Статья опубликована в журнале “Промышленные АСУ и контроллеры”. 2005. №11, с. 49– 52.

И.З. Альтерман, А.А. Шалыто

Формальные методы программирования логических контроллеров

Предлагаются методы непосредственного построения программ на языке функциональных блоков по графам переходов. Примеры реализации рассмотрены для контроллеров *Simatic S7*.

The present article describes the methods of developing programs based on function blocks language by the agency of transition graphs. The ways of programming are displayed in terms of *Simatic S7* controller .

Постановка задачи

При анализе программы, реализуемой контроллером, часто бывает так, что нет исходных текстов программ с комментариями, нет и полной документации, объясняющей функциональные особенности программы. А что, если программу изначально строить по формальным принципам, изложенным, например, в работе [1]. Тогда логика работы программы может быть легко восстановлена из ее текста за счет формального преобразования в изоморфный граф переходов, описывающий поведение объекта. Это позволит затратить минимум времени на анализ программы и ее модификацию в случае необходимости. Такой граф может служить не только компактным и наглядным способом представления алгоритма, но и общим языком общения между программистом-исполнителем и технологом-заказчиком.

В предлагаемой статье предлагается рассмотреть практические вопросы программирования логических контроллеров на основе автоматного подхода [1, 2].

При этом отметим, что программа в контроллере выполняется по шагам и циклически.

Суть автоматного программирования, состоит в том, что управляющая программа строится и функционирует как конечный автомат, который может находиться в каждый момент времени только в одном из N состояний. При этом в каждом цикле для текущего состояния (иначе ШАГА программы) вычисляются логические условия, позволяющие изменить состояние программы.

Такой подход однозначно определяет поведение программы в каждом цикле по отношению к объекту управления – обеспечивает ее детерминированность. Основное достоинство этого подхода, что искомая программа "строится" по формальным правилам.

Рассмотрим пример реализации программы для управления клапаном на основе автоматного подхода. Приведем словесное описание алгоритма [1].

1. При нажатии кнопки "Откр." (X1) клапан начинает открываться.
2. После его открытия срабатывает сигнализатор открытого положения, зажигается лампа "Откр." (X4) и управляющий сигнал с клапана снимается (Z2).
3. При нажатии кнопки "Закр." (X2) клапан начинает закрываться.
4. После его закрытия срабатывает сигнализатор закрытого положения, зажигается лампа "Закр." (X3) и управляющий сигнал с клапана (Z1) снимается.
5. Если в течение 3 с клапан не откроется или не закроется, то управляющий сигнал с клапана снимается и зажигается лампа контроля "Неисправность" (Z3).
6. Сброс сигнала контроля осуществляется нажатием кнопки "Разблок." (X5).

Схема связей “источник информации – управляющий автомат – исполнительные механизмы”, задающая интерфейс автомата, приведена на рис.1.

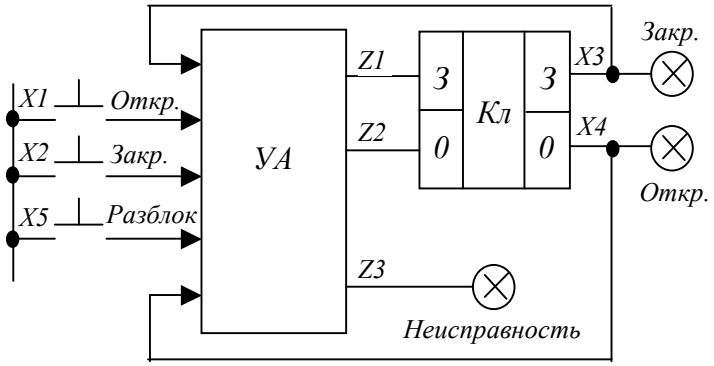


Рис. 1. Схема связей автомата

Построим по словесному описанию граф переходов автомата (рис.2).

