Применение SWITCH-технологии при разработке прикладного программного обеспечения для микроконтроллеров. Часть 4

Владимир ТАТАРЧЕВСКИЙ
arktur04@mail.ru

В предыдущих статьях цикла [1–3] мы подробно обсудили организацию обмена сообщениями в программном обеспечении, построенном на основе SWITCH-технологии, и начали рассмотрение механизма таймеров. В этом материале мы продолжим обсуждение таймеров, а также приведем пример реализации SWITCH-программы на конкретном примере.

Как уже упоминалось в предыдущей статье [3], под таймером мы будем понимать не аппаратный узел микроконтроллера, а программную конструкцию (виртуальный таймер), позволяющую ввести в программу, основанную на SWITCH-технологии, условия переходов, зависящие от времени. При этом мы будем подразделять таймеры на локальные и глобальные. Локальным таймером будем считать такой таймер, действие которого ограничено одним состоянием конечного автомата, и мы можем использовать его значение для описания условий переходов только данного состояния. Область действия глобального таймера распространяется на несколько состояний автомата. Программа может содержать множество локальных и глобальных таймеров, но все они управляются прерыванием единственного аппаратного таймера микроконтроллера.

Рассмотрим вопросы, связанные с реализацией и применением таймеров.

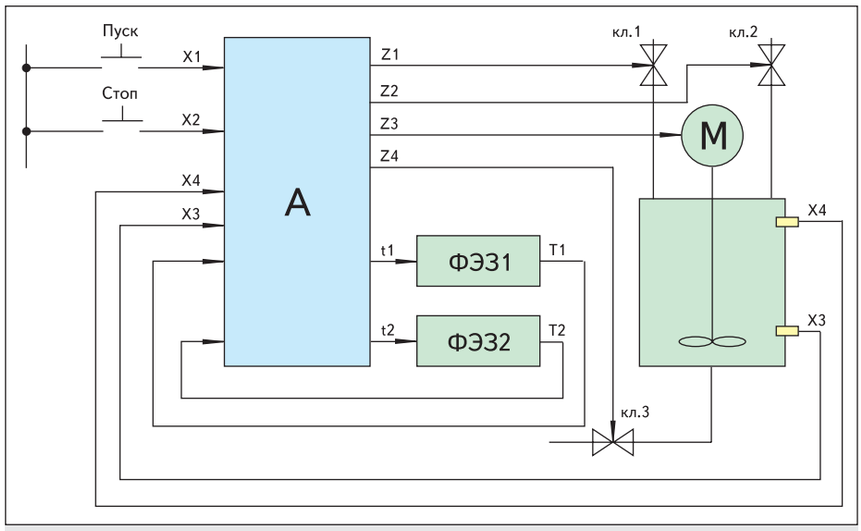

Рис. 1. Функциональная схема технологической установки: А — управляющий автомат; ФЭЗ1, ФЭЗ2 — функциональные элементы задержки; М — мотор; х3, х4 — датчики уровня жидкости; кл1, кл2, кл3 — клапаны

Локальные таймеры

Локальным таймером в предлагаемой модели является выделенная переменная, увеличивающее свое значение через фиксированные интервалы времени (например, каждую секунду). При входе в состояние значение этой переменной обнуляется (таймер инициализируется). В результате появляется возможность указывать в качестве условия выхода из состояния определенное значение данной переменной, соответствующее необходимому интервалу времени. Это позволяет упростить граф переходов конечного автомата. Наряду с локальными таймерами возможно также применение и глобальных таймеров, которые инициализируются в одном, а являются условиями выхода — в другом состоянии (состояниях).

Рассмотрим автомат, использующий таймеры. В качестве примера возьмем технологическую установку, описанную в работе [4]. Ее функциональная схема приведена на рис. 1.

