Санкт-Петербургский государственный университет информационных технологий,
механики и оптики.
Кафедра «Компьютерные технологии»

А.Д. Руднев, А.А. Шалыто

Управление контроллером привода гибкого диска (на примере контроллера *I8272A*)

Программирование с явным выделением состояний.
Проектная документация

Проект создан в рамках «Движения за открытую проектную документацию »

http://is.ifmo.ru

Санкт-Петербург

2006

Оглавление

Введение

Важно подчеркнуть значение простоты и элегантности, так как сложность приводит к нагромождению противоречий и, как мы уже видели, появлению ошибок. Я бы определил элегантность как достижение заданной функциональности при помощи минимума механизма и максимума ясности.

Фернандо Корбато, лауреат премии Turing Award.

Для алгоритмизации и программирования задач логического управления была предложена SWITCH-технология, которая называется также «автоматное программирование» или «программирование с явным выделением состояний» [1]. Первоначально эта технология была предложена предложенный для создания программного обеспечения систем логического управления. В дальнейшем она была расширенна для применения в событийно – управляемых системах [2].

В современном программировании одним из перспективных направлений использования данной технологии, по мнению авторов, является создание драйверов для операционных систем (ОС), например, таких как *QNX Neutrino* и *Linux*.

Применительно к указанным выше ОС, их ядра отлажены и, по-видимому, не содержат серьезных ошибок. Этого, однако, нельзя сказать о драйверах, которые могут серьезно нарушить стабильность работы операционных систем. Особенно серьезна эта проблема для ОС *Linux*, драйверы которой выполняются в привилегированном режиме и могут разрушить всю систему. Отметим, что в настоящее время только небольшая часть драйверов (в основном драйверы сетевых карт и протоколов) строятся с использованием конечных автоматов. Как будет показано ниже, применение автоматов может повысить надёжность и других типов драйверов [3].

Отметим также, что даже в тех случаях, когда автоматы применяют для описания поведения компонент, обычно отсутствует формальный переход от такого описания к коду программ [4].

Другим важным фактором, препятствующим созданию надежных драйверов, является практически полное отсутствие какой–либо документации [5].

Исходный код ядра ОС *Linux* имеет размер около 160 Мб, большая часть которого приходится на разнообразные модули поддержки оборудования. К сожалению, среди них нет ни одного документированного (или даже хотя бы полноценно комментированного), что мешает использованию накопленного опыта.

В настоящем проекте делается попытка решить указанные проблемы на примере разработки драйвера контроллера **накопителя на гибких магнитных дисках** (НГМД), применяемого в компьютерах на базе *Intel*-совместимых процессоров. При этом отметим, что контроллер реализован аппаратно, а система управления должна быть реализована программно.

Оригинальная версия контроллера появилась более 20 лет назад, и с тех пор его интерфейс в целях совместимости не изменялся. Алгоритм управления контроллером чрезвычайно сложен и неудобен, в отличие от жестких дисков, имеющих реализованные аппаратно средства управления и контроля ошибок. Именно по этой причине эта система управления выбрана с целью продемонстрировать удобство применения автоматов в рассматриваемой области.

Для демонстрации была выбрана операционная система *MS-DOS* и язык *C*, широко используемый в системном программировании. Применение *MS-DOS*, в отличие от современных ОС, обеспечивает возможность не углубляться в детали работы с прерываниями и вводом – выводом, доступным лишь в привилегированном режиме. Программа может быть модифицирована для работы в качестве драйвера для большинства ОС [6].

Следует отметить, что в создатель ОС *UNIX* К. Томпсон в интервью [3] на вопрос о текущей работе ответил: “Мы создали язык генерации машин с конечным числом состояний, так как селекторный телефонный разговор — это группа взаимодействующих машин с конечным числом состояний. Этот язык применяется в *Bell Labs* по прямому назначению — для создания указанных машин, а вдобавок с его помощью стали разрабатывать **драйверы**”.

Где посмотреть как разрабатываются драйверы сотрудниками К. Томпсона не известно. Наш же подход описывается ниже, а так как он выполнен в рамках “Движения за открытую проектную документацию”, то его документация опубликована на сайте http://is.ifmo.ru в разделе “Проекты”.

Еще один разговор с великими о построении драйверов состоялся в ходе визита Н. Вирта и Ю. Гутхнехта с СПбГУ ИТМО http://is.ifmo.ru/belletristic/_wirth_poch.pdf. При

этом на пресс-конференции Ю. Гутхнехт сказал, что дела в традиционном программировании обстоят неплохо, если не считать программирования драйверов, при создании которых обычно делается много ошибок.

После этого один из авторов показал гостям проект, в котором программы для драйверов написаны не традиционным путем, а формально и изоморфно по моделям — графам переходов конечных автоматов. Классики сильно удивились …

1. Постановка задачи

Имеется пользовательская программа, для которой должно быть обеспечено выполнение операций чтения сектора гибкого диска. (прочие операции, включая запись, для упрощения программы не рассматриваются).

Между пользовательской программой и гибким диском расположен контроллер I8272A, который управляется с помощью базовой системы ввода – вывода *BIOS*.

В настоящей работе в учебных целях вместо подпрограмм *BIOS,* которые не используются, разрабатывается модуль управления контроллером гибкого диска. Этот модуль осуществляет две основные операции – инициализацию контроллера и чтение сектора данных. Все операции рассчитаны на работу с современной конфигурацией дисководов, которая содержит один привод, читающий двухсторонние диски двойной плотности размером три с половиной дюйма [7].

Схема взаимодействия программной и аппаратной части представлена на рис. 1. Интерфейсный модуль обеспечивает возможность чтения сектора через разработанный модуль вызовом одной функции. Система автоматов взаимодействует с аппаратной частью при помощи специальной прослойки, обеспечивающей простой интерфейс (Приложение 1). Для обеспечения корректности работы при переносе на другую операционную систему требуется переписать только этот модуль.

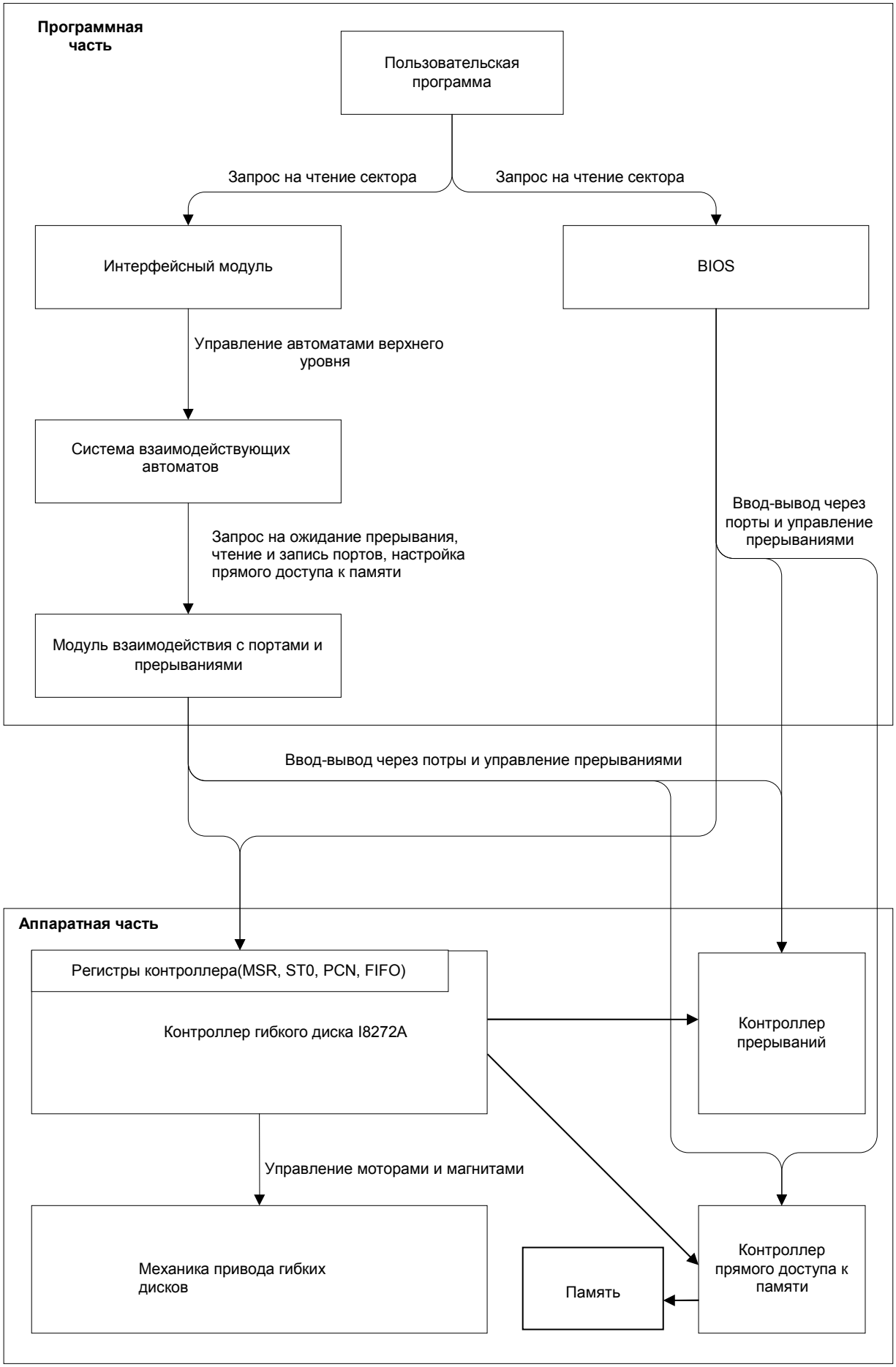

Рис.1 Схема взаимодействия программных и аппаратных частей системы

Контроллер дисковода *I8272A* не содержит буферов приема и передачи информации. Поэтому все операции ввода – вывода, особенно передача данных сектора, из которого выполняется считывание, чрезвычайно критичны по времени. Поэтому для приема данных будет использоваться контроллер прямого доступа к памяти (ПДП), обеспечивающий стабильную скорость передачи.

Приведем основные характеристики современных дисководов и дисков (табл.1). Они будут использоваться для загрузки в соответствующие регистры. Все значения представленны в шестнадцатиричной системе счисления.

Таблица 1. Параметры контроллера и дисков

Параметр	Значение	Код
Размер диска	1.44 Мб	
Количество секторов	2880	
Размер сектора	512 байт	0x02
Количество цилиндров(дорожек)	80 (0-79)	
Количество головок	2 (0-1)	
Количество секторов на цилиндре	8 (1-8)	
Пространство между секторами	27	0x1B
Скорость передачи данных	500 кб/с	0x00
Время загрузки головок	1 мс	0x01
Время разгрузки головок	240 мс	0x0E
Интервал шага головок	2 мс	0x0C

В табл. 2 приведены коды всех команд контроллера, используемых в данной работе.

Таблица 2. Коды команд

Команда	Код
Установка параметров	0x03
Рекалибровка	0x07
Поиск	0x0F
Чтение причины прерывания	0x08
Чтение сектора данных	0xE6

Обработка ошибок – одна из основных трудностей во взаимодействии с контроллером НГМД. Неудачно спроектированная микросхема контроллера *I8272A* плохо реагирует на нестандартные ситуации, возникновением которых программист управлять

не может, и склонна к зависанию. Специально для таких случаев предусмотрен контакт, принудительное замыкание которого приводит к аппаратному сбросу контроллера. Аналогичным образом ведут себя совместимые микросхемы *I82077A, FDC37C78, FDC37C93x, FDC765A, FDC9266, DC9267, W83627HF, W83877F*.

Другими источниками потенциальных ошибок являются механика привода, подверженная загрязнениям, и сами гибкие диски, имеющие низкую надежность хранения информации. При возникновении большинства ошибок требуется повторение операции, и, в случае если ошибки не прекращаются, перезагрузка микросхемы. При этом ошибка в передаче любого байта требует начать операцию сначала, так как контроллер не содержит никаких встроенных средств компенсации ошибок.

Отметим, что работа *BIOS* для рассматриваемой задачи не всегда достаточно надёжна. Поэтому, если контроллер перестаёт отвечать на команды, то зависает и пользовательская программа – *BIOS* не спасает от аппаратных сбоев.

Цель настоящей работы состоит в разработке модуля, заменяющего в этой части *BIOS ,* который маскирует аппаратные ошибки. Это достигается за счёт того, что модуль строится на основе автоматного подхода, в рамках которого при проектировании анализируются все состояния каждой компоненты программы и управляемого оборудования.

2. Интерфейс

Для демонстрации работы разработанного модуля, применяется учебная программа (Приложение 2), читающая первый физический сектор дискеты. Этот сектор содержит загрузочную запись и сведения о самом диске и файловой системе. Чтение демонстрационной программой выполняется дважды: сначала с использованием разработанного модуля, а затем при помощи базовой системы ввода вывода (*BIOS)*. Совпадение информации, считанной с дискеты обоими методами, означает корректную работу модуля. Пример вывода приведен на рис. 2.

```
                          Direct              BIOS
      jmp signature       EB 58 90         EB 58 90
          OEM name:       WINIMAGE         WINIMAGE
   Bytes per sector:           512              512
Sectors per cluster:             1                1
      Root entries:            224              224
     Total sectors:           2880             2880
       Volume label:
        File system:          FAT12            FAT12
_
```

Рис.2. Вывод тестирующей программы

Для запуска демонстрационной программы необходима ОС *MS-DOS*. Для её работы необходимо вставить любой диск в дисковод и запустить исполняемый файл этой программы.

3. Реализация

Как показано на рис. 1, разрабатываемая программа состоит из трёх частей: интерфейсного модуля, системы взаимодействующих автоматов и модуля взаимодействия с портами и прерываниями.

Рис. 3 является схемой взаимодействия некоторых из автоматов с модулем взаимодействия с портами и прерываниями. В его состав входит три процедуры: обработчик прерывания таймера, обработчик прерывания дисковода и настройка прямого доступа к памяти. Также он содержит семафор.

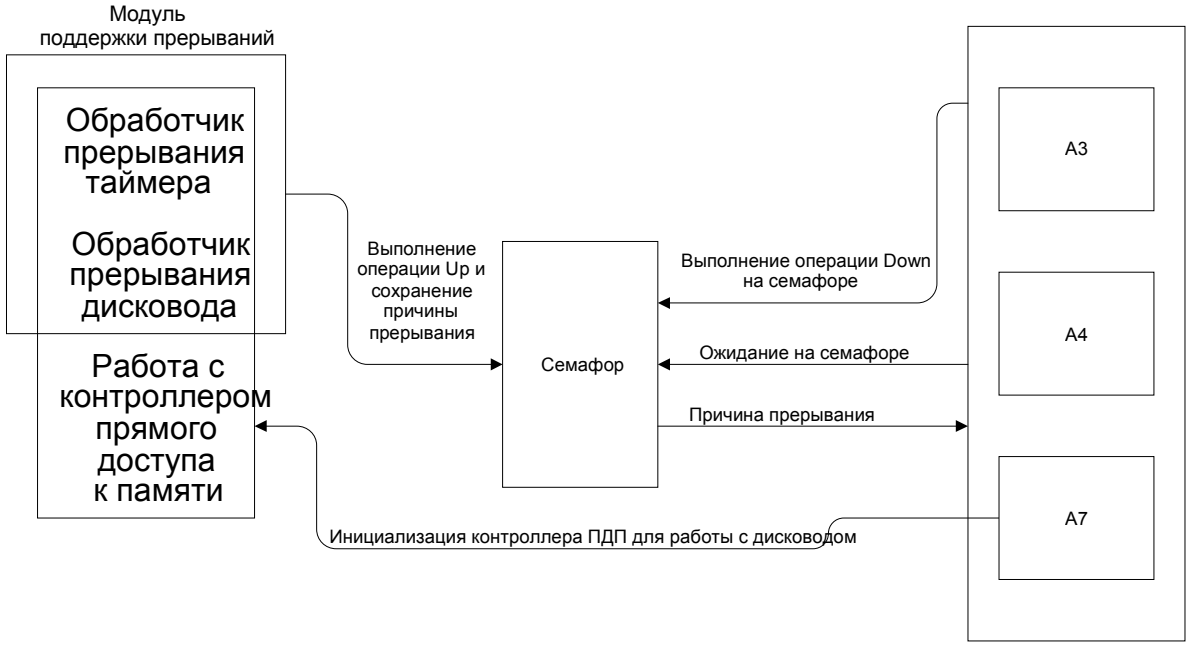

Рис. 3. Схема взаимодействия системы автоматов с модулем взаимодействия с портами и прерываниями.

