Автоматное программирование

Области применения

В соответствии с классификацией, введенной Д. Харелом, любую программную систему можно отнести к одному из следующих классов.

- Трансформирующие системы осуществляют некоторое преобразование входных данных, и после этого завершают свою работу. В таких системах, как правило, входные данные полностью известны и доступны на момент запуска системы, а выходные – только после завершения ее работы. К трансформирующим системам относятся, например, архиваторы и компиляторы.

- Интерактивные системы взаимодействуют с окружающей средой в режиме диалога (например, текстовый редактор). Характерной особенностью таких систем является то, что они могут контролировать скорость взаимодействия с окружающей средой – заставлять среду «ждать».

- Реактивные системы взаимодействуют с окружающей средой путем обмена сообщениями в темпе, задаваемом средой. К этому классу можно отнести большинство телекоммуникационных систем, а также системы контроля и управления физическими устройствами.

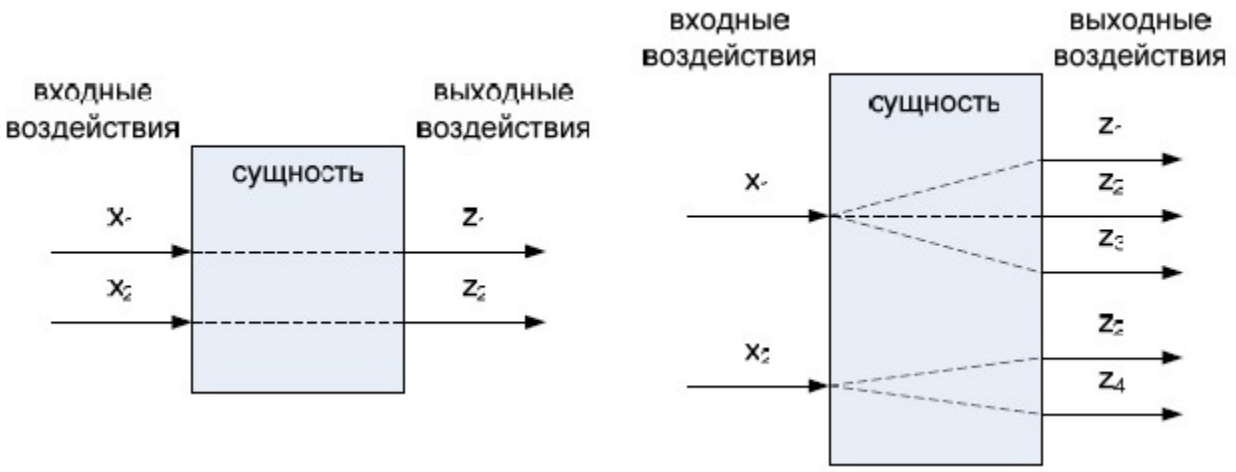
Конечные автоматы традиционно применяются при создании компиляторов, которые относятся к классу трансформирующих систем. Автомат здесь понимается как некое вычислительное устройство, имеющее входную и выходную ленту. Перед началом работы на входной ленте записана строка, которую автомат далее посимвольно считывает и обрабатывает. В результате обработки автомат последовательно записывает некоторые символы на выходную ленту.

Другая традиционная область использования автоматов – задачи логического управления – является подклассом реактивных систем. Здесь автомат – это, на первый взгляд, совсем другое устройство. У него несколько параллельных входов (чаще всего двоичных), на которые в режиме реального времени поступают сигналы от окружающей среды. Обрабатывая эти сигналы, автомат формирует значения нескольких параллельных выходов.

Таким образом, даже традиционные области применения конечных автоматов охватывают принципиально различные классы программных систем.

По мнению многих авторов, критерий применимости автоматного подхода лучше всего выражается через понятие сложное поведение. Сущность (объект, подсистема) обладает сложным поведением, если в качестве реакции на некоторое входное воздействие она может осуществить одно из нескольких выходных воздействий. При этом существенно, что выбор конкретного выходного воздействия может зависеть не только от входного воздействия, но и от текущего состояния сущности и от предыстории. Сложное поведение также называют поведением, зависящим от состояния (в англоязычной литературе используется термин state-dependent behavior). Соответственно, простое поведение можно назвать поведением, не зависящим от состояния.

Для сущностей с простым поведением реакция на любое входное воздействие зависит только от этого воздействия.



Сущность с простым поведением (слева) и со сложным поведением (справа)

Рассмотрим в качестве примера электронные часы. Пусть у них имеется только две кнопки, которые служат для установки текущего времени: кнопка «H» (Hours) увеличивает на единицу число часов , а кнопка «M» (Minutes) – число минут. Увеличение происходит по модулю 24 и 60 соответственно. Такие часы обладают простым поведением, поскольку каждое из двух входных воздействий (нажатие первой или второй кнопки) приводит к единственной, заранее определенной реакции часов.

Электронные часы

Рассмотрим теперь электронные часы с будильником. Дополнительная кнопка «A» (Alarm) служит в них для включения и выключения будильника. Если будильник выключен, то кнопка «A» включает его и переводит часы в режим, в котором кнопки «H» и «M» устанавливают не текущее время, а время срабатывания будильника. Повторное нажатие кнопки «A» возвращает часы в обычный режим. Наконец, нажатие кнопки «A» в обычном режиме при включенном будильнике приводит к выключению будильника.

Электронные часы с будильником

Поведение часов с будильником уже является сложным, поскольку одни и те входные воздействия (нажатия одних и тех же кнопок) в зависимости от режима инициируют различные действия.

В программных и программно-аппаратных вычислительных системах сущности со сложным поведением встречаются очень часто. Таким свойством обладают устройства управления, сетевые протоколы, диалоговые окна, персонажи компьютерных игр и многие другие объекты и системы.

Распознать сущность со сложным поведением в исходном коде программы очень просто: при традиционной реализации таких сущностей используются логические переменные, называемые флагами, и многочисленные запутанные конструкции ветвления, условиями в которых выступают различные комбинации значений флагов.

Такой способ описания логики сложного поведения плохо структурирован, труден для понимания и модификации, подвержен ошибкам.

Основная рекомендация по применению автоматного программирования очень проста: используйте автоматный подход при создании любой программной системы, в которой есть сущности со сложным поведением. Опыт показывает, что таким свойством обладает практически любая серьезная система. Однако обычно не все компоненты системы характеризуются сложным поведением. Поэтому данную выше рекомендацию можно дополнить еще одной: используйте автоматный подход при создании только тех компонент системы, которые являются сущностями со сложным поведением. Следование этим двум простым рекомендациям позволяет создавать корректные и расширяемые программные системы со сложным поведением.

Основные понятия

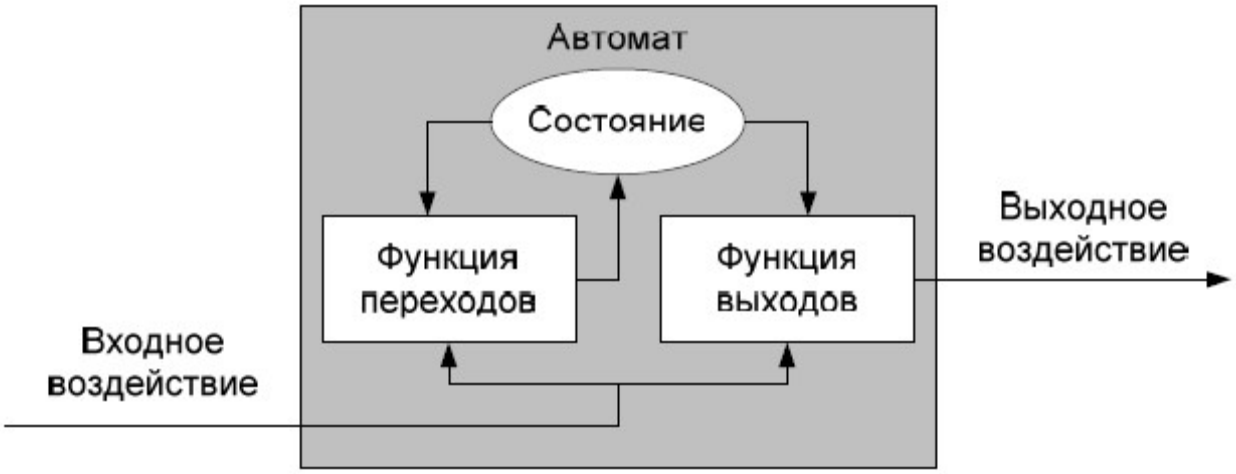
Базовым понятием автоматного программирования является «состояние». Это понятие с успехом применяется во многих развитых областях науки, например, в теории управления и теории формальных языков.

Основное свойство состояния системы в момент времени t_0 заключается в «отделении» будущего ($t > t_0$) от прошлого ($t < t_0$) в том смысле, что текущее состояние несет в себе всю информацию о прошлом системы, необходимую для определения ее реакции на любое входное воздействие, формируемое в момент t_0.

При описании понятия «сложное поведение» упоминалось, что реакция сущности со сложным поведением на входное воздействие может зависеть, в том числе, и от предыстории. При использовании понятия «состояние» знание предыстории более не требуется. Состояние можно рассматривать как особую величину, которая в неявной форме объединяет все входные воздействия прошлого, влияющие на реакцию сущности в настоящий момент времени. Реакция зависит теперь только от входного воздействия и текущего состояния.

Понятие входного воздействия также является одним из базовых для автоматного программирования. Чаще всего, входное воздействие – это вектор. Его компоненты подразделяются на события и входные переменные в зависимости от смысла и механизма формирования. Совокупность конечного множества состояний и конечного множества входных воздействий образует (конечный) автомат без выходов. Такой автомат реагирует на входные воздействия, определенным образом изменяя текущее состояние. Правила, по которым происходит смена состояний, называют функцией переходов автомата.

То, что в автоматном программировании собственно и называется (конечным) автоматом (см. рис.), получается, если соединить понятие автомата без выходов с понятием выходного воздействия. Такой автомат реагирует на входное воздействие не только сменой состояния, но и формированием определенных значений на выходах. Правила формирования выходных воздействий называют функцией выходов автомата.



Конечный автомат

Управляющие и вычислительные состояния

Для того, чтобы лучше разобраться в основных концепциях автоматного программирования, рассмотрим программу машины Тьюринга.

Пусть, например, необходимо реализовать функцию инкремент (увеличение целого числа на единицу). Пусть исходное число записано на ленте в двоичном виде слева направо, во всех остальных ячейках находится пустой символ b ('blank') и головка указывает на самый старший разряд числа. Тогда для увеличения числа на единицу можно предложить следующий алгоритм:

1. Двигаться вправо, пока не встретиться пустой символ.
2. Сдвинуться на одну ячейку влево.
3. Пока в текущей ячейке находится '1', заменять его на '0' и двигаться влево.
4. Если в текущей ячейке находится '0' или 'b', записать в ячейку '1' и завершить работу.

b	1	0	1	1	b
	↑				

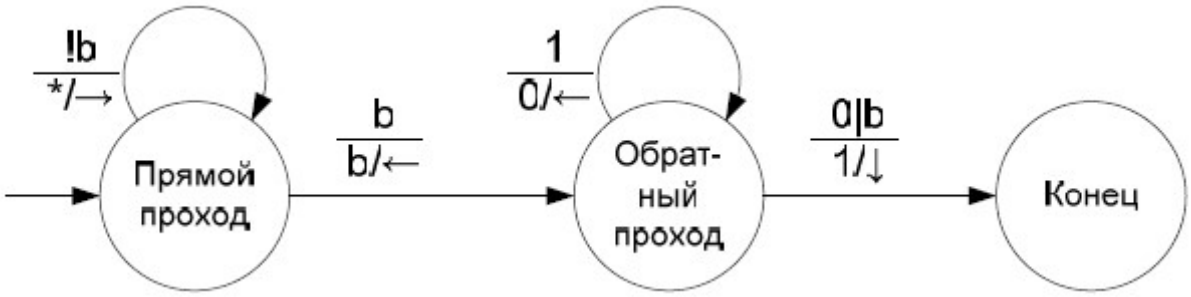
b	1	0	0	0	b
		↑			

b	1	0	1	1	b
				↑	

b	1	1	0	0	b
		↑			

b	1	0	1	0	b
			↑		

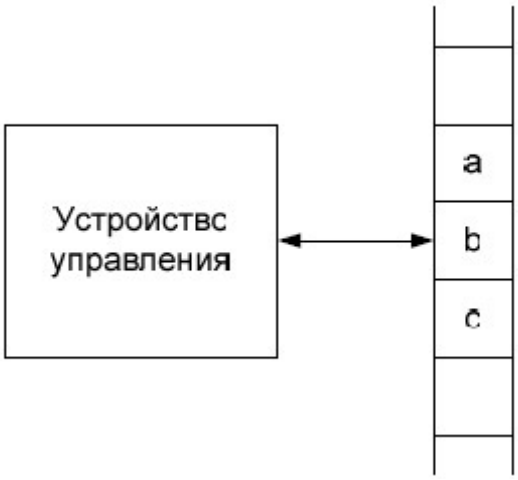
На рисунке ниже представлен граф переходов управляющего автомата машины Тьюринга, реализующей функцию инкремент. Здесь символ ‘*’ на месте записываемого символа означает «записать тот же самый символ, который был считан». Команды головке обозначаются стрелками (стрелка вниз означает «остаться на месте»).