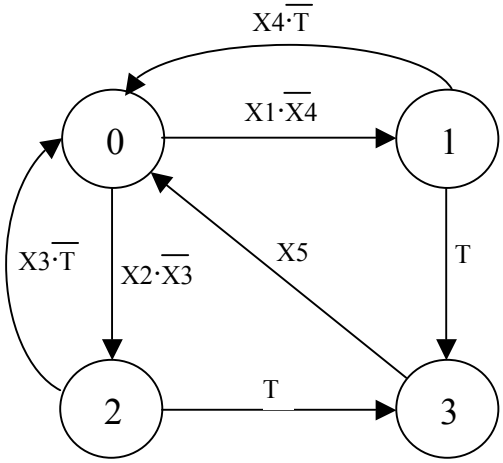


Рис.2. Граф переходов автомата

При анализе состояний графа переходов обычно решаются следующие вопросы:

- список команд на включение (команда активна);
- список команд на выключение (если для реализации графа переходов допускается использование команд с запоминанием – с памятью);
- максимальное/минимальное время активности состояния (Tmax/Tmin).

Обозначим через `Si` состояние автомата с номером `i`. Тогда для рассматриваемого примера определяем:

S_0 = { Z1=0; Z2=0; Z3=0; Tmax/Tmin не учитываются };

S_1 = { Z2=1; Tmax= 3 с };

S_2 = { Z1=1; Tmax= 3 с };

S_3 = { Z3=1; Tmax/Tmin не учитываются }

Переход из одного состояния графа в другое представляет собой логическое условие, которое всегда проверяется для текущего состояния. Если условие – истина, то переход выполняется. В случае множественных переходов из текущего состояния и активности одновременно нескольких условий, приоритетным будем считать тот переход, логическая функция которого должна быть вычислена по ходу выполнения программы раньше.
Обозначим `Tij` переход из состояния `Si` в состояние `Sj`. Тогда для рассматриваемого примера получим следующие логические выражения переходов:

$T_{01} = X1 \,\&\, \overline{X4}$; $T_{02} = X2 \,\&\, \overline{X3}$; $T_{10} = X4 \,\&\, \overline{T}$; $T_{20} = X3 \,\&\, \overline{T}$; $T_{13} = T$; $T_{23} = T$; $T_{30} = X5$

Рассмотрим два подхода, позволяющих выполнить формализованный перевод искомого графа переходов в программный код для программируемых контроллеров.

2. Метод шаговых меток

Метод шаговых меток основывается на кодировании состояний графа с помощью битовых ячеек памяти, которые будем называть "шаговыми метками". Представим программу управления клапаном из двух частей (рис.3), одна из которых отвечает за реализацию переходов, а вторая – за формирование команд для всех состояний. Назовем первую часть блоком формирования переходов (БФП), а вторую – блоком формирования команд (БФК). Еще раз отметим цикличность программы, которая вытекает из циклического характера работы программируемого контроллера.

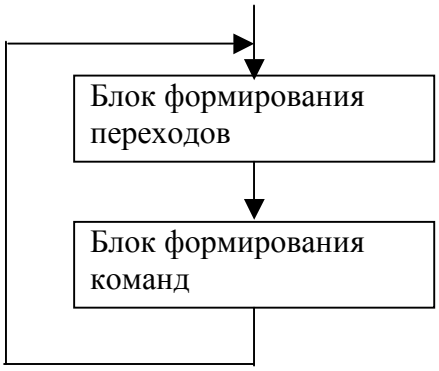


Рис.3. Составные части программы управления клапаном

Формальный переход к программе заключается в том, что пользователю предлагается наполнить типовые структуры блоков формирования переходов (рис.4) и команд (рис.5) конкретными данными для заданного графа переходов.

Управление шаговыми метками выполняется по триггерным функциям `Set` и `Reset`. При срабатывании перехода `Tij` текущая шаговая метка `SMi` сбрасывается и сразу же устанавливается новая активная метка `SMj`. При рестарте контроллера должна обеспечиваться установка в единицу шаговой метки, соответствующей исходному состоянию.

