Выбор технологии программирования встроенных систем

Во встроенных системах традиционно выделяют программную (Software) и аппаратную (Hardware) составляющие. Для создания каждой из них требуется технология, которая является частью технологии создания системы в целом. В статье обсуждаются классы аппаратной организации систем, в различной степени ограничивающие выбор технологии программирования. Приводятся примеры организации встроенного программного обеспечения микроконтроллеров.

Рустам Ковязин

rustam@d1.ifmo.ru

Введение

Встроенные системы (ВсС) используют для автоматизации во множестве отраслей, например в энергетике, металлургии, теплоснабжении. Область их применения также распространяется на бытовую технику, телекоммуникации, транспортные и аэрокосмические системы, промышленную автоматику и др. Многие управляющие системы на основе релейно-контактных схем, аналоговой техники, механических агрегатов активно заменяются системами с программным управлением. Благодаря этому повышается функциональность и надежность управляющих систем, расширяется область их применения.

Базовым принципом современной вычислительной техники является программируемость системы на всех уровнях. Программное управление во ВсС реализуется на основе микропроцессоров, микроконтроллеров и ПЛИС. Программирование этих устройств организуется в виде многоуровневой (иерархической) структуры, которая удобна для разработчиков. Такая структура программного обеспечения поддерживается технологиями, опирающимися на системные библиотеки, API, интерпретирующие ядра (виртуальные машины) и др. При обсуждении вопросов программирования ВсС будем пользоваться термином «*вычислитель*», обобщенно понимая под ним любое из перечисленных устройств (комплекс устройств), имеющее свой уровень программного управления.

Тогда *технологиями программирования* (ТП) ВсС можно считать технологии реализации программного управления средствами вычислителей системы. Для каждого вычислителя может быть использована своя ТП. ТП объединяет множество необходимых для создания программ понятий. Их можно разделить на несколько категорий:

- вычислительные модели;
- методики и типовые решения;
- языки программирования;
- инструментальные средства.

Между этими категориями существуют отношения следствия. ТП объединяет способ, процесс и средства алгоритмизации программного управления.

На ТП существенное влияние оказывают архитектурные особенности вычислителей. Далее рассмотрим свойства аппаратных вычислителей, которые в первую очередь определяют доступные ТП при создании ВсС на современной микропроцессорной элементной базе. Будут приведены примеры простых ТП, широко используемых при разработке ВсС.

Характеристики аппаратных вычислителей встроенных систем

Важным вопросом организации программного управления является выбор вычислителя. Существует множество классификаций вычислителей. Ниже перечислен ряд классификаций, имеющих значение для выбора ТП. Характеристики каждого класса, с одной стороны, ограничивают круг используемых (целесообразных) ТП. С другой стороны, они влияют на создание новых более эффективных в этом классе ТП. Перечислим классификации вычислителей в порядке ослабления их влияния на выбор ТП, сокращения вносимых ограничений.

Рассматриваемые классификации опираются на широко распространенные типы электронных компонентов [1].

Программное управление

В рамках программного управления вычислитель позволяет манипулировать своими ресурсами (регистрами, функциональными устройствами, портами ввода-вывода и т. п.). Выделим характерные способы доступа к ресурсам вычислителя и соответствующие им классы программного управления.

- *Последовательное* программное управление. Взаимодействие ресурсов вычислителя возможно лишь в определенных комбинациях (инструкции). Для управления всеми необходимыми ресурсами требуется последовательность инструкций. В любой момент времени ядро выполняет лишь одну инструкцию. Последовательные программы соответствуют принципам программного управления фон Неймана. Последовательными вычислителями является большинство программируемых процессоров.

- *Последовательно-параллельное* программное управление. Одновременно доступно несколько комбинаций ресурсов вычислителя, что позволяет организовать несколько последовательностей инструкций (программных потоков). Управление числом последовательностей и взаимодействие между ними должно быть обеспечено на уровне инструкций. Последовательно-параллельные вычислители часто реализуются программно в различных API (WIN32, POSIX).
- *Параллельное* программное управление. Все ресурсы вычислителя доступны в любой комбинации. Программное управление осуществляется не на уровне инструкций последовательного интерпретатора. Параллельными вычислителями в определенном смысле являются ПЛИС, а языками программирования — средства описания аппаратуры VHDL, Verilog HDL и др.