Приведем словесное описание работы системы:

1. В исходном состоянии все клапаны закрыты, мотор (двигатель) «смешиватаеля» выключен.
2. Этап 1. При нажатии кнопки «Пуск» осуществляется налив жидкости путем открытия клапана 1 до срабатывания датчика х3.
3. Этап 2. Осуществляется налив жидкости путем открывания клапана 2 до срабатывания датчика х4.
4. Этап 3. Включается двигатель «мешалки» на время t1.
5. Этап 4. Производится слив смеси путем открывания клапана 3 на время t2.
6. После выполнения пункта 5 система возвращается в исходное состояние.
7. При нажатии кнопки «Стоп» во время выполнения пунктов 2-4 система переходит к сливу смеси (пункт 5).

Построим по словесному описанию автомат Мура, использующий локальные таймеры (рис. 2).

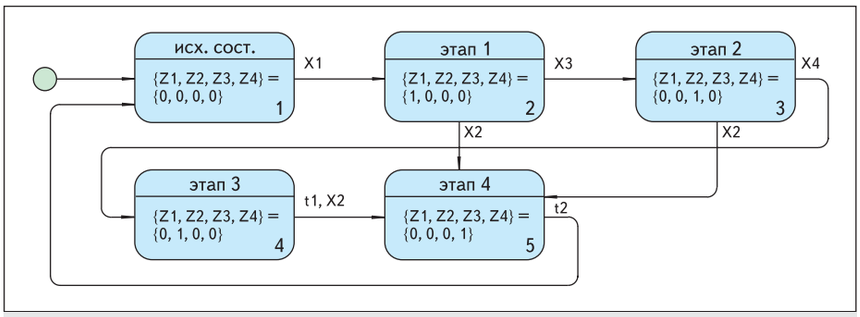

Рис. 2. Граф конечного автомата системы управления

Заметим, что граф переходов на рис. 2 состоит всего из пяти состояний. При этом за счет введения локального таймера (всего одной переменной на весь автомат) мы избавились от «искусственных» входов T1, T2, выходов t1, t2 и двух элементов задержки — ФЭЗ1 и ФЭЗ2. Можно заметить, что у данной схемы есть один недостаток — из всех «рабочих» состояний, кроме последнего, (из состояний 2–4) в состояние 5 ведут переходы с условием x2, соответствующим нажатию кнопки «Стоп». С учетом того, что в сложных технологических установках число состояний автомата в рабочем цикле может достигать нескольких десятков (и даже сотен), такие переходы могут сильно загромождать схему, затрудняя ее читаемость. Также подобный «прямой» подход может привести к ошибкам при разработке программы и особенно при ее модификации (например, из какого-либо состояния забыли провести переход по кнопке «стоп»). Теперь представим себе, что в процессе разработки было решено использовать в качестве кнопки «Стоп» не нормально-разомкнутую, а нормально-замкнутую систему (как это и имеет место в стандартных двухкнопочных постах «пуск-стоп»). Для этого необходимо произвести в соответствующих переходах из состояний 2–4 замену x2 на ~x2 (x2 инверсное). При этом небольшое, казалось бы, изменение является «ошибкоопасным», так как замену придется производить во всех рабочих состояниях. Поэтому целесообразно преобразовать граф переходов на рис. 2 в схему с параллельными потоками, изображенную на рис. 3.

Несмотря на то, что количество состояний в новой схеме увеличилось до шести, число переходов в ней уменьшилось, а ее наглядность возросла. И, что самое главное, возросла модифицируемость схемы. Теперь можно ввести любое количество состояний рабочего цикла вместо состояний 2–4, не заботясь о корректности срабатывания кнопки «Стоп», а замена условия останова системы (например, замена x2 на ~x2) влечет за собой изменение только в одном месте программы.