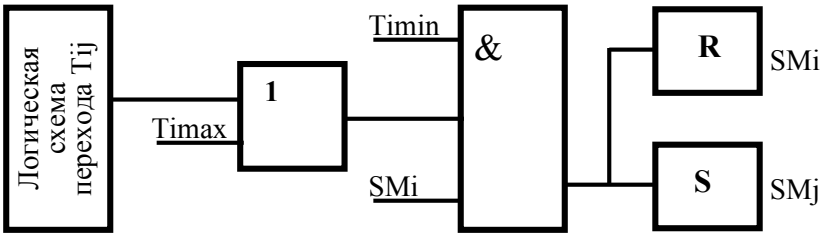


Рис. 4. Реализация перехода `Tij` в блоке формирования переходов

Активность команд ставится в прямую зависимость от активности шаговой метки. Если одна и та же команда активна сразу в нескольких состояниях графа, то она формируется по схеме "ИЛИ" от соответствующих шаговых меток. Если для i-го состояния графа предусмотрен учет времени его действия, то по фронту шаговой метки запускаются соответствующие таймеры задержки. Таймер `Timax` определяет максимальное время

действия i-го шага. При его срабатывании i-шаг заменяется на новый. Таймер `Timin` предназначен для задержки времени выполнения i-го шага минимум на время `Tmin`. Он используется в тех случаях, когда необходимо исключить преждевременный переход объекта в новое состояние.

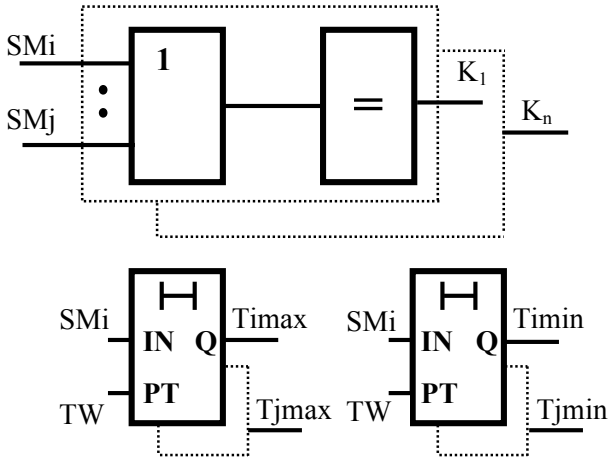


Рис.5. Компоненты блока формирования команд

В качестве примера реализации программы рассмотрим контроллеры фирмы *Siemens* серии *Simatic S7-300/400* [3]. Выполним управление клапаном на базе программного блока *FB* (Function Block), поскольку ти блоки имеют собственную статическую память, часть которой будем использовать для кодирования шаговых меток. Для этого, в разделе описания статической памяти блока, объявим переменную SM как битовый массив с инициализацией первой ячейки:

```
VAR      SM : ARRAY  [ 0 .. 3 ] OF BOOL  := TRUE, 3 (FALSE);
```

Для таймеров задержки будем использовать системную функцию процессора `TON` (SFB4), которая соответствует стандарту *IEC 1131-3*. Для оценки времени действия шагов один и два потребуется два таймера. Поэтому в разделе описания статической памяти блока FB объявим еще две переменные `T1max` и `T2max` с типом `SFB4`:

```
VAR      T1max : SFB 4;      T2max : SFB 4;
```

Для представления программы управления клапаном воспользуемся языком *FBD* (Function Block Diagram). Фрагмент блока формирования переходов для переходов T 20 и T13 приведен на рис.6.