Исходя из цели настоящей работы, основное внимание в ней уделяется реализации поведения программы на основе автоматного подхода. При этом программа реализуется с помощью восьми автоматов.

На рис. 4 и 5 представлены схемы взаимодействия автоматов.

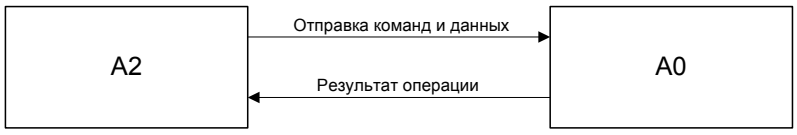

Рис. 4. Схема взаимодействия автоматов, обеспечивающих инициализацию контроллера

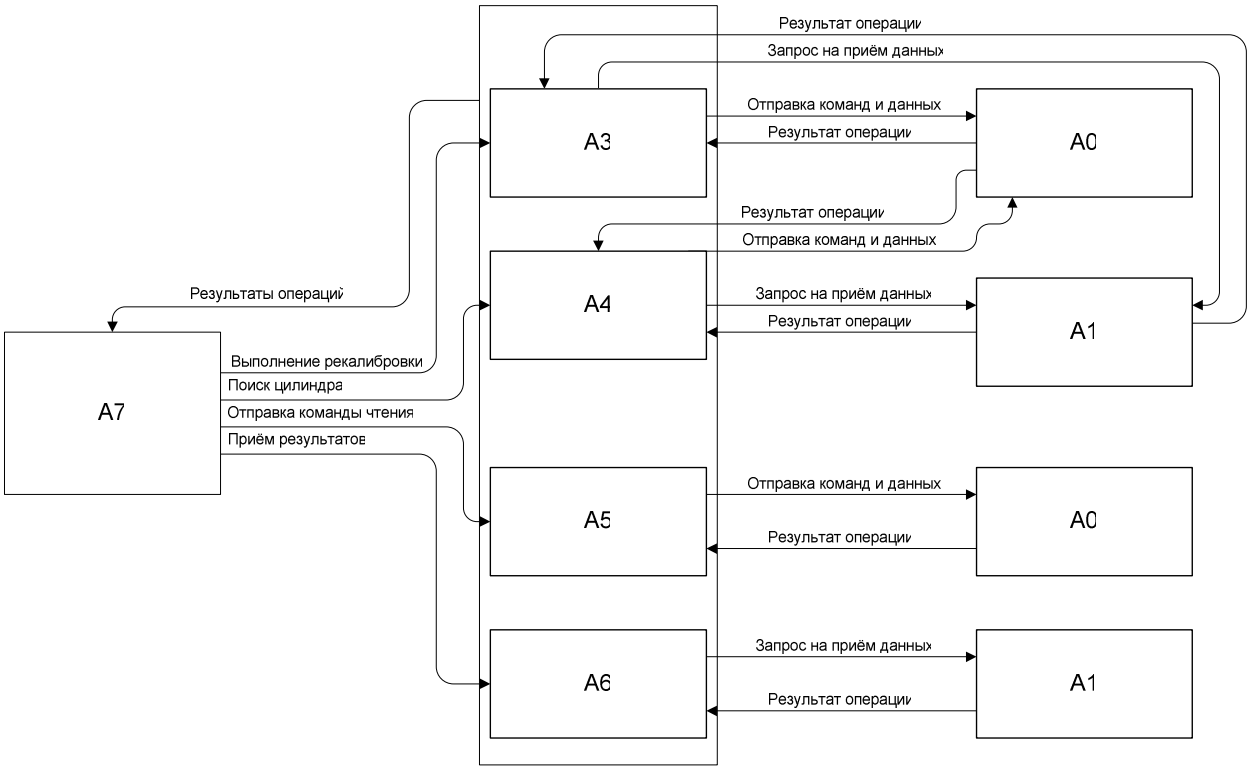

Рис. 5. Схема взаимодействия автоматов, обеспечивающих чтение сектора

4. Реализация автоматов

Входные переменные реализуются как функции, возвращающие логическое `(bool)` значение. Пример заголовка такой функции:

```
bool x0_1(void);
```

Некоторые входные переменные являются блокирующими - приостанавливают работу программы до получения результата. Примером таких переменных являются `x4_6` и `x7_9`. В современных операционных многозадачных системах поток исполнения, запросивший такую операцию, будет отложен до выполнения условия – в данном случае, прихода прерывания. При этом управление будет передано другому потоку. Это увеличит производительность системы в целом и уменьшает время отклика.

Выходные воздействия реализуются как функции, не принимающие и не возвращающие параметров:

```
void z0_2(void);
```

При выбранном способе реализации условия, записанные на ребрах графа переходов, переводятся в код наиболее естественным образом, например булева формула `!(x6_1|x6_2)` представляется в виде `!(x6_1() | x6_2())`.

Реализацию автоматов рассмотрим на примере автомата (рис. 6).

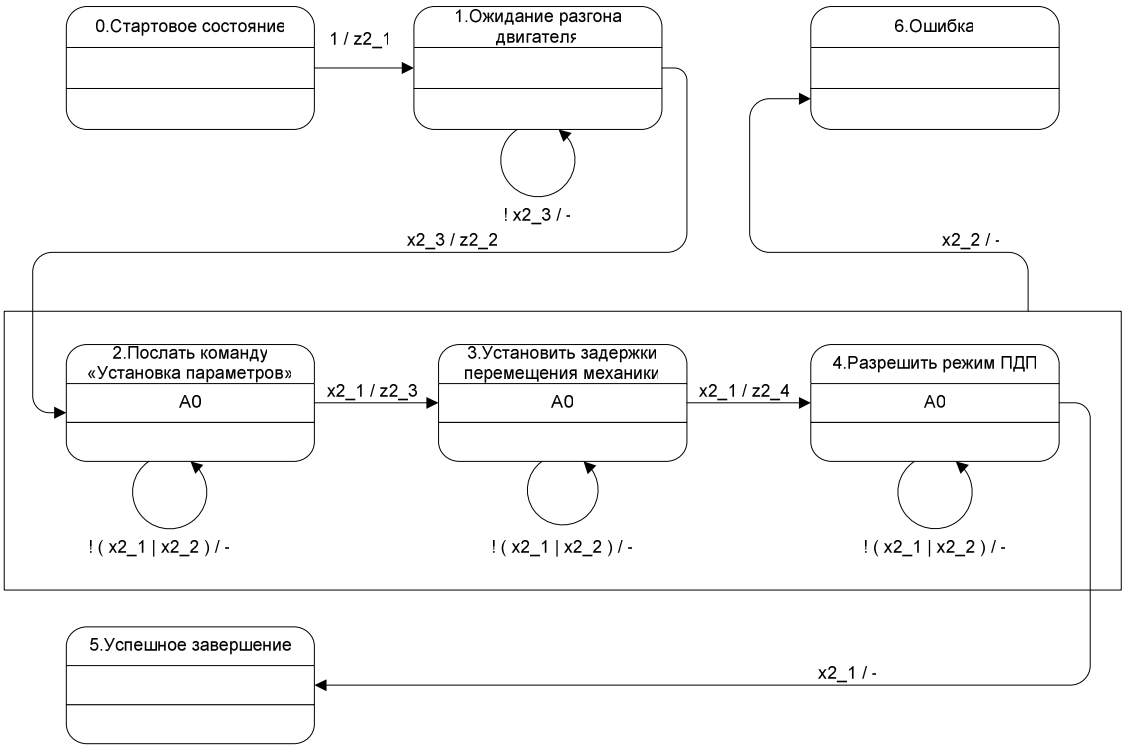

Рис. 6. Реализуемый автомат

Приведём функцию реализующий этот автомат.

```
void A2(void)
{

     LOG_AUTO_ENTER(2);

     switch(y2)
     {
          case 0:
          {
               z2_1();
               y2 = 1;
               break;
          }

          case 1:
          {
               if(!x2_3())
               {
                    y2 = 1;
                    break;
               }

               if(x2_3())
               {
                    z2_2();
                    y2 = 2;
                    break;
               }
               break;
```

```
}

case 2:
{
     if(!(x2_1() | x2_2()))
     {
          A0();
          break;
     }

     if(x2_1())
     {
          z2_3();
          y2 = 3;
          break;
     }

     if(x2_2())
     {
          y2 = 6;
          break;
     }


     break;
}

case 3:
{
     if(!(x2_1() | x2_2()))
     {
          A0();
          break;
     }

     if(x2_1())
     {
          z2_4();
          y2 = 4;
          break;
     }

     if(x2_2())
     {
          y2 = 6;
          break;
     }

     break;
}

case 4:
{
```

```
                if(!(x2_1() | x2_2()))
                {
                     A0();
                     break;
                }

                if(x2_1())
                {
                     y2 = 5;
                     break;
                }

                if(x2_2())
                {
                     y2 = 6;
                     break;
                }

                break;
           }

           case 5:
           {
                break;
           }

           case 6:
           {
                break;
           }
      }

      LOG_AUTO_EXIT(2);

}
```

В рассмотренной реализации макросы `LOG_AUTO_ENTER()` и `LOG_AUTO_EXIT()` протоколируют факты входа и выхода из тела функции автомата.

Автоматы, реализованные, как показано выше, приведены в Приложении 3.

Подобная реализация проста и обладает высоким быстродействием. Перейдем к формальному описанию автоматов.

5. Обозначение входных переменных (x)

Автомат A0

x0_1. Контроллер НГМД не готов к пересылке байта от ЦПУ.

x0_2. Время ожидания ввода–вывода не превысило максимальное.

Автомат A1

x1_1. Контроллер НГМД не готов к пересылке байта ЦПУ.

x1_2. Время ожидания ввода–вывода не превысило максимальное.

Автомат A2

x2_1. Автомат 0 перешел в «состояние 3 — успех».

x2_2. Автомат 0 перешел в «состояние 4 — ошибка».

x2_3. Время ожидания разгона двигателя дисковода истекло.

Автомат A3

x3_1. Автомат 0 перешел в «состояние 3 — успех».

x3_2. Автомат 0 перешел в «состояние 4 — ошибка».

x3_3. Автомат 1 перешел в «состояние 3 — успех».

x3_4. Автомат 1 перешел в «состояние 4 — ошибка».

x3_5. Время ожидания прерывания превысило максимальное.

x3_6. Контроллер прерывания сообщает о наличии прерывания от контроллера НГМД.

x3_7. Регистр `st0` показывает успешное окончание операции.

x3_8. Привод гибкого диска обнаружил нулевую дорожку.

x3_9. Число повторов поиска исчерпано.

Автомат A4

x4_1. Автомат 0 перешел в «состояние 3 — успех».

x4_2. Автомат 0 перешел в «состояние 4 — ошибка».

x4_3. Автомат 1 перешел в «состояние 3 — успех».

x4_4. Автомат 1 перешел в «состояние 4 — ошибка».

x4_5. Время ожидания прерывания превысило максимальное.

x4_6. Контроллер прерывания сообщает о наличии прерывания от контроллера НГМД.

x4_7. Регистр `st0` показывает успешное окончание операции.

x4_8. Привод гибкого диска обнаружил запрошенную дорожку.

x4_9. Число повторов поиска исчерпано.

Автомат A5

x5_1. Автомат 0 перешел в «состояние 3 — успех».

x5_2. Автомат 0 перешел в «состояние 4 — ошибка».

Автомат A6

x6_1. Автомат 1 перешел в «состояние 3 — успех».

x6_2. Автомат 1 перешел в «состояние 4 — ошибка».

Автомат A7

x7_1. Автомат 3 перешел в «состояние 11 — успех».

x7_2. Автомат 3 перешел в «состояние 10 — ошибка».

x7_3. Автомат 4 перешел в «состояние 12 — успех».

x7_4. Автомат 4 перешел в «состояние 11 — ошибка».

x7_5. Автомат 5 перешел в «состояние 11 — успех».

x7_6. Автомат 5 перешел в «состояние 10 — ошибка».

x7_7. Автомат 6 перешел в «состояние 9 — успех».

x7_8. Автомат 6 перешел в «состояние 8 — ошибка».

x7_9. Контроллер прерывания сообщает о наличии прерывания от контроллера НГМД.

x7_10. Время ожидания прерывания превысило максимальное.

x7_11.Регистр `st0` показывает успешное окончание операции.

x7_12. Регистр `st1` показывают ошибку сепаратора данных.

x7_13. Число повторов чтения исчерпано.

6. Обозначение выходных воздействий (z)

Автомат A0

z0_1. Обнулить счетчик тактов.

z0_2. Прочитать регистр *MSR*.

z0_3. Записать байт в *FIFO*.

Автомат A1

z1_1. Обнулить счетчик тактов.

z1_2. Прочитать регистр *MSR*.

z1_3. Прочитать байт из *FIFO*.

Автомат A2

z2_1. Выполнить сброс контроллера. Включить двигатель привода. Установить скорость передачи данных. Обнулить счетчик тактов.

z2_2. Установить автомат *A0* в состояние 0. Записать код команды «Установка параметров» в байт ввода – вывода.

z2_3. Установить автомат *A0* в состояние 0. Записать код задержки механики в байт ввода – вывода.

z2_4. Установить автомат *A0* в состояние 0. Записать код межсекторного интервала и режима ПДП в байт ввода – вывода.

Автомат A3

z3_1. Установить автомат *A0* в состояние 0. Записать код команды «Рекалибровка» в байт ввода – вывода.

z3_2. Установить автомат *A0* в состояние 0. Записать 0x00 в байт ввода – вывода.

z3_3. Обнулить счетчик тактов.

z3_4. Установить автомат *A0* в состояние 0. Записать код команды «Чтение причины прерывания» в байт ввода – вывода.

z3_5. Установить автомат *A1* в состояние 0.

z3_6. Считать байт ввода – вывода в переменную `st0`. Установить автомат *A1* в состояние 0.

z3_7. Считать байт ввода – вывода в переменную `pcn`.

z3_8. Уменьшить количество попыток поиска на единицу.

Автомат A4

z4_1. Установить автомат *A0* в состояние 0. Записать код команды «Поиск» в байт ввода – вывода.

z4_2. Установить автомат *A0* в состояние 0. Записать номер головки в байт ввода – вывода.

z4_3. Установить автомат *A0* в состояние 0. Записать номер дорожки в байт ввода – вывода.

z4_4. Обнулить счетчик тактов.

z4_5. Установить автомат *A0* в состояние 0. Записать код команды «Чтение причины прерывания» в байт ввода – вывода.

z4_6 . Установить автомат *A1* в состояние 0.

z4_7. Считать байт ввода – вывода в переменную `st0`. Установить автомат *A1* в состояние 0.

z4_8. Считать байт ввода – вывода в переменную `pcn`.

z4_9. Уменьшить количество попыток поиска на единицу.

Автомат A5

z5_1. Установить автомат *A0* в состояние 0. Записать код команды «Чтение сектора» в байт ввода – вывода.

z5_2. Установить автомат *A0* в состояние 0. Записать номер головки в байт ввода – вывода.

z5_3. Установить автомат *A0* в состояние 0. Записать номер дорожки в байт ввода – вывода.

z5_4. Установить автомат *A0* в состояние 0. Записать номер головки в байт ввода – вывода.

z5_5. Установить автомат *A0* в состояние 0. Записать номер сектора в байт ввода – вывода.

z5_6. Установить автомат *A0* в состояние 0. Записать код размера сектора в байт ввода – вывода.

z5_7. Установить автомат *A0* в состояние 0. Записать размер дорожки в байт ввода – вывода.

z5_8. Установить автомат *A0* в состояние 0. Записать размер межсекторного промежутка в байт ввода – вывода.

z5_9. Установить автомат *A0* в состояние 0. Записать 0xFF в байт ввода – вывода.

