# Применение SWITCH-технологии при разработке прикладного программного обеспечения для микроконтроллеров. Часть 8

**В предыдущих статьях цикла был рассмотрен ряд аспектов использования SWITCH-технологии при построении программного обеспечения встраиваемых систем на базе микроконтроллеров. В этой статье представлены алгоритмы, часто применяемые в программном обеспечении микроконтроллерных систем, и их реализация на основе SWITCH-технологии.**

**Владимир ТАТАРЧЕВСКИЙ**
arktur04@mail.ru

В подавляющем большинстве устройств на базе микроконтроллеров применяется клавиатура, имеющая, как правило, матричную организацию. Следовательно, в программе такого устройства должен содержаться алгоритм, производящий опрос клавиатурной матрицы и осуществляющий подавление дребезга контактов. Во многих случаях этот алгоритм также должен корректно обрабатывать различные сочетания нажатых клавиш. Алгоритмы такого рода не очень сложны и могут быть реализованы на основе циклов и задержек. Примеры программ подобного рода можно найти в [1, стр. 156]. Однако такая «очевидная» реализация имеет недостаток: будучи встроена в основную программу, эта подпрограмма блокирует исполнение основной программы на время своего цикла. Этого недостатка лишена реализация на основе SWITCH-технологии.

В качестве примера рассмотрим программу обслуживания ввода для клавиатуры, подключенной к микроконтроллеру по схеме, приведенной на рис. 1. Программа должна производить опрос клавиатурной матрицы, иметь защиту от дребезга контактов клавиатуры, корректно обрабатывать сочетания двух клавиш (например, сочетание клавиш Shift+6 должно определяться как нажатие «виртуальной» клавиши «стрелка влево»). Также программа должна иметь функцию автоповтора, подобно тому, как работает ввод с клавиатуры компьютера.

На рис. 2 приведен граф автомата, реализующего перечисленные функции.

Из рис. 2 видно, что устройство представляет собой автомат Мили (все действия производятся на переходах). Автомат использует один таймер (KEYB_TIMER). Величины задержек определяются следующими константами (они описаны в программе):

```
//период таймера 10 мс
#define debouce 10         //задержка для подавления дребезга
                           //(10 * 10 = 100 мс)
#define first_delay 50     //задержка первого повтора кода клавиши
                           //при ее удержании (50 * 10 = 500 мс)
#define auto_repeat 10     //задержка автоповтора кода клавиши при
                           //ее удержании (10 * 10 = 100 мс)
```

На рис. 3 показана временная диаграмма работы автомата при подаче на вход сигнала от кнопки с дребезгом.

На рис. 3 числа обозначают состояния автомата в соответствии с рис. 2. Временные задержки обозначены следующим образом: td (debounce) — задержка подавления дребезга, tfr (first repeat) — задержка перед первым повтором нажатия клавиши, tar (auto repeat) — период автоповтора нажатий клавиши.

При анализе рис. 2, 3 может сложиться впечатление, что такая программа предназначена для обработки нажатий только одной клавиши, тогда как необходимо обрабатывать нажатие всех клавиш клавиатуры. Кроме того, на рис. 2 не показан сам цикл сканирования клавиатурной матрицы. Однако никакой ошибки здесь нет. Цикл сканирования (он не приводится здесь, так как очень прост) активируется перед вызовом функции ProcessKeyFSM, реализующей автомат, изображенный на рис. 2. В нем формируется скан-код — число, соответствующее текущему («мгновенному») состоянию клавиатурной матрицы. Такой скан-код может представлять собой, например, 16-битовое беззнаковое целое, 12 младших разрядов которого соответствуют 12 кнопкам клавиа-