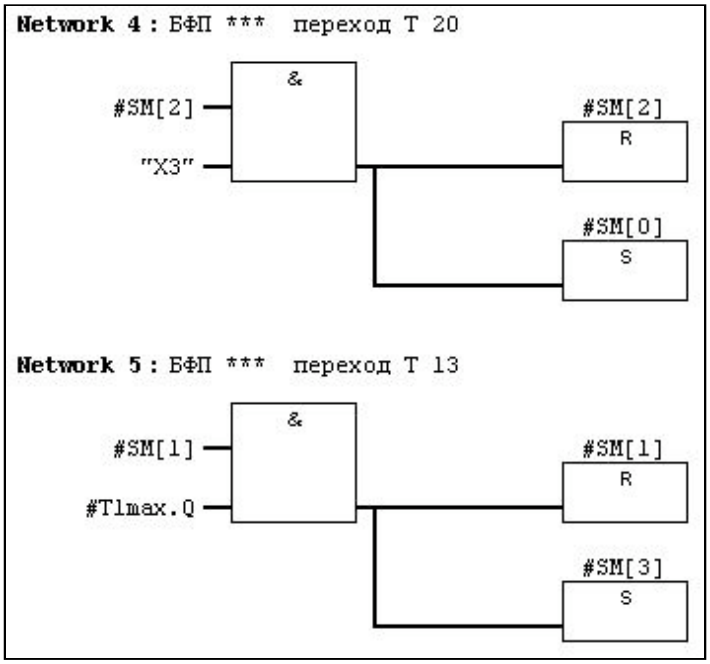


Рис. 6. Фрагмент программы для переходов T 20 и T13

Поскольку в рассматриваемом графе всего семь различных переходов, то в блоке FB записывается семь однотипных секций программного кода. Блок формирования команд (рис. 7) формирует три команды и управляет двумя таймерами.

