Крупномасштабное моделирование для встраиваемых приложений

Керри Гранд, Винод Редди, Ген Сасаки и Эрик Диллабер, компания MathWorks

АННОТАЦИЯ

Поскольку спрос на системы повышенной надежности и критически важное встраиваемое программное обеспечение повышается, многие организации приняли модельно-ориентированное проектирование для преодоления проблем, связанных со сложностью проекта, надежностью, качеством и временем выхода на рынок при разработке встраиваемых систем. Масштаб и объем проектов, применяющих модельно-ориентированное проектирование, продолжает быстро увеличиваться, что приводит к исключительно большим и сложным моделям. Как следствие, проектные группы стали больше, тем самым увеличив потребность во взаимодействии и сотрудничестве. Модельно-ориентированное проектирование облегчает параллельную разработку крупномасштабных проектов, позволяя нескольким проектным группам самостоятельно разрабатывать модели, интегрировать их с другими моделями, генерировать код производственного качества и осуществлять верификацию различных компонентов модели в рамках более крупной совместной инфраструктуры.

Для повышения эффективности крупномасштабного моделирования на протяжении всего цикла разработки, нужно глубокое понимание различных этапов и методов, требуемых для успешного применения модельно-ориентированного проектирования. Логичная архитектура разделения модели на компоненты, четкое определение интерфейсов до проектирования компонентов, поддержка этих интерфейсов, генерация кода производственного качества и инфраструктура, поддерживающая разработку – все эти аспекты должны быть учтены. Проект должен использовать одни и те же компоненты модели для проектирования, верификации, автоматической генерации документов и генерации кода производственного качества. Эта статья содержит лучшие практики по созданию инфраструктуры и развертыванию крупномасштабных моделей для встраиваемых приложений с применением модельно-ориентированного проектирования. Данная публикация предназначена тем, кто планирует разворачивать проект с более чем 100000 блоков Simulink и имеет опыт использования MATLAB, Simulink и Embedded Coder.

ВВЕДЕНИЕ

В течение последних нескольких лет многие компании разворачивали высокосложные и крупномасштабные встраиваемые системы, используя модельно-ориентированное проектирование [1, 2]. Благодаря возможности инженеров общаться с использованием моделей, осуществлению частых и быстрых итераций проектирования, а также генерации кода производственного качества, модельно-ориентированное проектирование для больших встраиваемых систем создает как новые возможности, так и вызовы. Определение большой модели является субъективным. Модель считается большой, если она слишком велика для понимания всех её деталей одним человеком, или если количество блоков превышает 100000, или если она содержит более 100 входов и выходов. Модель, считается большой, когда она требует значительных издержек для координации усилий по моделированию между несколькими дисциплинами или командами.

Проектная команда, работающая с большими моделями, предназначенными для встраиваемых приложений, часто подвергается влиянию общих проблем и может извлечь выгоду, используя проверенные в промышленности подходы. Эти проблемы относятся к архитектуре, проекту и реализации. В этой статье мы обсудим каждую из этих областей в деталях и рекомендуем подходы к преодолению этих проблем.

АРХИТЕКТУРА МОДЕЛИ

Архитектура модели является одной из ключевых задач разработки при проектировании крупномасштабных, высокосложных и взаимодействующих систем. Архитектура, прежде всего, определяет системные качества, такие как управляемость, производительность, модифицируемость и проверяемость. Модельно-ориентированное проектирование, фокусирующееся на системной модели [3], приводит к общим для всех вопросам по архитектуре модели, которые включают в себя:

- Как разбить модель на управляемые компоненты, чтобы помочь параллельной разработке и повторному использованию?
- Как определять и управлять согласованным набором интерфейсов?
- Как контролировать исполнение модели?
- Как архитектура модели влияет на тестирование?

В этом разделе мы рассмотрим вопросы, касающиеся разбиения на компоненты, управления интерфейсами, контроля выполнения и влияния архитектуры модели на тестирование.

КОМПОНЕНТИЗАЦИЯ

Компонент - это часть вашего проекта, элемент, модуль или узел, над которым вы можете работать без необходимости в более высокоуровневых частях модели. Вы можете выполнять проектирование на уровне компонентов, симуляцию, тестирование, генерацию кода и верификацию и валидацию. Если компонент является моделью, вы можете запустить его в Simulink как есть; если компонент является атомарной подсистемой в библиотеке, вам потребуется создать модель тестовой обвязки, передающую интерфейсы в эту подсистему. Небольшие элементы, такие, как библиотечные блоки, обычно играют роль вспомогательных функций и на самом деле не являются компонентом проекта (даже если вы проектируете и испытываете их так же, как компонент, они имеют более общий характер и повторное использование может происходить за счет некоторой потери эффективности). Кроме того, разделение модели на компоненты обеспечивает возможность контролировать меньшие участки проекта с использованием системы контроля версий и системы управления конфигурацией системы [4].

В этой статье рассматриваются две архитектурные конструкции: модели-ссылки (model reference) в режиме Accelerator и библиотеки (library), содержащие атомарные подсистемы (atomic subsystem). Хотя модели-ссылки и библиотеки могут использоваться и для других целей, эти конструкции, как правило, обеспечивают большую выгоду для крупномасштабных моделей. Выбор используемой конструкции должен быть основан на компромиссах между преимуществами, которые они предлагают.

Рекомендация 1.

Разбивайте модели верхнего уровня на компоненты с использованием моделей-ссылок в режиме Accelerator.

Модель-ссылка позволяет поместить компонент модели в отдельный независимый файл модели и использовать его как часть более крупной модели. Модели-ссылки создаются с помощью блоков Model, которые могут быть запущены в режиме Accelerator. В режиме Accelerator из модели генерируется код, который затем компилируется в оптимизированный (по отношению к памяти и циклам центрального процессора) формат для симуляции. В результате модели, которые слишком медленно симулируются, могут быть ускорены, если достаточное количество подсистем заменяются блокам Model. Использование модели-ссылки несколько ухудшает производительность при обновлении модели (update diagram), потому что каждая модель-ссылка проверяется на изменения во время инкрементального построения. При использовании одного экземпляра или всего нескольких экземпляров конкретной модели-ссылки, рекомендуется хранить в ней от 500 до 5000 блоков Simulink (как нижний порог). Если используются много экземпляров модели-ссылки, то модель-ссылка может быть еще меньше. Как вариант, для небольших компонентов, которые повторно используются во многих местах в пределах одного и того же файла модели, используют библиотечную модель с повторно используемой (reusable) атомарной подсистемой. Модель-ссылка позволяет пользователям применять систему контроля версий для повторно используемого компонента независимо от моделей, которые его используют. Наконец, когда пользователь обновляет модель Simulink, все блоки модели загружаются в

память. Поскольку модель-ссылка в режиме Accelerator выступает в качестве одного блока Simulink, требования к памяти значительно ниже по сравнению с обновлением атомарной подсистемы, содержащей такие же блоки.

Рекомендация 2.

Используйте атомарные подсистемы в библиотеках для компонентов, содержащих менее 500 блоков.

