УДК 004.415

А. А. Афанасьев, А. Б. Иванов

Сколковский институт науки и технологий

Разработка надёжного программного обеспечения для малых спутников с одноплатным бортовым компьютером

В этой статье мы представим наш подход для решения проблемы отсутствия строгого и надежного способа разработки программного обеспечения для малых спутников, основанный на фреймворке Behavior-Interaction-Priorities (BIP), а также обсудим возможность его использования на одноплатных бортовых компьютерах и решим возникающие при этом подходе сложности и задачи.

Ключевые слова: малые спутники, строгий подход, BIP фреймворк.

A. A. Afanasev, A. B. Ivanov

Skolkovo Institute of Science and Technology

Robust software design for nanosatellites with single board computer

In this paper, we will present our approach to solving the problem of lacking a rigorous and reliable way to develop robust software for nanosatellites, which is based on the usage of the Behavior Interaction Priorities (BIP) framework. We also discuss a possibility to use it in single board on board computers and resolve difficulties and challenges arising in testing and simulations.

Key words: nanosatellites, rigorous approach, BIP framework.

1. Введение

В современном процессе проектирования спутниковых систем наблюдается тенденция к экспоненциальному росту размеров программного обеспечния для бортовых компьютеров, что в конечном итоге приводит к созданию неэргономичных и чрезмерно сложных интерфейсов. Несмотря на то, что аппаратное обеспечение достаточно стандартизировано в форме так называемых COTS-компонентов (components-off-the-shelf), программное обеспечение всегда зависит от типа полезной нагрузки. Это приводит к дифференциации конструкций космических аппаратов в зависимости от учреждений, ответственных за производство этих систем. Каждый из этих проектов должен пройти процесс проверки, который становится дорогостоящим на поздних стадиях разработки. В качестве иллюстрации, на рис. 1 представлена широко известная V-диаграмма процесса разработки в системной инженерии [1].

Каждый уровень диаграммы соответствует уровню детализации проекта, слева – на этапе проектирования, справа – на этапе реализации и тестов. Исходя из структуры диаграммы, нетрудно заметить, что чем раньше ошибка в архитектуре аппарата будет замечена, тем быстрее и экономнее будет её исправить. Так, например, если ошибка обнаружена

на этапе определения системных требований, исправить её можно переосмыслением концепции плана операции или добавочными требованиями. Если же ошибка обнаружена на этапе тестов или верификации системы, придётся возвращаться обратно на этап дизайна и переделывать процесс разработки заново, что займёт намного большее количество времени и затраченных средств.

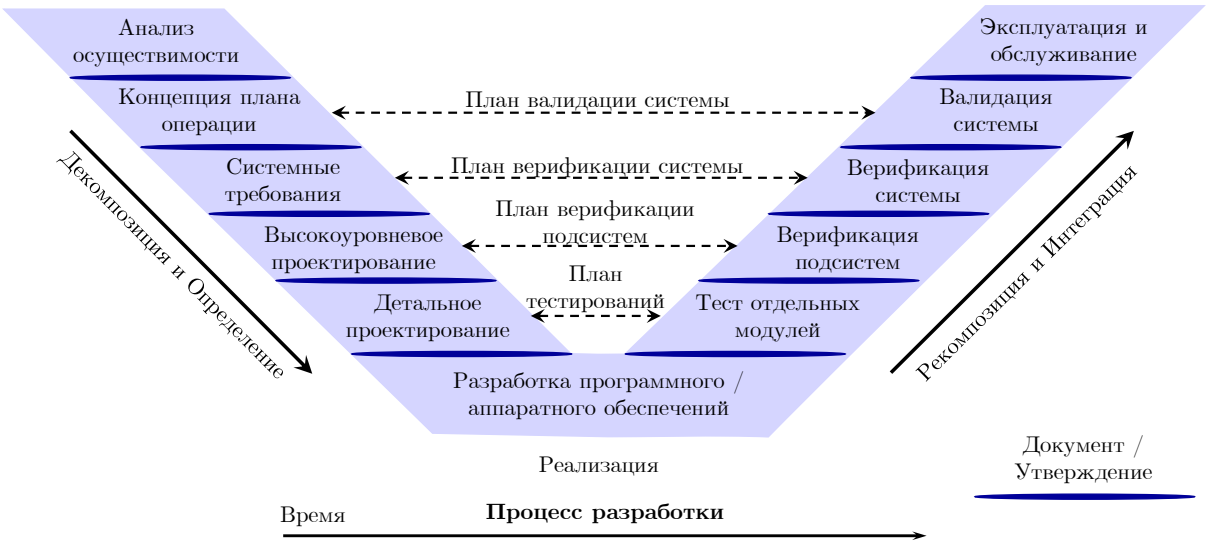

Рис. 1. V-диаграмма системной инженерии

Также стоит отметить специфику проектирования космических аппаратов – невозможно исправить ошибки программного обеспечения в то время, когда спутник уже выведен на орбиту. Это подразумевает использование *строгого подхода* при построении сложных систем. Однако доскональная проверка становится дорогостоящей на последних этапах проектирования.