Рассмотрим реализацию данного автомата (рис. 3) на языке C:

```
while(1) {
//Здесь должен производиться опрос входов системы управления

//основной поток программы (состояния 1–5 на рис. 3)
 switch(state){
   case 1: //состояние 1
       //устанавливаем выходы автомата
       z1 = z2 = z3 = z4 = 0;
       //условие перехода
       if(x1) state = 2;
       break;
   case 2: //состояние 2
       //устанавливаем выходы автомата
       z1 = 1;
       z2 = z3 = z4 = 0;
       //условие перехода
       if(x3) state = 3;
       break;
   case 3: //состояние 3
       //устанавливаем выходы автомата
       z1 = z2 = z4 = 0;
       z3 = 1;
       //условие перехода
       if(x4) state = 4;
       break;
   case 4: //состояние 4
       //устанавливаем выходы автомата
       z1 = z3 = z4 = 0;
       z2 = 1;
       //запускаем локальный таймер
       starttimer();
       //условие перехода
       if(GetTimer(TimerID) >= t_1) state = 5;
       break;
   case 5: //состояние 5
       //устанавливаем выходы автомата
       z1 = z2 = z3 = 0;
       z4 = 1;
       //запускаем локальный таймер
       starttimer();
       //условие перехода
       if(GetTimer(TimerID) >= t_2) state = 1;
       break;
 };
//контроль кнопки «стоп»
/*этот поток состоит из одного состояния и выполняется параллельно основному потоку программы. При активации входа x2 (кнопка «стоп»), этот поток принудительно переводит основной КА в состояние 5 (слив смеси и завершение работы).*/
 if((state > 1)&&(state < 5) && x2)
    state = 5;
//Здесь должна производиться установка выходов системы управления

};
```

Следует отметить, что в данном исходном тексте приведена скорее «условная» реализация, чем реальный код, в нем опущены многие подробности реализации, такие как объявления переменных и функций, опрос входов автомата и выдача воздействий на выходы.

Однако на примере данного исходного текста мы еще раз можем убедиться в преимуществе SWITCH-технологии перед традиционным подходом. В самом деле, SWITCH-программа всегда выполняется циклически, и на каждом цикле мы считываем состояние входных линий контроллера и устанавливаем выходные сигналы контроллера. Программа нигде не имеет «петель», вложенных циклов ожидания того или иного события, которые помешали бы нам обеспечить корректную работу параллельных потоков и функций обслуживания ввода-вывода.

Вернемся к таймерным условиям переходов. На рис. 3 из состояний 4 и 5 ведут переходы с пометками t1 и t2 соответственно. Это означает, что переход происходит по истечении времени t после *входа* автомата в состояние. Программно такое поведение реализуется достаточно просто:

```
case N:; //N — номер текущего состояния
   //---------------
   //здесь выполняем действие (деятельность) состояния
   //---------------
   starttimer(); //обнуляем таймер при входе в состояние
   if(GetTimer(TimerID) >= t) state = N_1; //если время t истекло, совершаем переход в состояние N_1
   break;
```

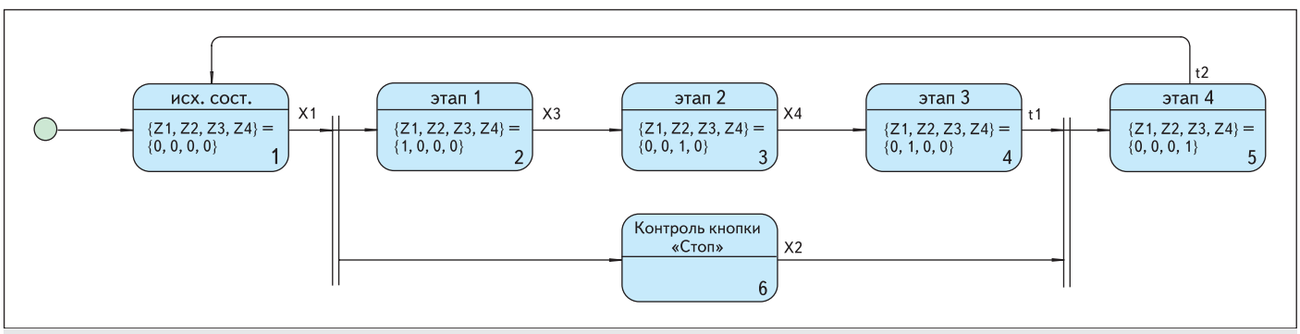

Рис. 3. Граф конечного автомата системы управления с параллельными потоками

Время t можно указывать как непосредственно в периодах аппаратного таймера (t >= 1234), так и в «естественных» единицах, объявив их в тексте программы:

```
#define sec 100 // период аппаратного таймера 10 мс
#define min 60 * sec
#define hour 60 * min
#define day 24 * hour
```