Как и модель-ссылка, библиотека также позволяют компоненту модели существовать в виде части более крупной модели, и этот компонент также может быть сохранен в виде отдельного файла модели. Тем не менее, библиотеки не могут быть использованы сами по себе. Они требуют родительскую модель и не уменьшают потребление памяти или циклов ЦПУ в обычном режиме симуляции. Как правило, библиотеки предназначены для вспомогательных функций низкого уровня, которые используются несколько раз в проекте. Основная разница между моделью-ссылкой и библиотекой в том, что библиотеки могут быть использованы с различными типами данных, частотой дискретизации и размерностями, в различных контекстах, без изменения проекта. Создание библиотечного компонента в виде атомарной подсистемы обеспечивает группировку блоков в единый модуль исполнения в модели. Таким образом, когда библиотеки используются для компонентов модели, рекомендуется использовать атомарные подсистемы в этих библиотеках. Сгенерированный код опционально может быть помещен в отдельную функцию и исходный файл. Библиотеки позволяют пользователям применять систему контроля версий для компонента независимо от моделей, которые используют его. Тем не менее, поскольку библиотеки являются контекстно-зависимыми и нуждаются в родительской модели для генерации кода, то код, сгенерированный для библиотечной модели, может отличаться для разных экземпляров этой библиотеки. Преимущество этого контекстно-зависимого подхода состоит в том, что библиотеки могут приспособиться к различным интерфейсным спецификациям. Тем не менее, для крупномасштабных компонентов модели это свойство скорее нежелательно, потому что интерфейсы, как правило, управляются и фиксируются, вплоть до определенных типов данных и размерностей.

Обратите внимание, что мы не рекомендуем разбивать модель системы, состоящей из миллиона блоков, на 200 моделей-ссылок, каждая из которых содержит 5000 блоков. Мы рекомендуем использовать модель-ссылку в режиме Accelerator на верхнем уровне модели и смешивать модели-ссылки в режиме Accelerator и атомарные библиотеки на более низких уровнях.

УПРАВЛЕНИЕ ИНТЕРФЕЙСАМИ

Как и в традиционном процессе разработки программного обеспечения, интерфейсы модели должны управляться с помощью централизованного хранилища для сбора информации об интерфейсах, такой, как тип данных, диапазон, описание, начальное значение, размерность и частота дискретизации. Эти интерфейсы существует на границе модели, включая внутренние состояния, параметры и сигнальные шины. Механизмы, используемые для управления интерфейсами, это объекты данных (data objects) Simulink для сигналов, параметров и шин.

Рекомендация 3.

Проектируйте с учетом портируемости и возможности повторного использования с помощью объектов данных Simulink.

Объекты данных Simulink для сигналов и параметров являются экземплярами классов Simulink или MPT (Module Packaging Tool) и существуют в базовом рабочем пространстве MATLAB. При использовании этих объектов данных задаются сигналы, состояния или параметры блока, связывающие информацию внутри самого объекта данных с конкретным компонентом модели. Например, сигнал в Simulink и объект данных MPT с одинаковыми именами имеют общие свойства. В результате, интерфейсными данными можно управлять отдельно от модели, что позволяет вести централизованный архив данных, внешний по отношению к модели - такой как словарь данных, а также улучшать повторное использование модели. Объекты данных для сигналов и параметров могут также описывать классы памяти, которые управляют кодом, генерируемым для объектов данных. Используя инструмент Custom Storage Class Designer (проектирование пользовательских классов памяти), вы можете создавать специализированные программные интерфейсы, такие, как уникальные методы доступа к данным и упаковки. Можно добавлять дополнительные свойства к существующим объектам данных Simulink, а также создавать новые объекты данных.

Мы рекомендуем наследовать пользовательские классы данных от класса MPT и наследовать все настраиваемые классы памяти MPT. Такой подход позволяет создавать новые пользовательские классы памяти или добавлять свойства без изменения установленных продуктов MATLAB/Simulink. Пользовательским классом данных можно управлять независимо от окружения MATLAB/Simulink и использовать, просто добавив расположение папки с классом в путь MATLAB.

Рекомендация 4.

Делайте шины виртуальными за исключением границ моделей-ссылок.

Другая форма управления интерфейсами заключается в использовании шин, которые группируют сигналы или другие шины для облегчения маршрутизации сигналов и снятия с разработчика необходимости управления большим количеством сигналов или интерфейсов. Шины создаются с помощью класса Simulink.Bus. В отличие от сигнала Simulink или MPT, класс Simulink.Bus аналогичен определению типа (typedef) в C. Класс Simulink.Bus только определяет тип структуры и не выделяет память под экземпляр в сгенерированном программном обеспечении. К Simulink.Bus не могут быть добавлены дополнительные свойства и определение класса памяти должно исходить от объекта данных Simulink или MPT. Шины также могут быть виртуальными, это означает, что они не влияют на сгенерированный код за исключением случаев, когда они пересекают границу модели-ссылки. Для модели-ссылки виртуальные шины преобразуются в невиртуальные шины; когда код генерируется с использованием Embedded Coder, такой интерфейс представлен в виде C структуры. За исключением модели-ссылки [5], где требуются невиртуальные шины, рекомендуется оставлять шины виртуальными, что позволит применить дополнительные оптимизации при генерации кода для сигналов. Атомарные подсистемы, которые преобразуются в модель-ссылку, потребуют конвертации шин в невиртуальные на границах подсистем.

Мы не рекомендуем помещать каждый интерфейс компонента в шину для облегчения маршрутизации сигнала. Нахождение баланса между количеством шин и отдельными входами/выходами является задачей проектирования интерфейса. Большая шина может оказаться нежелательной, если:

- излишне скрывает требования к интерфейсу
- увеличивает сложность компонентов, потому что шина должна быть упакована и распакована
- отрицательно влияет на оптимизации при генерации кода и пропускную способность, потому что некоторые конструкции моделирования могут привести к копированию шины

Сотни отдельных сигналов могут увеличить количество аргументов функции в сгенерированном программном обеспечении. Как результат, модель становится трудной для понимания из-за большого числа интерфейсов, и операции над смежными данными (в непрерывном регионе памяти) становятся невозможными. Как правило, шины имеют смысл там, где существует высокая связанность с компонентами модели.

Рекомендация 5.

Делайте интерфейсы ваших моделей явными за счет минимизации или ликвидации глобальных хранилищ данных, глобальных событий и глобальных блоков Goto/From.

Крупномасштабные модели, содержащие компоненты, часто используют одну из двух стратегий для обмена данными: глобальные данные или передача сигналов через всю модель. Обмен глобальными данными в Simulink, как правило, осуществляется за счет использования глобальных хранилищ данных и глобальных блоков Goto/From. В Stateflow, глобальные события могут быть вызваны с использованием функции global events. Основным недостатком использования глобальных данных является то, что источник и назначение становятся неявными, что усложняет отладку проекта, поскольку поток сигналов не представлен в модели. Между тем, соединения и обмен данными посредством сигналов представляют интерфейсы в явном виде. Мы рекомендуем делать интерфейсы модели явными за счет использования маршрутизации сигналов, при этом учитывая лучшие практики для хранилищ данных [6]. Явная маршрутизация сигналов дает дополнительное преимущество за счет уменьшения усилий, необходимых для конвертации компонента в модель-ссылку, поскольку глобальные блоки Goto/From и глобальные события Stateflow не могут пересекать границу модели-ссылки.

УПРАВЛЕНИЕ ВЫПОЛНЕНИЕМ

Рекомендация 6.

Используйте планирование при помощи вызова функций (function-call), если цель состоит в том, чтобы модель соответствовала существующей архитектуре программного обеспечения. В противном случае, позвольте Simulink определить наилучший порядок, исходя из анализа зависимостей данных.