Языки для создания последовательных (последовательно-параллельных) и параллельных программ различаются подмножествами вычислительных моделей, реализованных в них. Для параллельных программ базовой вычислительной моделью является модель дискретных событий, а для последовательных программ — синхронный автомат [2, 3]. Принципиальные отличия в базовых моделях программ ведут к тому, что обычно разработчики специализируются на разработке программ только одного из классов. Возможность разработки программ всех классов в соответствии с составляющими ТП влечет за собой необходимость владения характерными вычислительными моделями, методиками, языками и инструментальными средствами.

Разрядность вычислителя

Разрядность вычислителя определяет, какие данные им обрабатываются наиболее оперативно (*разрядность данных*), какие объемы данных вычислитель в состоянии обработать (*объем адресного пространства данных*) и насколько сложное программное управление в нем можно реализовать (*объем адресного пространства кода*).

Превышение пределов разрядной сетки влечет за собой трудности, в большинстве случаев скрываемые инструментальным обеспечением и незаметные на программном уровне. Результатом является значительное увеличение размера кода и замедление работы с «большими» данными.

- *4-битные* вычислители обычно используются в сложных электронных компонентах (контроллеры ЖКИ, приемопередатчики и т. п.). Программное управление алгоритмически простое и приближенное к непосредственному управлению аппаратурой.
- *8-битные* вычислители используются для решения низкоскоростных коммуникационных задач, задач логического управления невысокой и средней сложности. Программа обычно имеет однородную структуру [4].
- *16-битные* вычислители используются для задач средней сложности. В программе появляется структура: операционные среды, виртуальные машины и т. п. Появляется деление вычислителей на группы: логическое управление, математические вычисления, сигнальная обработка, коммуникации и др. Объем ресурсов вычислителя позволяет ввести аппаратную специализацию.
- *32-битные* вычислители обладают высокой производительностью и используются для сложных вычислений, обработки больших объемов информации и высокоскоростных коммуникационных протоколов. В структуре программы появляются многозадачные операционные системы, подсистемы управления ресурсами и т. п.

Распространенность архитектуры

Каждый вычислитель сопровождается арсеналом доступных инструментальных средств. Разработчики инструментальных средств стараются вывести свои продукты на высокий уровень, который задается такими известными фирмами, как Microsoft, Metrowerks и др. Уровень инструментальной поддержки определяется распространенностью архитектуры вычислителя.

- Инструментальные средства для *специализированных архитектур* обычно представлены в самом минимальном объеме.
- Инструментальная поддержка *архитектур ограниченного применения* может быть полноценной, но она обычно осуществляется разработчиками вычислителя.
- *Широко распространенные архитектуры* (x86, ARM, MCS-51, AVR, PICMicro и др.) имеют хорошую инструментальную поддержку, большое число примеров и готовых решений. Инструментальные средства создаются как разработчиками вычислителя, так и сторонними фирмами (Keil Software, HI-TECH, Metrowerks и др.).
- Процессор BcC может обладать *архитектурой инструментальной* машины, например, IBM PC с процессорами x86. С одной стороны, это дает существенное преимущество в разработке программ, а с другой стороны, инструментальное обеспечение ориентированно именно на инструментальную машину и требует «подгонки» под конкретный вычислитель.

Инструментальные средства для каждой архитектуры постепенно развиваются, но, к сожалению, не всегда в одном направлении. Разработчик, работающий с различными архитектурами, вынужден работать с разными инструментальными средами и средствами, пользовательскими интерфейсами, компонентами, знать и учитывать их свойства.

Физическая организация подсистемы памяти

Тип, емкость и особенности организации физического доступа к памяти со стороны процессора определяют максимально допустимый размер программы, вид и место хранения, способ запуска на исполнение и возможность модификации. Рассмотрим варианты организации адресуемой памяти, характерные для микроконтроллеров и микропроцессоров.