Увеличение числа на единицу с помощью машины Тьюринга

Отметим, что в графе переходов обозначены имена состояний управляющего автомата. Эти имена отражают смысл состояния и являются кратким описанием поведения машины в этом состоянии.

Итак, управляющий автомат машины Тьюринга, реализующей функцию инкремент, имеет три состояния. Сколько же состояний у этой машины в целом? Ее действия в каждый момент времени полностью определяются совокупностью состояния управляющего автомата, строки на ленте и положения головки. Отметим, что символы на ленте для устройства управления представляют собой входные воздействия, однако, относительно машины в целом они не являются входными (внешними), а формируют часть внутреннего состояния вычислителя. Всевозможных строк на ленте, как и положений головки, бесконечно много, поэтому и у машины Тьюринга бесконечное число состояний.



Машина Тьюринга

Однако, если задуматься, состояния управляющего устройства и состояния ленты имеют принципиально различное значения. В приведенном примере оказалось, что для того, чтобы задать алгоритм для машины Тьюринга, достаточно описать ее поведение в каждом из трех состояний управляющего автомата. Нам не потребовалось задавать действия машины для каждой из бесконечного числа возможных входных строк.

Можно сказать, что состояния управляющего автомата определяют действия машины, а состояние ленты – лишь результат этих действий.

Теперь очевидно, что состояния устройства управления и состояния ленты – совершенно разные понятия с точки зрения программирования на машине Тьюринга, и смешивать их не стоит. Первые следует явно перечислять, отображать на графе переходов, описывать алгоритм поведения в каждом из них. Вторые в программе в явном виде не участвуют, построить граф переходов между ними невозможно, а если бы это и удалось, то для понимания программы такой граф был бы бесполезен. Первые можно назвать качественными состояниями машины, а вторые – количественными. Состояния автомата называются управляющими, а состояния ленты – вычислительными.

Понятия управляющих и вычислительных состояний применимы не только к машине Тьюринга, но и к любой сущности со сложным поведением. Однако, если в машине Тьюринга отличить управляющие состояния от вычислительных не составляет труда (поскольку она по определению состоит из устройства управления и ленты), то для произвольной сущности явное выделение управляющих состояний – сложная задача. Причина сложности состоит в том, что различия между управляющими и вычислительными состояниями в общем случае трудно формализовать. Неформально основные различия сформулированы в таблице.

Управляющие и вычислительные состояния

Управляющие состояния	Вычислительные состояния
Их несколько	Их количество либо бесконечно, либо конечно, но очень велико
Каждое из них имеет вполне определенный смысл и качественно отличается от других	Большинство из них не имеет смысла и отличается от остальных лишь количественно
Они определяют действия, которые совершает сущность	Они непосредственно определяют лишь результаты действий

Опыт рассмотрения машины Тьюринга подсказывает, что для того, чтобы сделать программу простой и понятной, необходимо явно выделить управляющие состояния (идентифицировать их, дать им имена) и описать поведение сущности в каждом из них. Например, при реализации электронных часов с будильником можно выделить три управляющих состояния: «Будильник выключен», «Установка времени будильника» и «Будильник включен». В каждом из этих состояний реакция будильника на нажатие любой кнопки будет однозначной и специфической.

Спецификация структуры

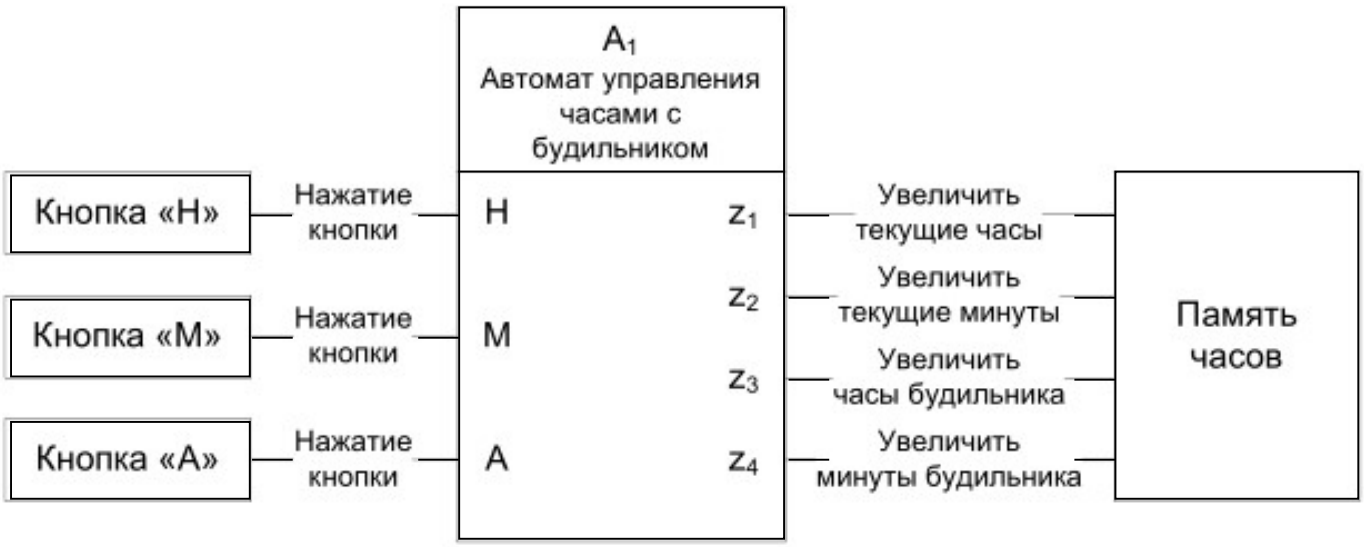
Как уже упоминалось и было продемонстрировано на нескольких примерах, одним из главных достоинств автоматного программирования является представление логики поведения системы в наиболее понятном и наглядном виде – с использованием графической нотации графов (или диаграмм) переходов. Однако не только логика поведения требует наглядного преставления. Спецификация структуры системы является, наряду со спецификацией поведения, одним из двух основных результатов процесса проектирования и также требует подходящей нотации.

В примерах, приводимых до сих пор, структура системы (объекты управления, автоматы и связи между ними) описывалась словесно. Как могли убедиться читатели, такое описание громоздко и не обладает свойством наглядности. В программировании

с явным выделением состояний для описания структуры системы используется графическая нотация схем связей автоматов.

Схема связей строится отдельно для каждого автомата системы. На ней в виде прямоугольника изображается сам автомат (рис. 2.20). Слева от него изображаются источники информации (в событийных системах их также называют поставщиками событий) – сущности из системы или внешней среды, которые формируют входные воздействия автомата. Для каждого события, входной переменной или предиката с номером состояния между автоматом и источником информации проводится линия, помеченная идентификатором и словесным описанием этого события (переменной, предиката). Таким образом, схема связей служит, в частности, для расшифровки сокращенных идентификаторов событий и переменных, используемых на диаграмме переходов.

Справа от автомата изображаются его объекты управления и подчиненные (вложенные или вызываемые) автоматы. Для каждой выходной переменной между автоматом и соответствующим объектом управления проводится линия, помеченная идентификатором и описанием переменной. Если вызываемому автомату передается одно или несколько событий, то для каждого из этих событий также проводится линия с пометками. Если объекты управления являются также и источниками информации – формируют часть входных переменных автомата – то они изображаются справа, а линии, соответствующие входным переменным изображаются в виде обратных связей.



Проектирование программ

1. По словесному описанию поведения управляемой системы строится спецификация структуры. По логике появления выходных воздействий определяется тип системы: с простым поведением или со сложным поведением. Если в системе нет элементов со сложным поведением, перейти к п. 3, иначе – к п. 2.

2. По спецификации структуры строится набор управляющих состояний. Строится управляющий автомат программной системы: управляющие состояния связываются между собой переходами, добавляются входные и выходные переменные, необходимые для реализации заданного в словесном описании поведения. Если система является событийной, определяется набор событий, обрабатываемых автоматом, они включаются в условия переходов наряду с входными переменными. Определяется такт работы автомата (с какой частотой читать входные воздействия и менять состояния автомата).

3. Входным переменным автомата сопоставляются запросы: булевы функции или (реже) двоичные переменные. Выходным переменным сопоставляются команды: процедуры. Если упомянутые подпрограммы являются недостаточно простыми для непосредственной реализации, для каждой из них производится цикл традиционного проектирования сверху вниз.

4. Вводятся переменные, необходимые для корректной работы упомянутых запросов и команд. Совокупность значений этих переменных составляет вычислительное состояние системы.

5. По построенному в п. 2 управляющему автомату и введенным в пп. 3, 4 переменным разрабатывается код программы.

6. В процедуре (функции), эмулирующей управляющий автомат, в обработчике каждого состояния проставляются точки останова, программа собирается и отлаживается.

Для примера рассмотрим, как применить этот метод для проектирования программного эмулятора часов с будильником, рассмотренных выше.

1. Составляется спецификация структуры (см. выше). Поведение программной системы является сложным, т.к. при одном входном воздействии – нажатие кнопки H – могут появиться разные выходные воздействия Z_1 или Z_3. Аналогичная ситуация с кнопкой M. Следует перейти к п. 2 и спроектировать управляющий автомат.

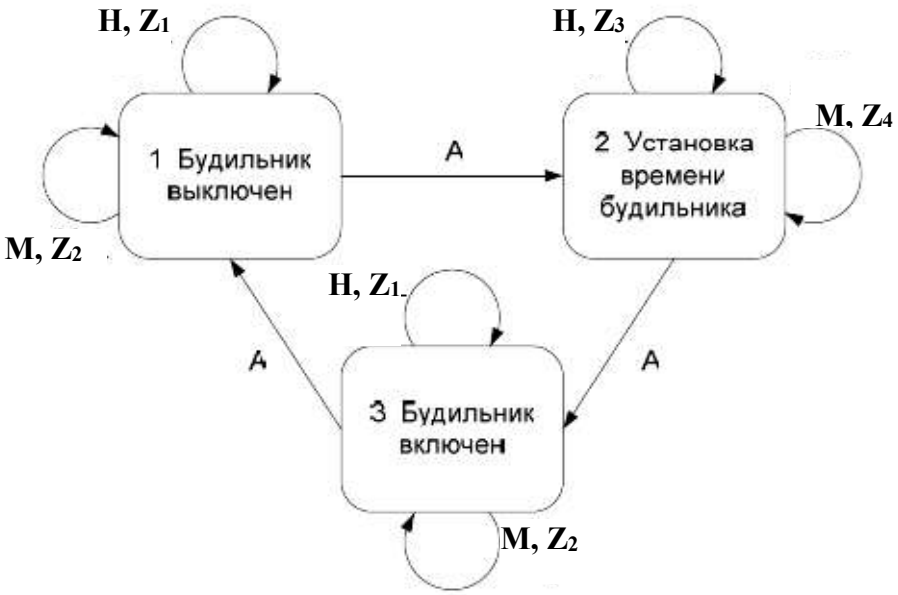
2. Из словесного описания поведения часов можно заключить, что они ведут себя по-разному в зависимости от того, включен или выключен будильник. Кроме того, у часов есть режим установки времени будильника. Следовательно, в данной системе целесообразно выделить три управляющих состояния: «1. Будильник выключен», «2. Установка времени будильника», «3. Будильник включен» (рис. 2.3).

Управляющие состояния эмулятора часов с будильником

В процессе выделения управляющих состояний приходится внимательно исследовать описание сущности со сложным поведением в поисках набора «ситуаций», в которых поведение сущности имеет качественные особенности.

Однако, некоторые формулировки в описании сущности могут упростить задачу поиска состояний. Например, понятие режим является синонимом понятия управляющее состояние (по крайней мере, в контексте программных и программно-аппаратных систем). Если в описании поведения системы упоминаются несколько режимов, то каждому из них наверняка будет соответствовать отдельное состояние.

В данном случае целесообразно спроектировать событийную систему. Автомат будет обрабатывать три события, соответствующие нажатию трех различных кнопок. На графе переходов (рис. 2.4) для обозначения событий используются названия кнопок. Переходы между состояниями осуществляются по нажатию кнопки «А». При нажатии других кнопок («Н» и «М») переходов между состояниями не происходит, но выполняются соответствующие действия (изменение текущего времени или времени срабатывания будильника). Эти действия обозначим выходными переменными z1–z4.