Автомат A6

z6_1. Установить автомат *A1* в состояние 0.

z6_2. Установить автомат *A1* в состояние 0. Считать байт ввода – вывода в переменную st0.

z6_3. Установить автомат *A1* в состояние 0. Считать байт ввода – вывода в переменную st1.

z6_4. Установить автомат *A1* в состояние 0. Считать байт ввода – вывода в переменную st2.

z6_5. Установить автомат *A1* в состояние 0. Считать байт ввода – вывода в переменную c.

z6_6. Установить автомат *A1* в состояние 0. Считать байт ввода – вывода в переменную h.

z6_7. Установить автомат *A1* в состояние 0. Считать байт ввода – вывода в переменную r.

z6_8. Считать байт ввода – вывода в переменную n.

Автомат A7

z7_1. Установить автомат *A3* в состояние 0. Установить число повторов поиска — три.

z7_2. Установить автомат *A4* в состояние 0. Установить число повторов поиска — три.

z7_3. Установить режим работы контроллера ПДП. Установить автомат *A5* в состояние 0.

z7_4. Обнулить счетчик тактов.

z7_5. Установить автомат *A6* в состояние 0.

z7_6. Уменьшить количество повторов чтения на единицу.

7. Автомат «Отправка байта контроллеру НГМД» (A0)

Описание автомата

Автомат обеспечивает передачу данных для контроллера НГМД через буфер *FIFO*. Эта базовая операция требуется для практически всех действий, таких как отправка команд и их параметров. Этот автомат имеет шесть состояний и повторяет определенную в документации фирмы *Intel* последовательность действий по взаимодействию с контроллером.

Состояние 0 («Стартовое состояние») соответствует ожиданию дисководом взаимодействия. При переходе в состояние «Прочитать регистр *MSR*» инициализируется таймер. После этого автомат готов к работе с контроллером.

Переход в состояние «Проверка регистра *MSR*» сопровождается чтением регистра. После этого выполняется проверка готовности контроллера к обмену данными в направлении «от центрального процессора», а, в случае неготовности - переход в состояние «Проверка тайм-аута» для проверки зависания контроллера.

В случае готовности контроллера происходит запись значения `fdc_io_byte` в регистр приемопередатчика и переход в состояние «Успешное завершение». При истечении максимального срока ожидания готовности контроллер считается «зависшим» и выполняется переход в состояние «Ошибка».

Описание состояний автомата

0. Стартовое состояние.
1. Подготовка к чтению регистра *MSR*.
2. Анализ регистра *MSR*.
3. Терминальное состояние «Успех».
4. Терминальное состояние «Ошибка».
5. Проверка тайм-аута.

Схема связей

На рис. 7 приведена схема связей автомата *A0*, описывающая его интерфейс.

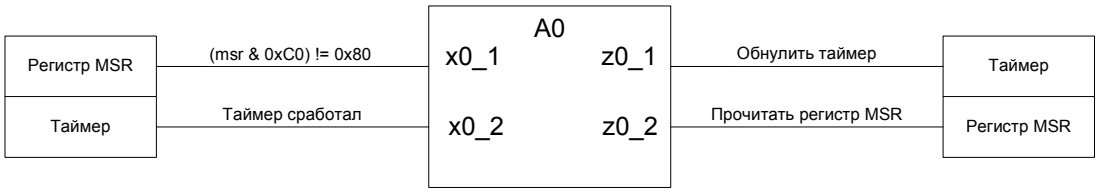

Рис.7. Схема связей автомата *A0*

Граф переходов

На рис. 8 приведен граф переходов автомата *A0*.

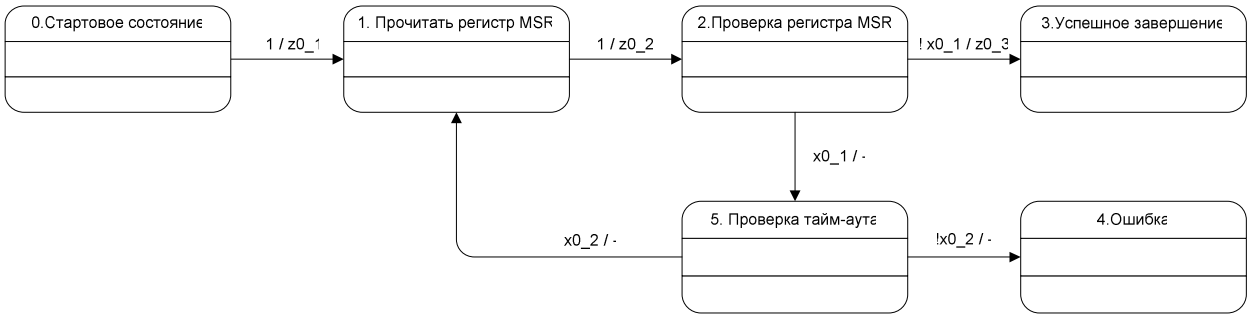

Рис. 8. Граф переходов автомата *A0*

8. Автомат «Прием байта от контроллера НГМД» (A1)

Описание автомата

Этот автомат обеспечивает передачу данных от контроллера НГМД через буфер *FIFO*. Это так же базовая операция, повторяющая автомат *A0* «Отправка байта», за исключением того, что в состоянии «Проверка регистра *MSR*» выполняется переход в состояние «Успешное завершение» при готовности контроллера передать данные для центрального процессора. При этом происходит чтение из регистра приемопередатчика в переменную `fdc_io_byte`.

Описание состояний автомата

0. Стартовое состояние.
1. Подготовка к чтению регистра *MSR*.
2. Анализ регистра *MSR*.
3. Терминальное состояние «Успех».
4. Терминальное состояние «Ошибка».
5. Проверка тайм-аута.

Схема связей

На рис. 9 приведена схема связей автомата *A1*, описывающая его интерфейс.

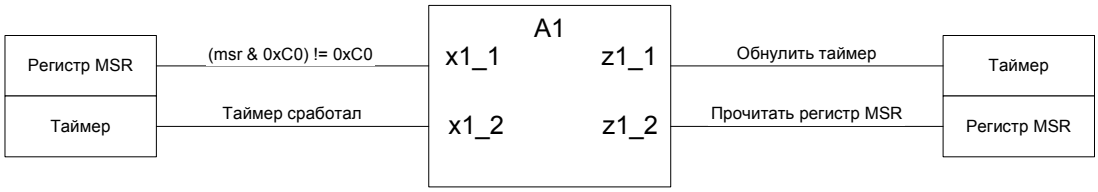

Рис. 9. Схема связей автомата *A1*

Граф переходов

На рис. 10 приведен граф переходов автомата *A1*.

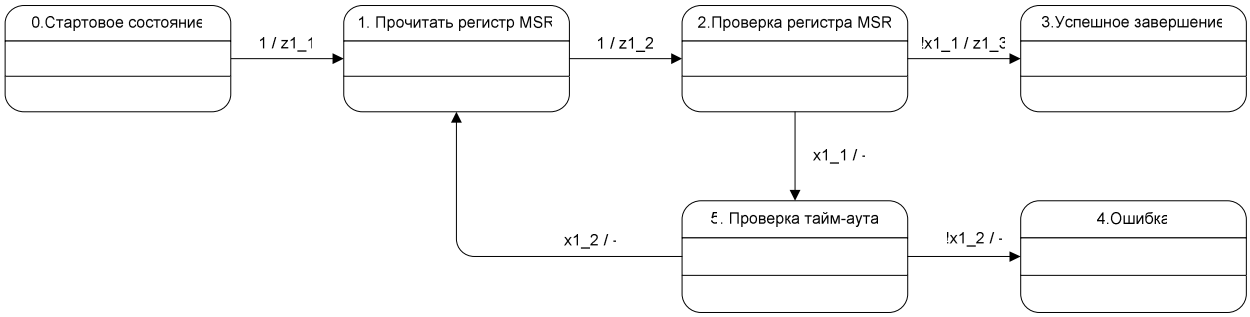

Рис. 10. Граф переходов автомата *A1*

9. Автомат «Инициализация контроллера» (A2)

Описание автомата

Этот автомат обеспечивает сброс и настройку контроллера, что необходимо до начала работы и при возникновении любой ошибки в дисководе.

При переходе из стартового состояния в состояние «Ожидание разгона двигателя» путем записи в управляющий регистр, последовательно выполняются сброс, запуск двигателя и инициализация таймера. В состоянии «Ожидание разгона двигателя» согласно спецификации компании *Intel* автомат должен находиться 0.5 с, что соответствует времени разгона привода, рассчитанного на диски размером 3.5 дюйма.

Затем при помощи вызова автомата *A0* контроллеру посылается команда «Установка параметров» и два ее аргумента. Эта команда устанавливает характерные для трехдюймовых дисков величины межсекторных, межцилиндровых интервалов, а также необходимую скорость передачи данных. Обнаружение ошибки на любом этапе приводит к переходу в состояние «Ошибка», после которого необходимо попытаться выполнить

инициализацию еще раз. Так как рассматриваемая команда относится числу команд без фазы выполнения и выдачи состояния, то успех пересылки команды обозначает успешное выполнение всей процедуры инициализации и переход в состояние «Успешное завершение».

Описание состояний автомата

0. Стартовое состояние.
1. Ожидание разгона двигателя. Дисковод начинает вращать диск.
2. Автомат посылает команду «Установка параметров» путем реализации вложенного автомата *A0*.
3. Автомат устанавливает задержки перемещения механики путем реализации вложенного автомата *A0*.
4. Автомат разрешает режим прямого доступа к памяти путем реализации вложенного автомата *A0*.
5. Терминальное состояние «Успех».
6. Терминальное состояние «Ошибка».

Схема связей

На рис. 11 приведена схема связей автомата *A2*, описывающая его интерфейс.

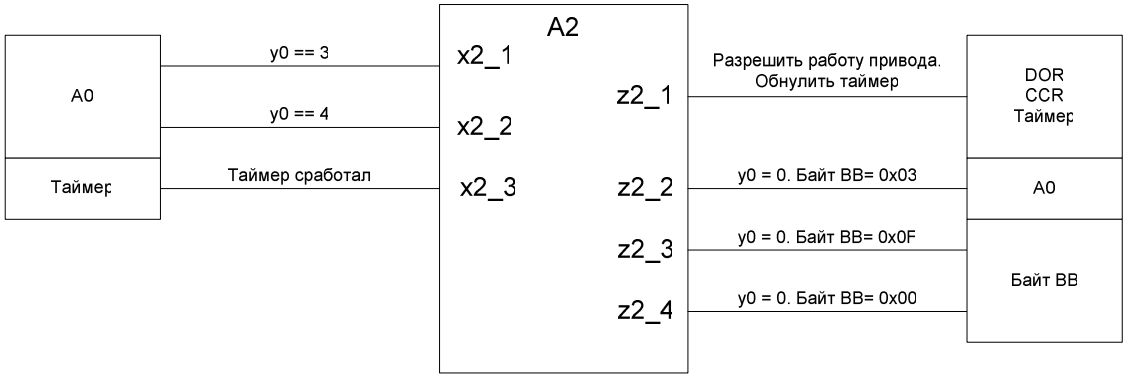

Рис. 11. Схема связей автомата *A2*

Граф переходов

На рис. 12 приведен граф переходов автомата *A2*.

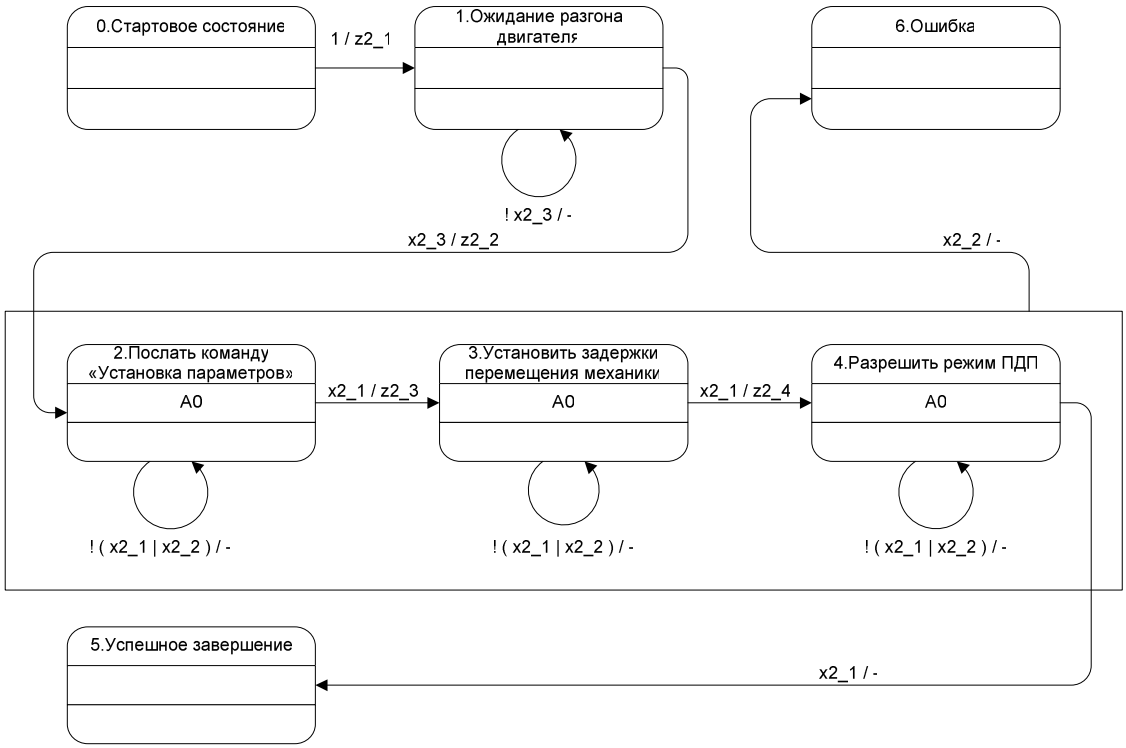

Рис. 12. Граф переходов автомата *A2*

10. Автомат «Рекалибровка» (A3)

Описание автомата

Этот автомат отвечает за выполнение команды «Рекалибровка». В случае, когда контроллер дисковода не может определить текущую дорожку при помощи считывания маркеров с диска, необходимо перевести блок головок привода в крайнее состояние до ограничителя. После этого блок может начать сканирование дорожек повторно. Так как при этом в соответствии с геометрией диска головка выполняет 77 шагов, то для нестандартных дискет, имеющих большее количество дорожек, или же при ошибках в работе механической части (привод может быть загрязнен), необходима многократная подача команды до достижения блоком головок упора.

В состояниях «Послать команду рекалибровки» и «Послать атрибуты» контроллеру указывается, на какой головке выполнять операцию. Так как у современных дисководов обе головки двигаются синхронно, то этот параметр можно всегда считать равным нулю. Эта команда имеет фазу выполнения, но не имеет фазы вывода результата. Если за заданный промежуток времени после подачи команды контроллер выставляет прерывание, то выполнение команды завершилось. В противном случае дисковод считается «зависшим» и выполняется переход в состояние «Ошибка». В случае успеха подается команда чтения причины прерывания и чтение ее результатов (состояния

«Послать команду чтения прерывания», «Прочитать регистр `st0`» и «Прочитать регистр `pcn`»).

В случае, если блок головок обнаружил ограничитель, то можно перейти в состояние «Успешное завершение». В противном случае выполняется проверка доступного количества повторов. Если предельное количество повторов не достигнуто, то выполняется переход из состояния «Проверка лимита повторов» обратно в состояние «Стартовое состояние».

