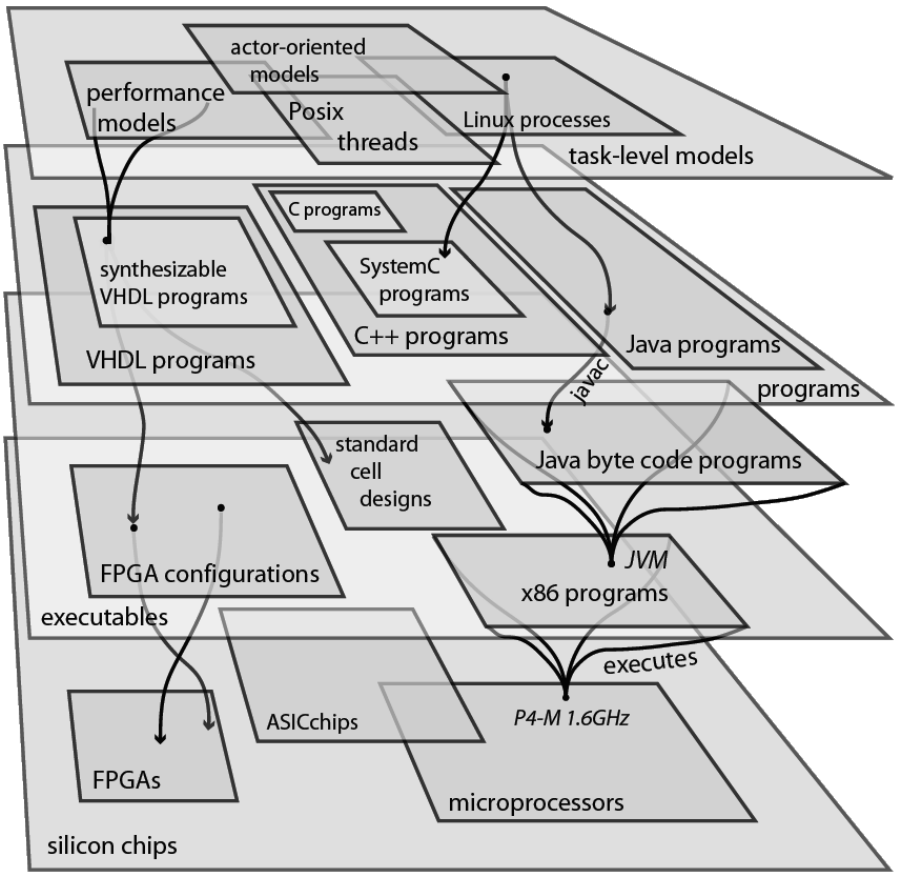


Рис. 2.4. Иерархия проектных платформ [61]

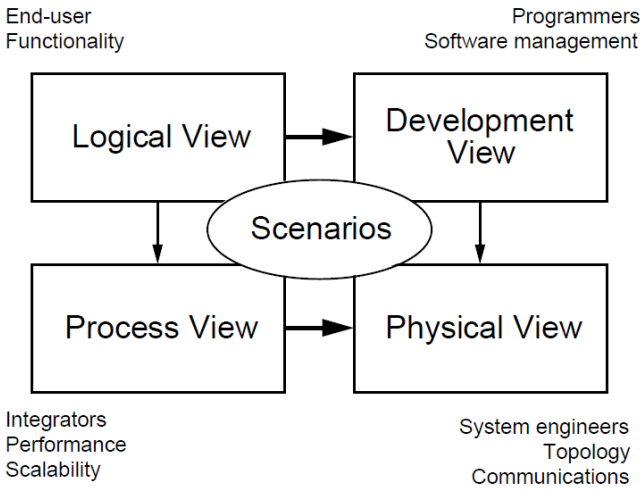


Рис. 2.5. Аспектное представление проекта [59]

- функциональный (если целесообразно, то по подфункциям / подсистемам…);
- надежностный;
- энергетический;
- синхронизации / синхронности / РМВ;
- информационной безопасности;
- конструкторско-технологический;
- повторного использования;
- масштабируемости, реконфигурируемости;
- документно-логистический и другие.
- Архитектурный агрегат (А-агрегат, АА) – элемент проекта. Обладает группами свойств в рамках каждого аспекта. Текущая информация о свойствах АА отражается в его спецификации, а связи между АА в пределах аспекта представляются списком ANL (aspect netlist). Совокупность АА на каждом шаге определяет текущее представление целевой системы и текущее состояние проекта.
- Архитектурная модель (А-модель) – модель проекта/целевой системы, включающая перечень аспектов, совокупность АА, аспектные списки связей (ANL) и спецификации для каждого АА.
- Архитектурная платформа – элемент повторного использования концептуальных решений (А-модель, полезная для повторного использования, зафиксированная в качестве самостоятельного продукта):
- перечень аспектов проектирования;
  - модель (модели) вычислений;

- внешняя логика взаимовлияния аспектов (критерии проектирования);
- перечень зафиксированных шаблонов повторного использования;
- элементная база.

- *Механизм* (Мх) – верифицированный и отложенный для повторного использования АА, снабженный набором метрик (характеристик).

Суть аспектного процесса проектирования состоит в том, что на начальном шаге проект разделяется на аспекты, дальше выполняется параллельное относительно независимое проектирование в рамках аспектов системы и в конечном итоге аспекты собираются воедино в реализацию системы.

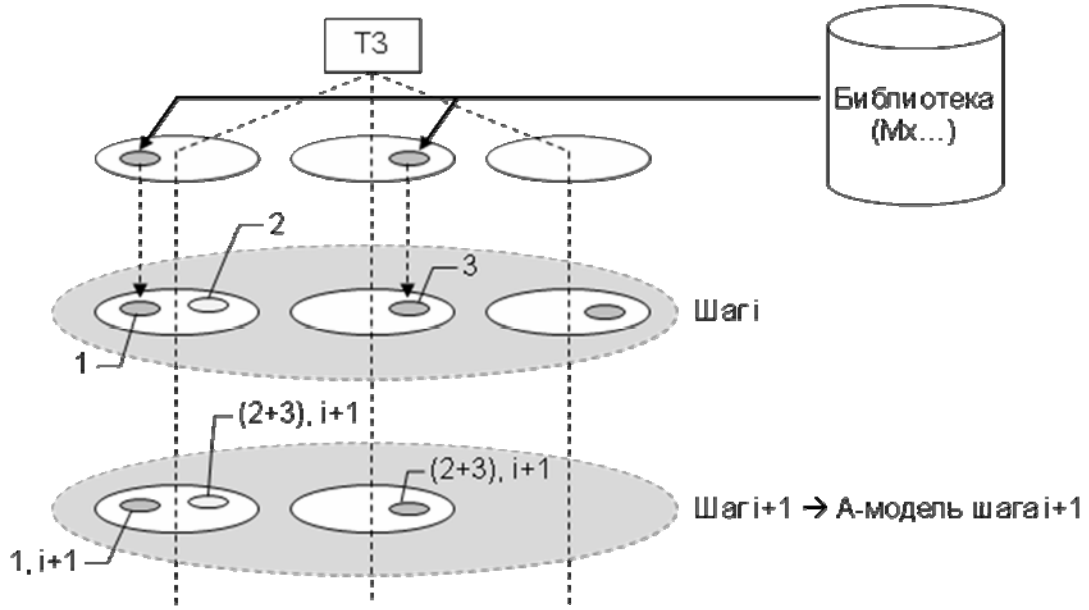


Рис. 2.6. Шаги аспектного процесса проектирования

На рис. 2.6 представлены шаги аспектного процесса проектирования, которые начинаются на этапе анализа технического задания (общего для проекта или частного для подсистемы проекта) с определения перечня концептуальных аспектов. Каждый шаг проекта предполагает проработку А-модели системы с выбранной степенью детализации, постепенно определяя для всех АА аспектное наполнение. На любом из шагов может происходить фиксация АА, отчуждение А-модели в виде архитектурной платформы или новых Мх для повторного использования (рис. 2.7, 2.8).