Управляющий автомат эмулятора часов с будильником

В качестве альтернативы событиям, можно было бы ввести три входные переменные, принимающие значения истина или ложь в зависимости от того, нажата ли соответствующая кнопка. Такое решение усложнило бы автомат, поэтому и был выбран событийный подход. В событийном варианте входные переменные отсутствуют.

3. Выходным переменным автомата, введенным на предыдущем шаге, сопоставим команды: z1 – «увеличить число часов текущего времени», z2 – «увеличить число минут текущего времени», z3 – «увеличить число часов времени срабатывания будильника», z4 – «увеличить число минут времени срабатывания будильника».

4. Наконец, введем четыре целочисленные переменные: «число часов», «число минут», «число часов будильника», «число минут будильника». Таким образом, совокупность значений текущего времени и времени срабатывания будильника определяет текущее вычислительное состояние системы.

5. После выполнения указанных выше шагов начинается стадия реализации эмулятора. Подпрограммы самого низкого уровня абстракции реализуются в терминах переменных, хранящих вычислительное состояние. Реализация автомата становится главной программой.

В приведенном примере логика поведения эмулятора часов с будильником описана с помощью автомата Мили. Выбор этой автоматной модели сделан на основе постановки задачи: из словесного описания поведения часов следует, что они совершают какие-либо действия только при нажатии соответствующих кнопок. В общем случае в программировании с явным выделением состояний могут применяться автоматы Мура, Мили или смешанные автоматы.

Множество управляющих состояний – более устойчивая характеристика системы, чем ее главная функция. Поэтому архитектура, построенная вокруг управляющих состояний, является более расширяемой.

Программная реализация часов с будильником на языке С

```
const int h = 1;
const int m = 2;
const int a = 3;
int e; // Текущее событие
int y; // Текущее управляющее состояние
// Реализация объекта управления
int hrs;
int mins;
int alarmHrs;
int alarmMins;

void z1() {
  hrs = (hrs + 1) % 24;
}

void z2() {
  mins = (mins + 1) % 60;
}

void z3() {
  alarmHrs = (alarmHrs + 1) % 24;
}

void z4() {
  alarmMins = (alarmMins + 1) % 60;
}

void main () {
...
// Реализация управляющего автомата
        while(1)
                e = ScanKbd();
                switch (y) {
                    case 1: // Будильник выключен
                                if (e == h) { z1(); }
                                else if (e == m) { z2(); }
                                else if (e == a) { y = 2; }
                                break;
                    case 2: // Установка времени будильника
                                if (e == h) { z3(); }
                                else if (e == m) { z4(); }
                                else if (e == a) { y = 3; }
                                break;
                    case 3: // Будильник включен
                                if (e == h) { z1(); }
                                else if (e == m) { z2(); }
                                else if (e == a) { y = 1; }
                                break;
                }
                DelayMs(100);
        }
}
```

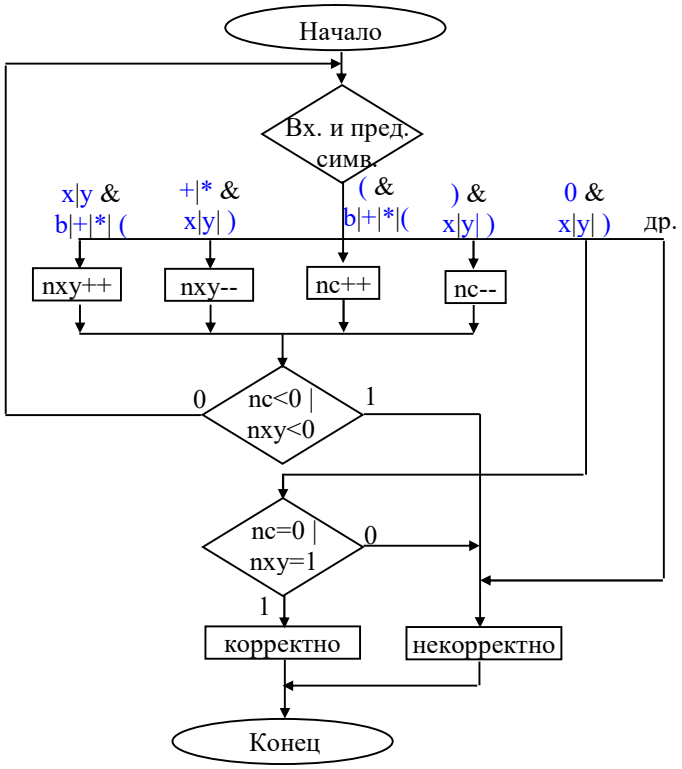
Примеры проектирования трансформирующих систем

Пример 1.

Проверка корректности арифметических выражений выполняется во многих трансформирующих системах – компиляторы, табличные процессоры, СУБД и др.

Разработаем программу проверки корректности арифметических выражений в алфавите {x, y, +, *, (,) , ┤ } императивным и автоматным подходами.

Императивное программирование	Автоматное программирование
Словесное описание С помощью переменной nxy будем проверять соответствие количества символов x, y и знаков + или *. С помощью переменной nc будем проверять соответствие количества скобок. Для проверки правильности сочетания символов в выражении будем запоминать предыдущий символ. В символьном массиве записано выражение. Перед началом выражения запишем пробел, а в конце – нулевой символ. В начальном состоянии nc=0, nxy=0. Словесный алгоритм 1) анализ входного символа (начинаем со 2-го): - если x или y, если предыдущий символ – пробел или + или * или (, то nxy++ и переходим к п. 2, иначе выражение некорректно, переходим в конец; - если + или *, если предыдущий символ – x или y или) , то nxy-- и переходим к п. 2, иначе выражение некорректно, переходим в конец; - если (, если предыдущий символ – пробел или + или * или (, то nc++ и переходим к п. 2, иначе выражение некорректно, переходим в конец; - если) , если предыдущий символ – x или y или) , то nc-- и переходим к п. 2, иначе выражение некорректно, переходим в конец; - если 0, если предыдущий символ – x или y или) , то переходим к п. 4, иначе выражение некорректно, переходим в конец; 2) если nc<0 или nxy<0, выражение некорректно, переходим в конец; 3) переходим к п. 1; 4) если nc=0 и nxy=1, выражение корректно, иначе – некорректно.	Смоделируем автомат с магазинной (стековой) памятью, распознающий арифметические выражения. 1) Множество входных символов: {**x**,**y**,+,*,(,), ┤ }. 2) Множество магазинных символов: { **A, B,** ∇} 3) Множество состояний: **t**, является также и начальным состоянием автомата. 4) Алгоритм работы автомата. - Цепочка **принимается**, если при ее окончании всем левым скобкам нашлись правые, для всех арифметическим знаков нашлись соответствующие «**x**» или «**y**». - Цепочка **отвергается**, если: 1. Количество правых скобок превысило количество левых. 2. Количество «**x**» или «**y**» превысило количество арифметических знаков более чем на единицу. 3. Входная цепочка прочитана до конца, а левым скобкам не нашлось пары. 4. Входная цепочка прочитана до конца, а некоторым арифметическим знакам не нашлось соответствующих «**x**» или «**y**». **Алгоритм** работы: - Если входная головка читает «**(**», то в магазин заталкивается **A**. - Если входная головка читает «**)**», то из магазина выталкивается **A**. - Если входная головка читает «+» или «*», то в магазин заталкивается **B**. - Если входная головка читает «**x**» или «**y**», то из магазина выталкивается **B**. - Цепочка **принимается**, если при достижении символа ┤ магазин пуст ∇. - Цепочка **отвергается**, если: 1. На входе остаются правые скобки «**)**» или «**x**» или «**y**», а магазин пуст ∇. 2. При достижении символа ┤ в магазине остаются символы **A** или **B**.



Код программы

```
void main (void) {
char   i;                       // счетчик вх. симв.
char M[11] = " x+y*(x+y)"; // входной массив
signed char nxy=0;              // для проверки x,y,+,*
signed char nc=0;               // для проверки скобок
  for(i=1;i!=11;i++) { // обр-ка вх. симв.
    switch (M[i]) {
      case 'x':
      case 'y': if((M[i-1]==32)||(M[i-1]=='+')||
                   (M[i-1]=='*')||(M[i-1]=='('))
                   {nxy++; break;}
                   else goto 10;
      case '+':
      case '*': if((M[i-1]=='x')||(M[i-1]=='y')||
                   (M[i-1]==')')) {nxy--; break;}
                   else goto 10;
      case '(': if((M[i-1]==32)||(M[i-1]=='+')||
                   M[i-1]=='*')||(M[i-1]=='('){nc++;break;}
                   else goto 10;
      case ')': if((M[i-1]=='x')||(M[i-1]=='y')||
                  (M[i-1]==')')) {nc--; break;}
                   else goto 10;
      case 0:   if((M[i-1]=='x')||(M[i-1]=='y')||
                   (M[i-1]==')')) goto 20;
                   else goto 10;
      default:  goto 10;
    }
    if((nc<0)||(nxy<0)) goto 10;
  }
10: printf("Выражение некорректно"); goto 30;
20: if((nc==0)&&(nxy==1))
    printf("Выражение корректно");
    else printf("Выражение некорректно");
30:
}
```

Стек. симв.	Входные символы				
	+, *	x,y	(	)	⊣
A	↓B	Отв.	↓A	↑	Отв.
B	↓B	↑	↑↓A↓B	Отв.	Отв.
∇	↓B	Отв.	↓A	Отв.	Доп.

5. В начальном состоянии стек содержит **B∇**. В символьном массиве записано выражение. В конце запишем нулевой символ.

Код программы

```
void main (void) {
char   i;                       // счетчик вх. симв.
char M[10] = "x+y*(x+y)";  // входной массив
char S[10];                     // стек глубиной 10

  S[8]='B';    // инициал-я стека
  S[9]=0;
  for(i=0;i!=10;i++) { // обр-ка вх. симв.
    switch (M[i]) {
      case '+':
      case '*': Stack_In('B'); break;
      case 'x':
      case 'y': if(S[0]=='B') {Stack_Out(); break;}
                   else goto 10;
      case '(': if(S[0]=='A')||(S[0]==0))
                     Stack_In('A');
                   else if(S[0]=='B') {Stack_Out();
                          Stack_In('A'); Stack_In('B');}
                   break;
      case ')': if(S[0]=='A') {Stack_Out(); break;}
                   else goto 10;
      case 0:   if(S[0]==0) goto 20;
                   else goto 10;
      default:  goto 10;
    }
  }
10: printf("Выражение некорректно"); goto 30;
20: printf("Выражение корректно");
30:
}
```

Пример 2.

Пусть, к примеру, требуется написать на языке Си программу, читающую из потока стандартного ввода текст, состоящий из строк, и для каждой строки печатающую первое слово этой строки и перевод строки. Ясно, что для этого во время чтения каждой строки следует:

1) пропустить пробелы, если таковые есть в начале строки;

2) читать буквы, составляющие слово, и печатать их, пока слово не кончится (то есть либо не кончится строка, либо не будет встречен пробельный символ);

3) когда первое слово успешно считано и напечатано, необходимо дочитать строку до конца, ничего при этом не печатая.

4) если встретился символ перевода строки, следует напечатать перевод строки и продолжить с начала.

При возникновении (на любой фазе) ситуации «конец файла» следует прекратить работу.

Императивная программа

Программа, решающая эту задачу в традиционном императивном стиле (программа – последовательность команд), может выглядеть, например, так:

```
#include <stdio.h>

int main() {
  char c;

  do {
    c = getchar();
    while (c == ' ') c = getchar();
    while (c != ' ' && c != '\n' && c != EOF) putchar(c), c = getchar();
    putchar('\n');
    while (c != '\n' && c != EOF) c = getchar();
  } while (c != EOF);

  return 0;
}
```

Программа в автоматном стиле

Ту же задачу можно решить, применив мышление в терминах конечных автоматов. Заметим, что разбор строки разделяется на три фазы:

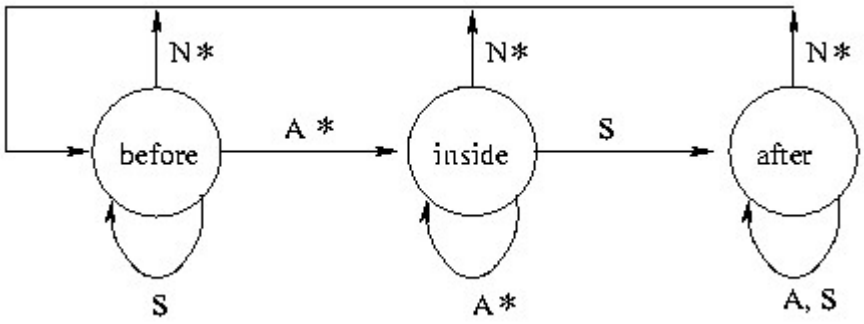
1) пропуск лидирующих пробелов;

2) печать слова;

3) пропуск символов остатка строки.