Описание состояний автомата

0. Стартовое состояние.
1. Автомат посылает команду «Рекалибровки» путем реализации вложенного автомата *A0*.
2. Автомат посылает атрибуты команды «Рекалибровки» путем реализации вложенного автомата *A0*.
3. Автомат ожидает прерывание, выполняя блокирующий вызов.
4. Автомат посылает команду чтения прерывания путем реализации вложенного автомата *A0*.
5. Автомат читает содержимое регистра `st0` путем реализации вложенного автомата *A1*.
6. Автомат читает содержимое регистра `pcn` путем реализации вложенного автомата *A1*.
7. Проверка регистра `st0` на ошибку.
8. Проверка регистра `pcn` на ошибку.
9. Проверка превышения лимита повтора.
10. Терминальное состояние «Успех».
11. Терминальное состояние «Ошибка».

Схема связей

На рис. 13 приведена схема связей автомата *A3*, описывающая его интерфейс.

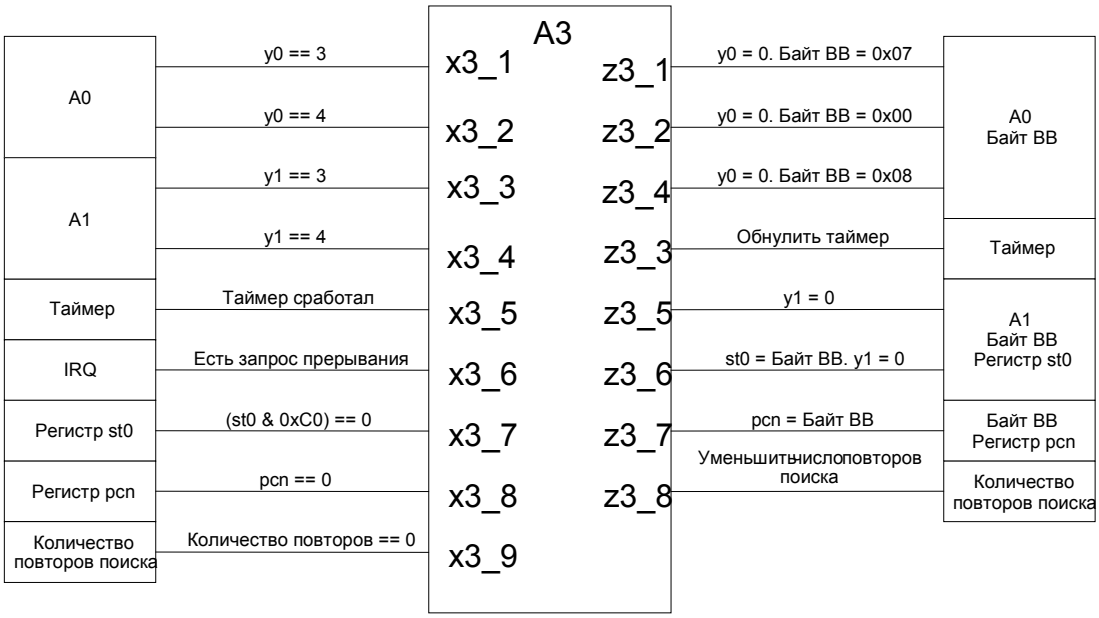

Рис. 13. Схема связей автомата *A3*

Граф переходов

На рис. 14 приведен граф переходов автомата *A3*.

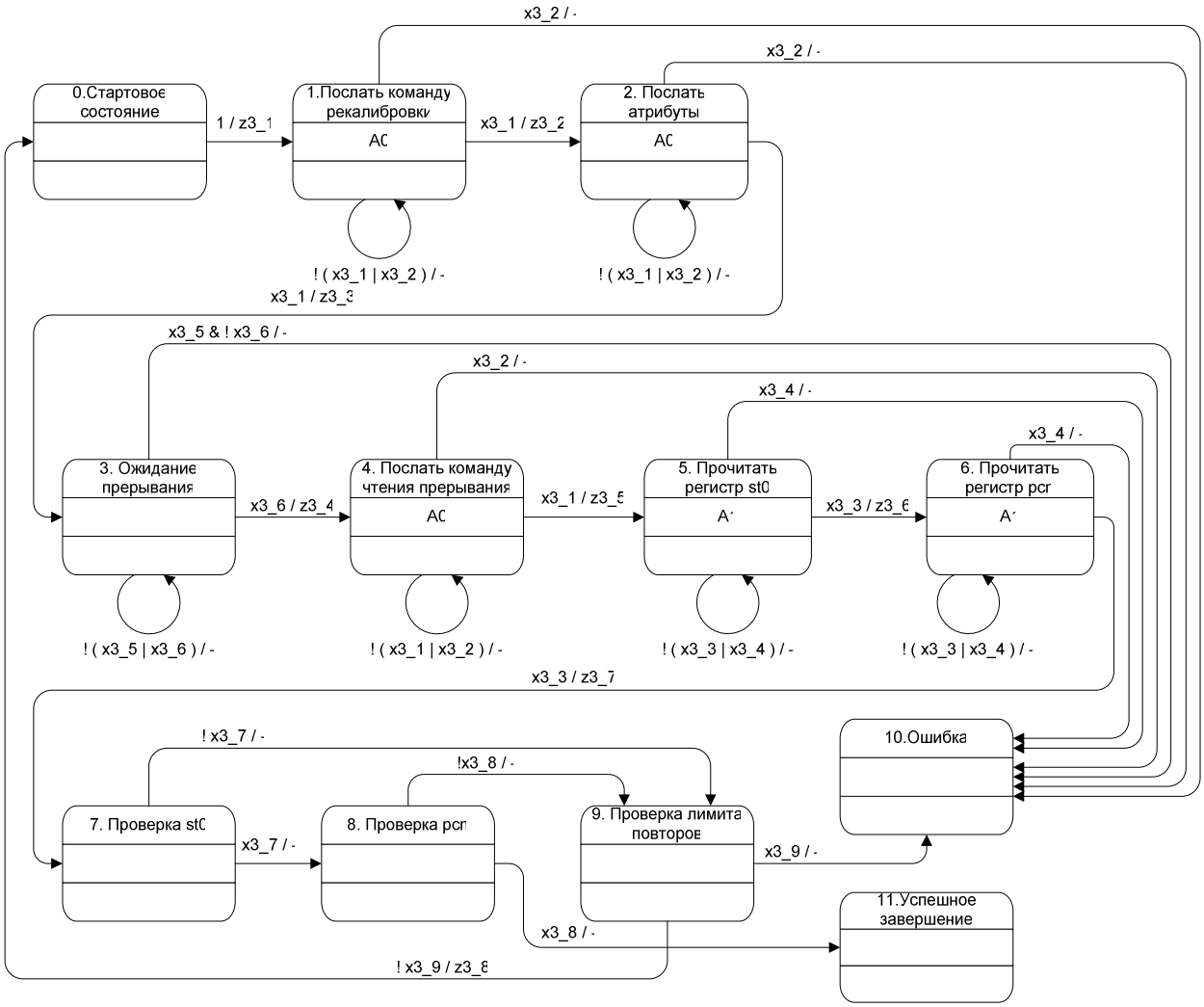

Рис. 14. Граф переходов автомата *A3*

11. Автомат «Поиск» (A4)

Описание автомата

Этот автомат отвечает за выполнение команды «Поиск». При помощи этой команды привод устанавливает головки на заданную дорожку, готовясь к считыванию данных с нее. Он аналогичен операции «Рекалибровка», отличаясь от нее лишь кодом и параметрами команды, и проверкой успешности выполнения. В аргументы команды входит отправка номера цилиндра и головки, на которую необходимо позиционировать механику дисковода. При окончании выполнения операции необходимо сравнить номер запрошенного цилиндра и возвращенного в регистре `pcn` значения. Они должны быть равны. В противном случае поиск необходимо произвести повторно.

Описание состояний автомата

0. Стартовое состояние.
1. Автомат посылает команду «Поиск» путем реализации вложенного автомата *A0*.
2. Автомат посылает номер головки путем реализации вложенного автомата *A0*.
3. Автомат посылает атрибуты команды «Поиск» путем реализации вложенного автомата *A0*.
4. Автомат ожидает прерывание, выполняя блокирующий вызов.
5. Автомат посылает команду чтения прерывания путем реализации вложенного автомата *A0*.
6. Автомат читает содержимое регистра `st0` путем реализации вложенного автомата *A1*.
7. Автомат читает содержимое регистра `pcn` путем реализации вложенного автомата *A1*.
8. Проверка регистра `st0` на ошибку.
9. Проверка регистра `pcn` на ошибку.
10. Проверка превышения лимита повтора.
11. Терминальное состояние «Успех».
12. Терминальное состояние «Ошибка».

Схема связей

На рис. 15 приведена схема связей автомата *A4*, описывающая его интерфейс.

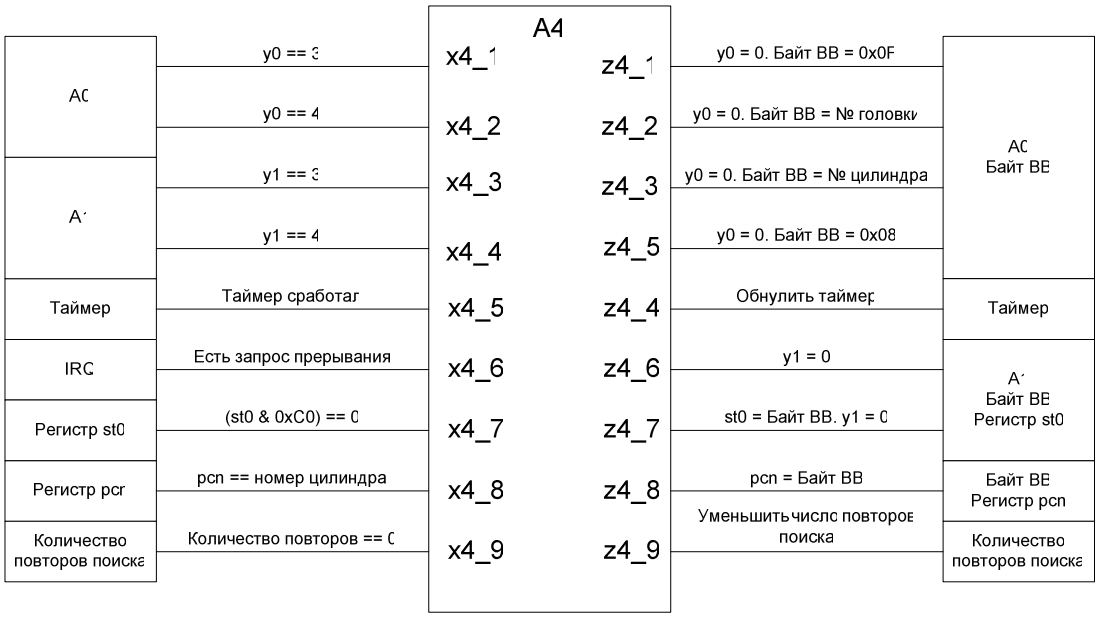

Рис. 15 Схема связей автомата *A4*

Граф переходов

На рис. 16 приведен граф переходов автомата *A4*.

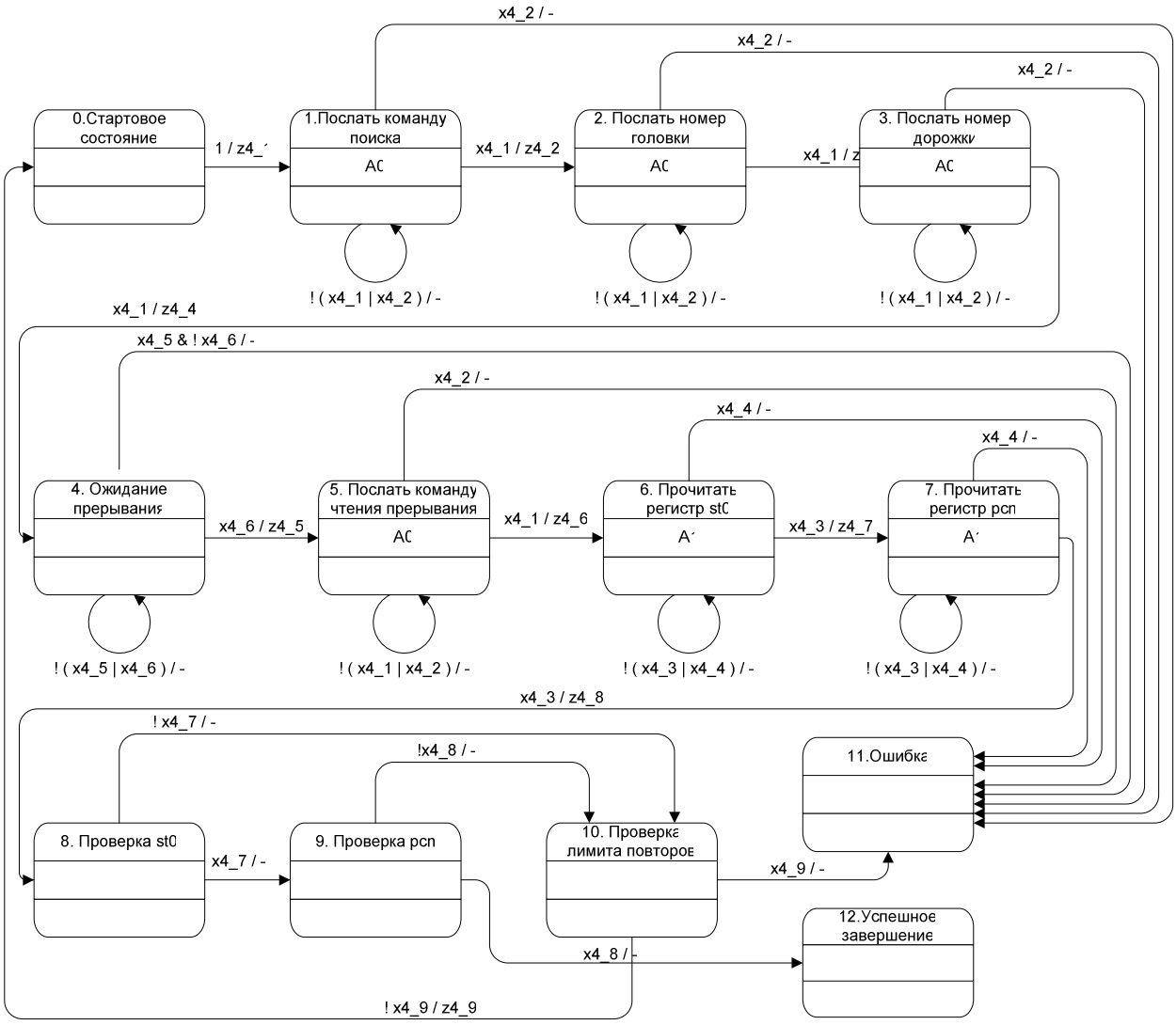

Рис. 16. Граф переходов автомата *A4*

12. Автомат «Отправка команды чтения» (A5)

Описание автомата

Это вспомогательный автомат, обеспечивающий корректную пересылку длинной (девять байт) команды чтения. Он обеспечивает последовательные вызовы автомата *A0*, который устанавливает дорожку, цилиндр, номер сектора, размер сектора и ряд других параметров.

Описание состояний автомата

0. Стартовое состояние.
1. Автомат посылает команду «Чтение» путем реализации вложенного автомата *A0*.
2. Автомат посылает номер головки путем реализации вложенного автомата *A0*.
3. Автомат посылает номер цилиндра путем реализации вложенного автомата *A0*.
4. Автомат посылает номер головки путем реализации вложенного автомата *A0*.
5. Автомат посылает номер сектора путем реализации вложенного автомата *A0*.

6. Автомат посылает код размера сектора путем реализации вложенного автомата *A0*.
7. Автомат посылает размер дорожки путем реализации вложенного автомата *A0*.
8. Автомат посылает межсекторный интервал путем реализации вложенного автомата *A0*.
9. Автомат посылает специальный размер путем реализации вложенного автомата *A0*.
10. Терминальное состояние «Успех».
11. Терминальное состояние «Ошибка».