Simulink предлагает разнообразные механизмы, позволяющие легко указать несколько скоростей (частот) выполнения задач, а также механизмы, применяющиеся для защиты данных. Синхронные компоненты обычно моделируются с помощью встроенных возможностей планировщика Simulink. Асинхронные компоненты обычно моделируются с помощью механизма вызова функций. Когда цель состоит в привязке компонента к существующей архитектуре программного обеспечения, обычно применяется планирование при помощи вызовов функций. Кроме того, блоки могут быть сгруппированы в атомарные подсистемы или модели-ссылки и явно вызываться при помощи механизма вызова функций или из диаграммы Stateflow для моделирования стандартных или существующих встраиваемых планировщиков. Такой подход предлагается для обеспечения соответствия архитектуры модели и существующей архитектуры встраиваемого программного обеспечения. На рисунке 1 показан подход с использованием подсистемы вызова функций (Function-call subsystem) для симуляции встроенного планировщика. В этом примере три периода выполнения (1 мс, 10 мс и 100 мс) были сгенерированы из блоков Function-Call Generator (генератор вызова функции). Учитывая полученное дерево вызовов для каждого периода выполнения, 1 мс планировщик вызывает Component_1 и Component_2, 10 мс планировщик вызывает Component_3 и Component_4, а 100 мс планировщик вызывает Component_5 и Component_6. Асинхронные периоды также могут быть смоделированы.

Преимуществом подсистем function-call является то, что обычно возникающие алгебраические петли могут быть автоматически решены путем указания порядка выполнения, что исключает необходимость добавления блока задержки в петле обратной связи. Тем не менее, следует соблюдать осторожность, чтобы избежать нарушения зависимостей данных в окончательной архитектуре модели [7]. Нарушения зависимостей данных происходят, когда сигнал Simulink не является актуальным до выполнения вызова подсистемы function-call или модели-ссылки. Хотя архитектура вызова функций обеспечивает прямое отображение между моделью и кодом, в этом часто нет необходимости, поскольку Simulink может определить порядок выполнения и обеспечить взаимодействие различных периодов выполнения через блоки Rate Transition.

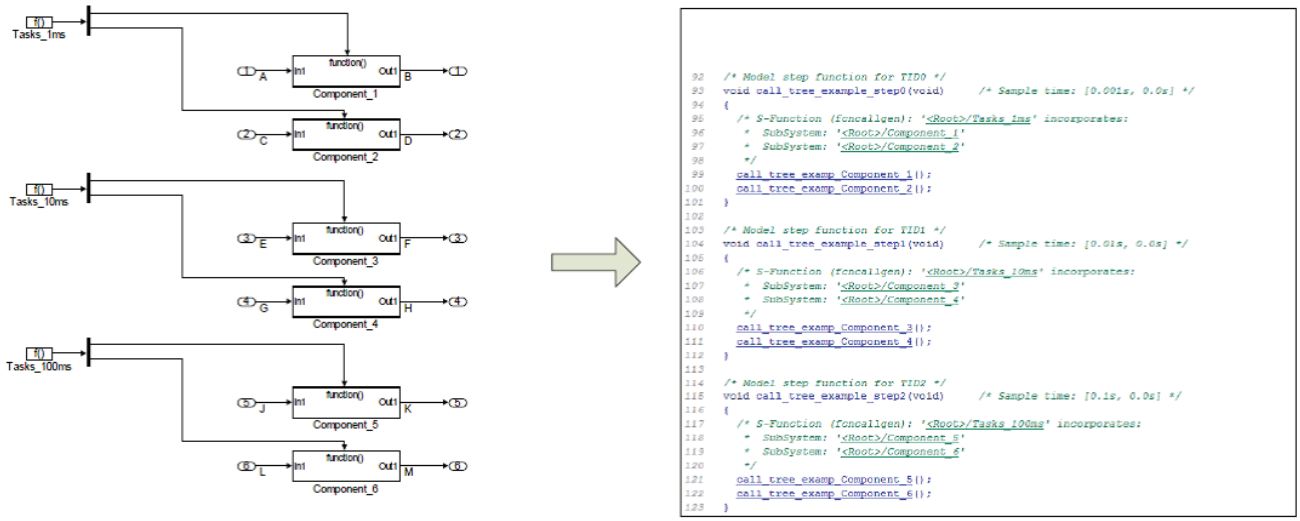

Рисунок 1. Управление выполнением подсистем с использованием блоков Function-Call Generator.

Рекомендация 7.

Избегайте применять настройки приоритета блоков для управления порядком выполнения в Simulink.

Simulink позволяет точечно управлять порядком выполнения, используя настройку приоритета блока, чего следует избегать по двум причинам. Во-первых, когда блок копируется, настройки его приоритета также копируются, что может привести к непреднамеренному порядку выполнения. Во-вторых, порядок выполнения блока, заданный приоритетом, должен быть осуществим (достижим) в Simulink.
В противном случае, приоритет будет проигнорирован с предупреждением или ошибкой, как это определено в конфигурации модели. Отладка этого может занять много времени.

Наследуемые и заданные частоты дискретизации представляют собой еще одну заботу. Для увеличения возможности повторного использования компонента, предпочтительно применять механизм, при котором Simulink распространяет частоты дискретизации от исходного блока к последующим блокам путем наследования. Этот механизм может использоваться путем указания наследуемой частоты дискретизации, т.е. (-1). Будьте внимательны к фильтрам и другим блокам с динамикой (дискретные блоки), потому что они, возможно, потребуют выполнения с частотой, отличной от их исходной, для обеспечения стабильности фильтра.

Сортировка блоков Simulink предполагает, что ничто не выполняется одновременно на нескольких процессорах, и что каждая частота прерывают другую обработку. Так что не может быть условий, при которых две или больше частей модели являются активными одновременно. В результате, симуляция в настоящее время не использует Parallel Computing Toolbox.

ВЛИЯНИЕ АРХИТЕКТУРЫ НА ТЕСТИРОВАНИЕ

Рекомендация 8.

Проектируйте архитектуру таким образом, чтобы инженеры могли самостоятельно проверять свои зоны ответственности в предметной области модели.

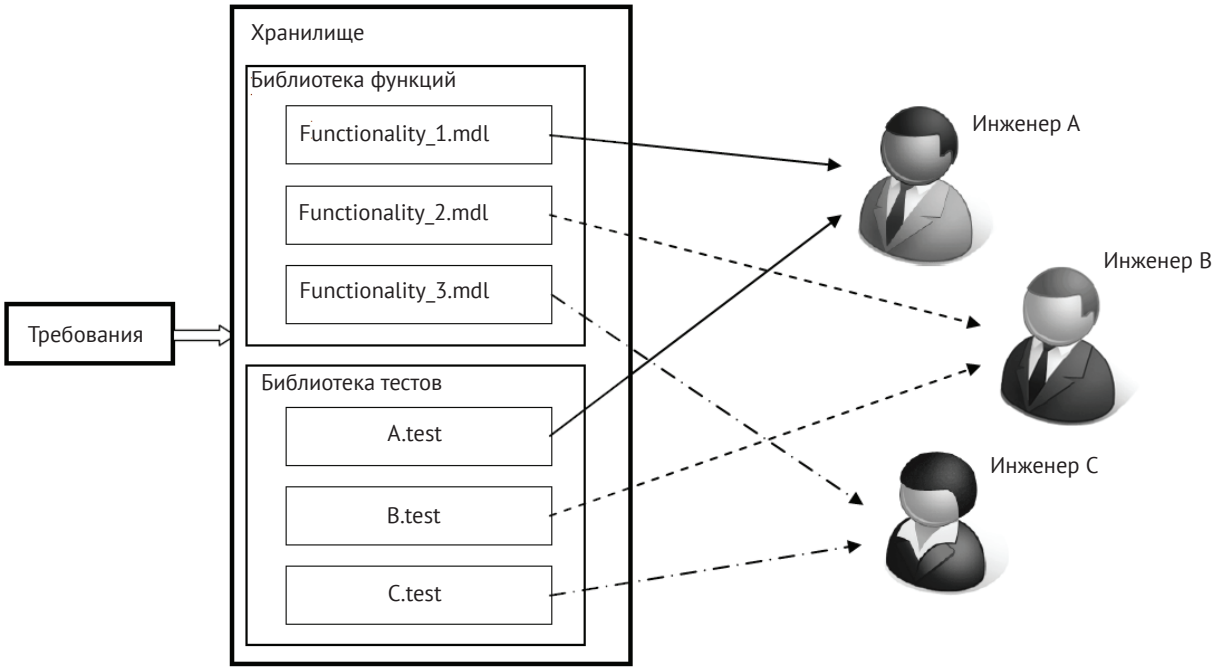

Рисунок 2. Модули должны быть разделены, чтобы позволить инженерам обращаться к ним без конфликтов.