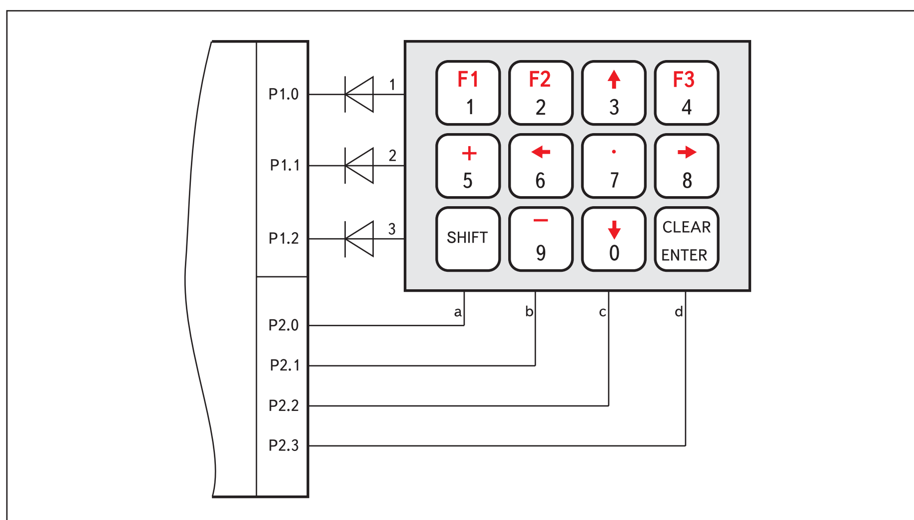


**Рис. 1.** Пример подключения матричной клавиатуры к микроконтроллеру

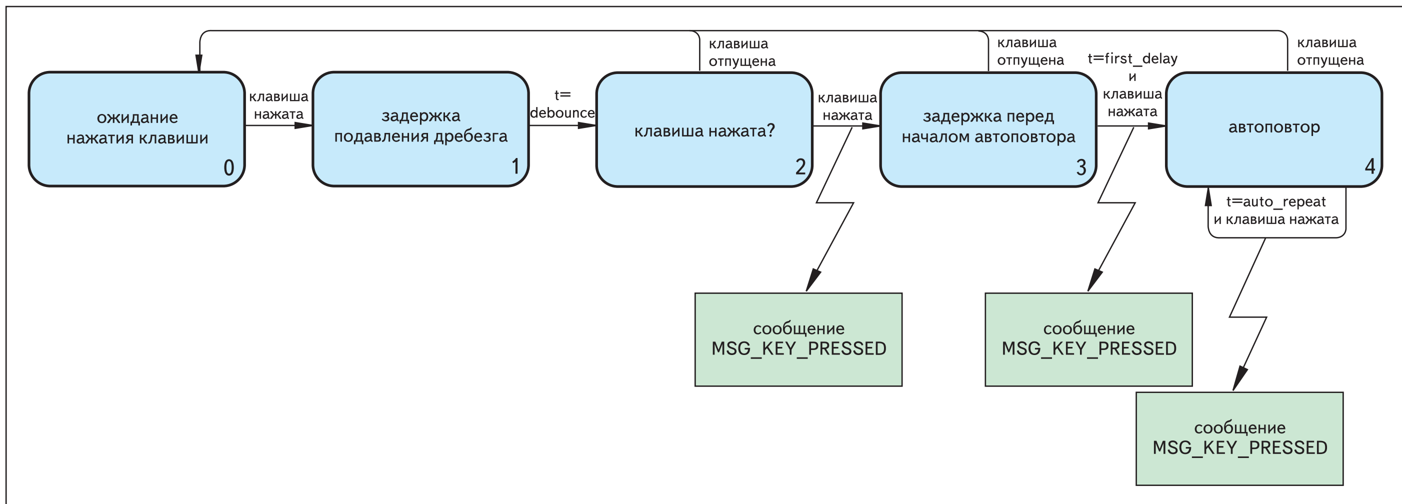


**Рис. 2.** Автомат драйвера клавиатуры

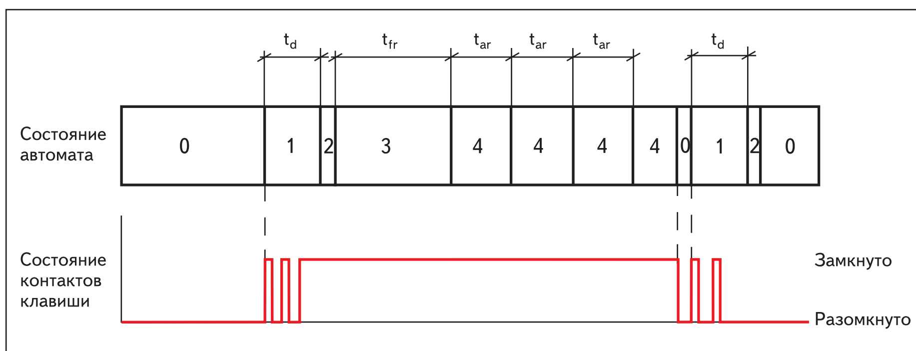


**Рис. 3.** Временная диаграмма работы автомата

туры (рис. 1), при этом нулю в разряде соответствует разомкнутая кнопка, а единице — замкнутая. Таким образом, каждой клавише и каждому сочетанию нажатых клавиш соответствует уникальный скан-код (если ни одна клавиша не нажата, скан-код равен нулю). В листинге программы (он приведен далее) скан-код клавиатуры содержится в переменной key_code. Также имеется переменная _key_code, в которой содержится состояние клавиатуры в предыдущем цикле. Она нужна для того, чтобы автомат мог корректно обработать ситуацию, при которой пользователь нажимает клавишу, удерживая при этом другую (или отпускает одну клавишу, удерживая нажатой другую). Для корректной обработки подобных ситуаций достаточно полагать, что изменение скан-кода клавиатуры равнозначно отпусканию клавиши. Тогда при изменении скан-кода автомат перейдет в состояние 0, и следующая комбинация клавиш будет обработана вновь. Таким образом, только переход 0-1 («клавиша нажата») реализуется буквально:

```
if (key_code > 0) {...};
```

Переходы 2-0, 3-0, 4-0 («клавиша отпущена») и 2-3, 3-4, 4-4, включающие в себя условие «клавиша нажата», реализуются как проверка совпадения текущего скан-кода и предыдущего его значения:

```
if (key_code == _key_code){
//клавиша нажата
}
else{
//клавиша отпущена
};
```

Теперь можно записать реализацию автомата на языке Си.