Схема связей

На рис. 17 приведена схема связей автомата *A5*, описывающая его интерфейс.

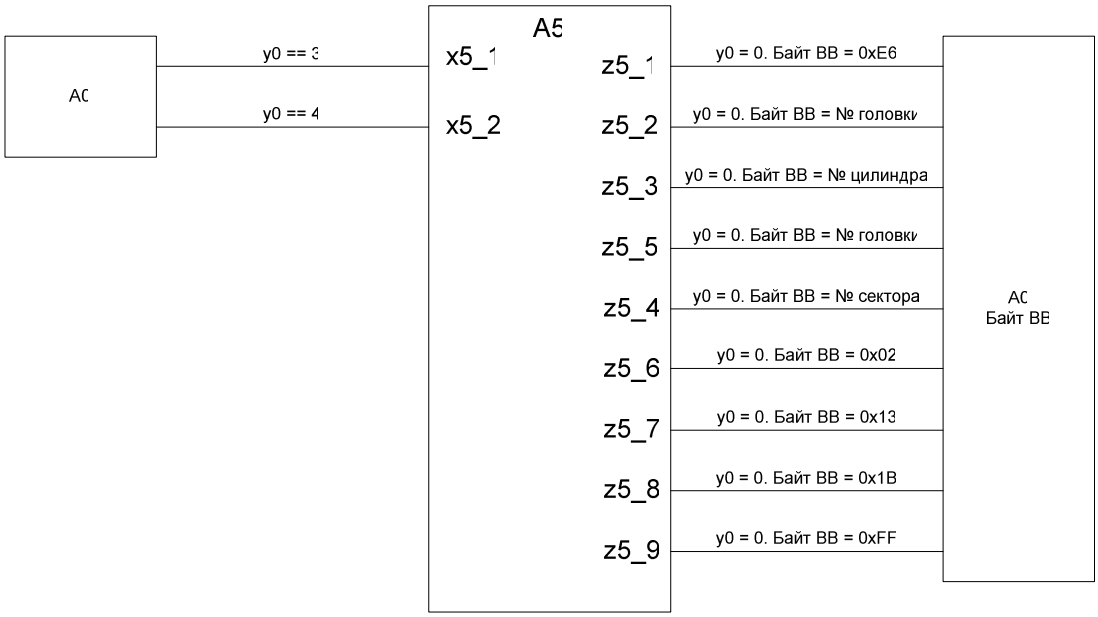

Рис. 17. Схема связей автомата *A5*

Граф переходов

На рис. 18 приведен граф переходов автомата *A5*.

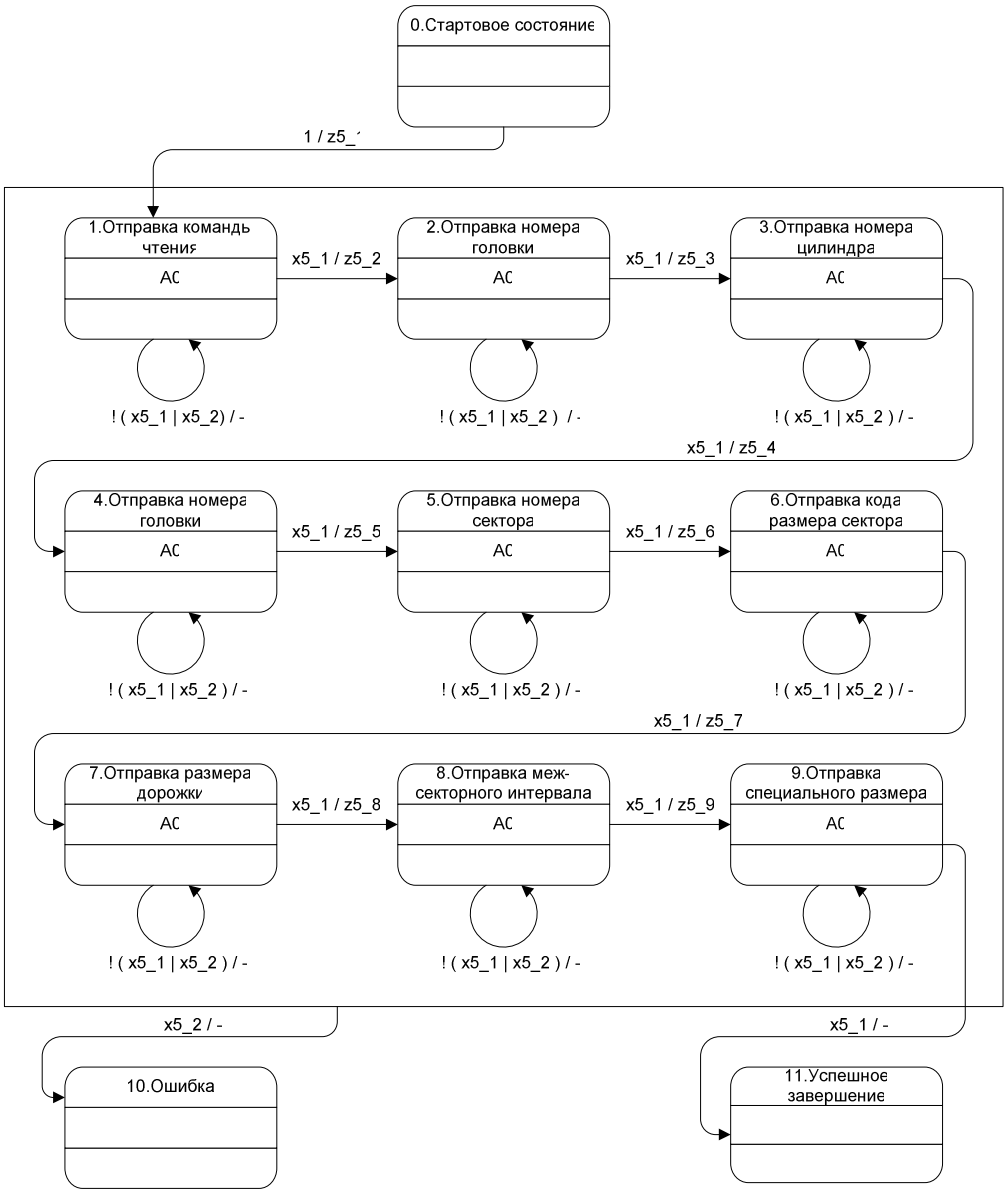

Рис. 18. Граф переходов автомата *A5*

13. Автомат «Чтение результатов» (A6)

Описание автомата

Вспомогательный автомат, обеспечивающий корректную пересылку семибайтного результата команды чтения. Он реализует последовательные вызовы автомата *A1*, который считывает состояние контроллера после операции чтения.

Описание состояний автомата

0. Стартовое состояние.
1. Автомат читает регистр `st0` путем реализации вложенного автомата *A1*.
2. Автомат читает регистр `st1` путем реализации вложенного автомата *A1*.
3. Автомат читает регистр `st2` путем реализации вложенного автомата *A1*.

4. Автомат читает регистр `c` путем реализации вложенного автомата *A1*.
5. Автомат читает регистр `h` путем реализации вложенного автомата *A1*.
6. Автомат читает регистр `r` путем реализации вложенного автомата *A1*.
7. Автомат читает регистр `n` путем реализации вложенного автомата *A1*.
8. Терминальное состояние «Успех».
9. Терминальное состояние «Ошибка».

Схема связей

На рис. 19 приведена схема связей автомата *A6*, описывающая его интерфейс.

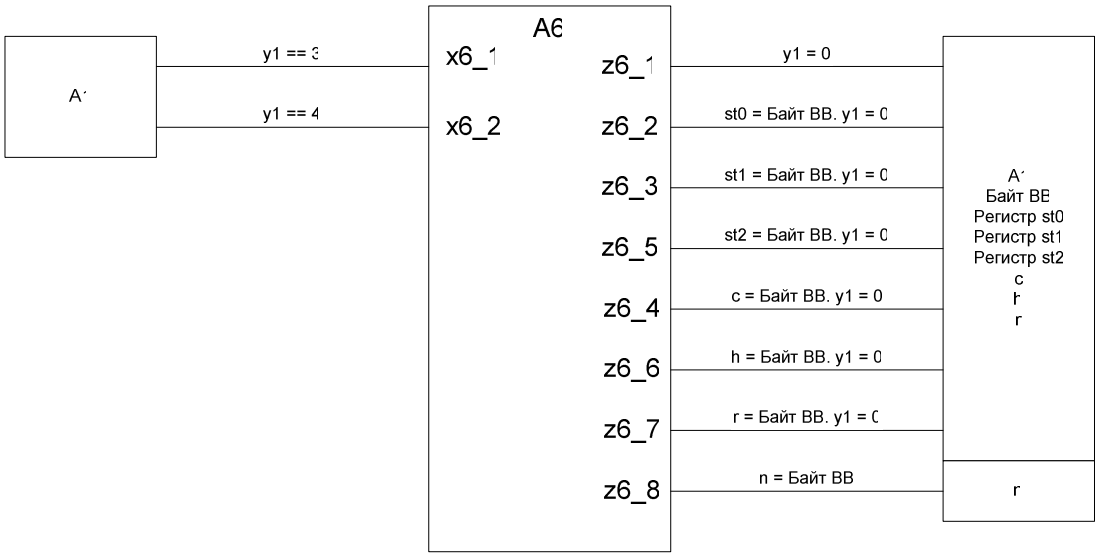

Рис. 19. Схема связей автомата *A6*

Граф переходов

На рис. 20 приведен граф переходов автомата *A6*.

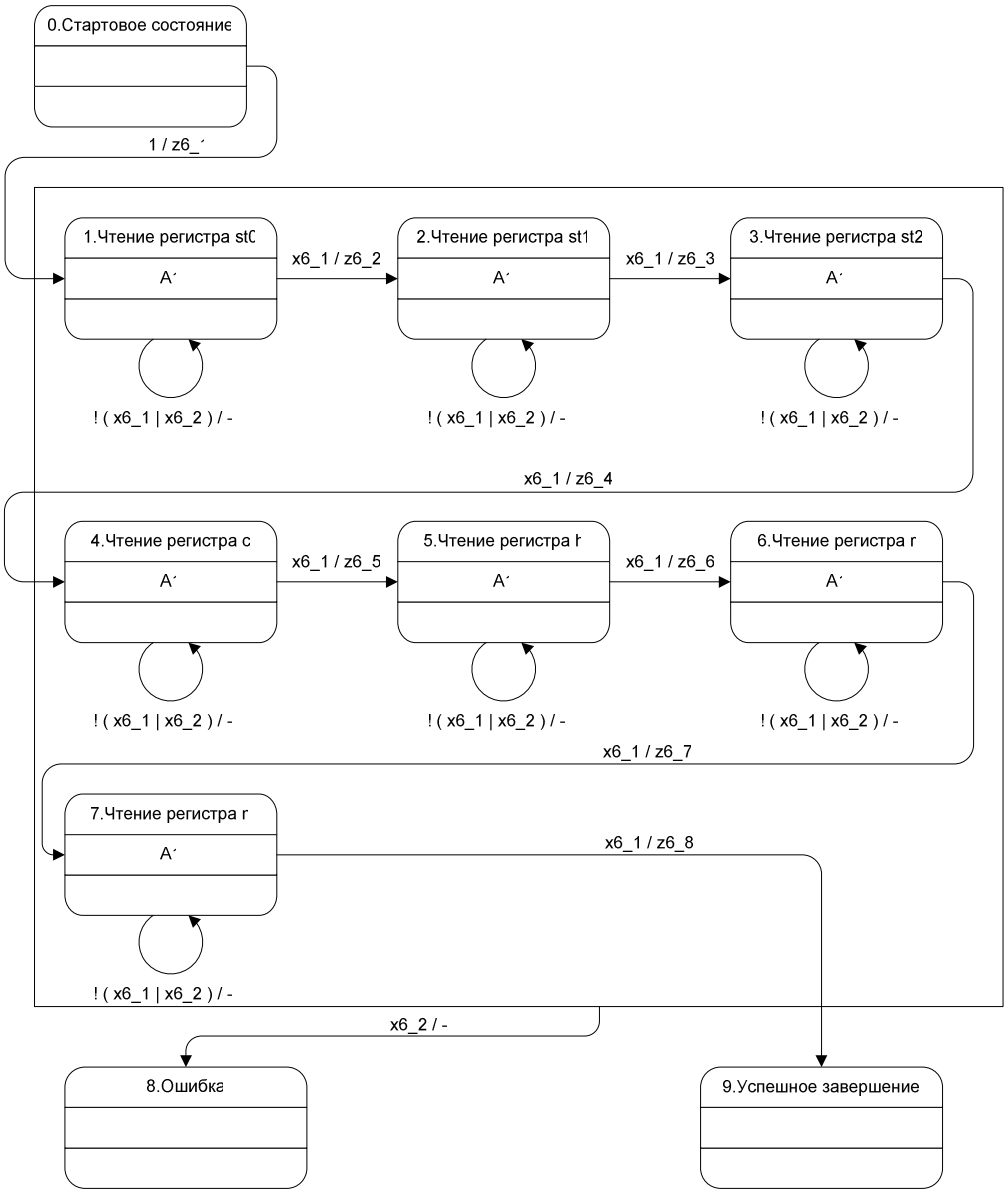

Рис. 20. Граф переходов автомата *A6*

14. Автомат «Чтение сектора» (A7)

Описание автомата

Основной автомат, выполняющий команду чтения сектора. Для его работы необходимо предварительно инициализировать контроллер дисковода (автомат *A2*).

Процесс чтения начинается с выполнения операции рекалибровки и поиска дорожки (состояния «Рекалибровка» и «Поиск»). Для компенсации возможного случайного сбоя при обращении к диску эти операции выполняются три раза. Если ошибки продолжают появляться, то это свидетельствует о нечитабельности диска или серьезной неисправности дисковода.

После инициализации контроллера прямого доступа к памяти, выполняется загрузка команды в состоянии «Отсылка команды чтения» (автомат *A5*). Если эта операция была успешна, то автомат ожидает прерывания в состоянии «Ожидание прерывания». Если контроллер дисковода вовремя выставляет сигнал прерывания, автомат считывает результат при помощи реализации вложенного автомата *A6.*

В случае, если произошла ошибка считывания данных, а не сбой оборудования, то чтение повторяется (состояние «Проверка счетчика повторов»). Это может произойти в случае нестабильной скорости вращения или поврежденном диске. Поэтому операция выполняется три раза. Если дисковод выставил флаг сбоя аппаратуры, или не удалось прочитать сектор с третьего раза, осуществляется переход в состояние «Ошибка». При отсутствии ошибок происходит переход в состояние «Успешное завершение», и буфер контроллера прямого доступа к памяти содержит корректные данные.

Описание состояний автомата

0. Стартовое состояние.
1. Автомат выполняет операцию «Рекалибровка» путем реализации вложенного автомата *A3*.
2. Автомат выполняет операцию «Поиск» путем реализации вложенного автомата *A4.*
3. Автомат отсылает команду «Чтение» с атрибутами путем реализации вложенного автомата *A5*.
4. Автомат ожидает прерывание.
5. Автомат выполняет чтение результата выполнения операции «Чтение» путем вызова автомат *A6*.
6. Проверка результата выполнения операции.
7. Проверка счетчика повторений.
8. Терминальное состояние «Успех».
9. Терминальное состояние «Ошибка».

Схема связей

На рис. 21 приведена схема связей автомата *A7*, описывающая его интерфейс.