При проектировании масштабной модели должен приниматься во внимание способ организации команды. Когда большая модель управляется несколькими инженерами, инженеры часто несут ответственность за отдельные участки модели. Если ответственность за тестирование также разделена, то инженеры должны тестировать только те части модели, которые относятся к их ответственности, как показано на рисунке 2. Рисунок 2 показывает архитектуру, при которой инженеры не конфликтуют при тестировании своей функциональности, потому что каждый инженер имеет права на сохранение (check-in) только своих файлов. Например, инженер A может получить (check-out) Functionality_1.mdl и A.test из хранилища, выполнить тест, добавить какие-то изменения в .mdl файл и сохранить файлы обратно в хранилище. Рассмотрим случай, когда A.test также проверяет функциональность в Functionality_2.mdl. Инженер может обнаружить проблемы с Functionality_2.mdl. Если оба инженера, A и B, внесли коррективы в Functionality_2.mdl, то будет конфликт при сохранении. Один из способов избежать такого конфликта - это запретить инженеру B тестировать, пока тестирует инженер A, но такой подход приводит к потерям производительности. Идеальным решением является такая архитектура модели, при которой нет никакого дублирования в собственности. Если перекрытия нельзя избежать, следует использовать систему контроля версий, чтобы избежать конфликтующих или потерянных изменений в моделях или тестах.

Рекомендация 9.

Внедряйте последовательный метод инспекции и записи сигналов в архитектуре модели.

Критерии успешности/провала при тестировании компонентов оцениваются с помощью мониторинга значений сигналов или состояний модели. Архитектура модели должна позволять измерять эти значения, что особенно важно, если модель имеет глобальные данные, которые перезаписываются многими функциями. Модель должна быть построена таким образом, чтобы значения сигнала могли быть записаны в определенных точках, и чтобы значения входных тестов могли быть введены в модель для симуляции и тестирования.

Механизм осуществления ввода сигнала и регистрации сигнала одинаков, в независимости от размера модели. В крупномасштабной модели многие задачи, связанные с тестированием, должны быть в значительной степени автоматизированы, чтобы инженер мог быстро создавать, выполнять и анализировать множество тестов. Поэтому важно, чтобы методы ввода и записи сигналов реализовывались согласованно для всех файлов модели.

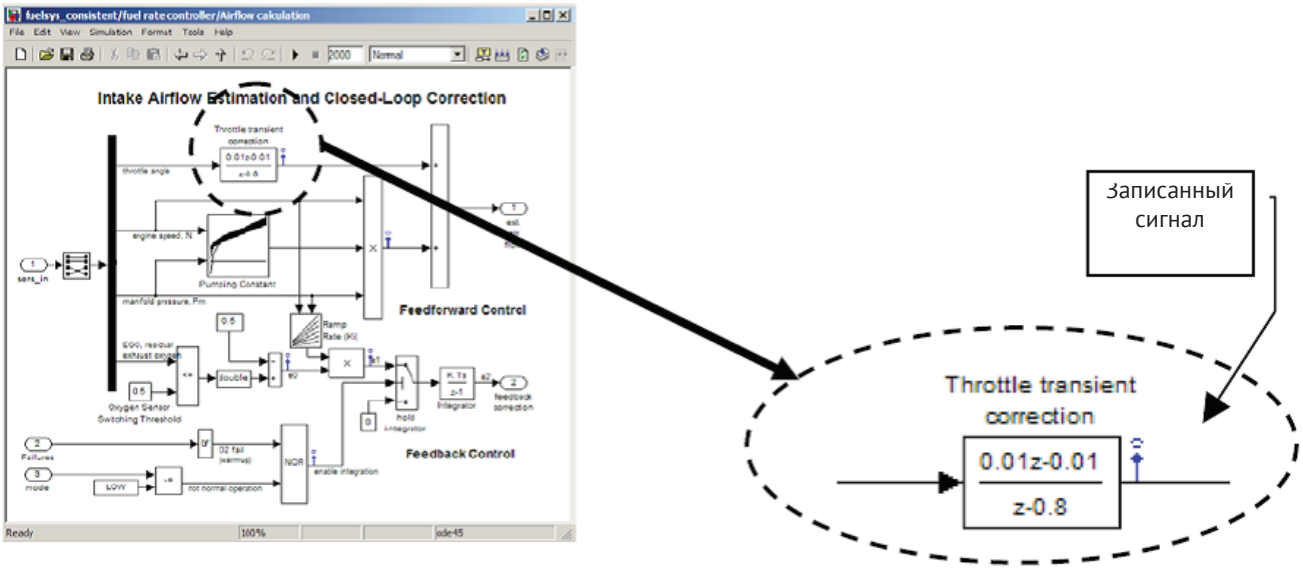

Рисунок 3. Последовательно наблюдаемая и контролируемая подсистема.

Рисунок 3 показывает модель, в которой захват промежуточных сигналов (не портов входа/выхода модели) реализуется с помощью записи сигналов. Ввод сигналов осуществляется исключительно за счет использования входных портов модели. Наличие такого последовательного метода позволяет инженеру правильно назначать тестовые входные данные и записанные данные в инструментах тестирования, таких как SystemTest.

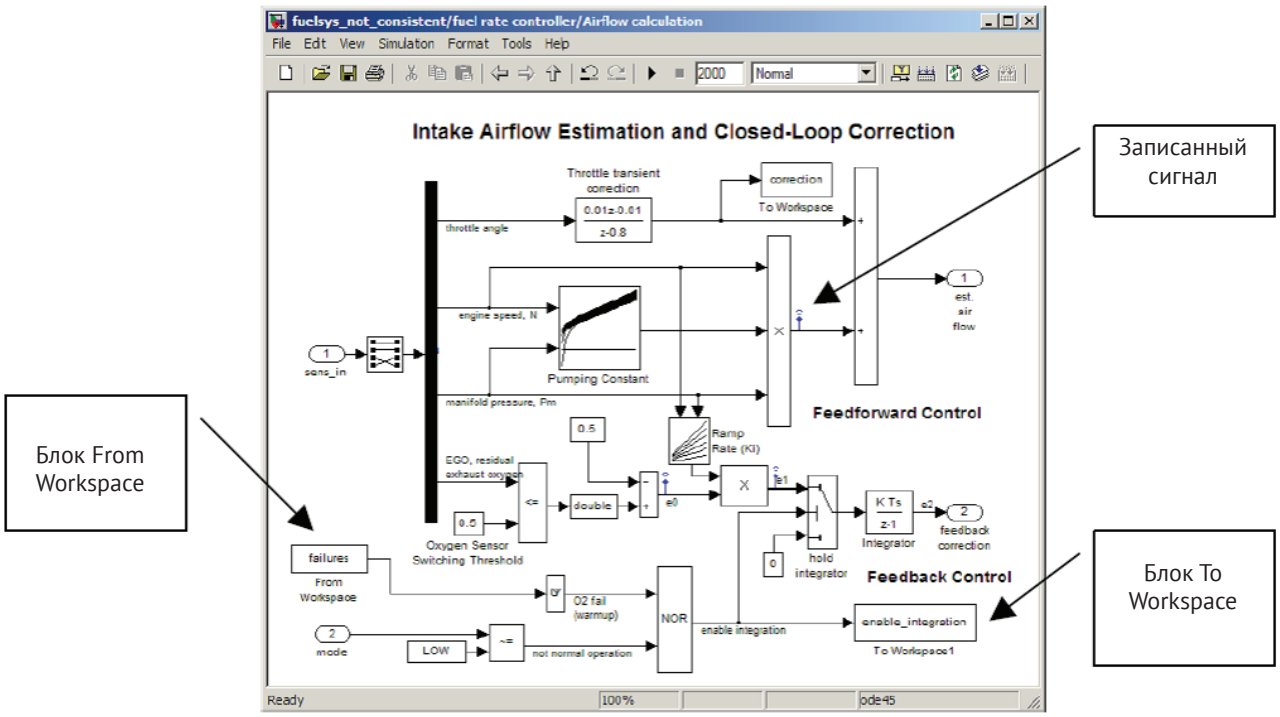

Рисунок 4. Непоследовательно наблюдаемая и управляемая подсистема.