В качестве проблем следует отметить:

- неформальность перехода шаг 1 – шаг 2 (заново порождаем архитектуру, но детальнее);
- эффект «перемешивания» механизмов, принадлежащих различным аспектам при реализации функциональности каждого из аспектов.

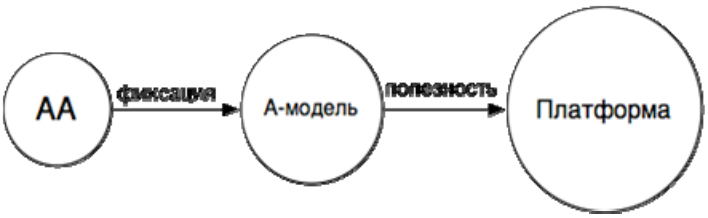


Рис. 2.7. Соотношение продуктов аспектного проектирования

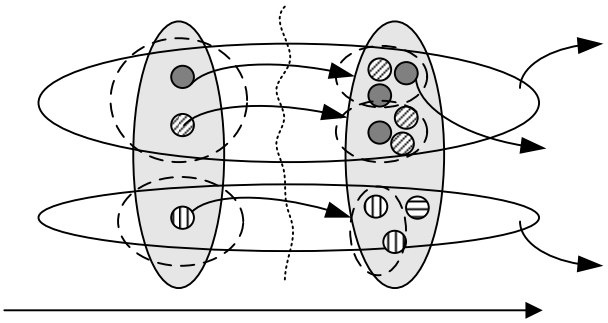


Рис. 2.8. Трансформации АА на шагах проекта

Выбор уровня детализации шагов, методов и средств проектирования внутри шага определяет разработчик, используя доступные и удобные для него инструментальные средства.

А-модель и ВПл рассматриваются как этапы эволюции архитектурных агрегатов. В свою очередь архитектурный агрегат может представляться идеей/решением, формирующейся исключительно в голове проектировщика.

Есть риск при использовании в проектировании стандартизованных элементов, потому что в этом случае проектировщик может никогда не подняться выше комбинаторного способа проектирования. В то же время желательно предоставлять проектировщику библиотеку стандартных элементов для улучшения соотношения «квалификация проектировщика» / размер [сложность] системы и улучшения условий повторного использования. Также необходимо предоставлять возможность создавать новые элементы такой стандартизованной библиотеки. Стандартизованные компоненты в проекте присутствуют в виде А-моделей. Причем, А-модели не предполагают свободное редактирование. Они зафиксированы и могут изменяться только при особых обстоятельствах. А-модель необходимо использовать в проекте одним из следующих способов:

- Определять процесс создания из А-модели редактируемого АА/набора АА (процессов может быть определено несколько и в результате каждого из этих процессов можно получать АА с разными значениями “свободных” свойств (не зафиксированных в рамках АА). После выполнения “процесса создания XXX из А-модели” необходимо выполнить верификацию (проверить, что созданное XXX соответствует А-модели).

- Определять процесс отбора из библиотеки УУУ компонентов соответствующих А-модели. Определяется путем верификации каждого из компонентов с помощью А-модели.

Шаги проектирования следует рассматривать не в качестве временных этапов, а скорее как промежутки между необходимостью создания отчетности для внешнего мира (логические этапы проекта).

2.3.2 Аспектное пространство процесса проектирования и целевой систем

Каждый процесс проектирования характеризуется набором аспектов проектирования, которые выделяет разработчик. Объединение всех таких аспектов даст некоторое множество, которое называется *аспектным пространством процесса проектирования* $\bigcup_i As_i$, отражающее архитектуру всего проекта в целом (framework). Количество рассматриваемых в процессе проектирования аспектов определяет количество элементов в аспектном пространстве процесса проектирования. Этот параметр процесса проектирования называется *аспектной полнотой*

$$F = \left\| \bigcup As_i \right\|.$$

Наряду с аспектным пространством процесса проектирования определяется понятие *аспектного пространства целевой системы*

$$AF = \prod_{i=1}^{\left\| \bigcup As_j \right\|} O_i \ ,$$

где O_i является средствами выражения одного из аспектов процесса проектирования, операция П – прямое произведение множеств. Аспектная полнота определяет размерность аспектного пространства целевой системы $F = \dim(AF)$.

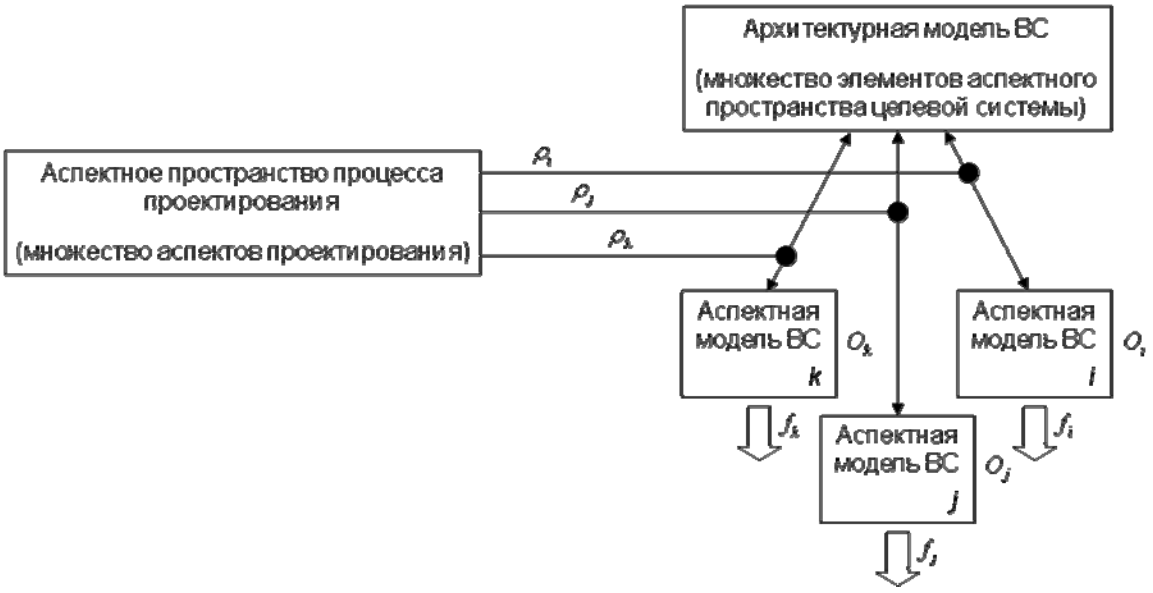


Рис. 2.9. Взаимосвязь аспектных пространств

На рис. 2.9 представлена взаимосвязь аспектных пространств.

Можно говорить о том, что все А-агрегаты являются элементами аспектного пространства целевой системы $aa \in AF$.

2.3.3 Архитектура ВсС, архитектурные агрегаты. Классификация архитектурных моделей

Как было показано в главе 1, очень узкий взгляд на ВС как на некоторую структурно-функциональную надстройку над известной вычислительной платформой, в роли которой выступает традиционная элементная база, не эффективен в современных условиях разработки ВсС. Существует точка зрения на целевую систему, в значительной мере расширяющая взгляд разработчика и демонстрирующая ему большее количество потенциальных решений задачи. Несомненно, структурно-функциональное рассмотрение ВсС крайне важно, но это всего лишь частный случай. Была разработана альтернатива такого взгляда.

2.3.3.1 Понятие архитектуры, архитектурные агрегаты

Альтернативой структурно-функциональному рассмотрению ВС выступает архитектурное рассмотрение [14]. Центральным понятием всего процесса проектирования ВС в этом случае становится *архитектура ВС – совокупность концептуальных аспектов ВС некоторого уровня детализации, адекватно отображающая проектируемую систему для данного уровня рассмотрения*. Данное определение архитектуры является обобщением традиционных трактовок этого понятия, и при необходимости может быть сведено к частным случаям.

