KiT#64(11).qxd 12/7/2006 1:29 PM Page 90

Применение SWITCH-технологии при разработке прикладного программного обеспечения для микроконтроллеров. Часть 1

Владимир ТАТАРЧЕВСКИЙ
arktur04@mail.ru

В предыдущей своей статье [1] автор предпринял попытку рассмотрения ряда проблем, возникающих при разработке прикладного программного обеспечения (ПО) встроенных систем (в дальнейшем будем называть его «программное обеспечение»). Многие положения статьи могли показаться спорными, но, без сомнения, затронутые в ней вопросы являются актуальными, что демонстрирует ряд читательских откликов. Между тем, существуют технологии программирования, существенно облегчающие разработку встроенного ПО.

Одной из технологий, облегчающих разработку встроенного ПО, является SWITCH-технология, упоминавшаяся в работе [1]. Сегодня мы рассмотрим один из вариантов SWITCH-технологии, применяемый при разработке ПО микроконтроллеров. Данный вариант реализации SWITCH-технологии был разработан автором для создания собственных проектов, и, конечно же, не является единственно возможным. Автор намерен и в дальнейшем развивать предлагаемую концепцию программирования, делая ее более гибкой и приспособленной к весьма широкому кругу задач. Автор также надеется, что данная публикация послужит своего рода примером для других программистов, и побудит их к публикации своих разработок.

В последнее время, по мере роста мощности микроконтроллеров, все большую популярность приобретают операционные системы реального времени (ОСРВ), такие как uC/OS II, Embedded Linux и т. п. Однако их применение не решает всех проблем, возникающих при разработке ПО. Во-первых, применение ОСРВ ограничено сравнительно мощными микроконтроллерами и практически исключено для наиболее массового сегмента микроконтроллеров — 8-разрядных устройств. Во-вторых, ОСРВ применяют там, где необходимо организовать выполнение и взаимодействие нескольких программных потоков, при этом многие проблемы разработки не только снимаются, но и могут усугубиться. Разработка приложений, состоящих из множества асинхронных потоков, выполняющихся зачастую с разными приоритетами, требует от программиста применения сложных средств организации взаимодействия потоков, что приобретает особую сложность при организации доступа различных потоков у аппаратным ресурсам микроконтроллера. Впрочем, достоинствам и недостаткам ОСРВ можно посвятить отдельную статью, а здесь достаточно сказать, что применение SWITCH-технологии в ряде случаев снимает необходимость использования ОСРВ за счет того, что, во-первых, взаимодействие между автоматами осуществляется более простым образом, чем между потоками ОСРВ, во-вторых, применение автоматов делает поведение программы абсолютно детерминированным, а взаимодействие потоков в ОСРВ требует дополнительных (и немалых) усилий для устранения коллизий. Однако SWITCH-технология вовсе не исключает применение операционных систем (в частности, она может использоваться при разработке ПО для MS Windows). Например, возможен вариант реализации программ по SWITCH-технологии, в котором различные автоматы выполняются в различных потоках, обмениваясь между собой сообщениями. SWITCH-технология может применяться для разработки систем со сложным поведением в различных предметных областях, она подходит и для создания систем жесткого реального времени. Технология может также использоваться для построения сложных обработчиков прерываний, что будет рассмотрено в последующих статьях.

В чем же заключаются преимущества SWITCH-технологии вообще и предлагаемой реализации в частности?

Преимущество первое (и главное): программы, построенные по предлагаемой технологии, легко документировать. В рамках этого стиля программирования [4] программа представляет собой совокупность конечных автоматов, взаимодействующих друг с другом и с «внешним миром». При этом в наглядной графической форме могут быть выражены как связи между автоматами, так и их внутренняя структура.