На рисунке 4 сбор данных осуществляется с помощью записанных сигналов и блоков To Workspace. Ввод сигналов для имитации отказов осуществляется блоком From Workspace, а другие входы являются входными портами модели. Хотя такая конструкция модели вполне допустима, настройка этих разрозненные методов в инструменте тестирования, скорее всего, потребует ручного вмешательства, и, следовательно, будет подвержена влиянию ошибок. Кроме того, не рекомендуется использовать блоки To и From Workspace в модели компонента. Ввод и запись сигнала посредством блоков To и From Workspace должны осуществляться вне модели с использованием тестовой обвязки.

ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ

Слишком часто соображения, касающиеся проектирования и реализация при совместной разработке, не до конца понимаются - или реализуются только в конце этапа разработки, что приводит к переделыванию. В данном разделе рассматриваются четыре области проектирования и реализации, которые оказывают влияние на крупномасштабное моделирование для встраиваемых приложений: последовательная среда для моделирования, поддерживающая совместную работу, модульность модели, выбора языка моделирования (Simulink, Stateflow и Embedded MATLAB) и выбор подхода к генерации кода производственного качества.

ПОСЛЕДОВАТЕЛЬНАЯ СРЕДА ДЛЯ МОДЕЛИРОВАНИЯ

При развертывании кода из большой модели чрезвычайно важно, чтобы совместная среда моделирования содержала последовательные настройки окружения MATLAB, рабочего пространства MATLAB, настройки конфигурации модели и настройки встраиваемого целевого вычислителя. Неспособность поддерживать согласованные параметры, как правило, приводит к ошибкам компиляции или времени выполнения после генерации окончательного кода для целевого вычислителя.

Рекомендация 10.

Создавайте согласованную среду моделирования с помощью startup.m, кэширования загрузки MAT-файлов, ссылок на
конфигурации модели, sl_customization.m и целевого системного файла (system target file, STF).

Рекомендация 11.

Спроектируйте модели и скрипты для поддержки параллельного построения моделей-ссылок с использованием Parallel Computing Toolbox в режиме Accelerator и при генерации кода.

Файл startup.m, помещенный в папку запуска MATLAB, выполняет команды при запуске MATLAB, что позволяет добавлять необходимые пути к папкам в пути MATLAB с помощью команды addpath. Поскольку этот метод поддерживает текущий сеанс MATLAB, он может быть использован с несколькими проектами. Этот файл запуска также следует использовать с параллельной симуляцией и генерацией кода, поскольку пути MATLAB должны быть согласованы для локальных рабочих процессов MATLAB в Parallel Computing Toolbox.

Рабочее пространство MATLAB должно быть последовательным при разработке одного компонента или при генерации производственного кода для конечного целевого вычислителя. Объекты рабочего пространства MATLAB обычно включают в себя объекты данных для сигналов и параметров, объекты конфигурации модели и параметры. Здесь могут возникнуть две проблемы. Во-первых, используемые файлы с параметрами для разработки компонентов и модели системы могут быть противоречивыми. Во-вторых, загрузка нескольких файлов MATLAB (обратите внимание, что файлы MATLAB являются текстовыми, а MAT-файлы бинарными) с большими массивами данных может занять значительное время и память системы. Скрипт для кэширования загрузки [8] может использоваться для решения этих двух вопросов. Когда объекты рабочего пространства MATLAB становятся необходимы, механизм кэширования загружает бинарный файл MAT, если его метка времени новее, чем файла MATLAB; загрузка бинарного файла может улучшить время загрузки и снизить потребление памяти.

Для полноты описания, здесь следует также упомянуть о концепции рабочего пространства модели. Хотя для управления данными чаще всего используется только рабочее пространство MATLAB, базовое рабочее пространство не поддерживает архитектуру разделения доступа к данным, поскольку каждая модель имеет доступ к данным рабочего пространства MATLAB. Могут также существовать другие требования, например, что данные должны быть инкапсулированы в модели из-за управления конфигурацией. В таких случаях может применяться рабочее пространство модели. Рабочее пространство модели позволяет указать один из трех источников данных: файл модели, бинарный MAT-файл или текстовый файл MATLAB. Следует отметить, что, хотя рабочее пространство модели эффективно применяется для обработки локальных данных в модели, объекты данных Simulink должны быть перемещены в рабочую область MATLAB перед генерацией кода. Simulink также предлагает методы управления данными: Simulink.saveVars используется для сохранения переменных рабочего пространства в файл MATLAB, а Simulink.findVars используется, чтобы обнаружить, какие переменные из рабочего пространства используются в модели.

Конфигурация модели - это набор значений параметров модели, в том числе тип решателя и время начала и остановки симуляции. По умолчанию, конфигурация хранится в самой модели. Когда возникают различия между конфигурациями компонентов, процесс генерации производственного кода может быть прерван с ошибкой. Проблему разнородных конфигураций можно легко решить с помощью ссылки на конфигурацию, которая определяется в модели и автономной конфигурации, которая находится в базовом рабочем пространстве MATLAB. Автономные конфигурации могут быть созданы путем создания объекта Simulink.ConfigSet. Используйте как минимум две автономных конфигурации для включения компонентов с одним и несколькими экземплярами модели-ссылки. В большинстве случаев, эти конфигурации похожи, за исключением настройки «Total number of instances allowed per top model» (Общее количество экземпляров, допустимое для модели верхнего уровня).

Руководства по стилю моделирования создают основу для успеха проекта и совместной работы над проектами, которые проводятся как внутри компании, так и с партнерами и субподрядчиками [9]. Рекомендуется, чтобы руковод-

ства по стилю моделирования были разработаны и применялись на уровне проекта или компании. Также желательно создание пользовательских настроек для интерфейса Simulink и подключения к внешним инструментам. Файл sl_customization.m в Simulink может регистрировать и применять различные настройки, включая пользовательские проверки моделей.

Целевой системный файл (system target file, STF) обеспечивает контроль над стадией генерации кода, процессом сборки и представлением целевого вычислителя для конечного пользователя. Он обеспечивает:

• Определения переменных, которые имеют основополагающее значение для процесса сборки, например, формат сгенерированного кода
• Главную точку входа в TLC программу верхнего уровня, которая генерирует код
• Информацию о целевом вычислителе, отображаемую в System Target File Browser
• Механизм для задания настроек генерации кода, специфичных для целевого вычислителя
• Механизм для наследования параметров от другого системного целевого файла

Как минимум, настройки генерации кода, специфичные для целевого вычислителя, должны быть зафиксированы, чтобы обеспечить последовательный процесс сборки. Это может быть легко осуществлено с использованием пользовательского STF, который наследует свойства от Embedded Real-Time (ERT) STF файла из Embedded Coder.