Как было предожено в аспектной модели процесса проектирования [16], в перечень концептуальных аспектов, составляющих архитектуру ВсС, помимо традиционных структурных и функциональных элементов входят элементы, вызванные другими факторами, влияющими на процесс проектирования, такие как надежность, энергопотребление, конструкция и т.д. В общем случае, являясь центральным понятием процесса проектирования, это архитектурное рассмотрение содержит в себе все составляющие соответственно процесса проектирования и создаваемого продукта. Все эти составляющие именуются *аспектами проектирования* (или просто *аспектами*).

Конкретный перечень аспектов формируется разработчиком с учетом условий проектирования. Условия определяются как ТЗ (прямо и/или косвенно), так и стилем, возможностями и общими целями самого коллектива разработчиков. На основании опыта работы в области создания ВсС можно выделить наиболее часто встречающиеся аспекты проектирования (список аспектов открытый, но в каждом случае конечный): функциональный, структурный, надежностный, конструктивно-технологический, энергетический, климатический, инструментальный, повторного использования, организационно-экономический, документный и др.

Структурный и функциональный аспекты отражают элементы традиционного структурно-функционального представления ВсС. Как было сказано, это рассмотрение не в состоянии отразить все нюансы процесса проектирования современных ВсС, поэтому необходимо рассматривать и остальные аспекты, которые в некоторых случаях при проектировании выходят на первое место и требуют отдельного внимания.

Надежностный аспект заостряет внимание разработчика на методах обеспечения надежности проектируемой системы, отслеживает эволюцию этого “параметра” в процессе проектирования, выдвигает определенные дополнительные требования. Конструктивно-технологический аспект отражает требования к конструкции ВсС, массогабаритным параметрам, технологичности производства. Энергетический аспект решает такую важную задачу как энергопотребление ВсС, определяет режимы пониженного энергопотребления, их роль и влияние на систему в целом. Климатический аспект должен отслеживать требования к системе по климатическому исполнению. Инструментальный аспект в качестве основной задачи рассматривает проектирование эффективного инструментария для проведения процесса проектирования. Аспект повторного использования на первое место ставит критерий повторного использования отдельных частей проектируемой системы. Организационно-экономический аспект определяет важные с точки зрения конкретного коллектива нюансы, такие как стратегии проекта, попутные задачи (освоение технологий или элементной базы, повышение квалификации и т.д.), перспективность разработки, территориальное расположение объектов и заказчиков, условия сопровождения, жизненный цикл системы и многое другое. Документный аспект определяет состав, объем и качество документирования ВС.

Необходимо отметить, что архитектурное рассмотрение проектируемой ВсС невозможно вне контекста самого процесса проектирования. Многие аспекты архитектурного рассмотрения только опосредованно влияют на проектируемую систему. Основное влияние такие аспекты оказывают собственно на процесс проектирования, который уже в свою очередь повлияет на целевую систему. Затруднительно проведение проектирования целевой системы без учета инструментальных, организационных, интеллектуальных возможностей коллектива, которые в конечном итоге материализуются в той или иной организации процесса проектирования.

При рассмотрении процесса проектирования с архитектурной точки зрения необходимо выделять два объекта верхнего уровня, с которыми имеет дело разработчик: архитектура всего проекта в целом (framework) и архитектура создаваемого продукта (проектируемой ВсС).

Каждый из перечисленных выше аспектов проектирования при рассмотрении характеризуется:

- средствами описания аспекта (O) – элементами некого множества и логикой их взаимодействия, посредством которых выражается суть рассматриваемой частной проблемы;
- аспектным проектором (ρ) – некоторой функцией, позволяющей из архитектурного описания выделить частную проблематику;
- характеристическими функциями (f) – позволяющими получать оценки, основываясь на представлении проблемы в терминологии средств описания аспекта.

Математически, аспект процесса проектирования можно определить как упорядоченную тройку элементов:

$$As = \{O, \rho, f\}, \tag{2.5}$$

$$f = \{\chi : O \to R, N, Z...\}, \tag{2.6}$$

где f – множество характеристических функций, а R, N и Z множества действительных, натуральных и целых чисел соответственно. Замена любого из элементов тройки приводит совершенно к новому аспекту.

Аспектная модель процесса проектирования определяет *архитектурный агрегат (А-агрегат, АА)* как базовый элемент процесса проектирования системы, объединяющий в себе различные точки зрения на целевую систему.

2.3.3.2 Классификация архитектурных моделей

Полностью независимое развитие аспектов с целью получения работоспособной системы в конечном итоге невозможно, так как каждый из них это всего лишь частный взгляд на ВС в целом. Например, желание уменьшить время работы определенного алгоритма потребует повышения тактовой частоты процессора, что в свою очередь приведет к повышению энергопотребления и, в итоге, возрастанию габаритов, которые могут быть жестко ограничены в ТЗ. Выходом в такой ситуации может стать использование другого алгоритма или вычислителя, что на первый взгляд значительно сложнее, но позволяет решить задачу.

Такое взаимное влияние аспектов происходит даже не на уровне А-модели, так как само проектное пространство не накладывает никаких ограничений на А-агрегаты. Продемонстрированные связи всего лишь отражают влияние существующей элементной базы и возможной практической реализуемости отдельных А-агрегатов проектного пространства. В процессе проектирования эти внешние факторы, задающие допустимые соотношения между отдельными аспектами (критерии проектирования), играют решающую роль, особенно, если существует необходимость получать реализуемую систему в заданных внешних условиях.

Кажется, что можно было бы создать оптимальные решения в каждой АСМ и получить идеальную систему. Однако это невозможно в силу ограничений,

накладываемых элементной базой. Суть этих ограничений такова, что не каждый А-агрегат, являющийся объединением аспектных характеристик, может быть реализован в элементной базе, доступной разработчику. Недоступность элементной базы может быть вызвана совершенно разными причинами: ограничениями производства печатных плат, недоступностью определенных микросхем на рынке, квалификацией и размером коллектива, неразвитостью технологий в мировом масштабе.

В процессе проектирования можно создать некоторую идеальную систему (модель), но которая в настоящий момент не может быть реализована силами коллектива. Это вовсе не означает, что эта же А-модель системы не может быть в то же самое время реализована никаким другим коллективом, или, что этот же коллектив не сможет реализовать ее через некоторое время, когда изменятся факторы, внешние для модели системы, но внутренние для коллектива.

Таким образом, мы приходим к важнейшему свойству А-модели – ее реализуемости. В общем случае разрабатываемая ВС не является математической абстракцией, она должна быть реализована в определенной элементной базе и обладать заданной функциональностью. При этом зачастую не столь важно, каким образом система была спроектирована. В нашем случае А-модель состоит из А-агрегатов. При этом при определении А-агрегата нигде не выдвигалось требования его реализуемости. На самом деле это требование вредно, так как автоматически привязало бы разработчика к определенной, зачастую неоправданно выбранной элементной базе.

Ниже приводится классификация А-агрегатов на основании их проекции в аспектное пространство. Пусть АА – некий А-агрегат, F – аспектная полнота, тогда возможны два взаимоисключающих варианта проекций:

$$\exists i \quad (0 < i \le F) \quad \rho_i(aa) = 0, \tag{2.7}$$

$$\forall i \quad (0 < i \le F) \quad \rho_i(aa) \neq 0. \tag{2.8}$$

А-агрегат, описывающий не все аспекты в рамках А-модели, называется *абстрактным А-агрегатом* [см. формулу (2.7)]. То есть при создании и использовании такого агрегата разработчик абстрагируется от тех или иных характеристик системы, которые для него не важны (произвольны) или будут доопределены впоследствии. Нетрудно заметить, что большинство технических решений, протоколов, интерфейсов, стандартов и являются такими абстрактными А-агрегатами, из которых разработчик по факту создает систему. В противоположность абстрактному А-агрегату, у которого существует хотя бы один из неопределенных аспектов, *полный А-агрегат* [см. формулу (2.8)] такой, у которого определены все аспекты, интересующие разработчика при проектировании конкретной ВсС.