- Память на кристалле (*внутренняя*) имеет фиксированный объем и конструктивно соединена с процессором. Большинство 8-битных микроконтроллеров имеют только такую память.
- *Внутренняя+внешняя* память позволяет варьировать объем доступной памяти за счет изменения емкости внешней памяти. Некоторые 8-битные вычислители позволяют подключать внешнюю память. При этом логически такая дополнительная память может как расширять существующие адресные пространства, так и создавать новые со специальными методами доступа.
- В 16-битных и 32-битных процессорах используется конструктивно *внешняя* память. Адресные пространства позволяют непосредственно работать с большими объемами физической памяти. Для создания программ нужны дополнительные средства. Внешняя физическая память за счет объема позволяет организовать сложные конфигурации логической памяти, хранение и загрузку на исполнение одного из нескольких приложений.

Размещение памяти является ограничением элементной базы (внутренняя память), влекущим использование определенных методик загрузки программы на исполнение. Деление памяти на физические блоки позволяет ее раздельно программировать, защищать, хранить «холодные загрузчики».

Архитектура памяти

Архитектура памяти определяет размещение кода и данных, способы работы с ними на логическом уровне. С другой стороны, архитектура памяти определяется инструкциями вычислителя, так что можно говорить об аналогичной архитектуре вычислителя.

- *Регистровая* архитектура выделяет множество ячеек памяти быстрого доступа (регистры) и один или несколько массивов памяти большого объема с относительно медленным доступом. Распространены два варианта организации последней:
  - Универсальная (*принстонская*) архитектура позволяет размещать информацию различного рода в одном адресном пространстве (код, данные, стек). Это облегчает модификацию кода программы, доступ к данным независимо от места их хранения.
  - Раздельная (*гарвардская*) архитектура содержит несколько адресных пространств. Это защищает код от модификации и не позволяет оперировать разными данными в едином стиле (у данных однозначная интерпретация в зависимости от места их хранения).
- *Стековая* архитектура вместо регистров вводит структуру с доступом LIFO.
- Архитектура «*память-память*» в вычислениях оперирует непосредственно ячейками памяти без выделения «особенных» ячеек.

Каждая из архитектур в чистом виде сейчас используется редко. Они сохранились в интерпретирующих ядрах, обладающих несложной функциональностью. Все эти архитектуры взаимозаменяемы и могут быть реализованы в рамках друг друга [5]. Сегодня наиболее распространена регистровая архитектура памяти. Кроме вычислителей все три вида архитектур применяются в компиляторах для внутреннего представления программы: польская инверсная запись (стековая архитектура), триадная запись (регистровая архитектура), тетрадная запись (архитектура «память-память»).

Таблица. Влияние организации памяти регистровых архитектур на технологии программирования

	Универсальное адресное пространство	Раздельные адресные пространства
Инструментальное обеспечение	Облегченная генерация кода программы.	Облегченная начальная загрузка программы, ее хранение в ПЗУ.
Языки программирования	В большинстве случаев языки хорошо согласуются с существующими стандартами.	Требуются дополнительные синтаксические конструкции для организации хранения данных.
Методики	Сложные структуры управления и данных. Возможна модификация кода и смешанное хранение данных.	Упрощенные структуры управления и данных. Возможно раздельное программирование разных видов памяти.
Ограничения элементной базы	Размерность команд кода кратна размерности данных. Правила хранения данных и кода совпадают.	Объем памяти кода и стека ограничены, поскольку память обычно интегрирована в вычислитель.

Поскольку регистровые архитектуры наиболее распространены, то в таблице рассмотрим их влияние на компоненты ТП.

Примеры простых технологий программирования BcC

В качестве примеров рассмотрим две характерные для BcC ТП. В них присутствует несколько уровней программирования, ориентированных на различные категории пользователей. Эти ТП были использованы в контроллерных системах CSC-1.1 и SDK-5.0, разработанных фирмой ЛМТ. Более подробную информацию об этих системах можно найти в Интернете на сайте lmt.cs.ifmo.ru.