Разумное решение к поставленной проблеме – указать и проверить системные требования на ранних стадиях проектирования, убедившись в том, что модель работает должным образом (метод correctness-by-construction). Таким образом, строгий подход был введен для построения сложных систем – *architecture-based design*, который интегрирует более простые системы в сложные и проверяет, сохраняют ли они заданные свойства, технические требования и ограничения своих простейших составляющих. Язык моделирования SysML может использоваться для описания системы в целом, а затем для проверки некоторых свойств. Однако он недостаточно строгий, чтобы обеспечить автоматическую верификацию и валидацию программного обеспечения [11].

Поэтому в данной статье предлагается использовать фреймворк Behavior-Interaction-Priorities (BIP) [12]. BIP может использоваться для формального моделирования сложных систем и предоставляет набор инструментов для их проверки, валидации и генерации кода. Также BIP позволяет использовать итеративный подход при проектировании спутников и адаптируется к изменению аппаратного обеспечения.

Мы использовали BIP фреймворк, чтобы показать возможность спроектировать логику работы малых спутников и скомпилировать её в машинный код, который должен выполняться на одноплатном компьютере (например, BeagleBone Black с процессором Cortex-A8). Эта работа доказала свою техническую осуществимость и выявила ряд проблем с выбранным подходом. В этой статье мы представим основные принципы структуры BIP, нашу реализацию и задачи для реализации.

2. Преимущества строгого подхода

Наш подход заключается в использовании формальной структуры валидации для создания и проверки логики системы с последующим переносом кода C++ непосредственно

на микропроцессор. Он обеспечивает следующие качества всей системы (необходимые для спутникового оборудования [13]):

- *Надёжность* – разработчик не должен беспокоиться о роли аппаратного обеспечния при продумывании логики системы. Перед началом работы с драйверами оборудования можно будет смоделировать поведение всей системы, используя установленные шаблоны проектирования. Система также будет проверена на соответствие ряду правил проектирования. Таким образом, можно будет устранить ряд ошибок или неясностей, которые обычно обнаруживаются в конце цикла разработки проекта.

- *Модульность* – различные компоненты единой системы можно верифицировать отдельно перед их интеграцией. Так, например, можно не конкретизировать физический смысл потока данных от различных датчиков, а представить их в качестве циклически повторяющихся пакетов, чтобы проверить использующую их логику операций.

- *Портативность* – система будет иметь аппаратный интерфейс в виде низкоуровневых функций C или C++, которые реализованы для отражения возможностей выбранных компонентов. В случае замены компонентов нам нужно только заменить соответствующий драйвер, сохраняя при этом общую логику системы. Это было бы серьезным улучшением текущего состояния дел, где смена компонентов также требует модификации логики. Такие изменения чрезвычайно дороги, поскольку логика уже реализована в коде C ++ и требует большого количества реверс-инжиниринга.

3. BIP фреймворк

Наш подход основан на фреймворке BIP [14] для компонентно-ориентированного проектирования correct-by-construction приложений. BIP предоставляет простой, но мощный механизм для координации параллельно работающих компонентов, накладывая три уровня: Behavior (Поведение), Interaction (Взаимодействие) и Priority (Приоритет).

Во-первых, поведение компонентов описывается портами, которые обозначают переходы между компонентами, и данными, хранящимися в локальных переменных. Порты также могут экспортировать часть локальных переменных, предоставляя доступ к данным компонента.

Во-вторых, наборы взаимодействий определяют координацию компонентов. Взаимодействия – это наборы портов, которые определяют допустимые синхронизации между компонентами. Модель взаимодействия определяется структурой соединений (связей) [6].

В-третьих, приоритеты используются для наложения ограничений планирования и для разрешения конфликтов, когда несколько взаимодействий разрешены одновременно.

3.1. Характеристики BIP фреймворка

Состояние компонента BIP не может быть напрямую изменено любым другим компонентом. Порты компонентов BIP не являются ни данными ввода/вывода, ни методами, которые должны быть вызваны. Вместо этого порт обозначает *событие*, которое происходит, когда компонент выполняет переход, для которого этот порт является обозначением. Таким образом, соединение обеспечивает синхронизацию таких событий. Механизм BIP решает в каких компонентах должны выполняться какие переходы, основываясь на полной информации о соединениях и приоритетах в системе.

Атомы (Atoms). Системы BIP собираются из атомарных компонентов (атомов), соответствующих параллельно испоняемым процессам, таким как алгоритмы управления, драйверы шины и памяти и т.д. Атомы имеют непересекающиеся пространства состояний. Атом определяется соответствующими наборами портов, состояний, переходов, переменных и update-функциями, которые связаны с переходами.