Очевидно, что реализуемость А-агрегата можно обсуждать только для полных А-агрегатов. В противном случае для А-агрегата остаются произвольными или неопределенными аспекты, которые в конечной системе

должны существовать. В реализованной системе не может быть частей, которые неопределенны и произвольны. Данные аспекты могут быть неинтересны разработчику с точки зрения эволюции А-модели, но не с точки зрения ее реализуемости.

Назовем полный А-агрегат, который не может быть в настоящее время реализован конкретным коллективом в доступной ему элементной базе, *виртуальным*. Нужно сказать, что виртуальность А-агрегата может определяться не только объективной ограниченностью элементной базы, но и субъективными ограничениями, такими как требования ТЗ, пристрастия и интересы коллектива. Как упоминалось выше, большинство А-агрегатов в процессе проектирования являются не только виртуальными, но зачастую даже абстрактными. О реализации модели, в состав которой входят абстрактные А-агрегаты вообще не может быть и речи, поэтому виртуальным А-агрегатом имеет смысл называть только полный А-агрегат.

Рассматривая А-модель системы с точки зрения образующих ее А-агрегатов можно выделить 3 класса А-моделей:

- Абстрактная А-модель. В такой модели существует хотя бы один абстрактный А-агрегат.
- Виртуальная А-модель. В такой модели нет абстрактных А-агрегат, но существует хотя бы один виртуальный А-агрегат.
- Реализуемая А-модель. В данной модели нет ни одного абстрактного или виртуального А-агрегата.

Абстрактная А-модель принципиально нереализуема и требует дальнейшей доработки, если это модель конкретной целевой системы. Но у таких моделей есть свой способ применения, а именно такие модели следует рассматривать как некоторые, возможно стандартизованные шаблоны для построения конкретных систем. Абстрактными А-моделями, перешедшими в разряд общепринятых шаблонов, можно рассматривать стандартные интерфейсы (USART, I^2C), шины (PC104, PCI, USB), протоколы (CANopen, TCP/IP), вычислительные ядра (MCS51, x86, PowerPC, JAVA, ARM), ОС (QNX, MS Windows) и т.д. При этом перечисленные модели описывают различные перечни аспектов, что не мешает им быть общепризнанными стандартами. Некоторые абстрактные А-модели, не имеющие широкого применения и стандартизации, также можно использовать в качестве повторно используемых шаблонов в рамках коллектива. То есть конкретный коллектив может зафиксировать определенное удачное решение, обозначить направление разработок, обеспечить преемственность и т.д., определив абстрактную А-модель системы и развивая ее в тех или иных конкретных приложениях. Особенно такие модели и локальные (внутри коллектива) повторно используемые шаблоны интересны с хорошо проработанным инструментальным аспектом, так как именно инструментарий в подавляющем большинстве случаев требует повышения

коэффициента повторного использования, и адаптации под конкретные проекты.

Виртуальная А-модель не может быть реализована из-за виртуальных А-агрегатов, но она уже полностью определена и в принципе может быть реализована, если расширить доступную элементную базу или изменить внешние условия. Такие А-модели могут быть использованы и как абстрактные, но в большинстве случаев требуется доводить такие модели до реализации. Для этого нужно избавиться от виртуальных А-агрегатов. Этого добиваются двумя способами: изменением модели или изменением внешних факторов. В первом случае разработчики изменяют модель (проводят процесс проектирования) до тех пор, пока виртуальных А-агрегатов в ее составе не останется. Во втором случае модель остается неизменной, а меняются внешние факторы. Например, происходит уточнение ТЗ в сторону изменения ограничений; обучение коллектива; переход на другие вычислительные ядра; использование новых для коллектива технологий (например, использование ПЛИС, смена языка программирования, поверхностных монтаж и многослойные печатные платы).

2.3.4 Методы и средства аспектного анализа

2.3.4.1 Аспектные проекторы и аспектные модели

А-агрегаты могут иметь различную степень абстракции, представлять собой технические решения или конкретные вычислительные узлы, определять разное количество аспектов для описываемого элемента. Модель системы, выраженную в терминах А-агрегатов, будем называть *архитектурной моделью (А-моделью) системы*. Формально А-модель представляет собой множество А-агрегатов:

$$Am = \{aa : aa \in AF\}.$$

Как следует из определений А-агрегата и А-модели непосредственная работа с ней крайне затруднена. Разноплановость аспектов не позволяет построить развитую логику взаимодействия непосредственно между А-агрегатами как атомарными сущностями в контексте А-модели. Кроме того, различные А-агрегаты могут содержать одни аспекты и не определять другие, что также не позволяет создать сложные правила взаимодействия между ними. В контексте А-модели логика взаимодействия А-агрегатов вырождается до уровней иерархического включения/подчинения и частного взаимодействия друг с другом, но уже в контексте отдельного аспекта. Если в А-модели между А-агрегатами существует взаимодействие, то, по сути, оно начинает проявляться между определенными (одноименными) аспектами этих А-агрегатов. Перечень аспектов, в которых проявляется отношение между А-агрегатами, называется *аспектным спектром* этого отношения.

Каждый из аспектов проектирования определяет свой аспектный проектор [см. формулу (2.7)]. Аспектный проектор предназначен для выделения из А-модели описания отдельного аспекта:

$$\rho_i : AF \to O_i$$
$$PRJ = \bigcup_{i=1}^{F} \rho_i \quad .$$

Развитие А-модели происходит на уровне рассмотрения отдельных ее аспектов. Для этого А-модель системы проецируется в определенное аспектное пространство. При этом для всех А-агрегатов А-модели происходит абстрагирование от аспектов, кроме интересующего, и уничтожаются все отношения, в спектр которых не входит интересующий аспект. Получившаяся модель, отражающая конкретный аспект, называется *аспектной моделью (АСМ)*.

$\forall i : 1 \le i \le F$ верны следующие утверждения:

$$\rho_i(Am) \subset O_i \text{ и } \rho_i(aa) = a_i, \qquad (2.9), (2.10)$$

где a_i является i-ым аспектом А-агрегата АА. Выражение (2.9) демонстрирует возможность выделения АСМ. Выражение (2.10) демонстрирует получение определенного аспекта А-агрегата. Важными свойствами аспектного проектора являются следующие свойства:

$$\forall \rho \in PRJ, \forall A \subset AF, \forall B \subset AF \qquad A = B \Rightarrow \rho(A) = \rho(B), \qquad (2.11)$$

$$\forall \rho \in PRJ, \forall A \subset AF, \forall B \subset AF \qquad \rho(A \cup B) = \rho(A) \cup \rho(B). \qquad (2.12)$$

Примечание. Утверждение

$$\forall \rho \in PRJ, \forall A \subset AF, \forall B \subset AF \qquad \rho(A) = \rho(B) \Rightarrow A = B$$

в общем случае не верно.

А-модель позволяет разработчику отслеживать развитие различных аспектов проектируемой системы, учитывать эмпирическое влияние аспектов друг на друга, контролировать развитие проекта в целом и ВсС в частности. В каждый конкретный момент разработчик может выбрать “ведущий” аспект, который будет определять дальнейшее развитие проекта.

Нельзя рассчитывать что, выделив некоторый набор точек зрения на целевую систему, для каждой из таких точек зрения разработчик будет способен с нуля создать всю технологию проектирования и разработать необходимые инструментальные средства. Выделяемые аспекты должны представлять собой относительно традиционные области деятельности разработчика, достаточно хорошо развитые на данный момент. Отличием процесса проектирования становится рассмотрение ВС и всего процесса целиком как элемента аспектного пространства, а не в рамках только некоторой частной проекции.

Приведенные выше примеры аспектов позволяют говорить о самодостаточности АСМ. Самодостаточность АСМ следует понимать, как наличие у разработчика возможности провести полное проектирование системы, опираясь только на внутренние закономерности развития АСМ и характеристические функции аспекта [см. формулы (2.5), (2.6)]. Каждая из АСМ

обладает необходимыми средствами саморегулирования, чтобы обеспечивать процесс проектирования.