Преимущество второе: возможность повторного использования кода. Задумывались ли вы о том, почему столь высокую популярность завоевали средства быстрой разработки (Rapid Application Development — RAD) — такие, как Borland Delphi? Ответ прост: потому что программа в них состоит из компонентов, являющихся в высокой степени автономными «сущностями». Компонент име-

Как уже упоминалось в работе [1], SWITCH-технология, которая также называется «автоматное программирование», является отечественной технологией программирования, созданной и разрабатываемой А. А. Шалыто и его соавторами. Отличным введением в SWITCH-технологию может служить книга [2]. Кроме того, Анатолий Абрамович Шалыто является основателем «Движения за открытую проектную документацию» [3]. С материалами по этой технологии и «Движению» можно ознакомиться на сайте **http://is.ifmo.ru/**, а также в Википедии (**http://ru.wikipedia.org/**, статья «Switch-технология»).

KiT#64(11).qxd 12/7/2006 1:29 PM Page 91

ет ограниченное количество связей с остальной программой, его можно разрабатывать и тестировать отдельно, а применять многократно, и именно это свойство подобных систем делает разработку быстрой и удобной.

Автомат в предлагаемом стиле программирования также является своего рода «кирпичиком», автономной единицей программы. Его связи с остальными автоматами сведены к минимуму и унифицированы. Его, как и компонент RAD-системы, можно разработать отдельно, а затем применять в различных программных проектах.

Из сказанного вытекает и третье преимущество: программы, построенные по SWITCH-технологии, легко поддаются модификации. Так как количество связей между автоматами минимально, изменения в одном из них чаще всего не влекут за собой необходимость коррекции кода в других автоматах.

В данной части настоящей статьи рассматривается общая структура программы, построенной на основе SWITCH-технологии, и ее базовая конструкция — автоматы.

В дальнейшем будут рассмотрены механизм обработки сообщений и механизм таймеров, необходимый для придания программе временно́го детерминизма. После рассмотрения базовых понятий будет приведен пример проектирования реальной программы.

Все исходные тексты ПО в статье приведены на языке Си. Это не означает, что рассматриваемый стиль программирования может быть реализован только на языке Си — его можно переложить и на любой язык программирования, включая язык ассемблера и C++, однако именно язык Си наиболее подходит для демонстрации возможностей данной технологии благодаря своей широкой распространенности среди специалистов по микроконтроллерам.

Все исходные тексты программ взяты из реального проекта, написанного для микроконтроллера AT91SAM7S256 (на ядре ARM7). При разработке использовался компилятор IAR C/C++ Compiler for ARM 4.40A.

Для того чтобы не затруднять понимание исходных текстов особенностями архитектуры данного процессора, из текстов по возможности исключен аппаратно-зависимый код, который заменен в соответствующих местах комментариями. Это сделано еще и для того, чтобы подчеркнуть независимость технологии от архитектуры конкретного процессора. Читатель при желании сможет восполнить эти «пробелы», написав соответствующий код для своего «любимого» микроконтроллера.

Общая структура программы

Опишем излагаемый стиль программирования. Его суть можно сформулировать следующим образом: программа представляет собой совокупность конечных автоматов, выполняющихся параллельно и обменивающихся между собой сообщениями. Другим ключевым свойством предлагаемой концепции является широкое использование таймеров, которые предназначены для привязки работы программы к реальному времени.

Итак, автоматы должны выполняться параллельно. Как достичь данного эффекта без использования многозадачной ОС? На самом деле, ничего сложного здесь нет. Все дело в особой структуре автоматной программы. В работе [5] приведены слова известного разработчика ядра ОС Linux Алана Кокса: «Потоковое программирование нужно тем, кто не умеет использовать конечные автоматы».

Рассмотрим структуру программ рассматриваемого класса более подробно. Будем для простоты считать, что каждый конечный автомат (КА) описан в отдельном модуле программы и имеет, как минимум, две внешние функции:

```
void InitFSM(void);
void ProcessFSM(void);
```

Вместо букв FSM в объявлении функций подставим имя автомата. При этом, например, функции:

```
void InitPasswordEditor(void);
void ProcessPasswordEditor(void);
```

будут принадлежать автомату PasswordEditor, описанному в модуле password_editor.c.