Для примера возьмём рис. 2 [15], на котором изображена BIP-модель проблемы «обедающих философов» [2]. Проблема формулируется следующим образом:

> *Условие*: N философов сидят вокруг круглого стола, перед каждым философом стоит тарелка с едой. Вилки лежат на столе между каждой парой ближайших философов. Каждый философ может либо есть, либо размышлять. Философ может есть только тогда, когда держит две вилки – взятую справа и слева. Каждый философ может взять ближайшую вилку (если она доступна) или положить – если он уже держит её. Взятие каждой вилки и возвращение её на стол являются раздельными действиями, которые должны выполняться одно за другим.
>
> *Задача*: разработать модель поведения, при которой ни один из философов не будет голодать, то есть будет вечно чередовать приём пищи и размышления.

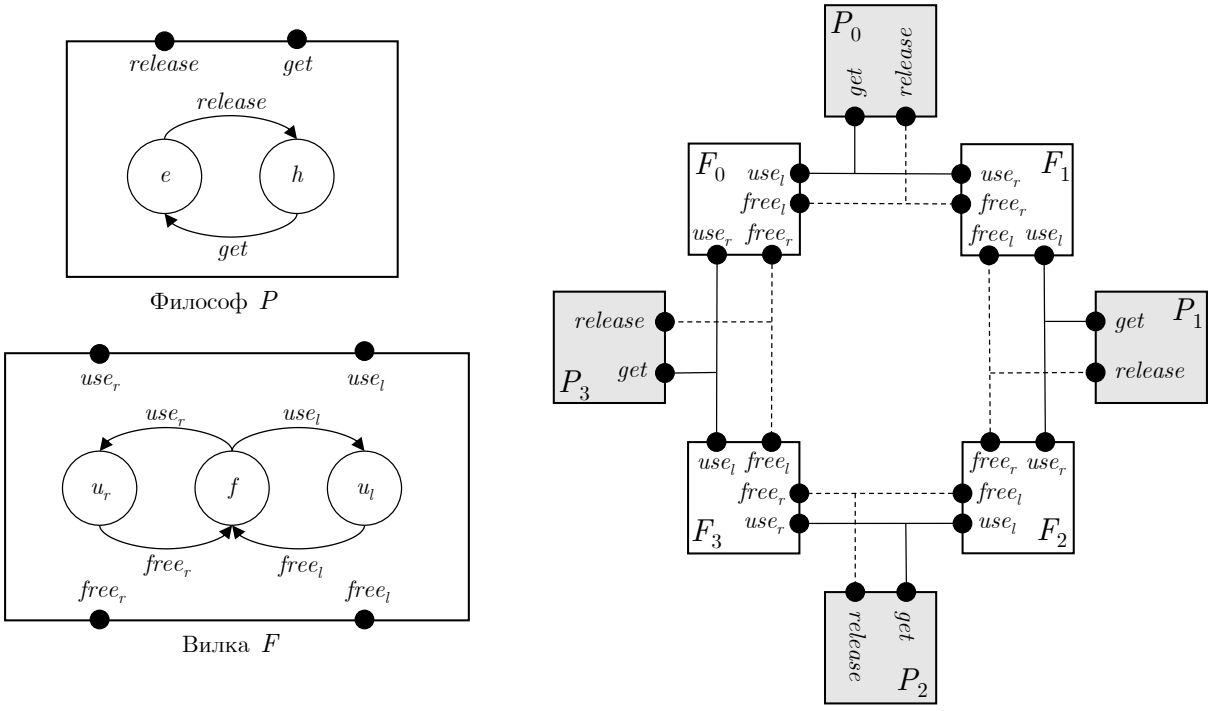

Рис. 2. BIP компонента из четырёх обедающих философов

Данная система состоит из 8 атомов, 4 из которых являются философами P, другие 4 – вилками F. Атом философа P может находиться в 2-х состояниях – e (eating) и h (hungry). Переход из одного состояния в другое отмечен также и в названии соответствующего порта: *release* (положить вилку на стол) и *get* (взять видку со стола). Атом вилки F чуть сложнее и может находиться в 3-х состояниях – u_r (used in right), u_l (used in left) и f (finished). Переходы подчинятюся следующей логике: если вилка свободна, то только тогда её можно использовать одному из философоф (справа или слева). Описание соединений для этого примера в следующем параграфе.

Соединения (Connectors). У системы на рис. 2 имеется 8 соединений, 4 из которых синхронизируют взятие вилок одним из философов (порты use_r, use_l, get), другие 4 синхронизируют возвращение вилок на стол (порты $free_r, free_l, release$). Это означает, например, что переход философа P_0 из состояния h в состояние e обязательно должно сопровождаться переходом вилки F_0 из состояния f в состояние u_l и переходом вилки F_1 из состояния f в состояние u_r (т.е. соответствующие порты переходов синхронизированы).