Различные АСМ могут быть выражены различными языковыми средствами (чертежи, схемы, алгоритмы, временные диаграммы и т.д.), и даже одна и та же АСМ может быть выражена по-разному для разных проектов или групп разработчиков. При этом для многих АСМ в настоящее время существует множество формальных и полуформальных способов описания и анализа (например, для функциональной модели можно перечислить автоматные модели, диаграммы потоков данных, диаграммы сообщений и т.д.).

2.3.4.2 Характеристические функции аспектных моделей, ортогональность аспектов

Помимо самодостаточности отдельных АСМ необходимо обеспечить *независимость* отдельных аспектов между собой. Для формализации независимости АСМ необходимо определить операцию получения А-модели из ее АСМ.

Для каждого из аспектных проекторов определяется обратный аспектный проектор – операция перевода элемента АСМ в аспектное пространство целевой системы. Если рассмотреть некоторый аспект i, его АСМ a_i, то обратный аспектный проектор ρ_i^{-1} для аспектного проектора ρ_i определяется следующим образом:

$$\rho_i(\rho_i^{-1}(a_i)) = a_i . \qquad (2.13)$$

Аспект i является *независимым* от аспекта j, если для аспектных проекторов выполнено

$$\forall a_j \quad \rho_i(\rho_j^{-1}(a_j)) = 0 . \qquad (2.14)$$

Аспекты являются *ортогональными*, если они взаимно независимы.

В процессе проектирования все аспекты ортогональны, если

$$\forall i : 0 < i \le F \quad \forall a_i \quad \forall j : 0 < j \le F \quad \rho_j(\rho_i^{-1}(a_i)) = \begin{cases} a_i, \text{если } j = i \\ 0, \text{если } j \ne i \end{cases} . \qquad (2.15)$$

Примером такой операции может стать поэлементное прямое произведение

$$\rho_i^{-1}(a) = \{0 \times 0 \times \ldots \times a \times \ldots \times 0\} ,$$

где a расположено на i-ой позиции.

Независимость отдельных аспектов может быть сформулирована как отсутствие влияния аспектных пространств друг на друга. Другими словами, при выборе аспектов процесса проектирования необходимо требовать их ортогональность. Нужно заметить, что ортогональность аспектов должна присутствовать не в рамках решаемой частной задачи или АСМ, а в рамках именно аспектного пространства процесса проектирования. Конкретные множества точек, представляющие АСМ, могут и не иметь влияния друг на

друга в некоторых частных случаях, хотя для другого случая данное влияние начнет проявляться.

Для АСМ аспект определяет множество характеристических функций [см. формулу (2.6)]. Предлагается использовать условное обозначение (2.16) для записи метрик i-ого аспекта архитектурной модели

$$\chi_i(A) = \chi_i(\rho_i(A)). \qquad (2.16)$$

Задача таких функций дать формальную оценку АСМ, установить ее непротиворечивость, убедиться, что характеристики модели не выходят за внешние ограничения и удовлетворяют требованиям, накладываемым на данный аспект системы. Ввиду независимости (ортогональности) аспектов, их характеристические функции также будут ортогональны, и непосредственно не будут зависеть от развития других АСМ.

Это легко показать, пользуясь определением ортогональности аспектных проекторов [см. формулу (2.15)]. Продемонстрируем это свойство для независимого аспекта.

Рассмотрим два аспекта проектного пространства *i* и *j*. Пусть имеются две базовых АСМ as_i и as_j. as_i остается фиксированной, а as_j развивается и изменяется. В результате развития as_j получаем as'_j. А-модель системы A превращается в A':

$$\chi_i(A) = \chi_i(A').$$

Другими словами эволюция аспекта не влияет на значения характеристических функций других независимых от него аспектов А-модели.

Если в процессе проектирования аспекты были выбраны таким образом, что они все взаимно ортогональны, то значения характеристических функций изменяются только в связи с изменениями соответствующей им АСМ. Это важное свойство позволяет проводить независимое проектирование в рамках АСМ самостоятельно, а самодостаточность отдельных АСМ позволяет это осуществить на практике.

Тем не менее, так как ортогональность аспектов достижима далеко не всегда, отметим следующее:

- если аспект [в основном] ортогонален, то целесообразно выделение его в частный проект по внутреннему частному техническому заданию (ЧТЗ);
- если аспект не ортогонален, то ЧТЗ аспекта может выступать как совокупность явно сформулированных ограничений, напоминаний, пожеланий (рекомендаций).

2.3.5 Аспектная классификация ВсС

Аспектная модель процесса проектирования ВсС позволяет строить ряд классификаций для ВсС и процессов их проектирования на основе так называемого *спектрального состава концептуальных и локальных аспектов*.

На рис. 2.10 показан результат преобразования традиционной *Y–диаграммы ВС* в *А-диаграмму (аспектную диаграмму) ВсС*:

- центр диаграммы – в зависимости от выбранной трактовки концентрических окружностей представляет собой конечную точку процесса проектирования (реализованная ВсС) или начальную точку (исходное ТЗ);
- концентрические окружности – соответственно, уровни иерархии системы / проекта или шаги процесса проектирования;
- оси по числу соответствуют выделяемым (концептуальным и локальным) аспектам проекта;
- возможно выделение не всех уровней в рамках локального аспекта;
- точки на пересечениях осей и окружностей – значимые категории объектов или процессов целевой системы или проекта ВсС.

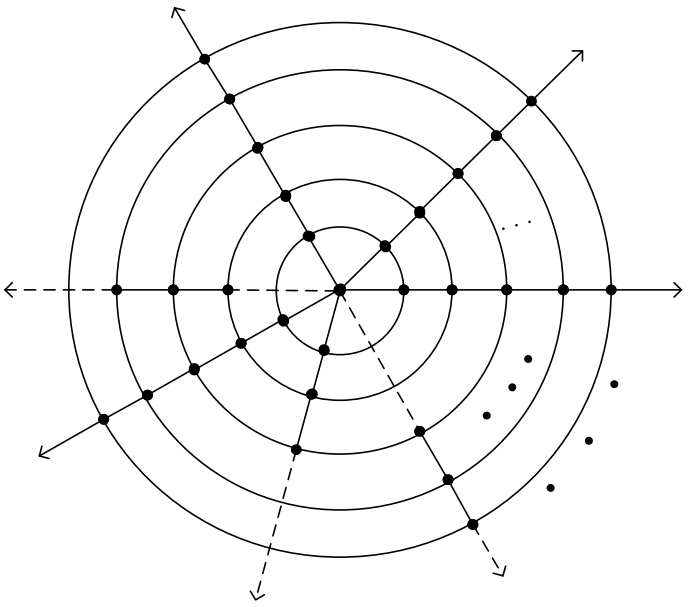


Рис. 2.10. А-диаграмма ВсС

На основе относительного веса аспекта в рамках проекта в целом или на каждом из шагов можно получать оценки предполагаемой реализации или предполагаемого процесса проектирования.

На этапе анализа ТЗ выявленный состав наиболее важных аспектов предполагаемого процесса проектирования позволяет разработчику достаточно точно оценить категорию будущей ВС и выбрать набор соответствующих шаблонов. Веса аспектов, обозначенные даже только в виде согласованных экспертных оценок представителей заказчика и аналитика – и архитектора со

Функцио-
нальность

стороны исполнителя, позволяют направлять работы по генерации вариантов архитектуры ВсС и выстраивать инфраструктуру проекта. Аспектная спектральная классификация может эффектирно использоваться и для ранжирования готовых платформ (шаблонов проектирования) ВсС. На рис. 2.11а представлен пример спектра аспектов: Ф – функциональный, И – инструментальный, Эп – энергопитания, Н – надежностный, РВ – реального времени, Д – документный, С – синхронизации, Точн – точностной и др.