При этом функция InitFSM, как следует из названия, инициализирует автомат (это что-то вроде конструктора в объектно-ориентированном программировании), а функция ProcessFSM отвечает за работу автомата. Главная особенность последней функции: она не должна выполнять продолжительных во времени действий, связанных с ожиданием какого-либо флага или с истечением временного интервала — то есть в ней не должно быть конструкций типа:

```
while(flag == 0);
или
for(i = 0; i < delay; i++);
```

Как будет показано далее, в таких конструкциях просто нет необходимости.

Задачей главного цикла программы является поочередный вызов функций ProcessFSM всех автоматов, составляющих программу:

```
//main.c
#include «messages.h»      //модуль обработки сообщений
#include «timers.h»        //модуль таймеров
#include «fsm1.h»          //модуль автомата fsm1
#include «fsm2.h»          //модуль автомата fsm2
#include «fsm3.h»          //модуль автомата fsm3

void main()
{
 InitTimers();             //инициализация таймеров
 InitMessages();           //инициализация механизма
                           //обработки сообщений
 InitFSM1();               //инициализация автомата FSM1
 InitFSM2();               //инициализация автомата FSM2
 InitFSM3();               //инициализация автомата FSM3

 SendMessage(MSG_FSM1_ACTIVATE); //активируем автомат FSM1

 // главный цикл программы
 while(1)
 {
   ProcessFSM1();          //итерация автомата FSM1
   ProcessFSM2();          //итерация автомата FSM2
   ProcessFSM3();          //итерация автомата FSM3
   ProcessMessages();      //обработка сообщений
 };
}
```

Теперь становится понятно, каким образом обеспечивается «многопоточность» системы: в каждой итерации главного цикла поочередно вызываются Process-функции каждого автомата — каждому автомату выделяется время для выполнения какого-либо элементарного действия (или, возможно, просто для передачи управления далее по списку). Такой порядок работы напоминает кооперативную многозадачность в ранних версиях Windows: программа выполняет какое-либо действие и передает управление дальше, но если она «зависает», то «зависает» вся система. Именно для предотвращения подобной ситуации вводится явный запрет на действия в автоматах, которые занимают продолжительное или неопределенное время.

Рассмотрим базовую структуру, используемую в приведенном выше коде — автоматы.

Автоматы

Вне сомнений, большинству читателей «КиТ» теория КА знакома не понаслышке. Однако в различных литературных источниках для терминов теории КА приводятся несколько различные определения. Поэтому, не приводя здесь строгих формальных определений, уточним некоторые базовые понятия теории КА. Итак, автомат Мура — это КА, у которого выход является функцией состояния — выходные воздействия определены

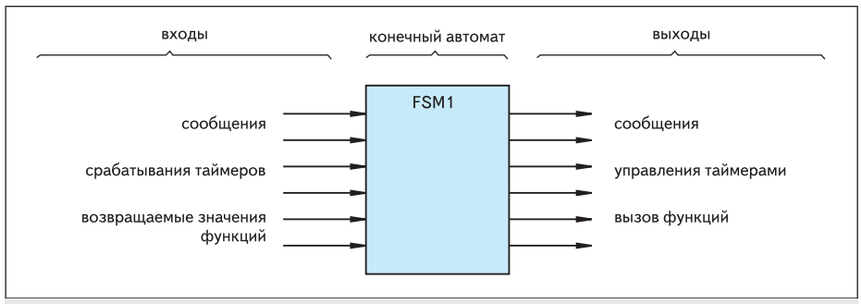

Рис. 1. Конечный автомат с входами и выходами

KiT#64(11).qxd 12/7/2006 1:29 PM Page 92

Ключевым понятием в рассматриваемой технологии программирования является конечный автомат (КА). Теория КА интенсивно развивалась в 50-е — 60-е годы прошлого века и нашла широкое применение во многих областях техники. Особенно плодотворной она оказалась при разработке трансляторов.

При этом КА рассматривается как своеобразное средство для «перевода» цепочек символов языка А в цепочки символов языка В, а правила перевода задаются структурой КА. Такое применение КА привело к тому, что до настоящего времени наиболее полное изложение теории КА можно найти в учебниках по компиляторам и математической лингвистике.