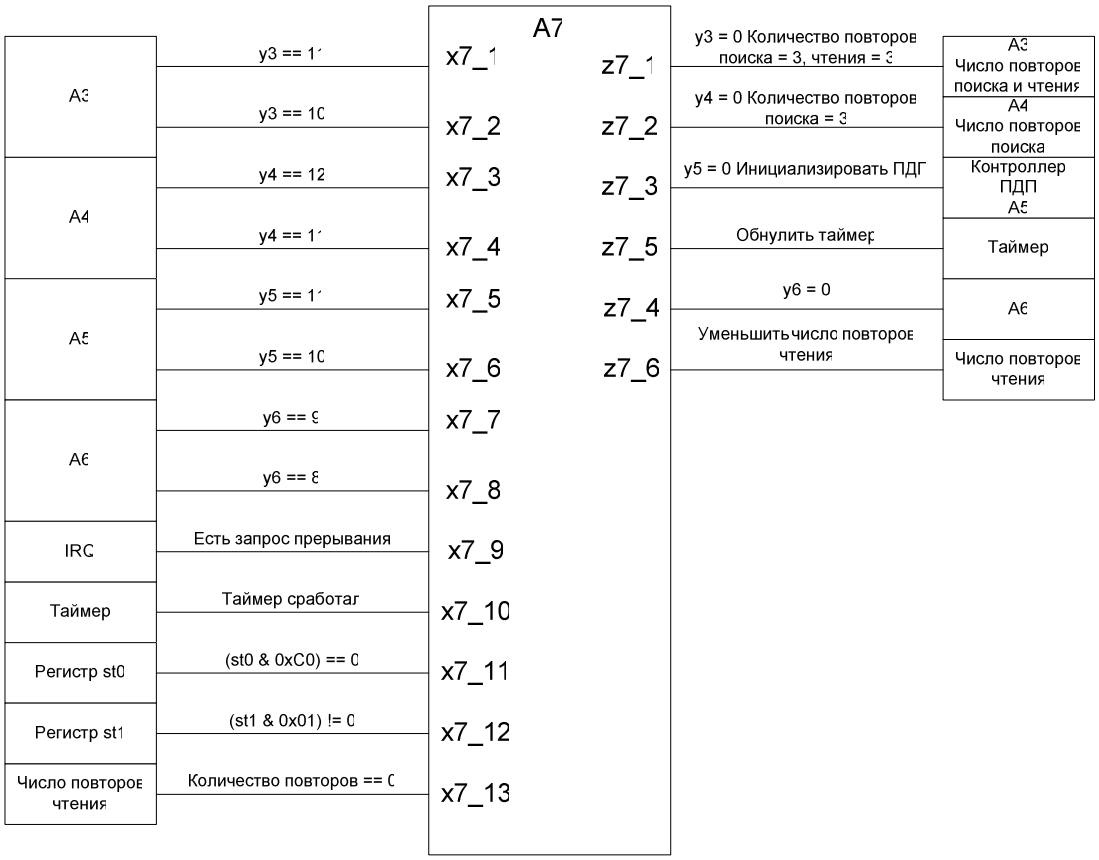

Рис. 21. Схема связей автомата *A7*

Граф переходов.

На рис. 22 приведен граф переходов автомата *A7*.

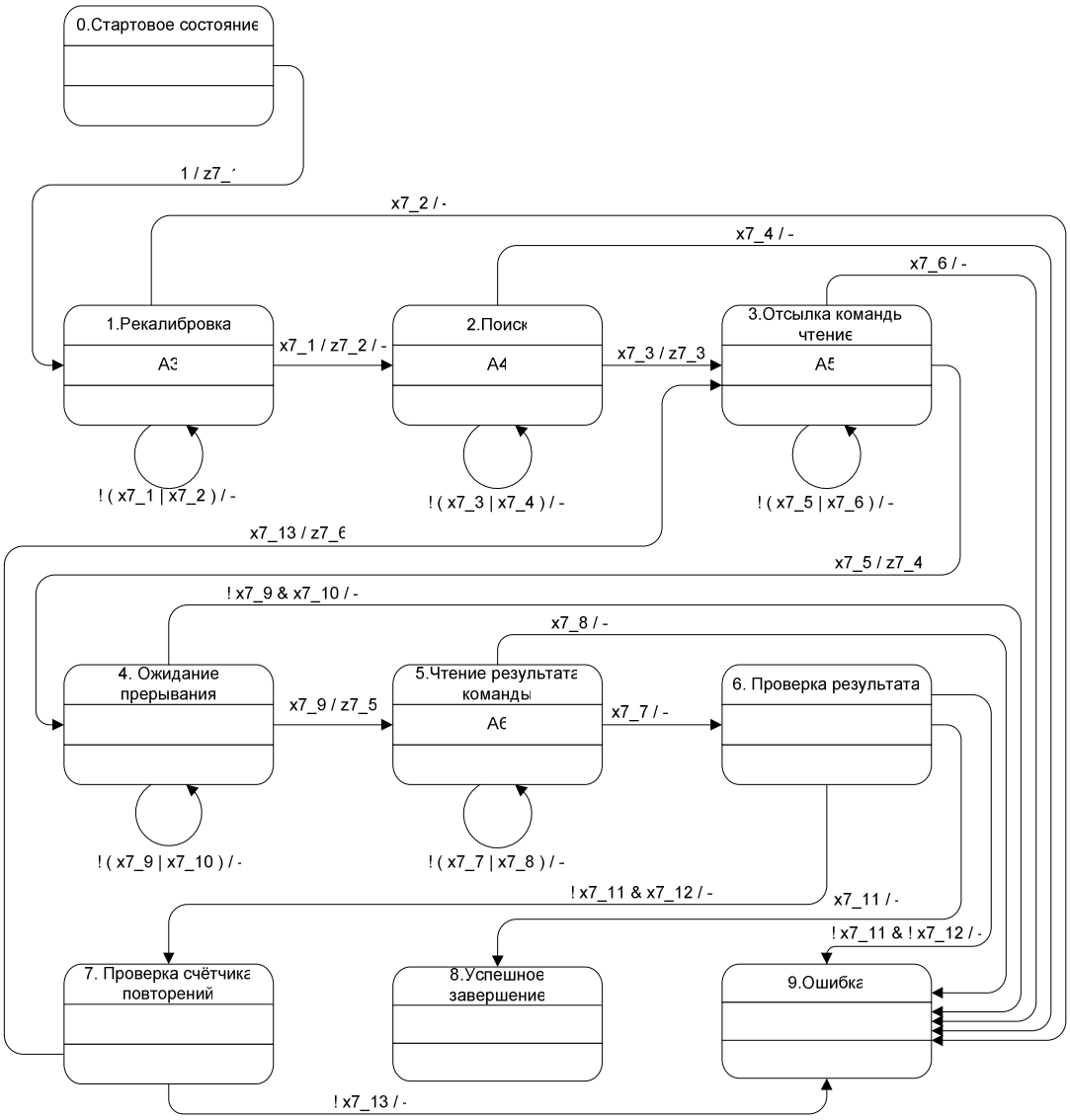

Рис. 22. Граф переходов автомата *A7*

Заключение

1. Разработанная система управления контроллером накопителя на гибких магнитных дисках является по построению весьма надежной.
2. Состояния автоматов соответствуют состояниям контроллера, что позволяет специалистам по микроэлектронике не только проверять правильность работы с контроллером, но и самостоятельно вносить изменения, не обладая большими знаниями в области программирования.
3. С использованием протоколирования (Приложение 4) значительно упрощается процесс отладки, что немаловажно при написании драйверов для современных операционных систем.

Источники

1. *Шалыто А. А.* SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998. http://is.ifmo.ru/books/switch/1
2. *Шалыто А. А., Туккель Н. И.* SWITCH-технология – автоматный подход к созданию программного обеспечения «реактивных» систем //Программирование. 2001. № 5. http://is.ifmo.ru/works/switch/1/
3. *Кун Д.*, Урбан Д., Хамильтон С. *Unix* и не только. Интервью с Кеном Томпсоном. //Открытые системы. 1999. № 4. http://www.osp.ru/os/1999/04/15.htm
4. *Танненбаум Э.* Компьютерные сети. СПб.: Питер, 2002.
5. *Шалыто А. А.* Новая инициатива в программировании. Движение за открытую проектную документацию //Мир ПК. 2003. № 9. http://is.ifmo.ru/works/open_doc/
6. *Танненбаум Э.* Современные операционные системы. СПб.: Питер, 2002.
7. *Кулаков В. В*. Программирование дисковых подсистем. СПб.: Питер, 2002.

Приложение 1. Исходные тексты модулей, реализующих вспомогательные функции

fdd-supp.h

```
#ifndef __FDD_SUPP_H__
#define __FDD_SUPP_H__

#include <bios.h>
#include <dos.h>
#include <stdio.h>
#include <stdlib.h>

#define bool  int
#define true  (-1)
#define false (0)


#define SECTOR_SIZE 512

void setup_fdc_dma(void);
void setup_timer(void);
void setup_interrupt(void);

void flush_timer(void);
void flush_interrupt(void);

bool interrupt_pending(void);

unsigned char inb(unsigned port);
void outb(unsigned port, unsigned char data);


extern char * fdc_dma_buf;
extern unsigned ticks;
extern int fdd_int;
#endif
```

fdd-supp.cpp

```
#include "fdd-supp.h"

char * fdc_dma_buf;
unsigned ticks;
int fdd_int = 0;

void interrupt (far* old_int)(...) = NULL;
void interrupt (far* old_int_timer)(...) = NULL;

void interrupt far fdd_interrupt(...)
```

```
{
     fdd_int++;
     asm {
          mov al, 0x20
          out 0x20, al
     }
}

void interrupt far timer_interrupt(...)
{
     ticks++;
     asm {
          mov al, 0x20
          out 0x20, al
     }
}


void setup_fdc_dma(void)
{
     unsigned int * p_ofs = (unsigned int *) &fdc_dma_buf;
     unsigned int * p_seg = (unsigned int *) &fdc_dma_buf;
     p_seg++;

     unsigned int w_ofs = *p_ofs;
     unsigned int w_seg = *p_seg;

     unsigned long address = ((unsigned long) (w_seg)) * 16 +
(unsigned long) (w_ofs);
     unsigned char * address_bytes = (unsigned char *)&address;

     asm {
          cli
     }

     outb(12, 0x46);
     outb(11, 0x46);
     outb(4, address_bytes[0]);
     outb(4, address_bytes[1]);
     outb(0x81, address_bytes[2]);
     outb(5, 0xFF);
     outb(5, 0x1);
     outb(10, 0x2);

     asm {
          sti
     }
}

void setup_timer(void)
{
     old_int_timer = getvect(8);
     setvect(8, timer_interrupt);
```

```
}

void setup_interrupt(void)
{
     old_int = getvect(14);
     setvect(14, fdd_interrupt);
     fdc_dma_buf = (char *) malloc(SECTOR_SIZE);
}

void flush_timer(void)
{
     setvect(8, old_int_timer);
}

void flush_interrupt(void)
{
     free(fdc_dma_buf);
     outb(0x3F2, 0);
     setvect(14, old_int);
}

bool interrupt_pending(void)
{
     if(fdd_int > 0)
     {
          fdd_int = 0;
          return true;
     }

     return false;
}

unsigned char inb(unsigned port)
{
    unsigned char data = inp(port);

     return data;
}

void outb(unsigned port, unsigned char data)
{
     outp(port, data);
}
```

Приложение 2. Исходные тексты демонстрационной программы

fdd-demo.cpp

```
#include <stdio.h>
#include <conio.h>

#include "fdd-supp.h"
#include "fdd-auto.h"

struct boot_rec
{
    unsigned char jmp[3];
    char          oem_name[8];
    unsigned int  bytes_per_sector;
    unsigned char sectors_per_cluster;
    unsigned int  reserved_sectors;
    unsigned char num_fats;
    unsigned int  root_entries;
    unsigned int  total_sectors;
    unsigned char media_type;
    unsigned int  fat_size;
    unsigned int  sectors_per_track;
    unsigned int  num_heads;
    unsigned long hidden_sectors;
    unsigned long total_sectors_long;
    unsigned char drv_num;
    unsigned char junk1;
    unsigned char extended;
    unsigned long volume_id;
    char          volume_label[11];
    char          fs_type[8];
};

unsigned char * bios_io_buf = NULL;

bool init_fdc(void)
{
     y2 = 0;
     while((y2 != 5) & (y2 != 6))
     {
          A2();
     }


    return (y2 == 5) ? true : false;
}

bool read_sector(int _track, int _head, int _sector)
```

```
{
     y7 = 0;
    track  = _track;
    head   = _head;
    sector = _sector;
     while((y7 != 8) & (y7 != 9))
     {
          A7();
     }


    return (y7 == 8) ? true: false;
}

bool read_sector_bios(int _track, int _head, int _sector)
{
    biosdisk(_DISK_READ, 1, _head, _track, _sector, 1,
bios_io_buf);
    return true;
}

void print_data(void)
{
    boot_rec * dbr = (boot_rec *) fdc_dma_buf;
    boot_rec * bbr = (boot_rec *) bios_io_buf;

    printf("                    %20s%20s\n\r",
        "Direct",
        "BIOS");

    printf("%20s            %02X %02X %02X            %02X %02X
    %02X\n\r",
        "jmp signature",
        dbr->jmp[0],
        dbr->jmp[1],
        dbr->jmp[2],
        bbr->jmp[0],
        bbr->jmp[1],
        bbr->jmp[2]);

    printf("%20s            ", "OEM name:");
    fwrite(dbr->oem_name, 8, 1, stdout);
    printf("            ");
    fwrite(bbr->oem_name, 8, 1, stdout);

    printf("\n\r");
    printf("%20s               %5u               %5u\n\r",
        "Bytes per sector:",
        dbr->bytes_per_sector,
        bbr->bytes_per_sector);

    printf("%20s               %5u               %5u\n\r",
        "Sectors per cluster:",
```

```
        dbr->sectors_per_cluster,
        bbr->sectors_per_cluster);

    printf("%20s               %5u               %5u\n\r",
        "Root entries:",
        dbr->root_entries,
        bbr->root_entries);

    printf("%20s               %5u               %5u\n\r",
        "Total sectors:",
        dbr->total_sectors,
        bbr->total_sectors);

    printf("%20s        %08X           %08X\n\r",
        "Volume id:",
        dbr->volume_id,
        bbr->volume_id);

    printf("%20s             ", "Volume label:");
    fwrite(dbr->volume_label, 11, 1, stdout);
    printf("         ");
    fwrite(bbr->volume_label, 11, 1, stdout);
    printf("\n\r");

    printf("%20s               ", "File system:");
    fwrite(dbr->fs_type, 8, 1, stdout);
    printf("            ");
    fwrite(bbr->fs_type, 8, 1, stdout);
    printf("\n\r");
}

int main(void)
{
    clrscr();
     setup_timer();
     setup_interrupt();

     if(!init_fdc())
     {
          printf("Error initializing fdc\n\r");
          return 1;
     }

     if(!read_sector(0 , 0, 1))
     {
          printf("Error reading sector\n\r");
          return 1;
     }

     flush_interrupt();
     flush_timer();

    bios_io_buf = (unsigned char *) malloc(512);
```

```
     if(!read_sector_bios(0 , 0, 1))
     {
          printf("Error reading sector via bios\n\r");
          return 1;
     }


    print_data();
    getch();
    free(bios_io_buf);
     return 0;
}
```