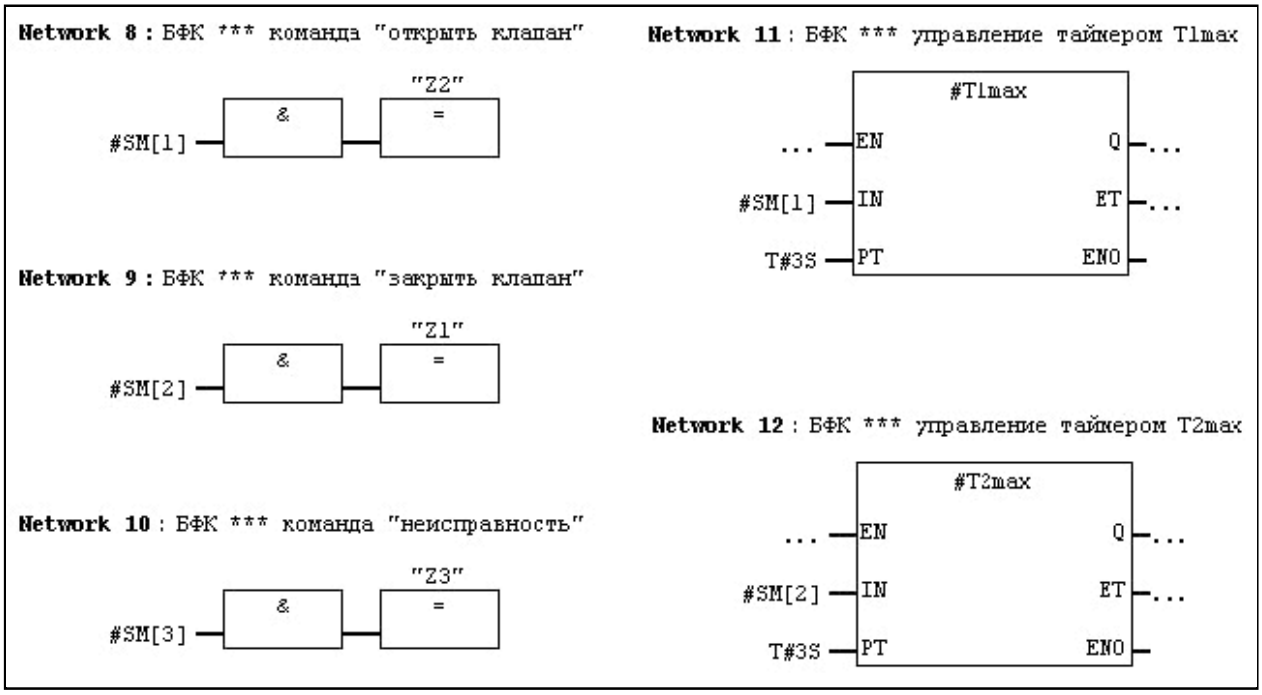


Рис. 7. Фрагмент программы для блока формирования команд

Предложенный подход значительно сокращает время на разработку и тестирование. Управляющие программы с использованием шаговых меток имеют четкую и ясную структуру, по которым можно легко восстановить граф переходов, а, следовательно, и поведение объекта. Однако, при этом ресурсы процессора используются недостаточно эффективно. Действительно, в каждом цикле происходит вычисление логических условий переходов и формирование команд не только для активного, а для всех состояний, что увеличивает время цикла. Количество используемых таймеров прямо зависит от количества “временных” состояний, что также нерационально. Указанных недостатков при реализации графа переходов можно избежать, если воспользоваться методом шаговых блоков, который в полной мере соответствует SWITCH-технологии [1].

Метод шаговых блоков

Метод шаговых блоков использует многозначное кодирование состояний графа, а программа выполняет условные вызовы процедур по текущему номеру состояния. Будем называть эти процедуры "шаговыми блоками". Шаговый блок `SBi` решает **только** задачи для i-го состояния: реализация возможных переходов, формирование команд на объект управления и управление таймерами для учета времени. Структура программы (рис.8) отражает условный характер вызова шаговых блоков по числовой переменной `SW`, используемой для хранения номера активного шага.

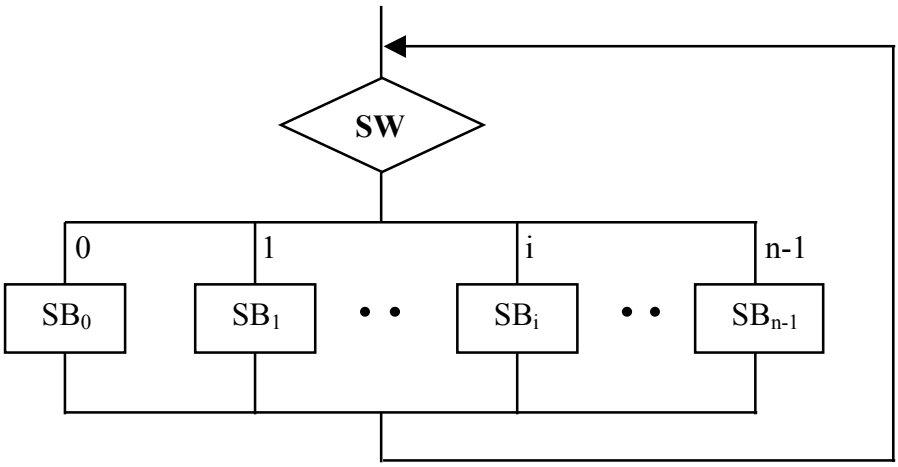
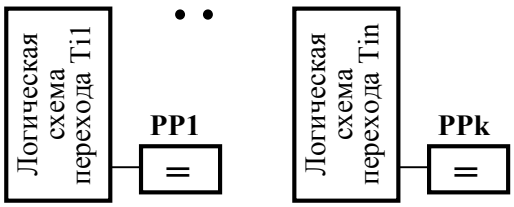
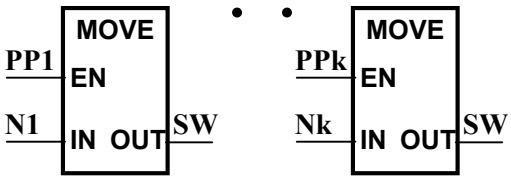
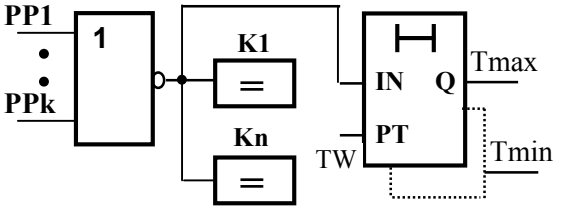


Рис.8. Структура программы для метода шаговых блоков

Шаблон для блока `SBi, приведенный в таблице 1,` позволяет формализовать процесс разработки программы. Результаты вычисления функций переходов для каждого шага запоминаются во временных переменных `PPi`. При выполнении хотя бы одного условия перехода блокируются все команды шага и изменяется числовая переменная `SW`, что приводит к выполнению нового шага уже на следующем цикле процессора.