Динамика изменения весов аспектов на фазах и шагах проекта в совокупности позволяет характеризовать желаемый тип процесса проектирования или получить классификацию типовых потоков проектирования, которые предлагаются (предопределяются) производителями САПР в области ВсС, СБИС (рис. 2.11б). Например, аспекты энергопотребления, конструкции и площади кристалла или ограничений элементной базы в различных случаях могут активно использоваться в проектировании только на шагах фазы реализации, начиная с архитектурного проектирования или «с провалами» в рамках ряда шагов по ходу проектирования.

Рис. 2.11. Аспектная классификация ВсС (а) и процессов проектирования ВсС (б)

Исходный спектр аспектов, зафиксированный на этапе ТЗ, может сопоставляться с «интегральным» спектром аспектов проекта, обновляемом на каждом шаге, что позволяет контролировать и корректировать по мере необходимости развитие проекта.

Выводы

1. Система архитектурных абстракций ВсС включает четыре группы понятий. Они охватывают вопросы организации вычислительного процесса, архитектурную организацию системы, базовые элементы и средства оценки проектных решений. Выделяются абстракции вычислительного и невычислительного характера.

2. Вычислительный механизм рассматривается как базовый элемент представления и накопления технических решений в абстрактной форме, не привязанной непосредственно к способу конечной реализации. Понятие распространено на механизмы невычислительного характера, используемые в проектировании ВсС.
3. Определение виртуальной вычислительной машины в контексте проектирования ВсС позволяет представлять иерархические гетерогенные вычислительные архитектуры и унифицировать комплексные функциональные блоки с точки зрения вариантов их реализации.
4. Пространство поиска проектных решений ВсС включает: пространство проекта, пространство целевой системы и пространство поиска технических решений.
5. Рассматривать координату проектного пространства, которая представляет фазы проектирования и исполнения в решении целевой задачи ВсС необходимо с единых позиций. При этом проектирование целевой системы представляется как организация вычислительного процесса в заданных ограничениях.
6. В современных технологиях проектирования присутствуют различные трактовки и варианты использования понятия вычислительная платформа. Введено понятие архитектурная платформа. Дано определение проектной абстракции вычислительная платформа как фиксируемому для повторного использования функционально значимому вычислительному [техническому] решению.
7. Продемонстрирована классификация ВсС на основе понятия «проектная вычислительная платформа», которая позволяет в значительной мере систематизировать базовые вычислительные архитектуры и связанные с ними технологии проектирования ВсС, как широко используемые, так и перспективные.
8. Приведенные шаблоны процессов архитектурного проектирования ВсС для случаев общего и частных технических заданий основаны на традиционной и аспектной моделях проектирования.
9. Для архитектурных моделей ВсС определено понятие аппаратной и программной реализации вычислительного устройства и введена классификация процессоров по критерию «программируемости – способу реализации». Показано место и роль проектирования микроархитектуры в высокоуровневом проектировании ВсС. Приведен метод проектирования микроархитектуры ВсС на основе композиции «шаблон – конфигурация».
10. Аспектная модель процесса проектирования ВсС включает следующие основные положения:
    - обобщение понятия архитектуры ВС, увязывающее в себе все аспекты процесса проектирования современных ВС. В качестве важного объекта

проектирования рассматривается А-модель, состоящая из А-агрегатов, представляющих базовые элементы моделирования;

- классификации А-агрегатов и А-моделей;
- определения и формальные математические описания базовых понятий А-модели ВС, таких как аспектное пространство процесса проектирования, полнота аспектного пространства, аспектные проекторы, аспектные модели, характеристические функции;
- формально, с использованием характеристических функций и аспектных проекторов, определено понятие независимости аспектов. На его основе показано удобство использования ортогональных аспектов в процессе проектирования;
- в контексте архитектурного проектирования сформулировано понятие “золотой” модели. В качестве поведенческого аспекта проектирования ВС рассматривается модель вычислений;
- понятие “архитектурная платформа” выступает в качестве мощного инструмента повторного использования концептуальных решений в процессе проектирования ВсС.

11. Аспектная классификация как самих ВсС, так и процессов их проектирования позволяет наглядно представлять свойства систем и критерии их создания, контролировать процесс проектирования, сравнивать проектные платформы, анализировать выполненные проекты ВсС.
12. Следующие основные положения ставят аспектную технологию проектирования ВсС в один ряд с другими перспективными методиками проектирования современных ВС:
    - широко используется архитектурное представление проектируемой системы, определяется понятие архитектурная модель;
    - в рамках проектирования значительную роль играют процессы моделирования;
    - объединены в процессе проектирования требования к целевой системе (продукту) и вторичные требования процесса проектирования (окружения);
    - унифицировано рассмотрение вычислительных процессов на всем протяжении процесса проектирования, вплоть до последней фазы реализации;
    - унифицированы трактовки традиционных понятий hardware и software;
    - выделены “ортогональные” области активности в процессе проектирования, развитие системы в рамках отдельных аспектов;
    - использованы шаблоны абстрактных и виртуальных архитектурных моделей для повышения коэффициента повторного использования;

- ограничения элементной базы учитываются в качестве внешних требований к процессу проектирования.

Следует отметить, что аспектная технология находится в начале своего становления. Необходимы активные исследования в части формализации данного подхода с дальнейшим развитием САПР соответствующего профиля.

Список сокращений и обозначений

А-агрегат, АА	архитектурный агрегат
А-модель (АМ)	архитектурная модель
А-платформа	архитектурная платформа
АСМ	аспектная модель
ВС	вычислительная система
ВсС	встраиваемая или встроенная ВС, embedded system
ВВМ	виртуальная вычислительная машина
ВМх	вычислительный механизм
ВПО	встроенное программное обеспечение
ИУС	информационно-управляющая система
МоС	модель вычислений
Мх	механизм
ОС	операционная система
ОСРВ	операционная система реального времени
ПЛИС	программируемая логическая интегральная схема
ПК	персональный компьютер
ПО	программное обеспечение
РИУС	распределенная ИУС
РМВ	реальный масштаб времени
САПР	система автоматизированного проектирования
СБИС	сверхбольшие интегральные схемы
СРВ	система реального времени
ТЗ	техническое задание
УСО	устройство сопряжения с объектом
ЯВУ	язык высокого уровня
ANL	aspect netlist
API	application programming interface
CFSM	codesign finite state machine
CPLD	complex programmable logic device
DFD	data flow diagram
DT	Design-Time
FSM	finite state machine
FW	firmware
HW	hardware
IAP	in-application programming
ICP	in-circuit programming
ISA	Instruction Set Architecture
ISP	in-system programming
RT	Run-Time
SOC, СнК	System-On-a-Chip, система на кристалле
SW	software

Литература

1. Булычев Д. Прототипирование встроенных систем на основе описания макроархитектуры. // Диссертация на соискание учёной степени кандидата физ-мат наук. СПбГУ. 2004.
2. Бухтеев А.В. Методы и средства проектирования систем на кристалле. // Chip News. № 4 (77). 2003. С. 4–14.
3. Вирт Н. Аппаратная компиляция. // Открытые системы. № 4-5. 1998. С. 7–12.
4. Иванов А.Н. Технологическое решение REAL-IT: создание информационных систем на основе визуального моделирования. // Системное программирование. Сб. статей под. ред. А.Н.Терехова и Д.Ю.Булычева. СПб. 2004. С. 89–100.
5. Кини Р.Л., Райфа Х. Принятие решений при многих критериях: предпочтения и замещения. М. Радио и связь. 1981.
6. Ключев А., Кустарев П., Платунов А. Контроллеры с микроэнергопотреблением в распределенных системах управления. // Компоненты и технологии. №7. 2001. С. 80–83.
7. Ключев А.О., Кустарев П.В., Платунов А.Е. Распределенные системы управления. // Сб. тезисов ДИМЭБ. СПб. 1997. С. 216–217.
8. Ковязин Р.Р. Постников Н.П. Создание локальных регуляторов на базе виртуальной машины с динамическим набором инструкций. // Научно-технический вестник СПбГУИТМО. Выпуск 32. Информационные технологии: теория, методы, приложения. СПб. СПбГУИТМО. 2006. С. 55–62.
9. Ковязин Р.Р., Постников Н.П. Разработка проблемно-ориентированных процессоров. // Сборник тезисов докладов конференции молодых ученых. Выпуск 5. СПб: СПбГУ ИТМО. 2010. С. 16–17.
10. Колонка редактора. // Встраиваемые системы. №1. 2009. С.6–10.
11. Кучински К. URL: http://cs.lth.se/edan15.
12. Непейвода Н.Н., Скопин И.Н. Основания программирования. Москва-Ижевск. Институт компьютерных исследований. 2003. 868 с.
13. Парфенов В.В., Терехов А.Н. RTST – технология программирования встроенных систем реального времени. // Сб. Системная информатика. Вып.5. Новосибирск. Сибирская издательская фирма РАН. 1997.
14. Платунов А.Е. Архитектурная модель цифровых вычислительных систем для встроенных применений. // Изв. вузов. Приборостроение. Т.44. №3. 2001. С.8–15.
15. Платунов А.Е. Заказные вычислительные платформы информационно-управляющих систем. Презентация компании ЛМТ. // Электронные компоненты. № 12. 2005. С. 42.
16. Платунов А.Е., Постников Н.П. Единое проектное пространство плюс аспектная технология – перспективная парадигма проектирования встраиваемых систем. // Научно-технический вестник СПбГУ ИТМО.