Назовём эти три фазы состояниями before, inside и after.

Программа будет реализовывать (моделировать) работу конечного автомата, изображённого на рисунке. Буквой N на диаграмме обозначен символ конца строки, буквой S — символ пробела, буквой A — все остальные символы. За один шаг автомат делает ровно один переход в зависимости от текущего состояния и прочитанного символа. Некоторые переходы сопровождаются печатью прочитанного символа; такие переходы на диаграмме обозначены звёздочками.



Программа теперь может выглядеть, например, так:

```
#include <stdio.h>

int main() {
  enum states { before, inside, after } state;
  int c;

  state = before;
  while ((c = getchar()) != EOF) {
    switch (state) {

      case before:
        if (c == '\n') putchar('\n');
        else if (c != ' ') putchar(c), state = inside;
        break;

      case inside:
        switch (c) {
          case ' ':
            state = after; break;
          case '\n':
            putchar('\n'), state = before;
            break;
          default: putchar(c);
        }
        break;

      case after:
        if (c == '\n') putchar('\n'), state = before;
```

```
    }
  }

  return 0;
}
```

Несмотря на то, что код явно стал более длинным, у него имеется одно несомненное достоинство: чтение (то есть вызов функции getchar()) теперь выполняется ровно в одном месте. Кроме того, вместо четырёх циклов, использовавшихся в предыдущей версии, цикл теперь используется только один. Тело цикла (за исключением действий, выполняемых в заголовке) представляет собой шаг автомата, сам же цикл задаёт цикл работы автомата.

Часто по условию задачи некоторые действия сущности не зависят от ее состояния, входной предыстории. Эти действия можно выполнить перед обработчиком состояний моделирующего автомата (блоком switch-case) или после него.

В рассматриваемом примере можно сэкономить объём кода, если заметить, что действия, выполняемые по символу «конец строки», от состояния не зависят. Программа, эквивалентная предыдущей, но написанная с учётом такого замечания, будет выглядеть так:

```
#include <stdio.h>

int main() {
  enum states { before, inside, after } state;
  int c;

  state = before;
  while ((c = getchar()) != EOF) {
    if (c == '\n') putchar('\n'), state = before, continue;
    switch (state) {
      case before:
        if (c != ' ') putchar(c), state = inside;
      break;
      case inside:
        if (c == ' ') state = after;
        else putchar(c);
      case after:
        break;
    }
  }

  return 0;
}
```

Разработать классическим (императивным) и автоматным подходами программу, читающую из потока стандартного ввода текст, состоящий из строк, и для каждой строки печатающую 2 первых слова этой строки и перевод строки.

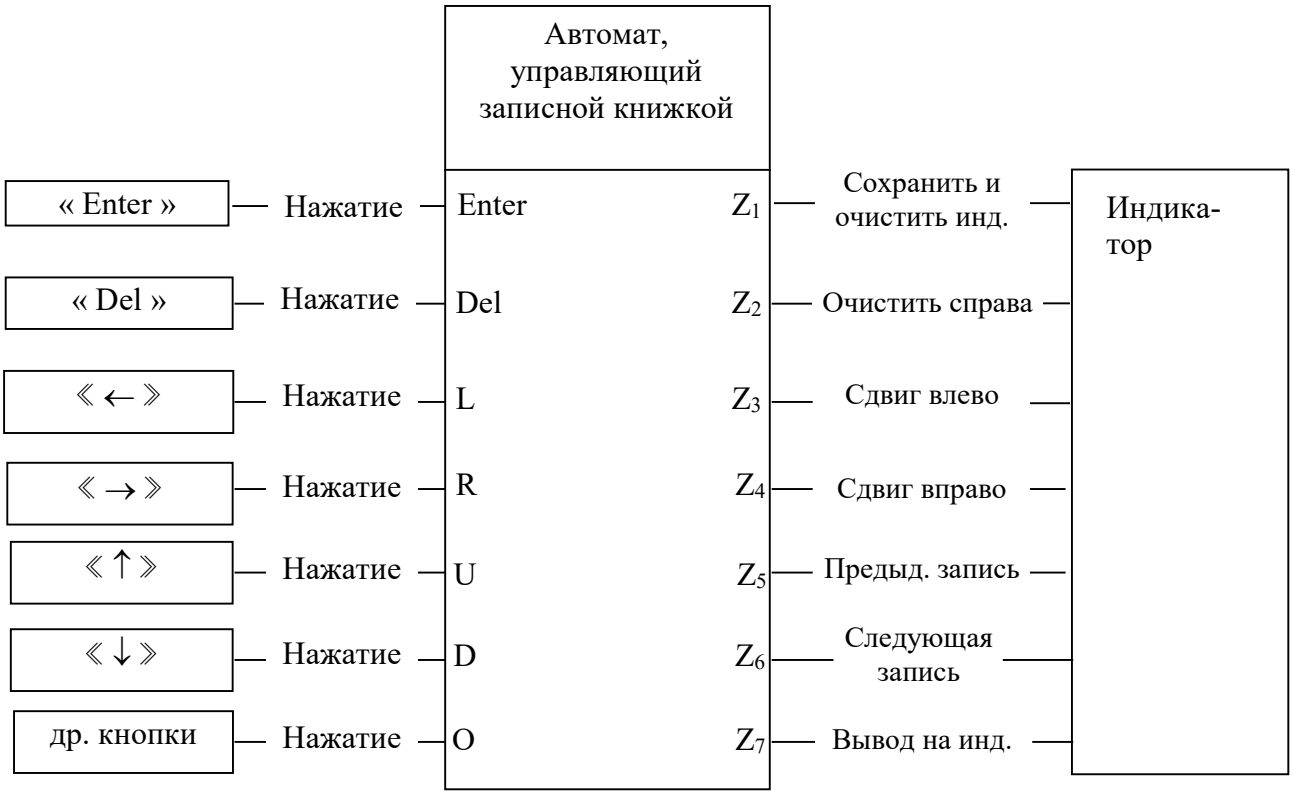
Примеры проектирования интерактивных систем

Задача 1. *Электронная записная книжка*

При включении питания на однострочном индикаторе отображается курсор. Записная книжка готова к вводу новых строк текста. При нажатии кнопки «Enter» происходит запись строки в память и очистка индикатора. Курсор ставится в начальную позицию. При нажатии кнопки «Del» происходит удаление символов справа от курсора. При нажатии кнопок «←» и «→» курсор перемещается в соответствующую сторону. При нажатии кнопок «↑» и «↓» происходит перелистывание записей книжки. При нажатии остальных кнопок происходит ввод соответствующих символов.

1. Построим структурную схему программной системы.

Структурная схема системы управления электронной записной книжкой



Управляющий автомат электронной записной книжки представляет собой типичный функциональный преобразователь, в котором каждому входному сигналу сопоставлен определенный выходной сигнал. Данная задача не требует автоматного программирования, так как в управляющей структуре отсутствуют сущности со сложным поведением. По структуре можно сразу разработать код программы.

Код программы

```
void main (void) {

char  key;       // идентификатор кнопки
char  i;         // счетчик вх. симв.
char  j;         // счетчик строк массива
char S[17][3]; // строковый буфер

  while(1) {     // обр-ка вх. симв.
     key = ScanKbd();
     switch (key) {
       case 13: Save_Buf(); Lcd_Clear(); break;
       case 46: Lcd_Erase(i); break;
       case 36: Lcd_Shift_Left(); break;
       case 38: Lcd_Shift_Right(); break;
       case 37: if(j>0) j--; Lcd_Put_Str(j); break;
       case 40: if(j<2) j++; Lcd_Put_Str(j); break;
       default:  Put_Buf(key); Lcd_Put_Str(j); break;
     }
     DelayMs(100);
  }
}
```

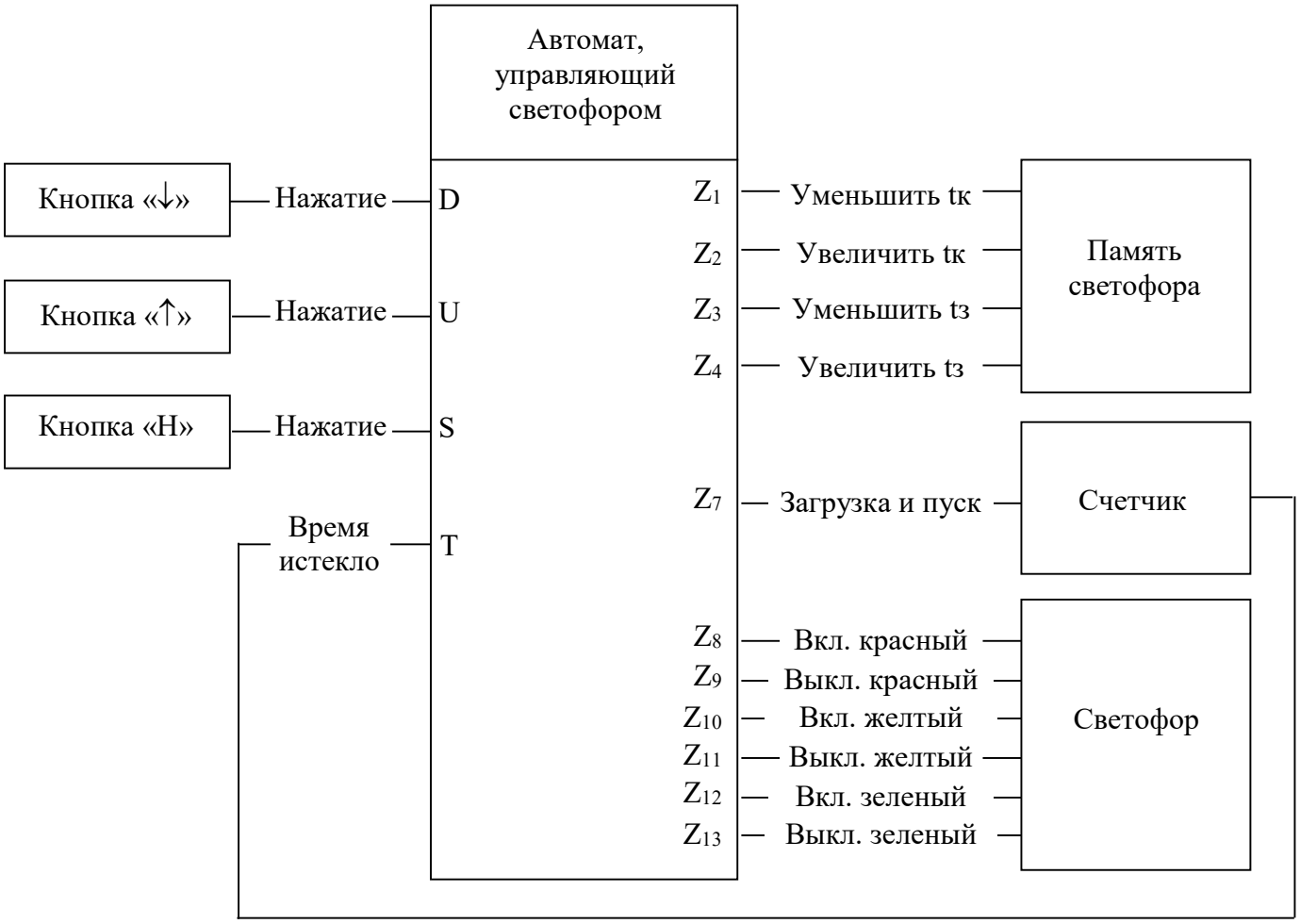
Задача 2. *Программа управления светофором*.

В начальном состоянии горит красный сигнал. На индикаторе отображается слово «красный». Через $\mathbf{t_к}$ включается желтый сигнал. На индикаторе отображается слово «желтый». Через 2 секунды красный и желтый сигналы гаснут, включается зеленый сигнал на время $\mathbf{t_з}$. На индикаторе отображается слово «зеленый». Затем 3 секунды зеленый сигнал мигает с частотой 1 Гц и скважностью 50%. По окончании мигания зеленый сигнал выключается, на 2 секунды включается желтый сигнал, на индикаторе - слово «желтый», затем желтый сигнал выключается и включается красный сигнал, на индикаторе - слово «красный». Далее работа светофора повторяется.

При нажатии на кнопку «**Н**» в режиме горения красного сигнала светофор входит в режим настройки и включается мигающий желтый цвет. На индикаторе отображается «tк=ХХс». После следующего нажатия на кнопку «**Н**» значение tк сохраняется и на индикаторе отображается «tз=ХХс». После следующего нажатия на кнопку «**Н**» значение tз сохраняется и светофор переходит из режима настройки в состояние горения красного цвета. Значения tк и tз изменяются с помощью кнопок «◄» и «►» в интервале от 5с до 30с с шагом 1с.

1. Построим структурную схему программы управления светофором и определим, имеет ли смысл автоматный подход программирования.