Соединения определяют набор взаимодействий между компонентами на основе типов портов, которые могут быть либо триггерами, либо синхронами (рис. 3а). Если все соединённые порты являются синхронами, то происходит синхронизация типа *рандеву*, т.е. требуемое взаимодействие может произойти, только если все участвующие компоненты разрешают переход на этих портах (рис. 3б). Если у соединения есть хотя бы один триггер, то синхронизация происходит по типу *широковещания*, т.е. разрешённые взаимодействия – это все непустые подмножества подключенных портов, содержащих по крайней мере один из портов триггера (рис. 3б). Более сложные соединения могут быть построены иерархически (рис. 3в).

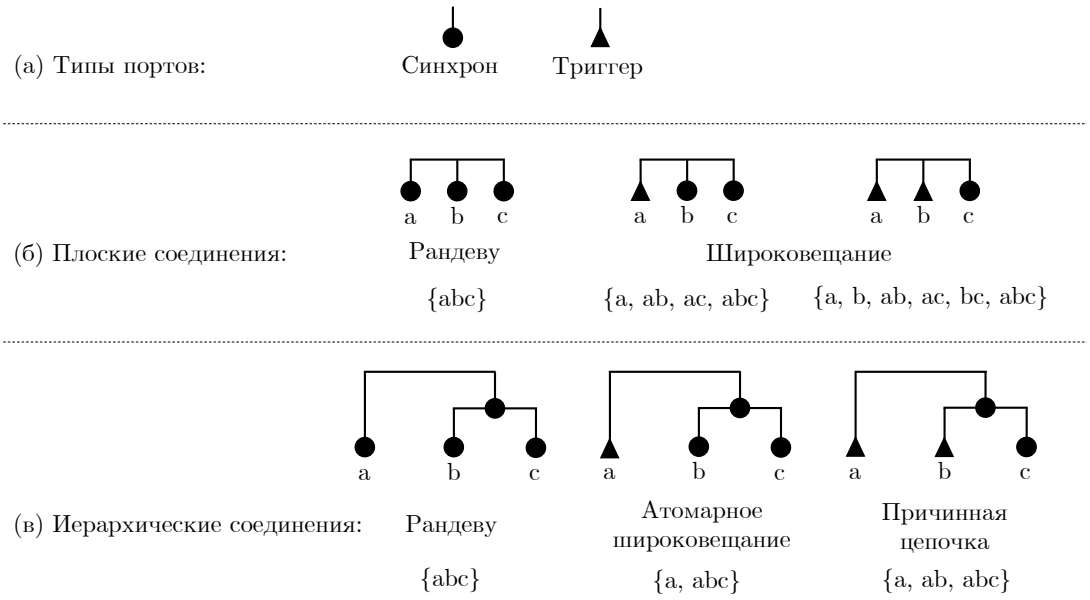

Рис. 3. Типы соединений в BIP фреймворке

Приоритеты (Priorities). На рис. 2 не показан пример использования приоритетов в системе BIP, однако легко представить, зачем они нужны. Если соединители разрешают несколько переходов одновременно с участием одних и тех же портов, можно самостоятельно указать, какой именно переход должен исполняться сначала. Иначе говоря, можно установить приоритет взаимодействия. В общем случае нет необходимости устанавливать приоритеты во всех конфликтных ситуациях: согласно семантике BIP, одному из возможных взаимодействий выбирается максимальный приоритет недетерминированным путём [6].

Комплексы (Compounds). Наконец, комплексные компоненты состоят из наборов подкомплексов (атомов и / или комплексов), соединений и приоритетов. Комплекс может экспортировать порты, определенные в соединениях, для взаимодействия с другими компонентами в большем комплексе. Так, на рис. 2 4 философа и 4 вилки, соединённые в одну компоненту, являются комплексом.

3.2. Обеспечение свойств – Архитектуры

Чтобы преодолеть проблему экспоненциального роста пространства состояний при апостериорной проверке системы, нами будут использоваться *архитектуры* – шаблоны проектирования, которые так ограничивают поведение набора компонентов, чтобы поведение составвных компонент (комплексов) соответствовало заданному свойству (т.е. сохраняло его).

Известно, что в задаче об «обедающих философах» основная цель – избежать взаимной блокировки, например, когда все N философов будут сидеть с вилкой в левой руке и ждать, пока кто-нибудь не положит на стол свою. Модель на рис. 2 не решает данную

проблему. Один из способов решения проблемы – добавление координатора, который будет сохранять очерёдность питающихся философов. Так, на рис. 4 показана простая BIP модель для взаимного исключения между двумя задачами [9]. Он состоит из двух компонентов, моделирующих собственно задачи, и одного компонента-координатора C. Четыре бинарных соединения синхронизируют каждое из действий b_1, b_2 (или f_1, f_2) в задачах с действием t (или r) координатора (обозначения здесь: b – begin, f – finish, t – take, r – release).