Таблица 1. Типовая структура шагового блока

N	Действия	Программное решение
1.	Установка признаков активации переходов	
2.	Формирование нового слова состояния (SW) N1 - номер шага для 1-го перехода Nk - номер шага для k-го перехода	
3.	Формирование команд шага (без памяти) и управление таймерами	

Количество таймеров на всю программу сводится к минимуму: один для вычисления времени `Tmax`, другой – `Tmin`. Если необходимо, то в шаговом блоке выполняется запуск таймеров, а по завершении шагового блока таймеры сбрасываются и могут быть использованы в уже следующем блоке.

Рассмотрим реализацию шаговых блоков для контроллеров *Simatic S7*. Как и в первом случае, воспользуемся блоком FB. В разделе описания статической памяти блока объявим одну переменную `SW` и один таймер `Tmax`. Массив `PP` логических переменных, в котором хранятся результаты переходов, разместим в разделе временных переменных.

```
VAR             SW : INT;       Tmax : SFB 4;
VAR_TEMP        PP:  ARRAY  [ 0 .. 7 ]  OF BOOL ;
```

Используем инструкцию языка STEP7, имеющую сокращение JL (Jump to Labels - множественный переход) для выбора блоков `SBi` (рис.9). Список переходов объявляется командами `JU` (Jump Unconditional – безусловный переход), расположенными вслед за командой `JL` и завершается командой, символьный адрес которой указывается непосредственно в инструкции `JL`. Выбор перехода зависит от значения аккумулятора (от 0 до 255). При нуле в аккумуляторе выбирается первая команда `JU` из списка, при единице – вторая и т.д.

Network 1 : Переход к шаговому блоку

```
        L     #SW      // загрузка в аккумулятор
        JL    NOSB
        JU    SB0      // переход, если SW = 0
        JU    SB1      // переход, если SW = 1
        JU    SB2      // переход, если SW = 2
        JU    SB3      // переход, если SW = 3
NOSB:   JU    ERR      // переход, если SW >= 4
```

Рис.9. Фрагмент программы для вызова шагового блока

Рассмотрим пример для шагового блока `SB1` (рис.10), который соответствует первому состоянию графа.

Начало любого шагового блока отмечается той меткой, которая была указана в списке переходов. Секции (network) пять и шесть соответствуют действиям один и два шаблона (т
из
«с
пр