Тогда условие перехода может выглядеть, например, так: (GetTimer(TimerID) >= 5*min + 30*sec), что гораздо нагляднее. Разумеется, мы можем описывать и сложные условия, например:

```
If((GetTimer(TimerID) >= t) && GetInput(1)) state = N_1; //если время t истекло и вход 1 активен, совершаем переход в состояние N_1
```

Функция starttimer(), обнуляющая таймерную переменную, должна выполняться только при первом входе в состояние, что решается, например, следующим образом:

```
int state, _state; //state — переменная состояния автомата
//_state — вспомогательная переменная, предназначенная для определения факта входа автомата в состояние
void starttimer(char TimerID)
{
   if(state <> _state)
       ResetTimer(TimerID); // обнуляем таймерную переменную при смене состояния автомата
   _state = state;
}
```

Разумеется, мы должны инкрементировать переменную таймера через равные промежутки времени в обработчике прерывания *аппаратного* таймера микроконтроллера [3].

И наконец, следует заметить, что при использовании в программе нескольких параллельных автоматных потоков мы должны ввести отдельную таймерную переменную для каждого потока.

У приведенной программы есть один недостаток. В состояниях 2, 3 (рис. 3) программа ожидает срабатывания датчиков x3 и x4 соответственно. Если какой-либо из этих датчиков не сработает (например, по причине отсутствия жидкости в трубопроводе), программа будет находиться в соответствующем состоянии неограниченное время. Приведенную ранее программу можно несколько улучшить, если ввести переходы по времени из состояний 2, 3 (рис. 3) в некое состояние, которое можно назвать «аварийным». Вообще, локальные таймеры очень удобны для описания реакции системы на различные нештатные ситуации, для контроля исправности системы по времени выполнения каких-либо операций и т. п. И в данном случае SWITCH-технология демонстрирует явное преимущество перед «традиционным» подходом. Приведем цитату из [5]: «…алгоритм должен быть дополнен «нехорошими» условиями и, соответственно, действиями для выхода из «нехороших» ситуаций. Это отразить на схеме алгоритма, сильно не переделывая его, не просто… Такая схема алгоритма обычно очень громоздка, и поэтому она на практике почти никогда не используется: схему алгоритма один раз еще можно построить, но учитывать в ней *все* особенности реального алгоритма, а тем более вносить в нее изменения на всех этапах жизненного цикла программы не будет практически *никто*».

Изложенный подход к применению таймеров в SWITCH-программах имеет ряд очевидных преимуществ перед «прямым» использованием аппаратного таймера микроконтроллера. Во-первых, появляется возможность организации неограниченного количества «виртуальных» таймеров, управляемых одним аппаратным таймером (экономия ресурсов контроллера). Во-вторых, не требуется перепрограммировать аппаратный таймер для того, чтобы производить отсчет различных интервалов времени. В-третьих, в программе может производиться одновременный отсчет нескольких временных интервалов в разных потоках.

Как уже упоминалось ранее, локальные таймеры имеют ограниченную функциональность, с их помощью мы можем всего лишь описывать условия переходов по времени, прошедшему с момента входа автомата в состояние. Гораздо большей гибкостью обладают глобальные таймеры, которые будут рассмотрены в следующей статье цикла.

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

Литература

1. Татарчевский В. А. Применение SWITCH-технологии при разработке прикладного программного обеспечения для микроконтроллеров. Часть 1 // Компоненты и технологии. 2006. № 11.
2. Татарчевский В. А. Применение SWITCH-технологии при разработке прикладного программного обеспечения для микроконтроллеров. Часть 2 // Компоненты и технологии. 2006. № 12.
3. Татарчевский В. А. Применение SWITCH-технологии при разработке прикладного программного обеспечения для микроконтроллеров. Часть 3 // Компоненты и технологии. 2007. № 1.
4. Шалыто А. А. SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука. 1998.
5. Вавилов К. В., Шалыто А. А. Что плохого в неавтоматном подходе к программированию контроллеров? //Промышленные АСУ и контроллеры. 2007. № 1.