ВЫБОР ЯЗЫКА МОДЕЛИРОВАНИЯ

Высококачественный проект программного обеспечения сводит к минимуму время, необходимое для создания, изменения, тестирования и поддержки программного обеспечения, при этом достигая приемлемой производительности времени выполнения. Каждый проектное решение должно приниматься в контексте всей системы и должно находить точное описание в языке проектирования. Семейство продуктов MATLAB предоставляет богатый набор вариантов моделирования - Simulink, Stateflow и Embedded MATLAB - для проектирования, симуляции и генерации кода производственного качества для систем, содержащих компоненты из разных инженерных областей. Каждый из этих вариантов моделирования подходит для выражения определенных аспектов проектирования, и неправильный выбор, будучи полностью функциональным, может привести к низкому качеству проекта. Хотя не оптимальный проект может не привести к проблемам целостности проекта в малой системе, это приведет к значительному бремени в масштабной системе, требуя нежелательных изменений на более поздних стадиях проекта.

Рекомендация 12.

Используйте инструменты:
- Simulink для потоков сигналов и систем управления с обратной связью
- Stateflow для комбинаторной логики, планировщиков и конечных автоматов
- Embedded MATLAB для матричных вычислений и вычислений в одну строку

Simulink идеально подходит для проектирования алгоритмов управления, а Stateflow идеально подходит для комбинаторной логики, планировщиков и машин состояния. Хотя вы могли бы использовать только один способ моделирования, чтобы описать весь проект, это может привести к не интуитивному проекту, который имеет неприемлемую производительность во время выполнения и который трудно проверять и поддерживать. Например, Simulink содержит блоки, которые называются, «For Iteration Subsystem» и «While Iterator Subsystem» для реализации простых циклов for и while, соответственно. Если у вас есть более сложные циклы, такие, как вложенные циклы или те, в которых индекс цикла изменяется, то это проще реализовать, используя Stateflow.

Embedded MATLAB полезен, когда вы получаете ваши алгоритмы из существующего кода MATLAB или при разработке алгоритмов, использующих матричные вычисления. Некоторые продвинутые вычисления более естественно выражаются в виде уравнений или процедурного языка, а не в виде блок-схемы. Вы могли бы реализовать уравнения в Simulink, используя несколько блоков умножения и деления, но лучше просто ввести уравнение в одну строку, в результате чего улучшается читаемость и поддерживаемость. Embedded MATLAB облегчает миграцию алгоритмов из кода MATLAB в Simulink. Для улучшения тестируемости, масштабируемости, поддерживаемости и производительности проекта, мы рекомендуем, тем не менее, перенести некоторые участки или весь код MATLAB в

конечном счете в Simulink и Stateflow. Например, в целях отладки очень просто записать промежуточный сигнал в Simulink, создав тестовую точку (test point). Такой подход не требует изменения модели, а достаточно только изменить свойства сигнала.

ЭКСПОРТ АЛГОРИТМА И ПОЛНАЯ ПОДДЕРЖКА ЦЕЛЕВОЙ СИСТЕМЫ

Традиционные мероприятия модельно-ориентированного проектирования, включающие масштабное моделирование для генерации кода производственного качества, используют либо экспорт алгоритма или построение целиком исполняемого файла для системы реального времени. Требуется учесть много аспектов перед тем, как выбрать один из этих подходов, а также при настройке - включая генерацию кода производственного качества (production code generation, PCG). Эти аспекты включают используемое окружение (и существующее программное обеспечение), повторное использование кода для разных целевых процессоров, а также необходимость сборки под ключ.

Рекомендация 13.

Определяйте стратегию развертывания производственного кода при проектировании архитектуры.

Рисунок 5 демонстрирует подход с экспортом алгоритма, использующий генерацию производственного кода. При таком подходе разработчик генерирует код для модели контроллера, выделенный на рисунке. Сгенерированный код C нужно будет интегрировать с операционной системой реального времени (real-time operating system, RTOS) и с другими программными компонентами, написанными вручную, такими, как драйвера устройств ввода и вывода и низкоуровневое встраиваемое программное обеспечение. Процессы компиляции и компоновки управляются с помощью внешней среды сборки или интегрированной среды разработки. Автоматизацию для управления процессами компиляции и компоновки можно осуществить в среде моделирования. Это наиболее распространенный подход к развертыванию производственного кода, особенно когда рукописные драйверы и планировщики уже существует.

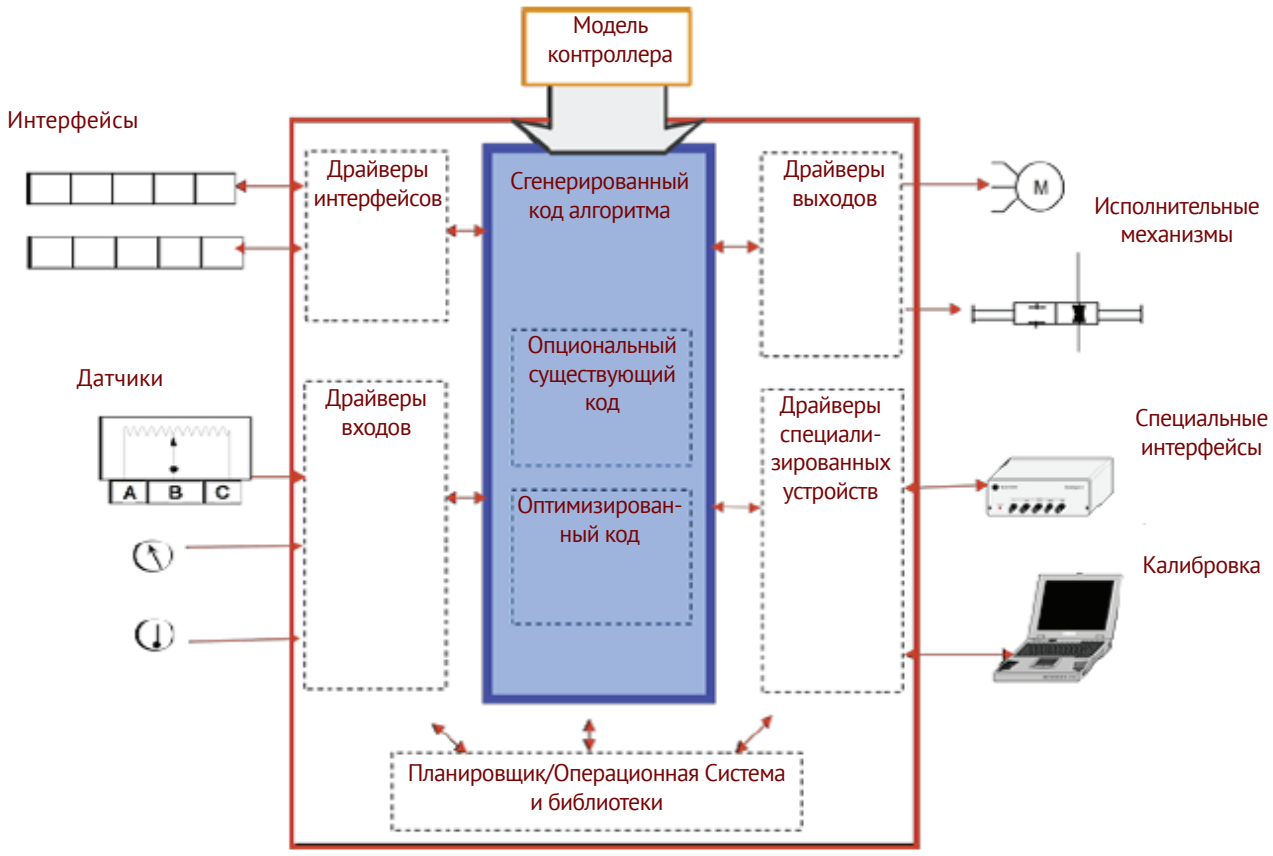

Рисунок 5. Подход с экспортом алгоритма при генерации производственного кода.