Вып. 11. Актуальные проблемы анализа и синтеза сложных технических систем. СПб. СПбГУ ИТМО. 2003. С.121–128.
17. Платунов А.Е., Постников Н.П. Формализация архитектурного проектирования информационно-управляющих систем. // Тезисы докладов XXXI научно-технической конференции ППС. СПб. ГИТМО(ТУ). 2000. С. 122.
18. Подиновский В.В., Ногин В.Д. Парето-оптимальные решения многокритериальных задач. М. Наука. 1982. 256 с.
19. Смирнов О.Л. Автоматизация технологического проектирования: Учеб. Пособие. СПб. СПбГУАП. 2001. 66 с.
20. Смит Дж., Наир Р. Архитектура виртуальных машин. // Открытые системы. URL: www.osp.ru/os/2005/05-06/185586/_p1.html.
21. Соболь И.М., Статников Р.Б. Выбор оптимальных параметров в задачах со многими критериями. М. Наука. 1981.
22. Терехов А.Н. RTST – технология программирования встроенных систем реального времени. // Сб. "Записки семинара кафедры системного программирования "CASE-средства RTST++". Вып.1. СПб. Издательство СПбГУ. 1998.
23. Терехов А.Н. Тезисы диссертации на соискание учёной степени доктора физ.-мат. наук. СПбГУ. 1991. URL: http://ant.tepkom.ru/publications/doc/Terekhov_Doct_thesis.pdf.
24. Терехов А.Н., Романовский К.Ю., Кознов Дм.В., Долгов П.С., Иванов А.Н. Real: методология и CASE-средство для разработки систем реального времени и информационных систем. // Программирование. №5. 1999. С. 44–52.
25. Топорков В.В. Модели и методы системного синтеза. М. Моск. энерг. ин-т. 1997.
26. Топорков В.В. Модели распределенных вычислений. Физматлит. 2004. 320 с.
27. Шалыто А.А. SWITCH – технология. Алгоритмизация и программирование задач логического управления. СПб. Наука. 1998. 628с.
28. Штойер Р. Многокритериальная оптимизация. М. Радио и связь. 1992.
29. ARTEMIS Industry Association. URL: https://www.artemisia-association.org.
30. Balarin F., Giusto P., Jurecska A., Passerone C., Sentovich E., Tabbara B., Chiodo M., Hsieh H., Lavagno L., Sangiovanni-Vincentelli A., Suzuki K. Hardware-Software Co-Design of Embedded Systems: The POLIS approach. Kluwer Academic Publishers. 1997. 297 p.
31. Baleani M., Ferrari A., Mangeruca L., Sangiovanni-Vincentelli A., Peri M., Pezzini S. Fault-tolerant platforms for automotive safety-critical applications // In Proc. of the Intl. Conf. on Compilers, Architectures and Synthesis for Embedded Systems. ACM Press. 2003. P. 170–177.

32. Barabanov A., Bombana M., Fominykh N., Gorla G., Terekhov A. Reusable objects for optimized DSP design // Embedded Microprocessor Systems. IOS Press. 1996. P. 433–442.
33. Boulytchev D., Lomov D. An Empirical Study of Retargetable Compilers. In Perspectives of System Informatics, Springer Berlin. Heidelberg. 2001.
34. Buck J.T., Ha S., Lee E.A., Messerschmitt D.G. Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems. // Int. Journal of Computer Simulation special issue on “Simulation Software Development”. April 1994. Vol. 4. P. 155–182.
35. Buck J.T., Ha S., Lee E.A., Messerschmitt D.G. Ptolemy: A mixed-paradigm simulation/prototyping platform in C++. // In Proceedings of the C++ At Work Conference, Santa Clara, CA. November 1991.
36. Chang H., Cooke L., Hunt M., Martin G., McNelly A., Todd L. Surviving the SOC Revolution: A Guide to Platform-Based Design. Kluwer Academic Publishers. November, 1999.
37. Clarke E.M., Emerson E.A., Sistla A.P. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications. // ACM Trans. on Programming Languages and Systems. Vol. 8. April 1986. P. 244–263.
38. CoFluent Design. URL: http://www.cofluentdesign.com.
39. Computing Curricula 2001. URL: *http://www.acm.org/education/curric_vols/cc2001.pdf/view.*
40. Davare A., Densmore D., Meyerowitz T., Pinto A., Sangiovanni-Vincentelli A., Yang G., Zeng H., Zhu Q. A Next-Generation Design Framework for Platform-Based Design // Conference on Using Hardware Design and Verification Languages (DVCon) 2007, San Jose California, February 2007. 8 p.
41. Densmore D., Passerone R., Sangiovanni-Vincentelli A. A Platform-Based Taxonomy for ESL Design. IEEE Design and Test of Computers, September 2006.
42. Dictionary of Computing. New York. Oxford University Press. 1983. 393 p.
43. Edwards S., Lavagno L., Lee E.A., Sangiovanni-Vincentelli A. Design of embedded systems: Formal models, validation, and synthesis. // Proceedings of the IEEE. March 1997.
44. Ellervee P., Kumar1 S., Jantsch A., Svantesson B., Meincke T., Hemani A. IRSYD: An Internal Representation for Heterogeneous Embedded Systems // NORCHIP’98 – The 16th NORCHIP Conference. Lund, Sweden. November 9-10, 1998. P. 214-221.
45. Ferrari A., Sangiovanni-Vincentelli A. System Design: Traditional Concepts and New Paradigms. // Proceedings of the 1999 Int. Conf. On Comp. Des. Austin. October 1999.
46. Gajski D. Silicon Compilers. Addison-Vesley. 1987.
47. Hardware-Software Codesign. // IEEE Design & Test of Computers, January-March 2000. P. 92–99.