Структурная схема системы управления светофором



Согласно условию задачи команды управления светофором зависят от того, какой свет горит в данный момент, то есть от текущего состояния светофора. **Входной сигнал** «время горения истекло» ведет к **разным выходным сигналам** в зависимости от состояния светофора. **Нажатие кнопки** «↓» и «↑» также ведет **к разной реакции программы**: меняется время горения красного или зеленого сигнала, либо ничего не меняется.

Анализ условия задачи и структурной схемы показал, что в данной задаче имеет смысл **автоматный подход программирования**.

2. По условию задачи выбираем внутренние состояния автомата:
1) горит красный свет
2) горят красный и желтый
3) горит зеленый
4) мигает зеленый
5) горит желтый
6) настройка времени горения красного цвета
7) настройка времени горения зеленого цвета

3. На базе условия задачи, структурной схемы и выбранных внутренних состояний строим таблицу переходов и выходов (матричная запись).

state	Красный	Красный-желтый	Зеленый	Миг. зеленый	Желтый	Настр. tк	Настр. tз
Красный	«Красный»	(T){Z_7(20), Z_{10}}	-	-	-	(key=S) Z_9	-
Красный-желтый	-	«Кр.-желтый»	(T) {Z_7($10t_з$), Z_9, Z_{11}, Z_{12}}	-	-	-	-
Зеленый	-	-	«Зеленый»	(T) Z_7(3)	-	-	-
Миг. зеленый	-	-	-	Z_{12}, П(0,5) Z_{13}, П(0,4)	(T){Z_7(20), Z_{10}}	-	-
Желтый	(T) {Z_7($10t_к$), Z_{11}, Z_8}	-	-	-	«Желтый»	-	-
Настр. tк	-	-	-	-	-	Z_{10}, П(0,5) Z_{11}, П(0,5) «tк=XXc» (key=D) Z_1 (key=U) Z_2	(key=S)
Настр. tз	(key=S) {Z_7($10t_к$), Z_8}	-	-	-	-	-	Z_{10}, П(0,5) Z_{11}, П(0,5) «tз=XXc» (key=D) Z_3 (key=U) Z_4

4. Код программы

```
const char D = 1; // Код нажатой кнопки
const char U = 2;
const char S = 3;
enum states {R,RY,G,GG,Y,SR,SG} state; // Состояния
char key;  // Идентификатор нажатой кнопки
char state; // Идентификатор управляющего состояния
char t = 5;  // Значение счетчика
char tr = 5;  // Время красного сигнала, с
char tg = 5; // Время зеленого сигнала, с
bit T = 0;    // Сигнал счетчика «время истекло»
bit RED = 1;      // Красный цвет
bit YEL = 0;      // Желтый цвет
bit GRN = 0;      // Зеленый цвет

void main (void) {
        t = 10*tr;
        state = R;
// Реализация управляющего автомата
        while (1) {
            key = ScanKbd();
            switch (state) {
                case R: // Красный
                        printf("Красный");
                        if (T) { state=RY;  T = 0;  t=20;  YEL=1; }
                        else if (key == S) { state=SR;  RED=0; }
                        break;
                case RY: // Красный-желтый
                        printf("Красный-желтый");
                        if (T) { state=G;  T = 0;  t=10*tg;  RED=0; YEL=0; GRN=1; }
```

```
                    break;
                case G: // Зеленый
                    printf("Зеленый");
                    if (T) { state=GG;  T = 0;  t=3; }
                    break;
                case GG: // Мигающий зеленый
                    GRN=1; DelayMs(500);
                    GRN=0; DelayMs(400);
                    if (T) { state=Y;  T = 0;  t=20;  YEL=1; }
                    break;
                case Y: // Желтый
                    printf("Желтый");
                    if (T) { state=R;  T = 0;  t=10*tr; YEL=0; RED=1; }
                    break;
                case SR: // Настройка времени красного сигнала
                    if (T) { T=0; t=5; YEL=1-YEL;} // Мигание желтого
                    printf("tк=%2dc", tr);
                    switch(key) {
                        case D: if (tr>5) tr--; break;
                        case U: if (tr<30) tr++; break;
                        case S: state=SG; break;
                    }
                    break;
                case SG: // Настройка времени зеленого сигнала
                    if (T) { T=0; t=5; YEL=1-YEL;} // Мигание желтого
                    printf("tз=%2dc", tg);
                    switch(key) {
                        case D: if (tg>5) tg--; break;
                        case U: if (tg<30) tg++; break;
                        case S: state=R;  T = 0;  t=10*tr; YEL=0; RED=1; break;
                    }
                    break;
            }
            DelayMs(100);          // такт работы автомата
            if(t==0) T=1; else t--;  // счетчик
        }
}
```

Пользователь вводит с помощью двух кнопок десятичное число от 0 до 9999, которое отображается на 4-разрядном цифровом индикаторе.

Первая кнопка включает и выключает режим ввода числа (при этом включается и выключается выделение изменяемого разряда), переключает между разрядами и выделяет только изменяемый разряд, вторая - изменяет значение разряда числа от 0 до 9 (по кольцу). Вторая кнопка действуют только в режиме ввода числа.

Ввод кода кнопки осуществляется функцией сканирования клавиатуры ScanKbd, вывод символьного массива на индикатор - функцией Put_LCD, включение и отключение подсветки разряда - функциями Sel_ON(n) и Sel_OFF(n), где n - номер разряда; сохранение введенного числа в переменную целого типа при выключении режима ввода – Save(). Сканирование кнопок и вывод на индикатор выполняются через 50 мс.

Составить структурную схему управляющего автомата, выделить управляющие состояния, построить таблицу или диаграмму переходов и выходов управляющего автомата и код программы на языке C++.

Пример проектирования систем реального времени

В системах реального времени программа постоянно отслеживает состояние внешней среды (нажатие кнопок, срабатывание датчиков и т.п.) и немедленно реагирует на события внешней среды, значимые для данной программной системы.

Причем время обработки события внешней среды не должно превышать время между значимыми событиями внешней среды, т. е. программа должна «успевать» за средой и не пропускать значимые события!

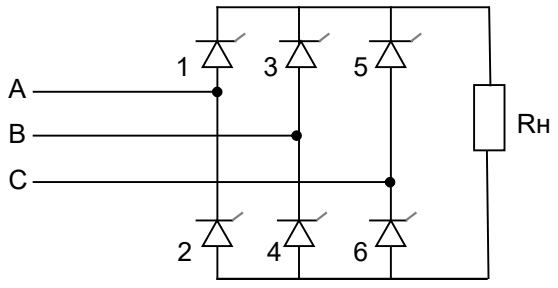
Такие жесткие требования легче удовлетворить с использованием автоматного подхода программирования.

Пусть требуется разработать программу управления тиристорами трехфазного полного моста, которая будет выполняться на микроконтроллере.

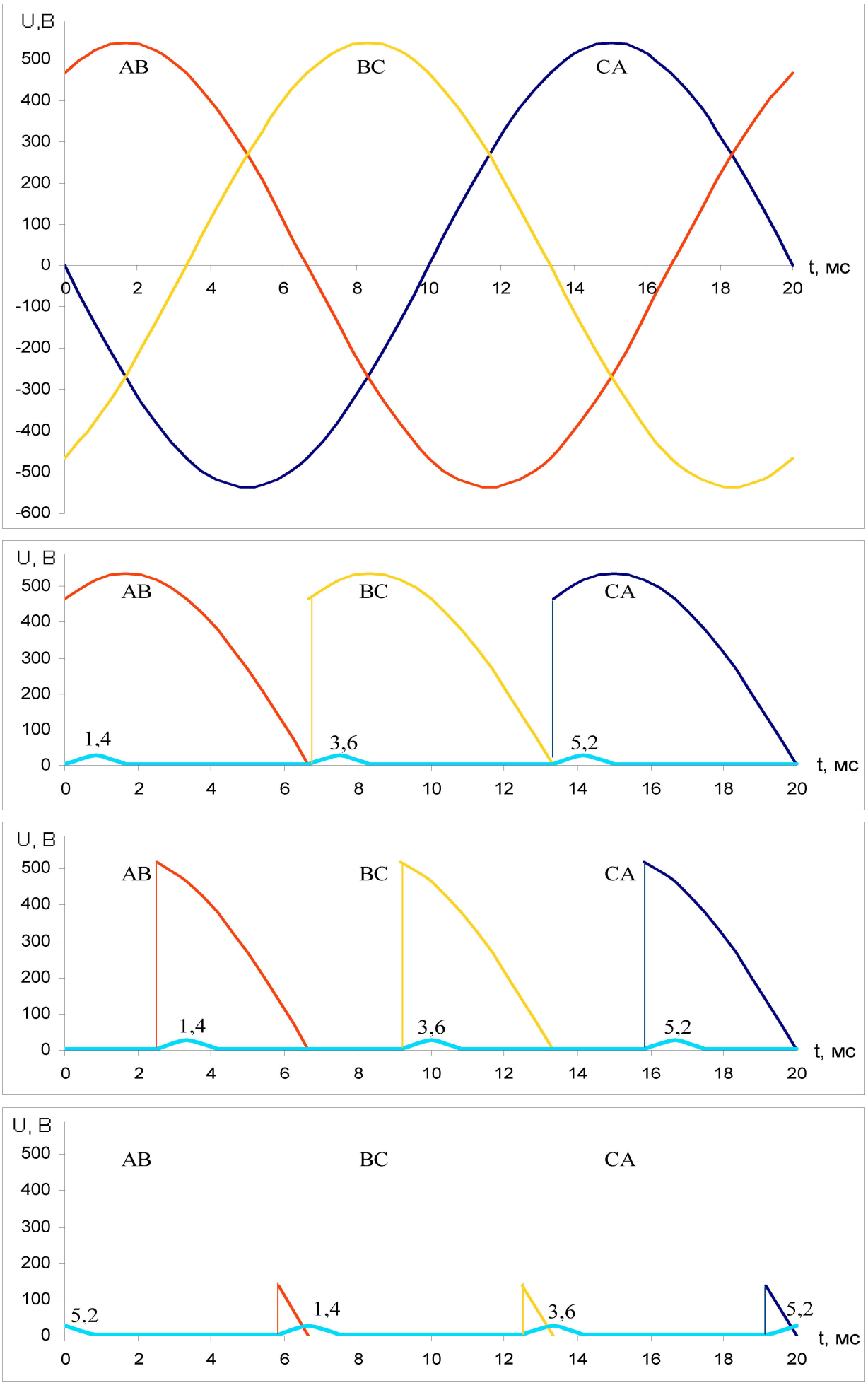
После включения питания программа производит инициализацию устройств микроконтроллера, проверку исправности датчиков тока и напряжения и ожидает включения тумблера пользователем. Если датчики тока и напряжения исправны и тумблер выключен, на индикаторе сообщение «Готов» (m1). Иначе если какой-либо датчик неисправен (показания больше 10А и 10В соответственно при выключенной нагрузке), на индикаторе «ДТ неисправен» (e1) или «ДН неисправен» (e2), иначе если тумблер включен, на индикаторе «Не готов» (e3), программа ожидает выключения тумблера, после чего переходит в состояние готовности с сообщением «Готов».

После включения тумблера (в режиме готовности) на нагрузку подается трехфазное напряжение пилообразной формы (см. ниже). Напряжение регулируется потенциометром от 20 до 220 В. На индикаторе «Работа XXXВ XXXА» (m2), где XXXВ — текущее значение напряжения, XXXА — текущее значение тока, измеренные датчиками напряжения и тока.

После выключения тумблера напряжение выключается и на индикаторе «Готов». При включении тумблера цикл работы повторяется.



Электрическая схема подключения нагрузки



Временные диаграммы напряжений на нагрузке при начальном, среднем и конечном углах управления тиристорами

В данной задаче программа должна в течение периода синусоиды линейного напряжения (20 мс) успевать выполнять целый ряд задач:
1) отслеживать начало периода линейного напряжения;
2) 3 раза за период точно отсчитывать время подпериода (3-й части периода)
3) читать значения с датчиков тока и напряжения;
4) преобразовывать значения с датчиков в действующие значения тока и напряжения;
5) выводить результаты измерения на индикатор;
6) читать оцифрованное значение напряжения с потенциометра;
7) преобразовывать его во время запуска тиристоров (угол управления);
8) отсчитывать это время от начала подпериода до запуска тиристоров;
9) выдавать управляющие импульсы на тиристоры не менее 1 мс (3 раза за период!).