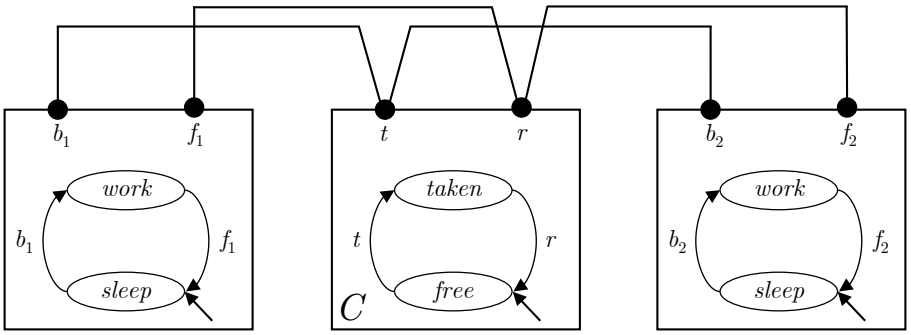

Рис. 4. Модель взаимного исключения в BIP фреймворке

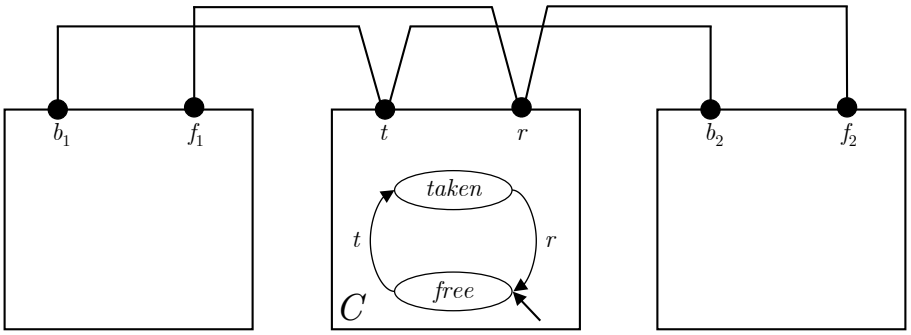

Рис. 5. Архитектура взаимного исключения

Архитектуру можно рассматривать как модель BIP, в которой некоторые атомарные компоненты рассматриваются как координаторы, а остальные являются параметрами. Когда архитектура применяется к набору компонентов, эти компоненты используются в качестве операндов для замены параметров архитектуры. Таким образом, на рис. 5 показана архитектура, которая обеспечивает свойство взаимного исключения для любых двух компонентов с интерфейсами $\{b_1, f_1\}$ и $\{b_2, f_2\}$, удовлетворяющая следующему свойству:

> *Как только компонент выполнил переход f, он не войдет в критическую секцию, если он не выполнит переход b.*

Для компонентов-операндов на рис. 4 критической секцией является состояние *work*.

Композиция архитектур основана на ассоциативном, коммутативном и идемпотентном операторе '$\oplus$' [16]. Если две архитектуры $\mathcal{A}_1$ и $\mathcal{A}_2$ обеспечивают соответствующие свойства Φ_1 и Φ_2, то их композиция $\mathcal{A}_1 \oplus \mathcal{A}_2$ обеспечит свойство $\Phi_1 \wedge \Phi_2$, что значит, что оба свойства сохранены композицией архитектур.

Поскольку архитектуры ограничивают поведение компонентов, к которым они применяются, отсутствие взаимной блокировки, как правило, не может быть гарантировано конструкцией. Вместо этого отсутствие взаимной блокировки должно быть проверено апостериори с помощью специальных инструментов, таких как DFinder [17].

4. Реализация

В качестве образца взята работа [18], в которой выполняется полный рабочий процесс проектирования (т.е. анализ требований, логическое проектирование, разработка BIP-кода, компиляция в микропроцессор и тестирование) подсистемы управления командами и данными (CDMS) для проекта CubETH [19], тем самым демонстрируя осуществимость использования концепта BIP для малых спутников. В качестве микроконтроллера был выбран EFM32GG880 ARM Cortex-M3 со 128 КБ ОЗУ и 1 МБ программной памяти, работающий на частоте 48 МГц. CubETH – это швейцарский проект [20] по созданию 1U кубсата для демонстрации новых технологий в области спутниковых систем навигации при использовании COTS-компонент аппартного обеспечивания.

Вся модель состояла из 56 атомов и не вместилась в доступную память модели спутника. Таким образом, обоснование было сделано для сокращенной модели программного обеспечения, содержащей 19 атомов и 60 соединений.