Приложение 3. Исходные тексты модулей, реализующих автоматы

fdd-auto.h

```
#ifndef __FDD_AUTO_H_
#define __FDD_AUTO_H_

#include "fdd-supp.h"

#define FDD_INT_TIMEOUT 60
#define FDD_IO_TIMEOUT 20
#define FDD_ENGINE_TIMEOUT 20

#define LOG

#ifdef LOG
#define PRINT_LEVEL        { for(int l = 0; l < log_level; l++)\
{ printf("\t"); }}
#define LOG_AUTO_ENTER(a)  { log_stack[log_stack_pos++] = a;\
log_level++; PRINT_LEVEL; printf("| Automa A%i is in state\
%i\n", a, y##a);}
#define LOG_AUTO_EXIT(a)   { log_stack_pos--; log_level--;}
#define LOG_AUTO_CHANGE(a) { PRINT_LEVEL; printf("| Automa A%i\
switched to state %i\n", a, y##a);}
#define LOG_X(a, c)        { int res = c; PRINT_LEVEL; \
printf("> %s invoked by A%i with result %s\n", #a,\
log_stack[log_stack_pos - 1], (res == 0) ? "false" : "true");\
return res;}
#define LOG_Z(a)           { PRINT_LEVEL; printf("< %s invoked\
by A%i\n", #a, log_stack[log_stack_pos - 1]);}
#define LOG_MAX_STACK 16
#endif
```

```
extern unsigned char fdc_io_byte;
extern unsigned char st0;
extern unsigned char st1;
extern unsigned char st2;
extern unsigned char pcn;
extern unsigned char msr;
extern unsigned char c;
extern unsigned char h;
extern unsigned char r;
extern unsigned char n;

extern unsigned char head;
extern unsigned char track;
extern unsigned char sector;

extern int read_retries;
extern int seek_retries;

void A0(void);
void A1(void);
void A2(void);
void A3(void);
void A4(void);
void A5(void);
void A6(void);
void A7(void);


extern int y0;
extern int y1;
extern int y2;
extern int y3;
extern int y4;
extern int y5;
extern int y6;
extern int y7;


bool x0_1(void);
bool x0_2(void);

bool x1_1(void);
bool x1_2(void);

bool x2_1(void);
bool x2_2(void);
bool x2_3(void);

bool x3_1(void);
bool x3_2(void);
bool x3_3(void);
bool x3_4(void);
bool x3_5(void);
```

```
bool x3_6(void);
bool x3_7(void);
bool x3_8(void);
bool x3_9(void);

bool x4_1(void);
bool x4_2(void);
bool x4_3(void);
bool x4_4(void);
bool x4_5(void);
bool x4_6(void);
bool x4_7(void);
bool x4_8(void);
bool x4_9(void);

bool x5_1(void);
bool x5_2(void);

bool x6_1(void);
bool x6_2(void);

bool x7_1(void);
bool x7_2(void);
bool x7_3(void);
bool x7_4(void);
bool x7_5(void);
bool x7_6(void);
bool x7_7(void);
bool x7_8(void);
bool x7_9(void);
bool x7_10(void);
bool x7_11(void);
bool x7_12(void);
bool x7_13(void);


void z0_1(void);
void z0_2(void);
void z0_3(void);

void z1_1(void);
void z1_2(void);
void z1_3(void);

void z2_1(void);
void z2_2(void);
void z2_3(void);
void z2_4(void);

void z3_1(void);
void z3_2(void);
void z3_3(void);
void z3_4(void);
```

```
void z3_5(void);
void z3_6(void);
void z3_7(void);
void z3_8(void);

void z4_1(void);
void z4_2(void);
void z4_3(void);
void z4_4(void);
void z4_5(void);
void z4_6(void);
void z4_7(void);
void z4_8(void);
void z4_9(void);

void z5_1(void);
void z5_2(void);
void z5_3(void);
void z5_4(void);
void z5_5(void);
void z5_6(void);
void z5_7(void);
void z5_8(void);
void z5_9(void);


void z6_1(void);
void z6_2(void);
void z6_3(void);
void z6_4(void);
void z6_5(void);
void z6_6(void);
void z6_7(void);
void z6_8(void);


void z7_1(void);
void z7_2(void);
void z7_3(void);
void z7_4(void);
void z7_5(void);
void z7_6(void);

#ifdef LOG
extern int log_level;
extern int log_stack[LOG_MAX_STACK];
extern int log_stack_pos;
#endif

#endif
```

fdd-a0.cpp

```
#include "fdd-auto.h"

void A0(void)
{
     LOG_AUTO_ENTER(0);

     switch(y0)
     {
          case 0:
          {
               z0_1();
               y0 = 1;
               break;
          }

          case 1:
          {
               z0_2();
               y0 = 2;
               break;
          }

          case 2:
          {
               if(x0_1())
               {
                    y0 = 5;
                    break;
               }

               if(!x0_1())
               {
                    z0_3();
                    y0 = 3;
                    break;
               }

               break;
          }

          case 3:
          {
               break;
          }

          case 4:
          {
               break;
          }

          case 5:
```

```
        {
            if(x0_2())
            {
                y0 = 1;
                break;
            }

            if(!x0_2())
            {
                y0 = 4;
                break;
            }

            break;
        }
    }

    LOG_AUTO_EXIT(0);
}
```

fdd-a1.cpp

```
#include "fdd-auto.h"


void A1(void)
{
     LOG_AUTO_ENTER(1);

     switch(y1)
     {
          case 0:
          {
               z1_1();
               y1 = 1;
               break;
          }

          case 1:
          {
               z1_2();
               y1 = 2;
               break;
          }

          case 2:
          {
               if(x1_1())
               {
                    y1 = 5;
               }

               if(!x1_1())
               {
                    z1_3();
                    y1 = 3;
               }

               break;
          }

          case 3:
          {
               break;
          }

          case 4:
          {
               break;
          }

          case 5:
```

```
        {
            if(x1_2())
            {
                y1 = 1;
                break;
            }

            if(!x1_2())
            {
                y1 = 4;
                break;
            }

            break;
        }
    }

    LOG_AUTO_EXIT(1);
}
```

fdd-a2.cpp

```
#include "fdd-auto.h"

void A2(void)
{

     LOG_AUTO_ENTER(2);

     switch(y2)
     {
          case 0:
          {
               z2_1();
               y2 = 1;
               break;
          }

          case 1:
          {
               if(!x2_3())
               {
                    y2 = 1;
                    break;
               }

               if(x2_3())
               {
                    z2_2();
                    y2 = 2;
                    break;
               }
               break;
          }

          case 2:
          {
               if(!(x2_1() | x2_2()))
               {
                    A0();
                    break;
               }

               if(x2_2())
               {
                    y2 = 6;
                    break;
               }

               if(x2_1())
               {
                    z2_3();
```

```
            y2 = 3;
            break;
        }

        break;
    }

    case 3:
    {
        if(!(x2_1() | x2_2()))
        {
            A0();
            break;
        }

        if(x2_2())
        {
            y2 = 6;
            break;
        }

        if(x2_1())
        {
            z2_4();
            y2 = 4;
            break;
        }

        break;
    }

    case 4:
    {
        if(!(x2_1() | x2_2()))
        {
            A0();
            break;
        }

        if(x2_2())
        {
            y2 = 6;
            break;
        }

        if(x2_1())
        {
            y2 = 5;
            break;
        }

        break;
    }
```

```
            case 5:
            {
                  break;
            }

            case 6:
            {
                  break;
            }
      }

      LOG_AUTO_EXIT(2);

}
```

fdd-a3.cpp

```
#include "fdd-auto.h"


void A3(void)
{

     LOG_AUTO_ENTER(3);

     switch(y3)
     {
          case 0:
          {
               z3_1();
               y3 = 1;
               break;
          }

          case 1:
          {
               if(!(x3_1() | x3_2()))
               {
                    A0();
                    break;
               }

               if(x3_2())
               {
                    y3 = 10;
                    break;
               }

               if(x3_1())
               {
                    z3_2();
                    y3 = 2;
                    break;
               }

               break;
          }

          case 2:
          {
               if(!(x3_1() | x3_2()))
               {
                    A0();
                    break;
               }

               if(x3_2())
```

```
        {
            y3 = 10;
            break;
        }

        if(x3_1())
        {
            z3_3();
            y3 = 3;
            break;
        }

        break;
    }

    case 3:
    {
      bool _x3_6 = x3_6();
        if(!(x3_5() | _x3_6))
        {
            break;
        }

        if(x3_5() & !_x3_6)
        {
            y3 = 10;
            break;
        }

        if(_x3_6)
        {
            z3_4();
            y3 = 4;
            break;
        }

        break;
    }

    case 4:
    {
        if(!(x3_1() | x3_2()))
        {
            A0();
            break;
        }

        if(x3_2())
        {
            y3 = 10;
            break;
        }
```

```
        if(x3_1())
        {
            z3_5();
            y3 = 5;
            break;
        }

        break;
    }

    case 5:
{
        if(!(x3_3() | x3_4()))
        {
            A1();
            break;
        }

        if(x3_4())
        {
            y3 = 10;
            break;
        }

        if(x3_3())
        {
            z3_6();
            y3 = 6;
            break;
        }

        break;
    }

    case 6:
    {
        if(!(x3_3() | x3_4()))
        {
            A1();
            break;
        }

        if(x3_4())
        {
            y3 = 10;
            break;
        }

        if(x3_3())
        {
            z3_7();
            y3 = 7;
            break;
```

```
        }

        break;
    }

    case 7:
    {
        if(!x3_7())
        {
            y3 = 9;
            break;
        }

        if(x3_7())
        {
            z3_7();
            y3 = 8;
            break;
        }

        break;
    }

    case 8:
    {
        if(!x3_8())
        {
            y3 = 9;
            break;
        }

        if(x3_8())
        {
            y3 = 11;
            break;
        }

        break;
    }

    case 9:
    {
        if(x3_9())
        {
            y3 = 10;
            break;
        }

        if(!x3_9())
        {
            z3_8();
            y3 = 0;
            break;
```

```
                }

                break;
            }

            case 10:
            {
            }

            case 11:
            {
            }

        }

        LOG_AUTO_EXIT(3);
}
```

fdd-a4.cpp

```
#include "fdd-auto.h"

void A4(void)
{

     LOG_AUTO_ENTER(4);

     switch(y4)
     {
          case 0:
          {
               z4_1();
               y4 = 1;
               break;
          }

          case 1:
          {
               if(!(x4_1() | x4_2()))
               {
                    A0();
                    break;
               }

               if(x4_2())
               {
                    y4 = 10;
                    break;
               }

               if(x4_1())
               {
                    z4_2();
                    y4 = 2;
                    break;
               }

               break;
          }

          case 2:
          {
               if(!(x4_1() | x4_2()))
               {
                    A0();
                    break;
               }

               if(x4_2())
               {
```

```
            y4 = 10;
            break;
        }

        if(x4_1())
        {
            z4_3();
            y4 = 3;
            break;
        }

        break;
    }

    case 3:
    {
        if(!(x4_1() | x4_2()))
        {
            A0();
            break;
        }

        if(x4_2())
        {
            y4 = 10;
            break;
        }

        if(x4_1())
        {
            z4_4();
            y4 = 4;
            break;
        }

        break;
    }

    case 4:
    {
      bool _x4_6 = x4_6();

        if(!(x4_5() | _x4_6))
        {
            break;
        }

        if(x4_5() & !_x4_6)
        {
            y4 = 11;
            break;
        }
```

```
        if(_x4_6)
        {
            z4_5();
            y4 = 5;
            break;
        }

        break;
    }

    case 5:
    {
        if(!(x4_1() | x4_2()))
        {
            A0();
            break;
        }

        if(x4_2())
        {
            y4 = 11;
            break;
        }

        if(x4_1())
        {
            z4_6();
            y4 = 6;
            break;
        }

        break;
    }

    case 6:
    {
        if(!(x4_3() | x4_4()))
        {
            A1();
            break;
        }

        if(x4_4())
        {
            y4 = 11;
            break;
        }

        if(x4_3())
        {
            z4_7();
            y4 = 7;
            break;
```

```
        }

        break;
    }

    case 7:
    {
        if(!(x4_3() | x4_4()))
        {
            A1();
            break;
        }

        if(x4_4())
        {
            y4 = 11;
            break;
        }

        if(x4_3())
        {
            z4_8();
            y4 = 8;
            break;
        }

        break;
    }

    case 8:
    {
        if(!x4_7())
        {
            y4 = 10;
            break;
        }

        if(x4_7())
        {
            y4 = 9;
            break;
        }

        break;
    }

    case 9:
    {
        if(!x4_8())
        {
            y4 = 10;
            break;
        }
```

```
                if(x4_8())
                {
                    y4 = 12;
                    break;
                }

                break;
            }

            case 10:
            {
                if(x4_9())
                {
                    y4 = 11;
                    break;
                }

                if(!x4_9())
                {
                    z4_8();
                    y4 = 0;
                    break;
                }

                break;
            }

            case 11:
            {
            }

            case 12:
            {
            }

        }

        LOG_AUTO_EXIT(4);
}
```

fdd-a5.cpp

```
#include "fdd-auto.h"

void A5(void)
{

     LOG_AUTO_ENTER(5);

     switch(y5)
     {
          case 0:
          {
               z5_1();
               y5 = 1;
               break;
          }

          case 1:
          {
               if(!(x5_1() | x5_2()))
               {
                    A0();
                    break;
               }

               if(x5_2())
               {
                    y4 = 10;
                    break;
               }

               if(x5_1())
               {
                    z5_2();
                    y5 = 2;
                    break;
               }

               break;
          }

          case 2:
          {
               if(!(x5_1() | x5_2()))
               {
                    A0();
                    break;
               }

               if(x5_2())
               {
```

```
            y4 = 10;
            break;
        }

        if(x5_1())
        {
            z5_3();
            y5 = 3;
            break;
        }

        break;
    }

    case 3:
    {
        if(!(x5_1() | x5_2()))
        {
            A0();
            break;
        }

        if(x5_2())
        {
            y4 = 10;
            break;
        }

        if(x5_1())
        {
            z5_4();
            y5 = 4;
            break;
        }

        break;
    }

    case 4:
    {
        if(!(x5_1() | x5_2()))
        {
            A0();
            break;
        }

        if(x5_2())
        {
            y4 = 10;
            break;
        }

        if(x5_1())
```

```
        {
            z5_5();
            y5 = 5;
            break;
        }

        break;
    }

    case 5:
    {
        if(!(x5_1() | x5_2()))
        {
            A0();
            break;
        }

        if(x5_2())
        {
            y4 = 10;
            break;
        }

        if(x5_1())
        {
            z5_6();
            y5 = 6;
            break;
        }

        break;
    }

    case 6:
    {
        if(!(x5_1() | x5_2()))
        {
            A0();
            break;
        }

        if(x5_2())
        {
            y5 = 10;
            break;
        }

        if(x5_1())
        {
            z5_7();
            y5 = 7;
            break;
        }
```

```
     break;
}

case 7:
{
     if(!(x5_1() | x5_2()))
     {
          A0();
          break;
     }

     if(x5_2())
     {
          y4 = 10;
          break;
     }

     if(x5_1())
     {
          z5_8();
          y5 = 8;
          break;
     }

     break;
}

case 8:
{
     if(!(x5_1() | x5_2()))
     {
          A0();
          break;
     }

     if(x5_2())
     {
          y4 = 10;
          break;
     }

     if(x5_1())
     {
          z5_9();
          y5 = 9;
          break;
     }

     break;
}

case 9:
```

```
            {
                if(!(x5_1() | x5_2()))
                {
                    A0();
                    break;
                }

                if(x5_2())
                {
                    y5 = 10;
                    break;
                }

                if(x5_1())
                {
                    z5_2();
                    y5 = 11;
                    break;
                }

                break;
            }


            case 10:
            {
                break;
            }

            case 11:
            {
                break;
            }
        }

        LOG_AUTO_EXIT(5);

}
```