Рассматриваемые технологии основаны на применении инструментального загрузчика и интерпретирующего ядра. Суть обеих ТП заключается в организации кода в памяти микроконтроллера так, чтобы облегчить загрузку пользовательских приложений и пользовательское программирование. Инструментальный загрузчик интересен тем, что его широкое применение во множестве вычислителей поддержано на аппаратном уровне. Использование интерпретирующего ядра позволяет решать более широкий круг задач, выделив новый уровень программирования.

Системы, в которых были использованы эти технологии, являются достаточно универсальными и допускают пользовательское программирование на уровне системного интегратора. Возможности существенно расширяются, если микроконтроллер имеет память с возможностью многократной перезаписи. В качестве вычислителя в рассматриваемых примерах ТП используются микроконтроллеры фирмы Microchip семейства PIC18FXXX с гарвардской архитектурой, RISC-ядром и Flash-памятью программ. Инструментальный загрузчик был использован в контроллере SDK-5.0, а интерпретирующее ядро — в контроллере CSC-1.1, который серийно производится в 4 модификациях.

Инструментальный холодный загрузчик

В персональном компьютере функции холодного загрузчика выполняет операционная система. Задачами холодного загрузчика являются инициализация аппаратуры системы, перевод ее в безопасное состояние и запуск пользовательского приложения на исполнение. Дополнение загрузчика инструментальными свойствами позволит ему не только запускать пользовательские приложения, но и программировать их в память контроллера.

В микроконтроллерах семейства PIC18FXXX используется Flash-память программного кода, который начинает исполняться с нулевого адреса (вектор RESET). В этом векторе обычно находится ссылка на пользовательское приложение. В системе с холодным загрузчиком ссылка указывает на загрузчик, задачей которого является инициализация периферии системы и ожидание сигнала от пользователя о его желании запрограммировать новое пользовательское приложение. Таким сигналом может быть определенное состояние дискретного входа (джампер, движковый переключатель, кнопка), логический сигнал на линиях RS232 или другом интерфейсе связи с инструментальной машиной. Контроллер может просто ожидать некоторое время передачи по инструментальному интерфейсу. Важно, чтобы загрузчик не воспринял по нему помеху, способную инициировать программирование и уничтожение существующего приложения. Если сигнал о программировании отсутствует или временной интервал ожидания истек, то загрузчик передаст управление пользовательскому приложению, хранящемуся в памяти контроллера. Если холодный загрузчик программирует приложение в энергозависимую память, то передача пользовательского приложения по инструментальному каналу обязательна. Такое решение чаще всего применяют в демонстрационных системах (Evaluation Kit).

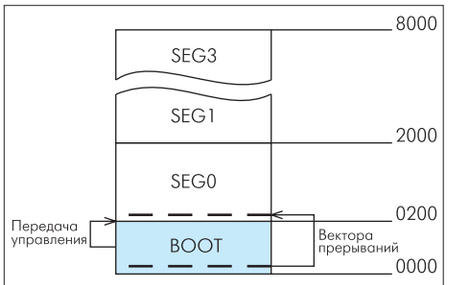

Рис. 1. Карта памяти SDK-5.0

Память программ в микроконтроллерах PIC18FXXX поделена на сектора: 4 сектора общего назначения и один загрузочный сектор (см. рис. 1). В начальных адресах памяти специально выделен небольшой сектор, позволяющий хранить в нем загрузчик отдельно от остального кода (пример аппаратной поддержки данной технологии). Каждый из секторов может быть защищен от стирания и записи. Это важно, так как и холодный загрузчик, и пользовательское приложение располагаются в одной памяти кода. Загрузчик потенциально может запрограммировать приложения поверх себя, что может привести к неработоспособности контроллера. Весь код загрузчика должен размещаться в одном загрузочном секторе (512 слов). Этот сектор защищен от записи и может быть запрограммирован только внешним программатором.