```
char key_state;//переменная состояния автомата

void ProcessKeyFSM(void){
switch (key_state){
/*состояние 0. В этом состоянии автомат ожидает нажатия на клавишу*/
case 0: //клавиша не нажата
if (key_code > 0){
_key_code = key_code;
ResetTimer(KEYB_TIMER);
key_state = 1;
};
break;
/*состояние 1. В этом состоянии вводится задержка на время debounce (ожидание завершения переходного процесса при замыкании контактов клавиши)*/
case 1: //задержка подавления дребезга
if (GetTimer(KEYB_TIMER) > debounce)
key_state = 2;
break;
/*состояние 2. В этом состоянии фиксируется факт нажатия клавиши и формируется сообщение MSG_KEY_PRESSED. Если клавиша не нажата, возврат в состояние 0*/
case 2: //если клавиша нажата, посылаем сообщение
if (key_code == _key_code){
ResetTimer(KEYB_TIMER);
SendMessage(MSG_KEY_PRESSED);
key_state = 3;
}
else
key_state = 0;
break;
/*состояние 3. В этом состоянии автомат формирует сообщение MSG_KEY_PRESSED, если пользователь удерживает клавишу в течение времени first_delay. Если клавиша отпущена, возврат в состояние 0*/
case 3:
if (key_code == _key_code){
if (GetTimer(KEYB_TIMER) >= first_delay){
ResetTimer(KEYB_TIMER);
SendMessage(MSG_KEY_PRESSED);
key_state = 4;
};
}
else
key_state = 0;
break;
/*состояние 4. В этом состоянии автомат формирует последовательность сообщений MSG_KEY_PRESSED с периодом auto_repeat, если пользователь продолжает удерживать клавишу. Если клавиша отпущена, возврат в состояние 0*/
case 4:
if (key_code == _key_code){
if (GetTimer(KEYB_TIMER) >= auto_repeat){
ResetTimer(KEYB_TIMER);
SendMessage(MSG_KEY_PRESSED);
};
}
else
key_state = 0;
break;
};
}
```