Как выполнить эти задачи за 20 мс?
1) Используется компаратор, выходной сигнал которого «1» (5 В), если Uл>0 и «0» (0 В), если Uл<0. Выход компаратора читается программой через 19-20 мс. Этот период удобнее выбрать в качестве такта управляющего автомата.
2) Используется аппаратный таймер микроконтроллера, дважды отсчитывающий 6666 мкс от начала периода.
3,6) Используется внутренний АЦП контроллера, освобождающий процессор от функции преобразования.
4,7) Используются арифметические операции, которые занимают до 1 мс процессорного времени! (при тактовой частоте контроллера 4 МГц)
5) Используется индикатор с внутренним контроллером индикации, внутренней памятью и интерфейсом универсального последовательного порта (UART), освобождающий процессор от операций вывода на индикатор. Скорость передачи данных на индикатор не меньше 57,6 кбит/с. В рабочем режиме выводится не вся строка, а только ток и напряжение, что сокращает время вывода на индикатор.
8) Используется прерывание по регистру сравнения, срабатывающее, когда значение регистра сравнения будет равно значению таймера.
9) Используется циклическое переключение управляющего вывода тиристора с «0» на «1» с периодом 30 мкс в течение 1 мс (время открытия тиристоров).

1. Построим структурную схему системы управления и определим, имеет ли смысл автоматный подход программирования.

Согласно условию задачи **включенное состояние тумблера ведет к разной реакции программы**: после включения питания – к выводу сообщения «не готов» и ожиданию выключения тумблера, а после выключения тумблера – к выводу сообщения «Работа XXXB XXXA» и включению тиристоров. Кроме того, **сигнал «Время истекло»** (прерывание по регистру сравнения) также **ведет к разным управляющим сигналам на тиристоры** в зависимости от текущего подпериода.

Анализ условия задачи и структурной схемы показал, что в данной задаче имеет смысл **автоматный подход программирования**.

Структурная схема системы управления тиристорами моста

2. По условию задачи выбираем внутренние состояния автомата:
1) не готов (если тумблер включен после инициализации);
2) готов (если тумблер выключен после инициализации);
3) работа (если тумблер включен в состоянии «готов»);

3. На базе условия задачи, структурной схемы и выбранных внутренних состояний строим таблицы переходов и выходов главного автомата и автомата, управляющего тиристорами (матричная запись).

Таблица переходов и выходов главного автомата A1

state	Не готов	Готов	Работа
Не готов	(SW) Z_1(e3)	(!SW) Z_1(m1)	-
Готов	-	ДТ? ДН? (ДТ>10) Z_1(e1) (ДН>10) Z_1(e2)	(SW) Z_1(m2)
Работа	-	(!SW) {Z_1(m1), Z_7}	Z_6(6600), A2, ДТ? ДН? Z_2(10), Z_4, Z_2(15), Z_3

Таблица переходов и выходов автомата A2, управляющего тиристорами

tm	ПП1	ПП2	ПП3
ПП1	(T=Ty) Z_8	(TI) {Z_7, Z_6(6600)}	-
ПП2	-	(T=Ty) Z_9	(TI) {Z_7, Z_6(6600)}
ПП3	(T=Ty) {Z_{10}, Z_7}	-	-

4. Код программы

```
const char PG = 1;      // Код состояния
const char IDLE = 2;
const char WORK = 3;
const char DT = 1;      // Каналы АЦП
const char DN = 2;
const char ANGLE = 3;
char state=PG;          // переменная для режима
char tm=0;              // счетчик частей периода
char Buf[17];           // буфер вывода
unsigned int ccpr1;     // для загрузки в регистр сравнения

void main(void)
{

        GIE = 0;        // отключаем все прерывания
        Init();         // инициализация МК
        GIE = 1;        // включаем прерывания

        while(1) {

                while (SS != 0) continue;       // ждем срез СС
                while (SS != 1) continue;       // ждем фронт СС

                switch(state) {         // выбор режима

                        case PG:    // режим проверки готовности
                                if(!SW) {printf("Готов"); state=IDLE;} // если тумблер отключен
                                else printf("Не готов");
                                break;
                        case IDLE: // режим ожидания
                                if(SW) {sprintf("Работа  В  А",Buf); state=WORK;} //если тумблер включен
                                else {
                                        adc_read(DT);
                                        if(ADRES>10) printf("Неисправен ДТ");
                                        adc_read(DN);
                                        if(ADRES>10) printf("Неисправен ДН");
                                }
                                break;
                        case WORK:  // рабочий режим
                                GIE=0;          // нельзя прерывать иниц-ю таймера и имп. на тир-ры
                                tm=0;           // первая часть периода
                                TMR1ON=0; TMR1=-6600; TMR1ON=1;
                                GIE=1;
                                adc_read(ANGLE);    // чтение угла упр-я с АЦП
                                CCP1IE = 0;             // запрет прерывания на время загрузки регистров
                                ccpr1=-6600+12*ADRES; // загрузка угла упр.
                                CCPR1H=ccpr1>>8;            // загрузка старшего байта регистра сравнения
                                CCPR1L=ccpr1&0xFF;          // загрузка младшего байта регистра сравнения
                                CCP1IE = 1;                 // разрешить прерывание по CCPR1
                                adc_read(DN);               // чтение ДН
                                voltage = ADRES;
                                adc_read(DT);               // чтение ДТ
                                current = ADRES;
```

```
                        Form_buf_lcd(voltage, 10);   // вывод напряжения до 10 позиции
                        Form_buf_lcd(current, 15);   // вывод тока до 15 позиции
                        if(!SW) {printf("Готов"); state=IDLE; TMR1ON=0;} // если тумблер откл.
                        break;
                }
        }
}

#pragma interrupt_level 0
void interrupt tmr1(void)  // обработчик прерываний
{
        if (TMR1IF) {                           // если переполнение таймера 1
                if (tm<2) {tm++;} else {tm=0;}  // счёт частей периода
                TMR1IF = 0;                     // сбрасываем флаг прерывания
                TMR1ON = 0;                     // выключаем таймер 1
                TMR1=-6600;                     // инициализация таймера 1
                TMR1ON = 1;                     // запуск таймера
        }

        if (CCP1IF) {
                CCP1IF=0;
                switch(tm) {
                        case 0: Pulse_Tyr_AB(); break;
                        case 1: Pulse_Tyr_BC(); break;
                        case 2: Pulse_Tyr_CA(); tm=0; TMR1ON=0; break;
                }
        }
}
```

После включения питания программа производит инициализацию устройств микроконтроллера, проверку исправности датчиков тока (ДТ). Если ДТ исправны (показания меньше I_0), на индикаторе сообщение «Готов» (m1) и программа ожидает включения тумблера пользователем. Иначе на индикаторе «ДТ неисправен» (e1), программа может перейти только в режим настройки.

После включения тумблера на нагрузку подается трехфазное напряжение пилообразной формы (см. выше). Напряжение регулируется кнопками «↓» и «↑» от 20 до 220 В с шагом 20 В (при этом позиция напряжения от 1 до 11). Позиция напряжения пересчитывается в угол управления по формуле ccpr1=-6600+500*(11-pos). При нажатии кнопки «•» устанавливается 220 В. На индикаторе «Работа XX A» (m2), где «XX A» — текущее значение тока, измеренное ДТ. После выключения тумблера напряжение выключается и программа входит в режим ожидания, на индикаторе «Готов». При включении тумблера цикл работы повторяется.

Вход в режим настройки возможен только из режима ожидания. В этом режиме кнопками «↓» и «↑» регулируется нулевой ток ДТ I_0 от 1 до 10 А. Вход и выход из режима с сохранением I_0 осуществляется кнопкой «•» .

Сканирование кнопок во всех режимах осуществляется функцией ScanKbd().

Составить структурную схему управляющего автомата, выделить управляющие состояния, построить таблицу или диаграмму переходов и выходов управляющего автомата и код программы на языке C++.

Моделирование несколькими автоматами

Сущность со сложным поведением можно смоделировать как одним так и несколькими взаимодействующими управляющими автоматами. Примеры, рассмотренные ниже, показывают 3 возможных способа моделирования сущности со сложным поведением несколькими автоматами.

Способ 1. Автоматы связаны через состояния (вложенные автоматы)

1. На структурной схеме только один главный автомат, некоторые состояния которого разбиты на подсостояния. Эти подсостояния обрабатывает вложенный автомат, который работает только в определенном состоянии главного автомата.

При таком моделировании есть особенность: при переходе главного автомата из состояния, в котором вызывается вложенный автомат, в другое состояние, вложенный автомат остается в том же состоянии!

2. Состояния автомата:
  1) горит красный свет
  2) горят красный и желтый
  3) горит зеленый
  4) мигает зеленый
  5) горит желтый
  6) мигает желтый
      6.1) настройка времени горения красного цвета
      6.2) настройка времени горения желтого цвета
      6.3) настройка времени горения зеленого цвета

3. Таблицы переходов и выходов

Главный автомат A1

State_1	Красный	Красный-желтый	Зеленый	Мигающий зеленый	Желтый	Мигающий желтый
Красный	«Красный»	(T){$Z_7(10t_ж)$, Z_{10}}	-	-	-	(S) Z_9
Красный-желтый	-	«Кр.-желтый»	(T) {$Z_7(10t_з)$, Z_9, Z_{11}, Z_{12}}	-	-	-
Зеленый	-	-	«Зеленый»	(T) $Z_7(3)$	-	-
Мигающий зеленый	-	-	-	Z_{12}, П(0,5) Z_{13}, П(0,4)	(T) {$Z_7(10t_ж)$, Z_{13}, Z_{10}}	-
Желтый	(T) {$Z_7(10t_к)$, Z_{11}, Z_8}	-	-	-	«Желтый»	-
Мигающий желтый	(S&(State_2=3)) {$Z_7(10t_к)$, Z_{11}, Z_8}	-	-	-		Z_{10}, П(0,5) Z_{11}, П(0,4) A2

Вложенный автомат A2

State_2	Настр. tк	Настр. tж	Настр. tз
Настр. tк	«tк=XXc» (key=D) Z_1 (key=U) Z_2	(key=S)	-
Настр. tж	-	«tж=XXc» (key=D) Z_3 (key=U) Z_4	(key=S)
Настр. tз	(key=S)	-	«tз=XXc» (key=D) Z_5 (key=U) Z_6