Основными сложностями были: 1) необходимость статического распределения памяти, 2) большой объем ОЗУ сгенерированного кода и 3) необходимость компиляции процессора BIP и сгенерированного кода с помощью специального компилятора ARM. Объем потребляемой памяти сгенерированного кода показан на рис. 6. Объем ОЗУ (128 МБ) на процессоре Cortex-A3 является основным ограничением, ограничивающим число атомов и соединений (и, следовательно, сложность модели), которые могут быть использованы одновременно. Доступно только статическое распределение, поэтому невозможно перераспределить данные в основную память. Это ограничение легко снимается на более мощных платформах. Выбранная архитектура использовалась из-за низкого энергопотребления и, в более общем смысле, режима ограничения мощности на 1U кубсате.

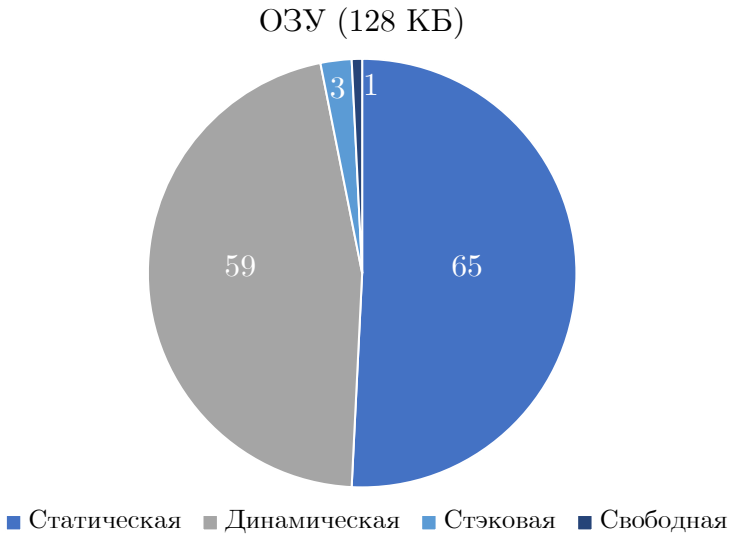

Рис. 6. Объём потребляемой оперативной памяти сокращённой BIP модели

Модель, разработанная в [18], была проанализирована для того, чтобы определить, какие паттерны проектирования повторяются. Эти шаблоны были формализованы как архитектуры BIP (см. раздел 3.2) и использовались в [9] для разработки новой модели BIP бортового программного обеспечения CubETH с нуля. Более конкретно, начиная с небольшого набора атомов, которые реализуют простейшие ключевые функциональные возможности программного обеспечения, мы систематически применяли вышеупомянутые архитектуры для сохранения свойств, налагаемых требованиями, сформулированными для бортового программного обеспечения. Поскольку свойства сохраняются посредством архитектурной композиции [16], все свойства, которые мы связали с требованиями CubETH, удовлетворяются конструкцией этой последней модели.

Архитектуры обеспечивают требуемые свойства, ограничивая совместное поведение компонентов-операндов (раздел 3.2). Из чего следует, что совместное применение архитектур может привести к взаимным блокировкам. Был использован инструмент DFinder [17] для проверки свободы от блокировок исследуемой модели. DFinder применяет композиционную проверку на моделях BIP, переоценивая набор достижимых состояний и символически проверяя, что пересечение полученного набора и набора блокировочных состояний пусто. Такой подход позволяет DFinder анализировать очень большие модели. Инструмент надежный, но неполный: из-за вышеупомянутого приближения он может давать ложные срабатывания, то есть потенциальные тупиковые состояния, которые недоступны в конкретной системе. Тем не менее модель исследования показала отсутствие блокировок. Таким образом, никакого дополнительного анализа достижимости не потребовалось.

Также нами была проверена возможность использовать полную модель BIP из [18] на одноплатном компьютере, которая была бы не урезана из-за недостатка программной и оперативной памяти. В качестве одноплатного компьютера был использован BeagleBone Black (рис. 7), в основе которого находится процессор ARM Cortex-A8. В отличие от микроконтроллера EFM32, в BeagleBone намного больше памяти – 512 МБ ОЗУ, однако и намного больше электропотребление, что сильно сказывается на применимости данного устройства в проектах с малыми спутниками. Мы попробовали откомпилировать процессор BIP напрямую внутри BeagleBone и, соответственно, компилировать модель для системы управления данными также напрямую внутри одноплатного компьютера, без использования тулчейна.