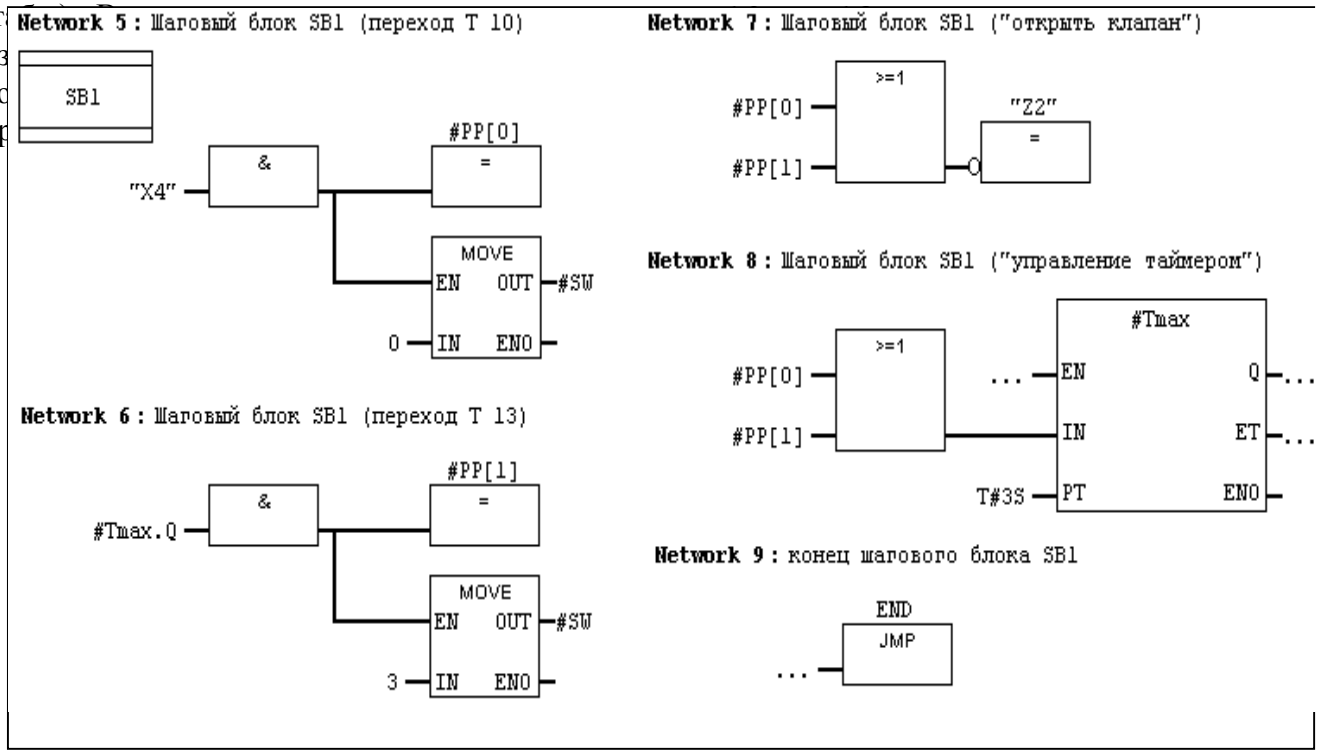


Рис.10. Фрагмент программы для шагового блока `SB1`

Для тестирования программы, реализующей граф переходов, пользователю достаточно наблюдать за одной единственной переменной SW. Изменяя вручную значение SW, можно контролировать выполнение программы с любого шага.

Если необходимо модифицировать управляющий граф, то пользователь может изменить только соответствующие шаговые блоки, а не всю программу. Это сводит к минимуму вероятность ошибок и сокращает время на разработку и обслуживание программы.

Типовая структура шаговых блоков позволяет легко восстановить логику работы программы и делает ее пригодной для длительного использования.

Метод шаговых блоков, как показывает опыт разработки АСУ ТП, например, для печей с шагающими балками (ОАО “Северсталь”), хорошо понимается инженерно-техническим персоналом заказчика.

Заключение

Рассмотренные формальные методы программирования логических контроллеров позволяют превратить искусство написания программ в простое и надежное ремесло, основанное, как и создание аппаратуры, на проектировании. Разработчик получает технологию для "**монтажа**" программы из готовых и понятных "**конструкций**". Ему достаточно только заполнить типовые программные структуры конкретными данными, взятыми из графа переходов.

Использование предлагаемых методов повышает качество программного продукта и продлевает его жизненный цикл. Простота предлагаемых решений снимает извечный вопрос о понимании программных текстов и делает их доступными даже специалисту "средней" квалификации.

Список литературы

1. ***Шалыто А.А.*** SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998. 628 с.
2. ***Шалыто А.А.*** Реализация алгоритмов логического управления программами на языке функциональных блоков //Промышленные АСУ и контроллеры. 2000. № 4, с.45 – 50.
3. ***SIMATIC*. Simatic S7/M7/C7**. Programmable controllers. SIEMENS. Catalog ST 70. 2002.

Альтерман И.З. – канд. техн. наук, руководитель группы АСУТП, ООО "КвадроТек", Москва, igoralterman@hotmail.ru

Шалыто А.А. – д-р техн. наук, профессор, заведующий кафедрой “Технологиии программирования” Санкт-Петербургского государственного университета информационных технологий, механики и оптики