Достаточно широкое применение КА нашли также в задачах синтеза цифровых устройств, в задачах описания поведения сложных систем (в отрасли связи даже был создан специальный язык описания телекоммуникационных систем, базирующийся на КА).

Однако в программировании (в том числе встроенных систем) КА не нашли широкого применения. По сути, единственным инструментальным средством программирования микроконтроллеров, основанным на КА и доступным на сегодняшний день, является IAR Visual State.

Несколько иначе обстоят дела с программируемыми логическими контроллерами (ПЛК). Для них создан стандарт IEC 61131-3, реализованный во множестве систем программирования, из которых, пожалуй, самыми известными являются системы CoDeSys (фирма Smart Software Solutions) и ISaGRAF (фирма ICS Triplex ISaGRAF).

Стандарт IEC 61131-3 включает в себя несколько языков, в том числе и язык SFC (Sequential Function Chart). В рамках этого языка программа представлена как множество шагов (steps), между которыми осуществляются условные переходы (transitions и jumps). Программа может совершать действия (actions) как в состояниях (шагах), так и на переходах. Язык SFC поддерживает также параллельное выполнение участков программы.

Следует, однако, отметить, что языки стандарта IEC 61131-3 не являются идеальным средством программирования, им присущи и некоторые недостатки. Часть из них устраняется за счет введения расширений стандарта. Одно из таких расширений, призванное придать языкам стандарта объектно-ориентированные свойства, описано в работе [6].

В рамках данной статьи не будут рассматриваться достоинства и недостатки IEC 61131-3. Отметим лишь тот факт, что данный стандарт, хорошо зарекомендовавший себя при разработке ПО для ПЛК, не очень хорошо подходит для программирования микроконтроллеров, и дело тут не только в отсутствии соответствующих программных средств.

Это связано с тем, что микроконтроллерные системы зачастую более сложны и предполагают более гибкое использование аппаратных ресурсов, чем позволяет стандарт IEC 61131-3. Именно поэтому базовыми средствами программирования микроконтроллеров до сих пор являются языки ассемблера и Си.

SWITCH-технология делает программирование независимым как от применяемых аппаратных средств, так и языка программирования [2].

в состояниях. Автомат Мили — это автомат, у которого выходные воздействия определены для переходов между состояниями. И, наконец, смешанный автомат — это автомат, у которого выходные воздействия могут быть определены как в состояниях, так и на переходах. Также напомним читателям, что в английском языке термину конечный автомат соответствует термин Finite State Machine, или, сокращенно FSM.

Под словом «автомат» здесь и далее будем понимать исключительно КА. Прежде чем перейти к рассмотрению программных реализаций автоматов, укажем, что автомат имеет входы, выходы и переменную состояния (рис. 1).

Под входом понимается сообщение, срабатывание таймера, результат выражения, которое имеет логическое значение (в том числе возвращаемое функцией логическое значение). Доступ к аппаратным ресурсам микроконтроллера (регистрам периферийных устройств, портам ввода-вывода и т. п.) выполняется также путем вызова функций.

Под выходом автомата понимается отправка сообщения; запуск, останов или сброс таймера; вызов функции.

При этом выделяется особая переменная состояния — переменная, которая определяет текущее состояние автомата. Эта переменная должна быть доступна только своему автомату, ее изменение из других автоматов недопустимо.

Автомат может осуществлять действия (action) и деятельности (activity) автомата. И деятельность, и действие автомата относятся к его выходной активности.

При этом действием называется выходное воздействие, выполняемое однократно при входе в состояние или на переходе, а деятельностью — выходное воздействие, выполняющееся непрерывно в течение всего времени нахождения автомата в определенном состоянии. Также возможна реализация действий, выполняющихся на выходе из состояния.

Автомат Мура [2] и автомат смешанного типа могут выполнять как действия, так и деятельности, а автомат Мили может выполнять только действия.