Рисунок 6 демонстрирует построение целиком исполняемого файла для системы реального времени, с использованием генерации производственного кода. При таком подходе разработчик будет генерировать код для всей модели контроллера «под ключ». Драйвера входов/выходов и интерфейсы RTOS также можно сгенерировать, но не обязательно, если существует рукописный код. Если создана полноценная поддержка целевой системы, то разработчик имеет полный контроль над конфигурацией своих драйверов ввода/вывода, параметрами сборки и настройками компоновщика - и все это из самой модели.

Еще одно соображение, касающееся сборки под ключ, состоит в усилиях, которые требуются для разработки и поддержания полного окружения генерации производственного кода. Существуют коммерческие готовые решения, устраняющие эти усилия, и такие решения должны рассматриваться в первую очередь. В противном случае, должны быть выделены ресурсы для поддержки развития пользовательского пакета целевой поддержки (target support package), который включает в себя периферийные блоки для симуляции и генерации кода, TLC (target language compiler) файлы для интеграции с RTOS, TLC файлы для профилирования кода, компоненты сборки и требуемые опции по верификации, такие, как Процессор-в-контуре(Processor-in-the-Loop).

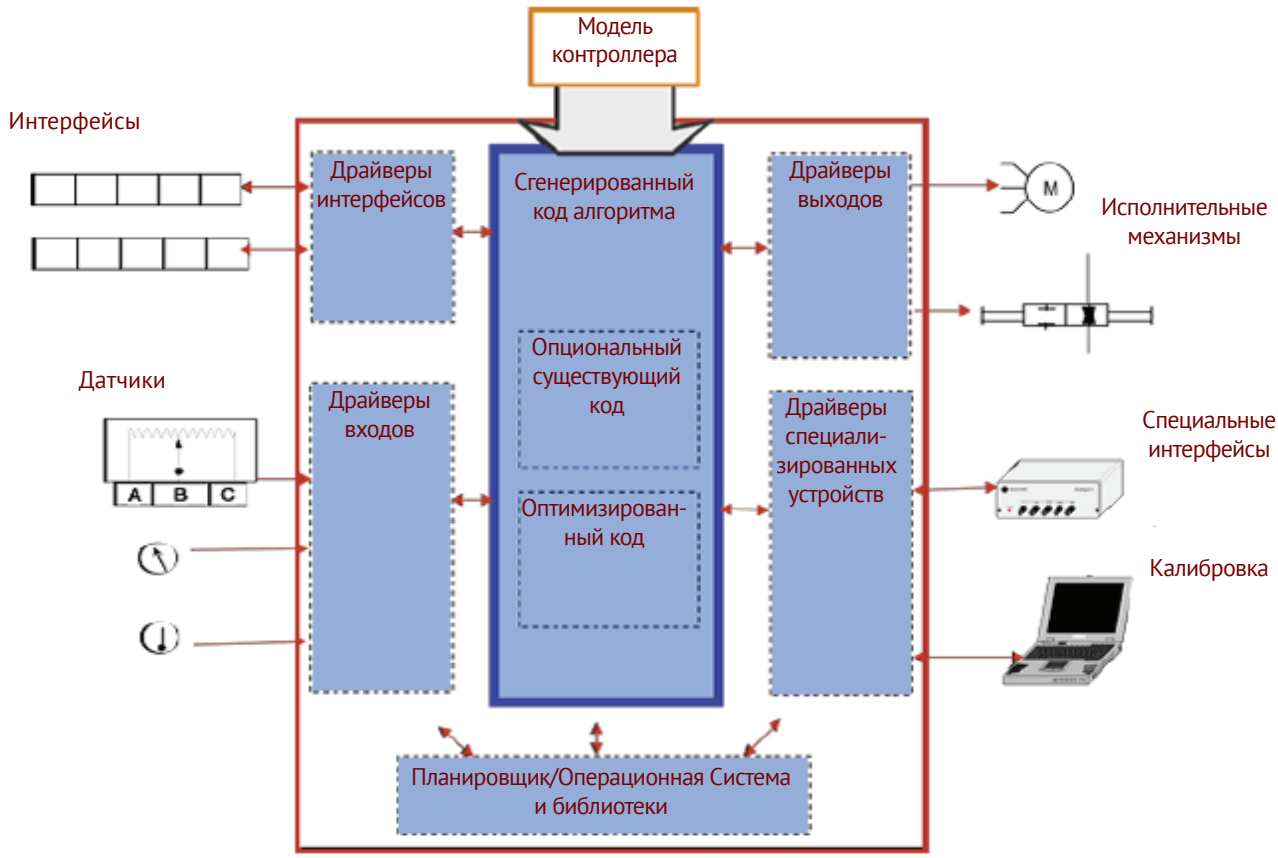

Рисунок 6. Полный исполняемый файл для системы реального времени, с использованием генерации производственного кода.

На рисунке 7 приводятся некоторые типичные решения, которые принимаются при переходе к генерации производственного кода. Подход с экспортом алгоритма обычно выбирают в следующих случаях:

- Повторное использование кода с различными аппаратными платформами или программными архитектурами
- Существование большой базы унаследованного программного обеспечения
- Отсутствие необходимости в полном исполняемом файле под ключ

Настройка наиболее подходящего окружения для экспорта алгоритма под вас потребует дополнительных решений. Во-первых, должно быть принято решение об использовании механизма экспорта функции (Export Functions) или полного построения модели. С экспортом функций, модель подсистемы проектируется при помощи вызовов функций и в сгенерированном коде не присутствуют функции планировщика или функции шага (step). Такой подход позволяет пользователю моделировать функциональность RTOS без влияния на сгенерированный код, а атомарные подсистемы затем используются для указания точек входа программного обеспечения. Далее идет решение о включении унаследованного кода. Если требуется симулировать существующее программное обеспечение, то можно использовать Legacy Code Tool для автоматизации этого процесса, используя программные интерфейсы MATLAB. API-интерфейсы позволят автоматизировать создание блоков за счет использования встроенных (inline) S-функций/файлов TLC, а также за счет создания файла зависимостей сборки (rtwmakecfg.m), используемого для указания исходного кода и библиотек, включённых в процесс построения. Встраиваемые процессоры и компиляторы для них часто имеют специализированные инструкции для поддержки определенных операций, которые используются в типичных встраиваемых приложениях. Такие инструкции для конкретного процессора выполняются гораздо быстрее, чем их эквиваленты на C, и могут существенно улучшить производительность кода. Библиотека подстановочных функций (Code Replacement Library) позволяет генерировать специфичный для процессора код, который использует конкретные инструкции процессора. Если размер модели большой, то специальные настройки позволят использовать функцию параллельного построения Embedded Coder. Пользователь может упаковать сгенерированный код для переноса его в среду сборки с помощью команды packNGo в MATLAB.

Настройка генерации полного исполняемого файла для выполнения в реальном времени также требует дополнительных решений. Во-первых, требует определить, существуют ли коммерческий пакет целевой поддержки для встраиваемого целевого процессора и компилятора. Требуется также принять во внимание поддержку периферийных устройств и требования к компоновке и карте памяти. Если эти коммерческие продукты отвечают требованиям для встраиваемой целевой системы, мы рекомендуем использовать коммерческие продукты. Если нет, то пользователь должен создать пользовательский пакет целевой поддержки, который включает в себя целевой системный файл, блоки для драйверов устройства, шаблон make-файла и другие необходимые файлы для адаптации процесса сборки (hook files) [10]. Кроме того, пользователь должен рассмотреть вопрос о включении устаревшего кода и специализированного кода за счет использования Legacy Code Tool и библиотеки подстановочных функций соответственно. Наконец, есть некоторые простые настройки, позволяющие включить функцию параллельного построения в Embedded Coder, включая определение пути MATLAB, динамического пути Java и рабочего пространства для локальных рабочих процессов MATLAB.