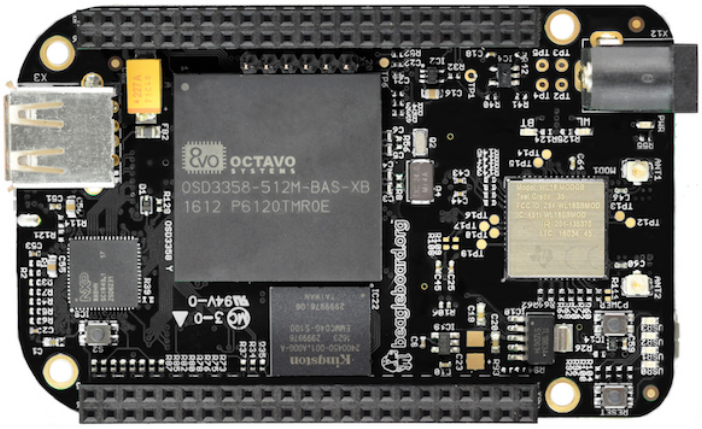

Рис. 7. BeagleBone Black

Так как компиляция процессора BIP внутри архитектуры ARM имеет свою специфику, к которой относится отказ от оптимизации при верификации сгенерированного кода из-за несоответствия предустановленной для BIP версии Java с доступными на BeagleBone, мы посчитали необходимым опубликовать трудные моменты сборки процессора BIP для одноплатного компьютера:

1) Компиляторы GCC и G++ версий не старше 4.8.5, headless Java SE не моложе 1.7.

2) Оптимизация должна быть пропущена, так как она использует предустановленный LPSolve java файл.

3) Реконфигурация процессора BIP сопровождается требуемыми вашей системой изменениями в PackageLoader'e.

Результаты сборки проекта на одноплатном компьютере показали, что оперативной и программной памяти хватает с избытком, однако увеличенное энергопотребление накладывает ограничения на использование подобного аппаратного обеспечения в наноспутниках по типу кубсата.

5. Заключение

В этой работе мы продемонстрировали осуществимость BIP-подхода для разработки спутникового программного обеспечения. Ограничения процессора Cortex-M3 привели к уменьшению модели. Однако демонстрация уменьшенной модели на плате кубсата прошла успешно. BIP-подход также возможно использовать на одноплатных компьютерах типа BeagleBone Black, которые позволяют использование гораздо большего количества памяти, однако для этого следует учесть энергетические параметры планируемой миссии.

Оказалось, что крайне полезно иметь возможность проверять логику проектирования перед компиляцией на аппаратное обеспечение. Кроме того, применение подхода проектирования на основе архитектуры BIP позволило разработать аналогичную модель, в которой все требуемые свойства обеспечиваются конструкцией. Этот подход требует только проверки модели на наличие взаимных блокировок, чего удалось добиться с помощью инструмента DFinder из набора инструментов BIP.

Эта работа показала, что для относительно небольших миссий, таких как кубсаты, фреймворк BIP можно использовать для разработки полного бортового программного обеспечения.

Литература

1. *Forsberg K., Mooz H.* The Relationship of System Engineering to the Project Cycle // INCOSE International Symposium. 1991. P. 57–65.
2. *Spangelo S.C., Cutler J., Anderson L., Fosse E., Cheng L., Yntema R., Bajaj M., Delp C., Cole B., Soremekum G., Kaslow D.* Model based systems engineering (MBSE) applied to Radio Aurora Explorer (RAX) CubeSat mission operational scenarios // 2013 IEEE Aerospace Conference. IEEE. 2013. P. 1–18.
3. *Bliudze S., Cimatti A., Jaber M., Mover S., Roveri M., Saab W., Qiang W.* Formal verification of infinitestate BIP models // Proceedings of the 13th International Symposium on Automated Technology for Verification and Analysis, ser. Lecture Notes in Computer Science, Finkbeiner B., Pu G., Zhang L., Eds. 2015. V. 9364. P. 326–343.
4. *Wilmot J., Fesq L., Dvorak D.* Quality Attributes for Mission Flight Software : A Reference for Architects // IEEE AeroSpace, Big Sky, MT. 2016. P. 1–7.
5. *Basu A., Bensalem S., Bozga M., Combaz J., Jaber M., Nguyen T.-H., Sifakis J.* Rigorous component-based system design using the BIP framework // Software, IEEE. 2011. V. 28, N 3. P. 41–48.
6. *Bliudze S., Sifakis J.* The algebra of connectors—structuring interaction in BIP // IEEE Transactions on Computers. 2008. V. 57, N 10. P. 1315–1330.
7. *Attie P., Bensalem S., Bozga M., Jaber M., Sifakis J., Zaraket F.* Global and Local Deadlock Freedom in BIP // ACM Trans. Softw. Eng. Methodol. 2018. V. 26, N 9. P. 1–48.
8. *Baier, C., Katoen, J.-P.* Principles of Model Checking // Representation and Mind Series. The MIT Press, Cambridge. 2008.
9. *Mavridou A., Stachtiari E., Bliudze S., Ivanov A., Katsaros P., Sifakis J.* Architecture-Based Design: A Satellite On-Board Software Case Study // In: Kouchnarenko O., Khosravi R. (eds) Formal Aspects of Component Software. FACS 2016. Lecture Notes in Computer Science, Springer, Cham. 2017. V. 10231. P. 260–279.
10. *Attie P., Baranov E., Bliudze S., Jaber M., Sifakis J.* A general framework for architecture composability // Formal Aspects of Computing. 2016. V. 18, N 2. P. 207–231.
11. *Bensalem S., Griesmayer A., Legay A., Nguyen T.-H., Sifakis J., Yan R.* D-Finder 2: towards efficient correctness of incremental design // Proceedings of the 3rd international conference on NASA Formal methods. 2011. P. 453–458.