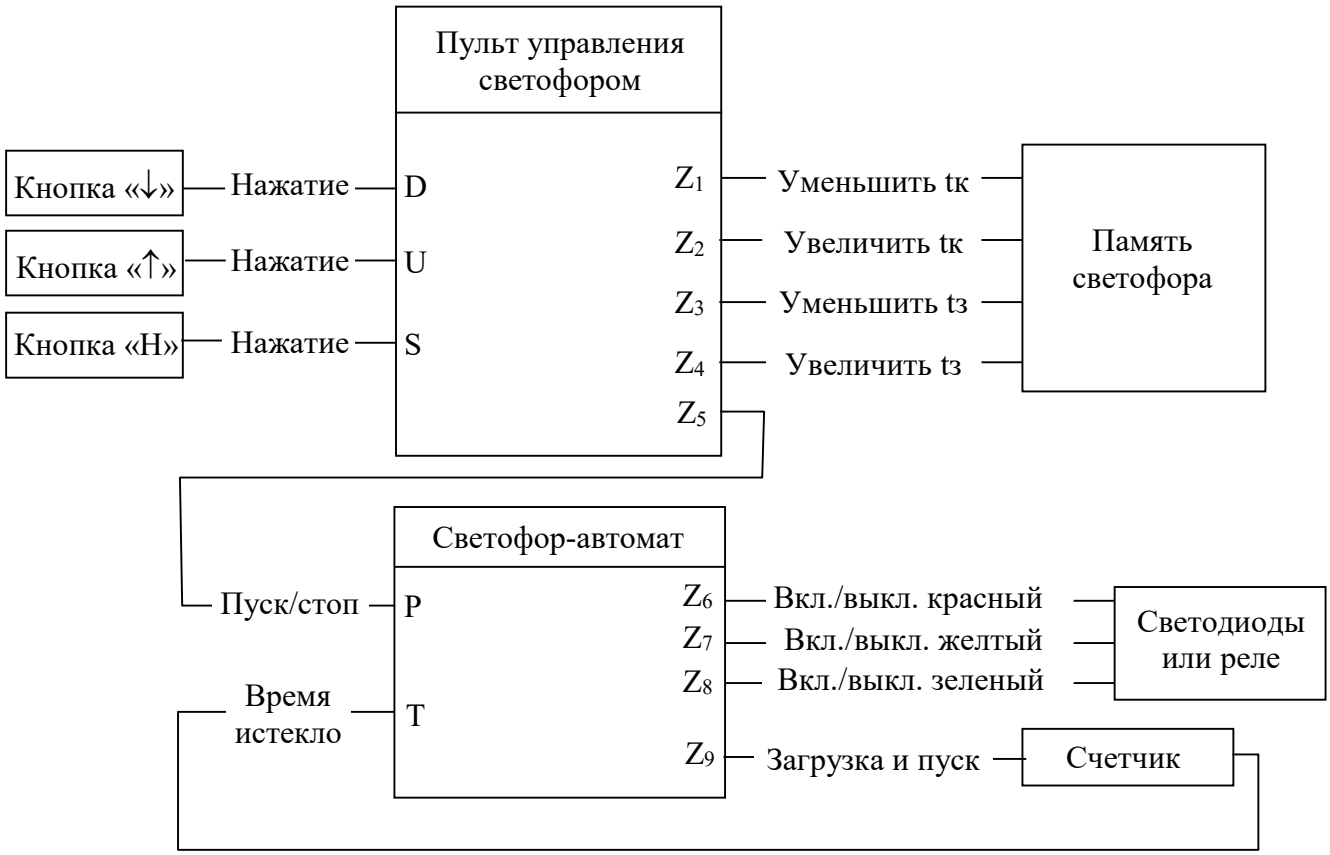
4. Код программы

```
enum states1 {R,RY,G,GG,Y,YY} state1; // Состояния главного автомата
enum states2 {SR, SY, SG} state2; // Состояния вложенного автомата
const char D = 1, U = 2, S = 3; // Код нажатой кнопки
char key;  // Идентификатор нажатой кнопки
char tr = 5, ty = 2, tg = 5;  // Время горения красного, желтого и зеленого сигнала, с
char t = 5;  // Значение счетчика
bit T = 0;    // Сигнал счетчика «время истекло»
bit RED = 1, YEL = 0, GRN = 0;   // Цвет: красный, желтый, зеленый

void main (void) {
        t = 10*tr;
        state1 = R;
// Реализация управляющего автомата
        while (1) {
            key = ScanKbd(); // сканирование кнопок
            switch (state1) {
                case R: // Красный
```

```
                printf("Красный");
                if (T) { state1=RY;  T = 0;  t=10*ty;  YEL=1; }
                else if (key == S) { state1=YY; t=0; RED=0; YEL=0; }
                break;
        case RY: // Красный-желтый
                printf("Красный-желтый");
                if (T) { state1=G;  T = 0;  t=10*tg;  RED=0; YEL=0; GRN=1; }
                break;
        case G: // Зеленый
                printf("Зеленый");
                if (T) { state1=GG;  T = 0;  t=3; }
                break;
        case GG: // Мигающий зеленый
                GRN=1; DelayMs(500);
                GRN=0; DelayMs(400);
                if (T) { state1=Y;  T = 0;  t=10*ty;  YEL=1; }
                break;
        case Y: // Желтый
                printf("Желтый");
                if (T) { state1=R;  T = 0;  t=10*tr; YEL=0; RED=1; }
                break;
        case YY: // Мигающий желтый
                if (T) { T=0; t=5; YEL=1-YEL;} // Мигание желтого
                switch(state2) {
                        case SR: // Настройка времени красного сигнала
                                printf("tк=%2dc", tr);
                                switch(key) {
                                        case D: if (tr>5) tr--; break;
                                        case U: if (tr<30) tr++; break;
                                        case S: state2=SY; break;
                                }
                                break;
                        case SY: // Настройка времени красного сигнала
                                printf("tж=%2dc", ty);
                                switch(key) {
                                        case D: if (ty>2) ty--; break;
                                        case U: if (ty<3) ty++; break;
                                        case S: state2=SG; break;
                                }
                                break;
                        case SG: // Настройка времени зеленого сигнала
                                printf("tз=%2dc", tg);
                                switch(key) {
                                        case D: if (tg>5) tg--; break;
                                        case U: if (tg<30) tg++; break;
                                        case S: state2=SR;
                                                state1=R; T = 0;  t=10*tr; YEL=0; RED=1; break;
                                }
                                break;
                }
                break;
        }
        DelayMs(100);           // такт работы автомата
        if(t==0) T=1; else t--;  // счетчик
    }
}
```

Способ 2. Автоматы связаны через входные и выходные сигналы

1. На структурной схеме главный автомат «Пульт управления светофором» управляет автоматом «Светофор-автомат» с помощью выходного сигнала Z_5 «Пуск/стоп».



2. Состояния автоматов

Главный автомат «Пульт управления светофором»:
1) Работа
2) Настройка времени горения красного цвета
3) Настройка времени горения зеленого цвета

«Светофор-автомат»:
1) горит красный свет
2) горят красный и желтый
3) горит зеленый
4) мигает зеленый
5) горит желтый
6) мигает желтый

3. Таблицы переходов и выходов

Автомат 1 - Пульт управления светофором

State_1	Работа	Настр. tк	Настр. tз
Работа	-	(key=S) !Z_5	-
Настр. tк	-	«tк=XXс» (key=D) Z_1 (key=U) Z_2	(key=S)
Настр. tз	(key=S) Z_5	-	«tз=XXс» (key=D) Z_3 (key=U) Z_4

Автомат 2 – Светофор-автомат

State_2	Красный	Красный-желтый	Зеленый	Мигающий зеленый	Желтый	Мигающий желтый
Красный	«Красный»	(T){$Z_9(20)$, Z_7}	-	-	-	(!P) !Z_6
Красный-желтый	-	«Красный-желтый»	(T) {$Z_9(10t_з)$, !Z_6, !Z_7, Z_8}	-	-	-
Зеленый	-	-	«Зеленый»	(T) $Z_9(3)$	-	-
Мигающий зеленый	-	-	-	Z_8, П(0,5) !Z_8, П(0,4)	(T) {$Z_9(20)$, !Z_8, Z_7}	-
Желтый	(T) {$Z_9(10t_к)$, !Z_7, Z_6}	-	-	-	«Желтый»	-
Мигающий желтый	(P) {$Z_9(t_к)$, !Z_7, Z_6}	-	-	-		Z_7, П(0,5) !Z_7, П(0,4)

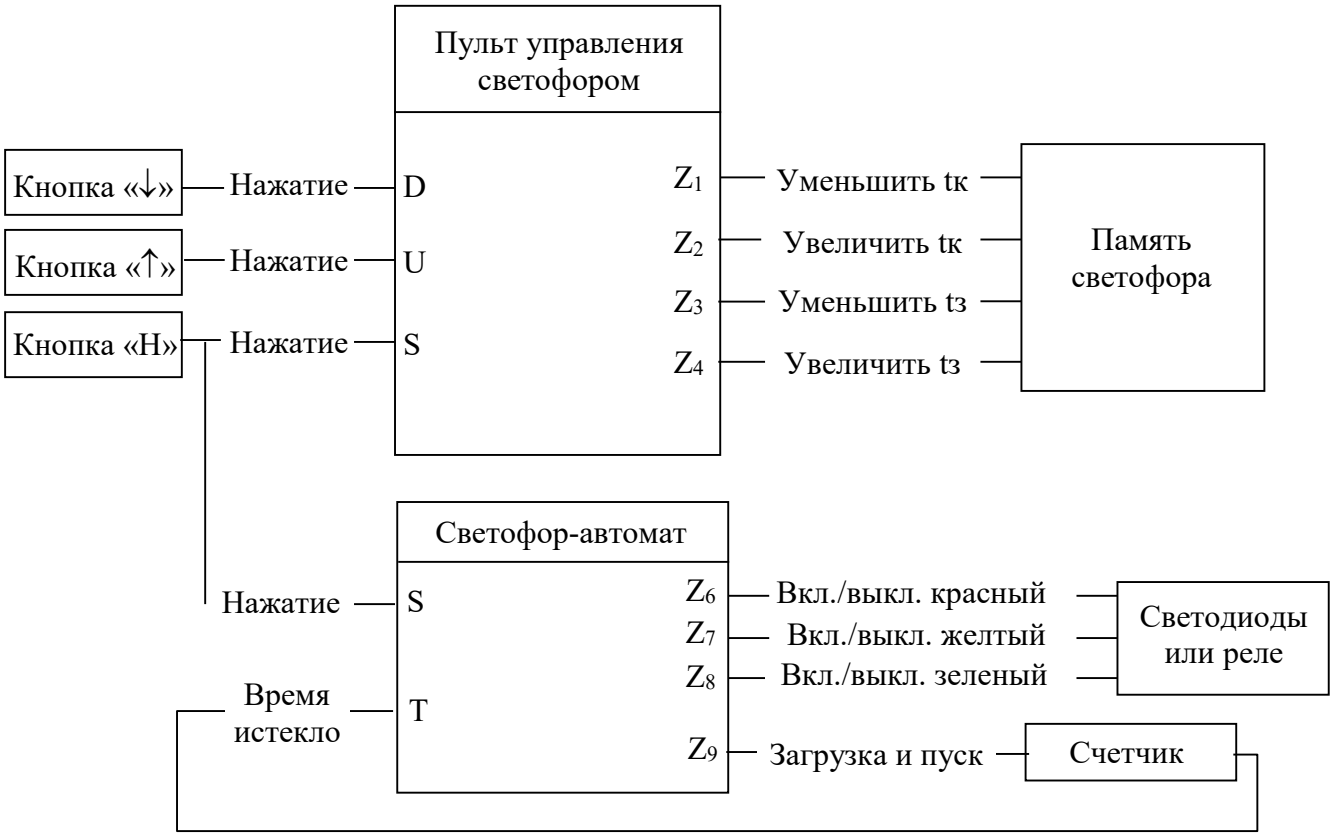
4. Код программы

```
void main (void) {

state1 = 0;       // Начальное состояние ПУ - работа
P=1;              // Светофор работает
t = 10*tr;        // Завести счетчик на время горения красного
state2 = R;       // Начальное состояние светофора – горит красный

// Реализация управляющих автоматов

while (1) {

    key = ScanKbd(); // сканирование кнопок

    switch (state1) {  // ПУ светофором

        case 0: // Работа
                if (key==S) {P=0; state1=1;}
                break;
```

```
        case 1: // Настройка tк
                printf("tк=%2dc", tr);
                switch(key) {
                        case D: if (tr>5) tr--; break;
                        case U: if (tr<30) tr++; break;
                        case S: state1=2; break;
                }
                break;
        case 2: // Настройка tз
                printf("tз=%2dc", tg);
                switch(key) {
                        case D: if (tg>5) tg--; break;
                        case U: if (tg<30) tg++; break;
                        case S: state1=0; P=1; break;
                }
                break;
    }

    switch (state2) {   // Светофор-автомат
        case R: // Красный
                printf("Красный");
                if (T) { state2=RY;  T = 0;  t=20;  YEL=1; }
                else if (!P) { state2=YY;  RED=0; }
                break;
        case RY: // Красный-желтый
                printf("Красный-желтый");
                if (T) { state2=G;  T = 0;  t=10*tg;  RED=0; YEL=0; GRN=1; }
                break;
        case G: // Зеленый
                printf("Зеленый");
                if (T) { state2=GG;  T = 0;  t=3; }
                break;
        case GG: // Мигающий зеленый
                GRN=1; DelayMs(500);
                GRN=0; DelayMs(400);
                if (T) { state2=Y;  T = 0;  t=20;  YEL=1; }
                break;
        case Y: // Желтый
                printf("Желтый");
                if (T) { state2=R;  T = 0;  t=10*tr; YEL=0; RED=1; }
                break;
        case YY: // Мигающий желтый
                if (T) { T=0; t=5; YEL=1-YEL;} // Мигание желтого
                else if (P) {state2=R; T=0; t=10*tr; YEL=0; RED=1;}
                break;
    }

    if(t==0) T=1; else t--;  // счетчик
    DelayMs(100);            // такт работы автомата

} // end while

} // end main
```

Способ 3. Автоматы независимы, но управляются одними входными сигналами

1. На структурной схеме автомат «Пульт управления светофором» и автомат «Светофор-автомат» независимы между собой, но имеют общее входное событие – нажатие кнопки «Н» , которое одновременно переключает «Пульт управления светофором» в состояние настройки и «Светофор-автомат» в состояние мигания желтого и наоборот.