В минимальном виде объявление автомата выглядит следующим образом:

```
//файл fsm1.h
#ifndef FSM1_h
#define FSM1_h

void InitFSM1(void);
void ProcessFSM1(void);
#endif
```

Перейдем к описанию автомата:

```
//файл fsm1.c
#include «fsm1.h»
#include «timers.h»
#include «messages.h»

char fsm1_state; //переменная состояния

void InitFSM1(void)
{
   fsm1_state = 0;
      //здесь можно выполнить инициализацию других
      //переменных автомата при их наличии
}

void ProcessFSM1(void)
{
   switch(fsm1_state)
   {
      case 0: //неактивное состояние
         if(GetMessage(MSG_FSM1_ACTIVATE))
         {
            fsm1_state = 1;
               //здесь выполняются действия, связанные
               // с активизацией компонента
            …
         };
         break;
   case 1: //активное состояние
      if(GetMessage(MSG_FSM1_DEACTIVATE))
      {
         fsm1_state = 0;
            //здесь выполняются действия, связанные с
            //деактивизацией компонента
         …
      };
      break;
   }
}
```

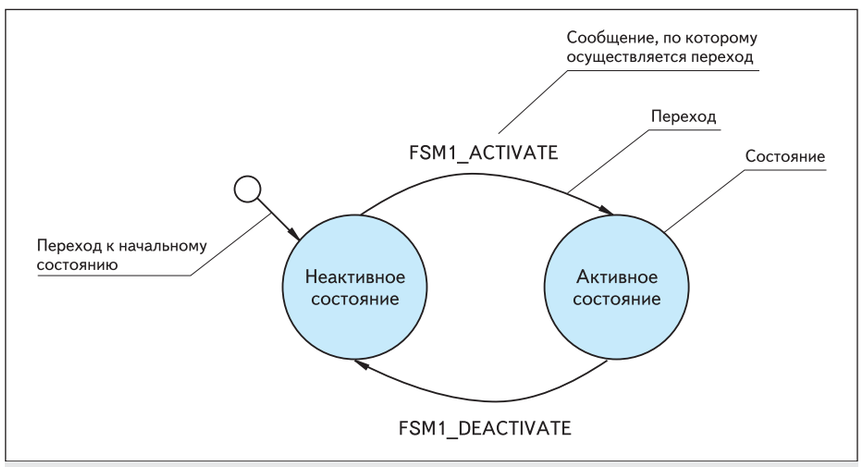

Рис. 2. Граф автомата

KiT#64(11).qxd 12/7/2006 1:29 PM Page 93

Приведенный исходный текст соответствует автомату Мили, который имеет два состояния: активное и неактивное, а также два входа: сообщения MSG_FSM1_ACTIVATE и MSG_FSM1_DEACTIVATE, которые переводят автомат в активное и неактивное состояние соответственно. Граф этого автомата изображен на рис. 2.

Этот автомат выполняет действия на переходах из одного состояния в другое (именно поэтому это автомат Мили), однако его достаточно легко преобразовать в автомат Мура или в смешанный автомат. Пока этот автомат не делает ничего полезного, это просто шаблон, на основе которого можно строить автоматы с более сложным поведением, что и будет сделано в дальнейшем.

В следующей части статьи будет продолжено рассмотрение реализации автоматов, а также механизмов таймеров и сообщений.

Автор выражает глубокую благодарность Анатолию Абрамовичу Шалыто за редактирование статьи и ценные замечания.

Литература

1. Татарчевский В. Некоторые мысли по поводу программирования встроенных систем //Компоненты и технологии. 2006. № 8.
2. Шалыто А. А. Switch-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука. 1998. **http://is.ifmo.ru/books/switch/1**
3. Шалыто А. А. Новая инициатива в программировании. Движение за открытую проектную документацию // Мир ПК. 2003. № 9. **http://is.ifmo.ru/works/open_doc/**
4. Непейвода Н.Н. Стили и методы программирования. М.: Интернет-Университет Информационных технологий. 2005. **http://www.intuit.ru/department/se/progstyles/9/1.html**
5. Зубинский А. FSM. **http://itc.ua/article.phtml?ID=19921&IDw=19**
6. Хесс Д. Объектно-ориентированные расширения МЭК 61131-3 // Современные технологии автоматизации. 2006. № 2.