Следует отметить еще один важный момент. Представленный автомат посылает сообщения основной программе при нажатии клавиш, но не сообщает, какая именно клавиша (или сочетание клавиш) нажата. Если в программе реализован какой-либо механизм сообщений с параметрами, то код нажатой клавиши можно передавать в качестве параметра сообщения. При использовании в программе простейшего механизма передачи сообщений без параметров, получение кода нажатой клавиши основной программой может быть реализовано следующим образом.

В программе перечисляются все используемые клавиши и их сочетания (сочетание кла-

виш, имеющее самостоятельный смысл, будем называть «виртуальной клавишей»):

```
#define KEY_0 1
#define KEY_1 2
...
#define KEY_9 10
#define KEY_ENTER 11
#define KEY_F1 12
#define KEY_F2 13
#define KEY_F3 14
//коды виртуальных клавиш (сочетания клавиша + «Shift»)
#define KEY_UP 15
#define KEY_DOWN 16
#define KEY_LEFT 17
#define KEY_RIGHT 18
#define KEY_MINUS 19
#define KEY_DOT 20
#define KEY_CLEAR 21
```

Также в программу вводится функция, преобразующая скан-код в код клавиши (ее реализация здесь не приводится):

```
char GetKeyCode(void);
```

Теперь основная программа, получив сообщение MSG_KEY_PRESSED, может получить код нажатой клавиши и предпринять какие-либо действия:

```
if(GetMessage(MSG_KEY_PRESSED)){
switch(GetKeyCode()){
case KEY_ENTER: //ENTER — завершение работы
...
break;
case KEY_CLEAR: //CANCEL — отмена
...
break;
```

Приведенный здесь модуль работы с клавиатурой играет в SWITCH-программе роль своеобразного драйвера клавиатуры. По сравнению с «традиционной» реализацией такой подход имеет ряд преимуществ. Одно из них уже было приведено: при работе драйвера клавиатуры основная программа не испытывает существенных задержек, связанных с работой антидребезгового алгоритма. Другие преимущества заключаются в следующем. Драйвер имеет высокую степень автономности: его работа не зависит от функционирования основной программы. Основная программа также не обязана «знать» о внутреннем алгоритме работы драйвера, она только получает от него сообщения о нажатии клавиш, и все, что нужно для обеспечения корректного взаимодействия основной программы и драйвера — описать несколько сообщений и коды виртуальных клавиш. Отсюда вытекает еще одно важное преимущество использования автоматов-драйверов: очень высокая степень модифицируемости программы. При внесении изменений в драйвер не требуется никаких изменений в основной программе и наоборот. При изменении аппаратной части устройства изменения вносятся только в драйвер. Даже при переносе программы на микроконтроллер другого типа, с другой организацией портов ввода/вывода потребуется только корректно переписать драйверы с учетом изменившейся архитектуры системы; те части программы, которые отвечают непосредственно за логику функционирования программы, остаются без изменений. Еще одним большим преимуществом автоматного подхода является высокая степень повторного использования ранее написанного кода. Написав драйвер один раз, его возможно использовать в последующих разработках многократно, уже не вникая в алгоритм его работы. Автономность драйвера, ограниченность его связей с основной программой также позволяют достичь эффективного разделения труда между программистами-разработчиками драйверов и программистами-разработчиками «основной» программы, то есть высокой степени параллельности их работы.

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

## Литература

1. Сташин В. В., Урусов А. В., Мологонцева О. В. Проектирование цифровых устройств на однокристальных микроконтроллерах. М.: Энергоатомиздат. 1990.