В микроконтроллерах PIC18FXXX таблица векторов прерываний располагается по адресам, находящимся в загрузочном секторе. Образ программ, генерируемый компиляторами для архитектуры PIC18FXXX, не является переносимым. В нем задается код программы, начиная с вектора RESET, то есть с нулевого адреса. Чтобы пользовательское приложение по адресам не пересекалось с холодным загрузчиком, его нужно расположить не с нулевого адреса, а с адреса, выходящего за пределы загрузочного сектора (стартовый адрес пользовательской программы должен быть не менее 0x0200). Нужен полный перенос образа программы, как будто ее адресное пространство начинается с этого адреса. Тогда холодный загрузчик для запуска приложения будет передавать управление на его стартовый адрес, располагающийся по известному смещению, а вектора прерываний загрузчика будут ссылаться на таблицу векторов пользовательского приложения.

Компилятор HI-TECH PICC Compiler предоставляет возможность такого размещения кода программы. Программа, размещающаяся в памяти не с нулевого адреса, будет исполняться некорректно без загрузчика, передающего ей управление. Таким образом, пользовательские приложения перестают быть самодостаточными программами. Пользовательское ПО может располагаться во всех секторах общего назначения. Внедрение инструментальной поддержки в загрузочный модуль устройства позволило конечному пользователю программировать его средствами инструментальной машины, в качестве которой выступает ПК.

Холодный загрузчик позволяет хранить в памяти кода одну программу и передавать ей управление. К сожалению, объем загрузочного сектора слишком мал, чтобы можно было сделать загрузчик более интеллектуальным, позволить хранить и передавать управление нескольким программам, организовать более удобное взаимодействие с пользователем.

ТП, используемые в загрузчике и в пользовательском ПО, определяются компилятором PICC. Для пользователя они предоставляются в полном объеме, основываясь на структурном и автоматном программировании. Применение ТП в загрузчике сложнее в силу ограничения объема его кода. Из-за этого пришлось частично пожертвовать структурным подходом, использовав глобальные данные и системные регистры специального назначения для хранения временных переменных, а также совмещение кода низкого и высокого уровня.

Интерпретирующее ядро

Применение интерпретирующих ядер (виртуальных машин) в программном обеспечении BcC описано в литературе [4]. Интерпретирующее ядро в явном виде делит исполняемую программу на прикладное (ППО) и системное (СПО) программное обеспечение. ППО выполняет целевую функцию системы. В СПО вместе с подсистемой драйверов функционирует интерпретирующее ядро (программно реализованный вычислитель). Вычислительные модели, реализуемые этим вычислителем, зависят от него и непосредственно не связаны с тем, какая ТП использовалась при создании СПО. Таким образом, для программирования

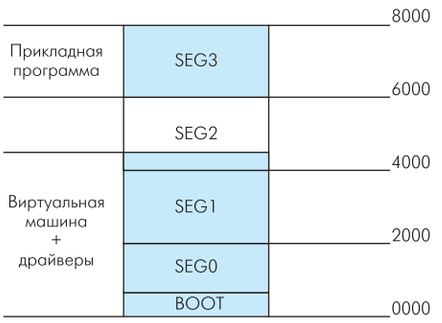

Рис. 2. Карта памяти CSC-1.1

контроллера с интерпретирующим ядром может потребоваться несколько ТП, поскольку он содержит несколько вычислителей. Если пользователю системы предоставляется возможность программирования, то только на уровне интерпретирующего ядра. Даже если такой возможности не предоставляется, наличие интерпретирующего ядра оправдано, так как позволяет разработчикам легко изменить целевую функцию системы без существенных изменений, вносимых в СПО.

Инструментальная поддержка в коде СПО позволяет программировать прикладную программу по инструментальному каналу. Для нее выделяется отдельный сегмент. Как и в примере с холодным загрузчиком, СПО имеет возможность программирования памяти кода, так что все сегменты кода, кроме сегмента с прикладной программой, защищены (см. рис. 2).