48. Hatley D.J., Pirbhai I.A. Strategies for Real-Time System Specification. Dorset House. 1988.
49. Information Technology. Portable Operating System Interface (POSIX). 2003.
50. ITEA Technology Roadmap for Software-Intensive Systems. 2 edition. May 2004.
51. J. E. Smith, R. Nair. Virtual Machines — Versatile Platforms for Systems and Processes. // Elsevier Inc. 2005. 649 p.
52. Jerraya A.A., Romdhani M., Marrec PH.LE, Hessel F., Coste P., Valderrama C., Marchioro G.F., Daveau J.M., Zergainoh N.-E. Multilanguage specification for system design and codesign. URL: http://tima-cmp.imag.fr/Homepages/cosmos/documents/asi.ps.
53. Jozwiak L., Nejah N., Figueroa M. Modern development methods and tools for embedded reconfigurable systems: A survey. // INTEGRATION, the VLSI journal, 07.2009, P. 1–33.
54. Keutzer K., Malik S., Newton R., Rabaey J., Sangiovanni-Vincentelli A. System Level Design: Orthogonalization of Concerns and Platform-Based Design. // IEEE Transactions on Computer-Aided Design of Circuits and Systems. Vol. 19, No. 12. December 2000.
55. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Videira Lopes, C., Loingtier, J.-M., and Irwin, J. Aspect-oriented programming. // Proceedings of the European Conference on Object-Oriented Programming (ECOOP), vol.1241, 1997, p.220–242.
56. Knudsen P.V. PACE: A Dynamic Programming Algoritm for Hardware/Software Partitionning. // Department of Computer science Technical University of Denmark. 1996.
57. Knudsen P.V., Madsen J. Integrating communication protocol selection with hardware/software codesign. // IEEE Transactions on Computer-Aided Design of Integrated Circuits. August 1999.
58. Kopetz H. REAL-TIME SYSTEMS. Design Principles for Distributed Embedded Applications. Kluwer Academic Publishers. 1997.
59. Kruchten P. Architectural Blueprints – The “4+1” View Model of Software Architecture. 1995
60. Lee E. A. What’s ahead for embedded software? // IEEE Computer. September 2000.
61. Lee E. Cyber Physical Systems: Design Challenges. // University of California, Berkeley Technical Report No. UCB/EECS-2008-8. 2008.
62. Lee E.A. Concurrent Models of Computation for Embedded Software. // UCB ERL Technical Memorandum M05/2. Department of Electrical Engineering and Computer Sciences, University of California. Berkeley, January 4, 2005.
63. Lee E.A. Embedded Software – An Agenda for Research. // Technical Memorandum UCB/ERL M99/63, University of California. Berkeley. December 15, 1999.

64. Lee E.A. Embedded Software. // Technical Memorandum UCB/ERL M01/26, University of California. Berkeley. November 1, 2001.
65. Lee E.A. Modeling Concurrent Real-Time Processes Using Discrete Events. // Annals of Software Engineering. Special Volume on Real-Time Software Engineering. 1998.
66. Lee E.A., Neuendorffer S., Wirthlin M.J. Actor-Oriented Design of Embedded Hardware and Software Systems // Journal of Circuits, Systems, and Computers, Version 2. November 20, 2002.
67. Lee E.A., Sangiovanni-Vincentelli A. Comparing Models of Computation. Proceedings of ICCAD. San Jose, California, USA. November 10-14, 1996.
68. Maciel P., Barros E., Rosenstiel W. A Petri Net Model for Hardware/Software Codesign. // In Design Automation for Embedded Systems. Vol. 4. October 1999. P. 243–310.
69. Martin G. Productivity in VC Reuse: Linking SOC platforms to abstract systems design methodology. // Forum on Design Languages: Virtual Components Design and Reuse. Lyon, France. August-September, 1999. P. 313–322.
70. Martin G., Chang H., et al. Surviving the SOC Revolution: A Guide to Platform Based Design. Kluwer Academic Publishers. September 1999.
71. Martin G., Sangiovanni-Vincentelli A. A Vision for Embedded Software // Proceedings of the 2001 international conference on Compilers, architecture, and synthesis for embedded systems, 2001. P. 1–7.
72. Mentor Graphics Corporation. URL: http://www.mentor.com.
73. Object Management Group. OMG Unified Modeling Language Specification. June 1999. URL: http://www.omg.org.
74. Patterson D.A., Hennessy J.L. Computer Organization and Design: The Hardware/software Interface. // Morgan Kaufmann. 2005.
75. Peterson J. L. Petri Net Theory and the Modeling of Systems. // Prentice-Hall Inc., Englewood Cliffs, NJ. 1981.
76. SAE AADL: A Society of Automotive Engineers Standard. URL: http://www.aadl.info.
77. Safonov V.O. Aspect.NET: concepts and architecture. // .NET Developers Journal. No. 10. 2004.
78. Sangiovanni-Vincentelli A. Defining platform-based design. // EEDesign. February 2002.
79. Sangiovanni-Vincentelli A. Quo Vadis SLD: Reasoning about Trends and Challenges of System-Level Design // Proceedings of the IEEE, March 2007. 95(3). P. 467–506.
80. Sangiovanni-Vincentelli A., Lee E.A. A framework for comparing models of computation. // IEEE Trans. Computer-Aided Design Integrated Circuits. December 1998.
81. Semiconductor Reuse Standard V2.0. Motorola Inc. 1999.

82. Sgroi M., Lavagno L., Sangiovanni-Vincentelli A. Formal Models for Embedded System Design. // IEEE Design & Test of Computers. April-June 2000. P. 2–15.
83. Shaw M. We can teach software better. // Computing Research News 4(4):2-12. September 1992.
84. Technology Roadmap on Software-Intensive Systems: The Vision of ITEA (SOFTEC Project) // ITEA Office, Eindhoven, March 2001. Chapter 7, "Engineering". P. 47–56.
85. Terekhov A.N. The main concepts of a new HLL computer “CAMCOH” // Computer Science Journal of Moldova. 1993. Vol.1. № 1(1). P. 22-27.
86. Tiwari V., Malik S., Wolfe A. Power analysis of embedded software: a first step towards software power minimization. // IEEE Transactions on VLSI Systems. December 1994.
87. Wirth N. Hardware Compilation: Translating Programs into Circuits. // IEEE Computer. No. 31(6). 1998. P. 25–31.

В 2009 году Университет стал победителем многоэтапного конкурса, в результате которого определены 12 ведущих университетов России с присвоением категории «Национальный исследовательский университет». Министерством образования и науки Российской Федерации была утверждена программа его развития на 2009–2018 годы. В 2011 году Университет получил наименование «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики»

КАФЕДРА ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Кафедра ВТ НИУ ИТМО создана в 1937 году и является одной из старейших и авторитетнейших научно-педагогических школ России.

Традиционно основной упор в подготовке специалистов на кафедре делается на фундаментальную базовую подготовку в рамках общепрофессиональных и специальных дисциплин, охватывающих наиболее важные разделы вычислительной техники.

Кафедра является одной из крупнейших в университете. Учебными курсами и научно-исследовательскими работами руководят 9 профессоров и 16 доцентов. На кафедре обучаются более 500 студентов и 30 аспирантов.

Кафедра имеет собственные компьютерные классы и специализированные исследовательские лаборатории, оснащенные современной вычислительной и оргтехникой, уникальным инструментальным и технологическим оборудованием, измерительными приборами и программным обеспечением.

В 2007-2008 гг. коллективом кафедры была успешно реализована инновационная образовательная программа СПбГУ ИТМО по научно-образовательному направлению «Встроенные вычислительные системы».

Начиная с 2009 года кафедра вычислительной техники является активным участником реализации программы развития национального исследовательского университета ИТМО, вошла в состав крупнейшего в НИУ ИТМО научно-исследовательского центра «Интеллектуальные системы управления и обработки информации».

Алексей Евгеньевич Платунов
Николай Павлович Постников

ВЫСОКОУРОВНЕВОЕ ПРОЕКТИРОВАНИЕ ВСТРАИВАЕМЫХ СИСТЕМ

(часть 1)

Учебное пособие

В авторской редакции
Редакционно-издательский отдел НИУ ИТМО
Зав. РИО Н.Ф. Гусарова
Лицензия ИД № 00408 от 05.11.99
Подписано к печати
Заказ №
Тираж 100 зкз.
Отпечатано на ризографе

Платунов А.Е., Постников Н.П.

ВЫСОКОУРОВНЕВОЕ ПРОЕКТИРОВАНИЕ ВСТРАИВАЕМЫХ СИСТЕМ

(Часть 1)

УЧЕБНОЕ ПОСОБИЕ

Санкт-Петербург

2011

Редакционно-издательский отдел
Санкт-Петербургского национального
исследоватиельского университета
информационных технологий, механики и
оптики
197101, Санкт-Петербург, Кронверкский пр., 49