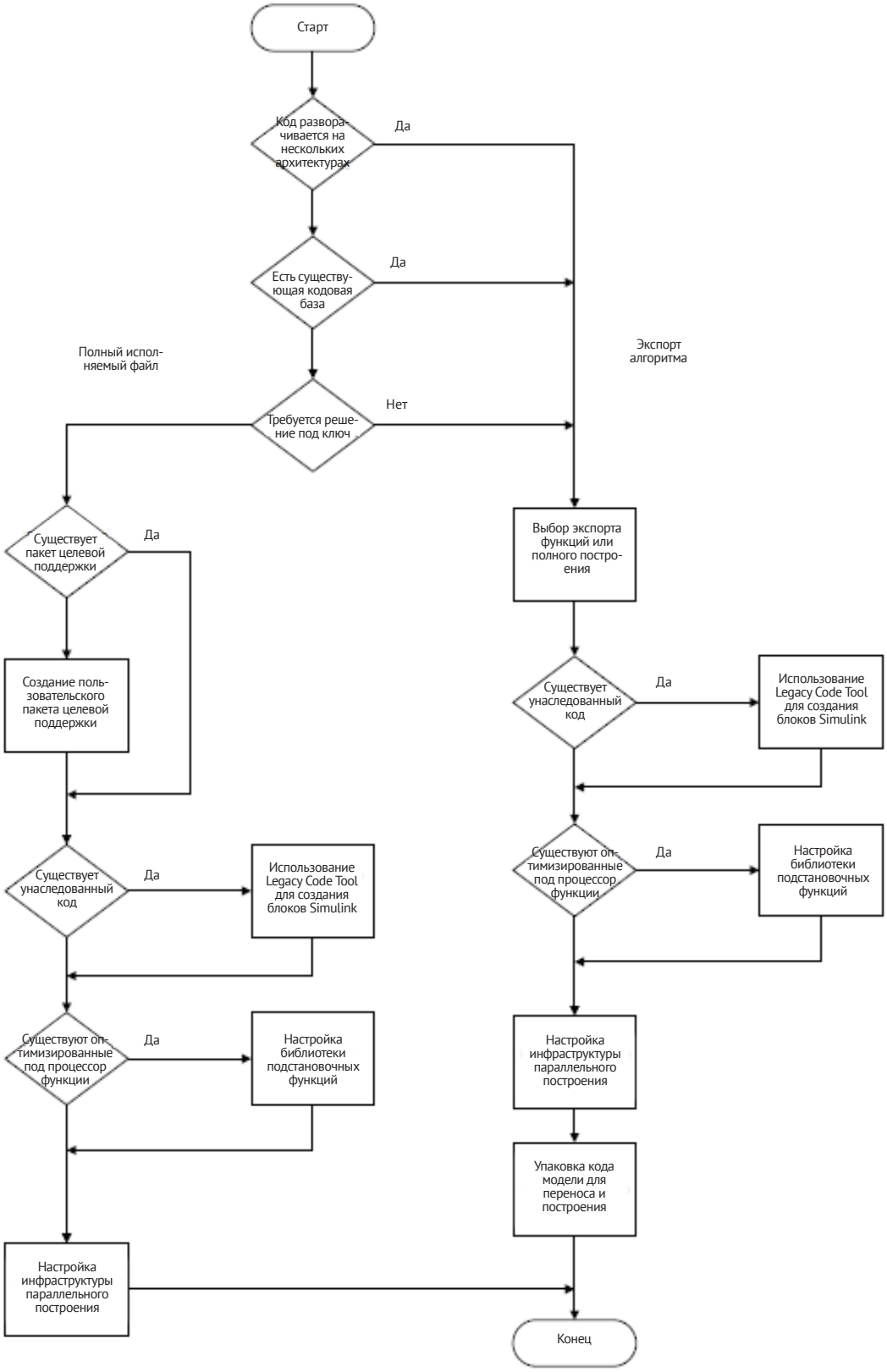

Рисунок 7. Диаграмма принятия решения о выборе экспорта алгоритма или полного исполняемого файла.

ЗАКЛЮЧЕНИЕ/ВЫВОДЫ

Широта и объем проектов, применяющих модельно-ориентированное проектирование для разработки встраиваемых систем, продолжает быстро увеличиваться, что приводит к появлению исключительно больших и сложных моделей. Поскольку инженеры могут общаться при помощи моделей, осуществлять частые и быстрые итерации проектирования и генерировать производственный код, модельно-ориентированное проектирование для больших встраиваемых систем создает как возможности, так и вызовы. Эта статья представляет некоторые проверенные рекомендации на основе накопленного в отрасли опыта крупномасштабного моделирования. Критически важно решить эти проблемы на ранней стадии проектирования, чтобы устранить неэффективное или неоптимальное проектирование модели, не поддерживающее эффективную генерацию кода производственного качества.

Ссылки

1. The MathWorks, "BAE Systems Achieves 80% Reduction in Software-Defined Radio Development Time with Model-Based Design," http://www.mathworks.com, May 2006.
2. The MathWorks, "GM Standardizes on Model-Based Design for Hybrid Powertrain Development," http://www.mathworks.com, May 2009.
3. Smith, Paul, Prabhu, Sameer, Friedman, Jonathan. "Best Practices for Establishing a Model-Based Design Culture," 2007-01-0777, The MathWorks, Natick, MA, 2007.
4. Walker, Gavin, Friedman, Jonathan, Aberg, Rob. "Configuration Management of the Model-Based Design Process," 2007-01-1775, The MathWorks, Natick, MA, 2007.
5. MATLAB Central, "Simulink Modeling: Buses Best Practices," http://www.mathworks.com/matlabcentral/fileexchange/23480, March 2009.
6. MathWorks Technical Solutions, What are Data Store blocks best practices for modeling and code generation using Simulink and Real-Time Workshop Embedded Coder?, June 2009.
7. MATLAB Central, "Two Methods for Breaking Data Dependency Loops in System Level Models," http://www.mathworks.com/matlabcentral/fileexchange/15368, Sept 2009.
8. MATLAB Central, "Fast Parameter Loading for MATLAB/Simulink," http://www.mathworks.com/matlabcentral/fileexchange/14898, May 2007.
9. The MathWorks, "Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow Version 2.1," http://www.mathworks.com/automotive/standards/maab.html, July 2007.
10.The MathWorks, "Real-Time Workshop Embedded Coder 5 - Developing Embedded Targets," http://www.mathworks.com, September 2000.
11.Eric Dillaber, Larry Kendrick, Wensi Jin and Vinod Reddy, The MathWorks, Inc. «Pragmatic Strategies for Adopting Model-Based Design for Embedded Applications,» SAE Paper 2010-01-0935.

ДЛЯ ЗАМЕТОК

Дополнительная информация и контакты

Информация о продуктах
matlab.ru/products

Пробная версия
matlab.ru/trial

Запрос цены
matlab.ru/price

Техническая поддержка
matlab.ru/support

Тренинги
matlab.ru/training

Контакты
matlab.ru
E-mail: matlab@sl-matlab.ru
Тел.: +7 (495) 232-00-23, **доб.** 0609
Адрес: 115114 Москва,
Дербеневская наб., д. 7, стр. 8

MathWorks®
Accelerating the pace of engineering and science