Архитектура программного вычислителя, разработанного в рамках системы CSC-1.1, имеет тип «память-память», тогда как используемый микроконтроллер имеет гарвардскую архитектуру. Программный вычислитель позволяет ППО работать с ресурсами контроллера как с ячейками памяти, обеспечивая все протоколы обмена данными на системном уровне. Моделью прикладной программы является конечный автомат Мили. Программа состоит из состояний, каждое из которых содержит последовательность операций. Все условные и безусловные переходы являются переходами между состояниями автомата. Инструкции интерпретирующего ядра представляют собой функциональные блоки, например, сумматоры, мультиплексоры, компараторы (есть некоторая аналогия с проектированием устройств на ПЛИС). Разработка приложений для такого вычислителя может оказаться затруднительной для разработчиков ППО, не представляющих основ языкового проектирования цифровых устройств на VHDL, Verilog HDL и др. Для применения в качестве входных языков более высокого уровня потребуется перенос ядра на систему с большей вычислительной мощностью. Организация интерпретирующих ядер на основе 8-битных контроллеров затруднительна. Данное ядро разрабатывалось как наиболее простое универсальное ядро, способное функционировать даже на 8-битных микроконтроллерах. Производительность ядра невысока, так как существенную долю процессорного времени «съедает» система драйверов, предоставляющая данные ядру для обработки.

Сравнение рассмотренных ТП

Описанные ТП схожи в инструментальном плане, однако уровень программирования у них отличается. Стиль прикладного программирования в системе с инструментальным загрузчиком совпадает со стилем самого загрузчика, так как для них обоих используются инструментальные средства одного класса. Загрузчик переводит интерфейсы связи с периферийными устройствами в безопасное состояние, а драйверы реализуются в пользовательском приложении. В системе с интерпретирующим ядром драйверы находятся в СПО, ППО не может повредить периферийные устройства. Независимо от действий ППО данные датчиков сохраняются в памяти системы. ППО производит их анализ и активирует управляющие воздействия, реализуемые опять же в рамках СПО. Таким образом, прикладное программирование в системе с программным вычислителем, с одной стороны, более безопасное, а с другой стороны — менее оперативное. Степень готовности системы с интерпретирующим ядром гораздо выше, чем у системы с загрузчиком. В первой системе пользователю остается разработать лишь ППО системы, а во второй системе нужно разрабатывать ПО в полном объеме. С другой стороны, в системе с загрузчиком структура ПО может быть произвольной, а не той, какая зафиксирована в системе с интерпретирующим ядром.

Заключение

Одной из особенностей рынка встроенных систем является жесткая конкуренция и сжатые сроки проектирования. Это в большинстве случаев не позволяет разработчикам создавать системы «с нуля», заставляя сокращать затраты за счет повторного использования аппаратных и программных решений. При этом к системам предъявляются высокие требования по функциональности, надежности и безопасности.

Удачный выбор вычислителей и технологий программирования в значительной мере определяет успех разработки, что влияет на широту области применения разработки, возможность оперативной смены ее профиля. Естественным ограничителем фантазии разработчиков «сверху» в этом выборе является стоимость как самой системы, так и ее создания.

Стиль разработки системного и прикладного ПО встроенных систем значительно отличается от стиля, традиционного для ПО персональных компьютеров. Выбор модели программирования и стиля написания встроенного ПО зависит не только от технического кругозора и личных пристрастий разработчика. Используемая аппаратная платформа оказывает влияние на архитектуру и стиль создаваемого ПО. При этом простейшие микроконтроллеры обладают достаточными функциональными возможностями для эффективного программирования в рамках «хитрых» вычислительных моделей.

Литература

1. Клингман Э. Проектирование микропроцессорных систем. М.: Мир. 1980.
2. L. Lavagno, A. Sangiovanni-Vincentelli, E. Sentovich. Models of Computation for Embedded System Design. 1998.
3. Непейвода Н. Н., Скопин И. Н. Основания программирования. Москва-Ижевск, Институт компьютерных исследований. 2003.
4. Ковязин Р., Платунов А. Программирование микроконтроллерных систем // Электронные компоненты. 2003. № 4.
5. M. A. Ertl. Implementation of Stack-Based Languages on Register Machines. 1996.