2. Состояния автоматов

«Пульт управления светофором»:
1) Работа
2) Настройка времени горения красного цвета
3) Настройка времени горения зеленого цвета

«Светофор-автомат»:
1) горит красный свет
2) горят красный и желтый
3) горит зеленый
4) мигает зеленый
5) горит желтый
6) мигает желтый

3. Таблицы переходов и выходов

Автомат 1 - Пульт управления светофором

State_1	Работа	Настр. tк	Настр. tз
Работа	-	(key=S)	-
Настр. tк	-	«tк=XXс» (key=D) Z_1 (key=U) Z_2	(key=S)
Настр. tз	(key=S)	-	«tз=XXс» (key=D) Z_3 (key=U) Z_4

Автомат 2 – Светофор-автомат

State_2	Красный	Красный-желтый	Зеленый	Мигающий зеленый	Желтый	Мигающий желтый
Красный	«Красный»	(T){$Z_9(20)$, Z_7}	-	-	-	(key=S) !Z_6
Красный-желтый	-	«Красный-желтый»	(T) {$Z_9(10t_з)$, !Z_6, !Z_7, Z_8}	-	-	-
Зеленый	-	-	«Зеленый»	(T) $Z_9(3)$	-	-
Мигающий зеленый	-	-	-	Z_8, П(0,5) !Z_8, П(0,4)	(T) {$Z_9(20)$, !Z_8, Z_7}	-
Желтый	(T) {$Z_9(10t_к)$, !Z_7, Z_6}	-	-	-	«Желтый»	-
Мигающий желтый	(key=S) {$Z_9(t_к)$, !Z_7, Z_6}	-	-	-		Z_7, П(0,5) !Z_7, П(0,4)

4. Код программы

```
void main (void) {

state1 = 0;       // Начальное состояние ПУ - работа
t = 10*tr;        // Завести счетчик на время горения красного
state2 = R;       // Начальное состояние светофора – горит красный

// Реализация управляющих автоматов

while (1) {

    key = ScanKbd(); // сканирование кнопок

    switch (state1) {  // ПУ светофором

        case 0: // Работа
```

```
                if (key==S) {state1=1;}
                break;
        case 1: // Настройка tк
                printf("tк=%2dc", tr);
                switch(key) {
                        case D: if (tr>5) tr--; break;
                        case U: if (tr<30) tr++; break;
                        case S: state1=2; break;
                }
                break;
        case 2: // Настройка tз
                printf("tз=%2dc", tg);
                switch(key) {
                        case D: if (tg>5) tg--; break;
                        case U: if (tg<30) tg++; break;
                        case S: state1=0;  break;
                }
                break;
    }

    switch (state2) {   // Светофор-автомат
        case R: // Красный
                printf("Красный");
                if (T) { state2=RY;  T = 0;  t=20;  YEL=1; }
                else if (key==S) { state2=YY;  RED=0; }
                break;
        case RY: // Красный-желтый
                printf("Красный-желтый");
                if (T) { state2=G;  T = 0;  t=10*tg;  RED=0; YEL=0; GRN=1; }
                break;
        case G: // Зеленый
                printf("Зеленый");
                if (T) { state2=GG;  T = 0;  t=3; }
                break;
        case GG: // Мигающий зеленый
                GRN=1; DelayMs(500);
                GRN=0; DelayMs(400);
                if (T) { state2=Y;  T = 0;  t=20;  YEL=1; }
                break;
        case Y: // Желтый
                printf("Желтый");
                if (T) { state2=R;  T = 0;  t=10*tr; YEL=0; RED=1; }
                break;
        case YY: // Мигающий желтый
                if (T) { T=0; t=5; YEL=1-YEL;} // Мигание желтого
                else if (key == S) {state2=R; T=0; t=10*tr; YEL=0; RED=1;}
                break;
    }

    if(t==0) T=1; else t--;  // счетчик
    DelayMs(100);           // такт работы автомата

} // end while

} // end main
```

Пользователь вводит с помощью двух кнопок десятичное число от 0 до 9999, которое отображается на 4-разрядном цифровом индикаторе.

Первая кнопка включает и выключает режим ввода числа (при этом включается и выключается выделение изменяемого разряда), переключает между разрядами и выделяет только изменяемый разряд, вторая - изменяет значение разряда числа от 0 до 9 (по кольцу). Вторая кнопка действуют только в режиме ввода числа.

Ввод кода кнопки осуществляется функцией сканирования клавиатуры ScanKbd, вывод символьного массива на индикатор - функцией Put_LCD, включение и отключение подсветки разряда - функциями Sel_ON(n) и Sel_OFF(n), где n - номер разряда; сохранение введенного числа в переменную целого типа при выключении режима ввода – Save(). Сканирование кнопок и вывод на индикатор выполняются через 50 мс.

Смоделировать систему управления двумя автоматами любым из перечисленных способом. Составить структурную схему системы управления, выделить управляющие состояния автоматов, построить таблицу или диаграмму переходов и выходов управляющих автоматов и код программы на языке C++.

Связь автоматного и объектно-ориентированного программирования
Автоматное программирование приложений в MS Windows

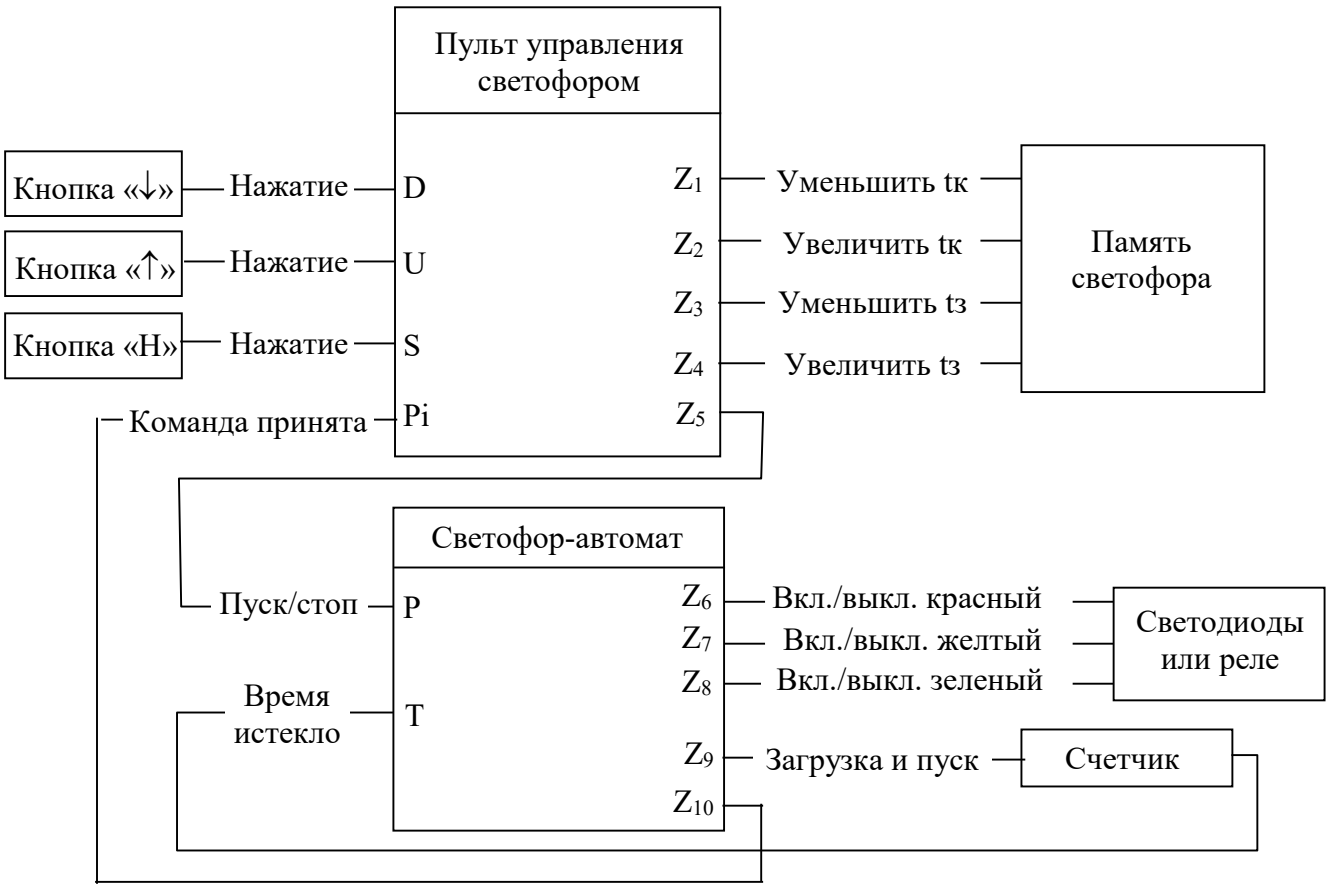
В теории объектно-ориентированного программирования считается, что **объект** имеет **внутреннее состояние** и способен получать **сообщения**, отвечать на них, отправлять сообщения другим объектам. Более приближенное к практике понятие *вызова метода объекта* считается синонимом понятия *отправки сообщения объекту*.

Подавляющее большинство языков программирования, работающих в ОС Windows, являются объектно-ориентированными. Такие языки рассчитаны на объекты с простым поведением, когда объект при вызове его метода имеет вполне определенную реакцию, не зависящую от его состояния и, тем более, от предыстории вызова его методов. С каждым методом связана определенная процедура. В связи с этим сочетать ООП с автоматным, выделяя объект со сложным поведением, довольно затруднительно и приводит к большому избыточному коду.

Наиболее удачным (на мой взгляд) способом автоматного программирования в таких языках является выделение кода автомата (или нескольких автоматов) в обработчик события “Время истекло” объекта **Таймер**.

Пример.

1. На структурной схеме главный автомат «Пульт управления светофором» управляет автоматом «Светофор-автомат» с помощью выходного сигнала (команды) $\mathbf{Z_5}$ «Пуск/стоп». Сам главный автомат переходит в другое состояние только тогда, когда получит от «Светофора-автомата» сигнал подтверждения приема команды **Pi**.



2. Состояния автоматов

Главный автомат «Пульт управления светофором»:
1) Работа
2) Настройка времени горения красного цвета
3) Настройка времени горения зеленого цвета

«Светофор-автомат»:
1) горит красный свет
2) горят красный и желтый
3) горит зеленый
4) мигает зеленый
5) горит желтый
6) мигает желтый

3. Таблицы переходов и выходов

Автомат 1 - Пульт управления светофором

State_1	Работа	Настр. tк	Настр. tз
Работа	(key=S) $!Z_5$	(!Pi)	-
Настр. tк	-	«tк=XXc» (key=D) Z_1 (key=U) Z_2	(key=S)
Настр. tз	(Pi)	-	«tз=XXc» (key=D) Z_3 (key=U) Z_4 (key=S) Z_5

Автомат 2 – Светофор-автомат

State_2	Красный	Красный-желтый	Зеленый	Мигающий зеленый	Желтый	Мигающий желтый
Красный	«Красный»	(T){$Z_9(20)$, Z_7}	-	-	-	(!P) $!Z_6$, Z_{10}
Красный-желтый	-	«Красный-желтый»	(T) {$Z_9(10t_з)$, $!Z_6$, $!Z_7$, Z_8}	-	-	-
Зеленый	-	-	«Зеленый»	(T) $Z_9(3)$	-	-
Мигающий зеленый	-	-	-	Z_8, П(0,5) $!Z_8$, П(0,4)	(T) {$Z_9(20)$, $!Z_8$, Z_7}	-
Желтый	(T) {$Z_9(10t_к)$, $!Z_7$, Z_6}	-	-	-	«Желтый»	-
Мигающий желтый	(P) {$Z_9(t_к)$, $!Z_7$, Z_6, Z_{10}}	-	-	-		Z_7, П(0,5) $!Z_7$, П(0,4)

4. Код программы на Object Pascal

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin

// Реализация управляющих автоматов

case state1 of  // ПУ светофором

        0: // Работа
                begin
                if (key=2) then P_out:=false;
                 if (not P_in) then state1:=1;
                end;
        1: // Настройка tк
                begin
                str(tr,s);
                s:='tк='+s+'c';
                edit1.Text:=s;
                case key of
                        1: if (tr>5) then tr:=tr-1;
                        3: if (tr<30) then tr:=tr+1;
                        2: state1:=2;
                end;
                end;
        2: // Настройка tз
                 begin
                str(tg,s);
                edit1.Text:='tз='+s+'c';
                case key of
                        1: if (tg>5) then tg:=tg-1;
                        3: if (tg<30) then tg:=tg+1;
                        2: P_out:=true;
                end;
                if (P_in) then state1:=0;
                end;
                end;

case state2 of  // Светофор-автомат

        0: // Красный
                begin
                edit1.Text:='Красный';
                if (T) then begin state2:=1; T:=false; tm:=20; YEL(1); end
                else if (not P_out) then begin state2:=5; tm:=5; RED(0); P_in:=false; end;
                end;
        1: // Красный-желтый
                begin
                edit1.Text:='Красный-желтый';
                if (T) then begin state2:=2; T:=false; tm:=10*tg; RED(0); YEL(0); GRN(1); end;
                end;
        2: // Зеленый
                begin
                edit1.Text:='Зеленый';
                if (T) then begin state2:=3; T:=false; tm:=5; end;
                end;
```

```
        3: // Мигающий зеленый
                begin
                if (T) then begin T:=false; tm:=5; k:=k+1; mig:=1-mig; GRN(mig); end;
                 if (k=6) then begin state2:=4; k:=0; tm:=20; YEL(1); end;
                end;
        4: // Желтый
                begin
                edit1.Text:='Желтый';
                if (T) then begin state2:=0; T:=false; tm:=10*tr; YEL(0); RED(1); end;
                end;
        5: // Мигающий желтый
                begin
                if (T) then begin T:=false; tm:=5; mig:=1-mig; YEL(mig); end
                else if (P_out) then begin state2:=0; tm:=10*tr; YEL(0); RED(1); P_in:=true; end;
                end;
                end;

if(tm=0) then T:=true else tm:=tm-1;  // счетчик

key:=0;

end;
```