12. *Pagnamenta M.* Rigorous software design for nano- and micro-satellites using BIP framework // Master's thesis, Space Center, EPFL. 2014.

13. *Ivanov A., Bliudze S.* Robust Software Development for University-Built Satellites // IEEE. 2017. URL: http://infoscience.epfl.ch/record/225659 (Дата обращения – 09.06.2019).

14. *Ivanov A.B., Masson L., Rossi S., Belloni F., Mullin N., Wiesendanger R., Rothacher M., Hollenstein C., Mannel B., Willi D., Fisler M., Fleischman P., Mathis H., Klaper M., Joss M., Styger E.* CubETH: Nano-satellite mission for orbit and attitude determination using low-cost GNSS receivers // 66th International Astronautical Congress. Jerusalem, Israel: International Astronautical Federation, IAF. 2015.

References

1. *Forsberg K., Mooz H.* The Relationship of System Engineering to the Project Cycle. INCOSE International Symposium. 1991. P. 57–65.

2. *Spangelo S.C., Cutler J., Anderson L., Fosse E., Cheng L., Yntema R., Bajaj M., Delp C., Cole B., Soremekum G., Kaslow D.* Model based systems engineering (MBSE) applied to Radio Aurora Explorer (RAX) CubeSat mission operational scenarios. 2013 IEEE Aerospace Conference. IEEE. 2013. P. 1–18.

3. *Bliudze S., Cimatti A., Jaber M., Mover S., Roveri M., Saab W., Qiang W.* Formal verification of infinitestate BIP models. Proceedings of the 13th International Symposium on Automated Technology for Verification and Analysis, ser. Lecture Notes in Computer Science, Finkbeiner B., Pu G., Zhang L., Eds. 2015. V. 9364. P. 326–343.

4. *Wilmot J., Fesq L., Dvorak D.* Quality Attributes for Mission Flight Software : A Reference for Architects. IEEE AeroSpace, Big Sky, MT. 2016. P. 1–7.

5. *Basu A., Bensalem S., Bozga M., Combaz J., Jaber M., Nguyen T.-H., Sifakis J.* Rigorous component-based system design using the BIP framework. Software, IEEE. 2011. V. 28, N 3. P. 41–48.

6. *Bliudze S., Sifakis J.* The algebra of connectors—structuring interaction in BIP. IEEE Transactions on Computers. 2008. V. 57, N 10. P. 1315–1330.

7. *Attie P., Bensalem S., Bozga M., Jaber M., Sifakis J., Zaraket F.* Global and Local Deadlock Freedom in BIP. ACM Trans. Softw. Eng. Methodol. 2018. V. 26, N 9. P. 1–48.

8. *Baier, C., Katoen, J.-P.* Principles of Model Checking. Representation and Mind Series. The MIT Press, Cambridge. 2008.

9. *Mavridou A., Stachtiari E., Bliudze S., Ivanov A., Katsaros P., Sifakis J.* Architecture-Based Design: A Satellite On-Board Software Case Study. In: Kouchnarenko O., Khosravi R. (eds) Formal Aspects of Component Software. FACS 2016. Lecture Notes in Computer Science, Springer, Cham. 2017. V. 10231. P. 260–279.

10. *Attie P., Baranov E., Bliudze S., Jaber M., Sifakis J.* A general framework for architecture composability. Formal Aspects of Computing. 2016. V. 18, N 2. P. 207–231.

11. *Bensalem S., Griesmayer A., Legay A., Nguyen T.-H., Sifakis J., Yan R.* D-Finder 2: towards efficient correctness of incremental design. Proceedings of the 3rd international conference on NASA Formal methods. 2011. P. 453–458.

12. *Pagnamenta M.* Rigorous software design for nano- and micro-satellites using BIP framework. Master's thesis, Space Center, EPFL. 2014.

13. *Ivanov A., Bliudze S.* Robust Software Development for University-Built Satellites. IEEE. 2017. URL: http://infoscience.epfl.ch/record/225659 (Requested on – 09.06.2019).

14. *Ivanov A.B., Masson L., Rossi S., Belloni F., Mullin N., Wiesendanger R., Rothacher M., Hollenstein C., Mannel B., Willi D., Fisler M., Fleischman P., Mathis H., Klaper M., Joss M., Styger E.* CubETH: Nano-satellite mission for orbit and attitude determination using low-cost GNSS receivers. 66th International Astronautical Congress. Jerusalem, Israel: International Astronautical Federation, IAF. 2015.

Поступила в редакцию 10.06.2019