fdd-a6.cpp

```
#include "fdd-auto.h"

void A6(void)
{

     LOG_AUTO_ENTER(6);

     switch(y6)
     {
          case 0:
          {
               z6_1();
               y6 = 1;
               break;
          }

          case 1:
          {
               if(!(x6_1() | x6_2()))
               {
                    A1();
                    break;
               }

               if(x6_2())
               {
                    y6 = 8;
                    break;
               }

               if(x6_1())
               {
                    z6_2();
                    y6 = 2;
                    break;
               }

               break;
          }

          case 2:
          {
               if(!(x6_1() | x6_2()))
               {
                    A1();
                    break;
               }

               if(x6_2())
               {
```

```
            y6 = 8;
            break;
        }

        if(x6_1())
        {
            z6_3();
            y6 = 3;
            break;
        }

        break;
    }

    case 3:
    {
        if(!(x6_1() | x6_2()))
        {
            A1();
            break;
        }

        if(x6_2())
        {
            y6 = 8;
            break;
        }

        if(x6_1())
        {
            z6_4();
            y6 = 4;
            break;
        }

        break;
    }

    case 4:
    {
        if(!(x6_1() | x6_2()))
        {
            A1();
            break;
        }

        if(x6_2())
        {
            y6 = 8;
            break;
        }

        if(x6_1())
```

```
        {
            z6_5();
            y6 = 5;
            break;
        }

        break;
    }

    case 5:
    {
        if(!(x6_1() | x6_2()))
        {
            A1();
            break;
        }

        if(x6_2())
        {
            y6 = 8;
            break;
        }

        if(x6_1())
        {
            z6_6();
            y6 = 6;
            break;
        }

        break;
    }

    case 6:
    {
        if(!(x6_1() | x6_2()))
        {
            A1();
            break;
        }

        if(x6_2())
        {
            y6 = 8;
            break;
        }

        if(x6_1())
        {
            z6_7();
            y6 = 7;
            break;
        }
```

```
                break;
            }

            case 7:
            {
                if(!(x6_1() | x6_2()))
                {
                    A1();
                    break;
                }

                if(x6_2())
                {
                    y6 = 8;
                    break;
                }

                if(x6_1())
                {
                    z6_8();
                    y6 = 9;
                    break;
                }

                break;
            }

            case 8:
            {
                break;
            }

            case 9:
            {
                break;
            }
        }

        LOG_AUTO_EXIT(6);

}
```

fdd-a7.cpp

```
#include "fdd-auto.h"

void A7(void)
{

     LOG_AUTO_ENTER(7);

     switch(y7)
     {
          case 0:
          {
               z7_1();
               y7 = 1;
               break;
          }

          case 1:
          {
               if(!(x7_1() | x7_2()))
               {
                    A3();
                    break;
               }

               if(x7_2())
               {
                    y7 = 9;
                    break;
               }

               if(x7_1())
               {
                    z7_2();
                    y7 = 2;
                    break;
               }

               break;
          }

          case 2:
          {
               if(!(x7_3() | x7_4()))
               {
                    A4();
                    break;
               }

               if(x7_4())
               {
```

```
            y7 = 9;
            break;
        }

        if(x7_3())
        {
            z7_3();
            y7 = 3;
            break;
        }

        break;
    }

    case 3:
    {
        if(!(x7_5() | x7_6()))
        {
            A5();
            break;
        }

        if(x7_5())
        {
            y7 = 4;
            break;
        }

        if(x7_5())
        {
            z7_4();
            y7 = 2;
            break;
        }

        break;
    }

    case 4:
    {
      bool _x7_9 = x7_9();
        if(!(_x7_9 | x7_10()))
        {
            break;
        }

        if(x7_10() & !_x7_9)
        {
            y7 = 9;
            break;
        }

        if(_x7_9)
```

```
        {
            z7_5();
            y7 = 5;
            break;
        }

        break;
    }

    case 5:
    {
        if(!(x7_7() | x7_8()))
        {
            A6();
            break;
        }

        if(x7_7())
        {
            y7 = 6;
            break;
        }

        if(x7_8())
        {
            y7 = 9;
            break;
        }

        break;
    }

    case 6:
    {
        if(!(x7_11() | x7_12()))
        {
            y7 = 9;
            break;
        }

        if(x7_11())
        {
            y7 = 8;
            break;
        }

        if((!x7_11()) & x7_12())
        {
            y7 = 7;
            break;
        }

        break;
```

```
            }

            case 7:
            {
                 if(!x7_13())
                 {
                      y7 = 9;
                      break;
                 }

                 if(x7_13())
                 {
                      z7_6();
                      y7 = 3;
                      break;
                 }

                 break;
            }


            case 8:
            {
                 break;
            }

            case 9:
            {
                 break;
            }
       }

       LOG_AUTO_EXIT(7);

}
```

fdd-auto.cpp

```
#include "fdd-auto.h"

#ifdef LOG
int log_level = 0;
int log_stack[LOG_MAX_STACK];
int log_stack_pos = 0;
#endif

unsigned char fdc_io_byte;
unsigned char st0;
unsigned char st1;
unsigned char st2;
unsigned char pcn;
unsigned char msr;
```

```
unsigned char c;
unsigned char h;
unsigned char r;
unsigned char n;

int read_retries;
int seek_retries;

unsigned char head;
unsigned char track;
unsigned char sector;

int y0 = 0;
int y1 = 0;
int y2 = 0;
int y3 = 0;
int y4 = 0;
int y5 = 0;
int y6 = 0;
int y7 = 0;
```

fdd-x.cpp

```
#include "fdd-auto.h"

bool x0_1(void)
{
     LOG_X(x0_1, ((msr & 0xC0) != 0x80));
}

bool x0_2(void)
{
     LOG_X(x0_2, (ticks < FDD_IO_TIMEOUT));
}


bool x1_1(void)
{
     LOG_X(x1_1, ((msr & 0xC0) != 0xC0));
}

bool x1_2(void)
{
     LOG_X(x1_2,  (ticks < FDD_IO_TIMEOUT));
}


bool x2_1(void)
{
     LOG_X(x2_1,  (y0 == 3));
}
```

```
bool x2_2(void)
{
     LOG_X(x2_2,  (y0 == 4));
}

bool x2_3(void)
{
     LOG_X(x2_3,  (ticks > FDD_ENGINE_TIMEOUT));
}

bool x3_1(void)
{
     LOG_X(x3_1,  (y0 == 3));
}

bool x3_2(void)
{
     LOG_X(x3_2,  (y0 == 4));
}

bool x3_3(void)
{
     LOG_X(x3_3,  (y1 == 3));
}

bool x3_4(void)
{
     LOG_X(x3_4,  (y1 == 4));
}

bool x3_5(void)
{
     LOG_X(x3_5,  (ticks > FDD_INT_TIMEOUT));
}

bool x3_6(void)
{
     LOG_X(x3_6,  (interrupt_pending() == true));
}

bool x3_7(void)
{
     LOG_X(x3_7,  ((st0 & 0xC0) == 0));
}

bool x3_8(void)
{
     LOG_X(x3_8,  (pcn == 0));
}

bool x3_9(void)
{
     LOG_X(x3_9,  (seek_retries == 0));
```

```
}


bool x4_1(void)
{
     LOG_X(x4_1,  (y0 == 3));
}

bool x4_2(void)
{
     LOG_X(x4_2,  (y0 == 4));
}

bool x4_3(void)
{
     LOG_X(x4_3,  (y1 == 3));
}

bool x4_4(void)
{
     LOG_X(x4_4,  (y1 == 4));
}

bool x4_5(void)
{
     LOG_X(x4_5,  (ticks > FDD_INT_TIMEOUT));
}

bool x4_6(void)
{
     LOG_X(x4_6,  (interrupt_pending() == true));
}

bool x4_7(void)
{
     LOG_X(x4_7,  ((st0 & 0xC0) == 0));
}

bool x4_8(void)
{
     LOG_X(x4_8,  (pcn == 0));
}

bool x4_9(void)
{
     LOG_X(x4_9,  (seek_retries == 0));
}


bool x5_1(void)
{
     LOG_X(x5_1,  (y0 == 3));
}
```

```
bool x5_2(void)
{
     LOG_X(x5_2,  (y0 == 4));
}


bool x6_1(void)
{
     LOG_X(x6_1,  (y1 == 3));
}

bool x6_2(void)
{
     LOG_X(x6_2,  (y1 == 4));
}


bool x7_1(void)
{
     LOG_X(x7_1,  (y3 == 11));
}

bool x7_2(void)
{
     LOG_X(x7_2,  (y3 == 10));
}

bool x7_3(void)
{
     LOG_X(x7_3,  (y4 == 12));
}

bool x7_4(void)
{
     LOG_X(x7_4,  (y4 == 11));
}

bool x7_5(void)
{
     LOG_X(x7_5,  (y5 == 11));
}

bool x7_6(void)
{
     LOG_X(x7_6,  (y5 == 10));
}

bool x7_7(void)
{
     LOG_X(x7_7,  (y6 == 9));
}
```

```
bool x7_8(void)
{
     LOG_X(x7_8,  (y6 == 8));
}

bool x7_9(void)
{
     LOG_X(x7_9,  interrupt_pending());
}

bool x7_10(void)
{
     LOG_X(x7_10,  (ticks > FDD_INT_TIMEOUT));
}

bool x7_11(void)
{
     LOG_X(x7_11,  ((st0 & 0xC0) == 0));
}

bool x7_12(void)
{
     LOG_X(x7_12,  ((st1 & 0x01) != 0));
}

bool x7_13(void)
{
     LOG_X(x7_13,  (read_retries == 0));
}
```

fdd-z.cpp

```
#include "fdd-auto.h"

#define R_DOR   0x3F2
#define R_CCR   0x3F7
#define R_MSR   0x3F4
#define R_FIFO  0x3F5

void z0_1(void)
{
     LOG_Z(z0_1);
     ticks = 0;
}

void z0_2(void)
{
     LOG_Z(z0_2);
     msr = inb(R_MSR);
}

void z0_3(void)
```

```
{
     LOG_Z(z0_3);
     outb(R_FIFO, fdc_io_byte);
}


void z1_1(void)
{
     LOG_Z(z1_1);
     ticks = 0;
}

void z1_2(void)
{
     LOG_Z(z1_2);
     msr = inb(R_MSR);
}

void z1_3(void)
{
     LOG_Z(z1_3);
     fdc_io_byte = inb(R_FIFO);
}


void z2_1(void)
{
     LOG_Z(z2_1);
     outb(R_DOR, 0);
     outb(R_DOR, 0x1C);
     outb(R_CCR, 0x00);
     ticks = 0;
}

void z2_2(void)
{
     LOG_Z(z2_2);
     y0 = 0;
     fdc_io_byte = 0x03;
}

void z2_3(void)
{
     LOG_Z(z2_3);
     y0 = 0;
     fdc_io_byte = 0x0F;
}

void z2_4(void)
{
     LOG_Z(z2_4);
     y0 = 0;
     fdc_io_byte = 0x00;
```

```
}


void z3_1(void)
{
     LOG_Z(z3_1);
    fdd_int = 0;
     y0 = 0;
     fdc_io_byte = 0x07;
}

void z3_2(void)
{
     LOG_Z(z3_2);
     y0 = 0;
     fdc_io_byte = 0x00;
}

void z3_3(void)
{
     LOG_Z(z3_3);
     ticks = 0;
}

void z3_4(void)
{
     LOG_Z(z3_4);
     y0 = 0;
     fdc_io_byte = 0x08;
}

void z3_5(void)
{
     LOG_Z(z3_5);
     y1 = 0;
}

void z3_6(void)
{
     LOG_Z(z3_6);
     st0 = fdc_io_byte;
     y1 = 0;
}

void z3_7(void)
{
     LOG_Z(z3_7);
     pcn = fdc_io_byte;
}

void z3_8(void)
{
     LOG_Z(z3_8);
```

```
     seek_retries--;
}


void z4_1(void)
{
     LOG_Z(z4_1);
    fdd_int = 0;
     y0 = 0;
     fdc_io_byte = 0x0F;
}

void z4_2(void)
{
     LOG_Z(z4_2);
     y0 = 0;
     fdc_io_byte = 0x04 * head;
}

void z4_3(void)
{
     LOG_Z(z4_3);
     y0 = 0;
     fdc_io_byte = track;
}

void z4_4(void)
{
     LOG_Z(z4_4);
     ticks = 0;
}

void z4_5(void)
{
     LOG_Z(z4_5);
     y0 = 0;
     fdc_io_byte = 0x08;
}

void z4_6(void)
{
     LOG_Z(z4_6);
     y1 = 0;

}

void z4_7(void)
{
     LOG_Z(z4_7);
     st0 = fdc_io_byte;
     y1 = 0;
}
```

```
void z4_8(void)
{
     LOG_Z(z4_8);
     pcn = fdc_io_byte;
}

void z4_9(void)
{
     LOG_Z(z4_9);
     seek_retries--;
}


void z5_1(void)
{
     LOG_Z(z5_1);
     y0 = 0;
     fdc_io_byte = 0xE6;
}

void z5_2(void)
{
     LOG_Z(z5_2);
     y0 = 0;
     fdc_io_byte = 0x04 * head;
}

void z5_3(void)
{
     LOG_Z(z5_3);
     y0 = 0;
     fdc_io_byte = track;
}

void z5_4(void)
{
     LOG_Z(z5_4);
     y0 = 0;
     fdc_io_byte = head;
}

void z5_5(void)
{
     LOG_Z(z5_5);
     y0 = 0;
     fdc_io_byte = sector;
}

void z5_6(void)
{
     LOG_Z(z5_6);
     y0 = 0;
     fdc_io_byte = 2;
```

```
}

void z5_7(void)
{
     LOG_Z(z5_7);
     y0 = 0;
     fdc_io_byte = 19;
}

void z5_8(void)
{
     LOG_Z(z5_8);
     y0 = 0;
     fdc_io_byte = 0x1B;
}

void z5_9(void)
{
     LOG_Z(z5_9);
     y0 = 0;
     fdc_io_byte = 0xFF;
}



void z6_1(void)
{
     LOG_Z(z6_1);
     y1 = 0;
}

void z6_2(void)
{
     LOG_Z(z6_2);
     y1 = 0;
     st0 = fdc_io_byte;
}

void z6_3(void)
{
     LOG_Z(z6_3);
     y1 = 0;
     st1 = fdc_io_byte;
}

void z6_4(void)
{
     LOG_Z(z6_4);
     y1 = 0;
     st2 = fdc_io_byte;
}

void z6_5(void)
```

```
{
     LOG_Z(z6_5);
     y1 = 0;
     c = fdc_io_byte;
}

void z6_6(void)
{
     LOG_Z(z6_6);
     y1 = 0;
     h = fdc_io_byte;
}

void z6_7(void)
{
     LOG_Z(z6_7);
     y1 = 0;
     r = fdc_io_byte;
}

void z6_8(void)
{
     LOG_Z(z6_8);
     n = fdc_io_byte;
}



void z7_1(void)
{
     LOG_Z(z7_1);
     y3 = 0;
     seek_retries = 3;
     read_retries = 3;
}

void z7_2(void)
{
     LOG_Z(z7_2);
     y4 = 0;
     seek_retries = 3;
}

void z7_3(void)
{
     LOG_Z(z7_3);
    fdd_int = 0;
     setup_fdc_dma();
     y5 = 0;
}

void z7_4(void)
{
```

```
     LOG_Z(z7_4);
     ticks = 0;
}

void z7_5(void)
{
     LOG_Z(z7_5);
     y6 = 0;
}

void z7_6(void)
{
     LOG_Z(z7_6);
     read_retries--;
}
```

Приложение 4. Фрагмент протокола работы

```
| Automa A7 is in state 2
> x7_3 invoked by A7 with result false
> x7_4 invoked by A7 with result false
     | Automa A4 is in state 1
     > x4_1 invoked by A4 with result true
     > x4_2 invoked by A4 with result false
     > x4_2 invoked by A4 with result false
     > x4_1 invoked by A4 with result true
     < z4_2 invoked by A4
| Automa A7 is in state 2
> x7_3 invoked by A7 with result false
> x7_4 invoked by A7 with result false
     | Automa A4 is in state 2
     > x4_1 invoked by A4 with result false
     > x4_2 invoked by A4 with result false
          | Automa A0 is in state 0
          < z0_1 invoked by A0
| Automa A7 is in state 2
> x7_3 invoked by A7 with result false
> x7_4 invoked by A7 with result false
     | Automa A4 is in state 2
     > x4_1 invoked by A4 with result false
     > x4_2 invoked by A4 with result false
          | Automa A0 is in state 1
          < z0_2 invoked by A0
| Automa A7 is in state 2
> x7_3 invoked by A7 with result false
> x7_4 invoked by A7 with result false
     | Automa A4 is in state 2
     > x4_1 invoked by A4 with result false
     > x4_2 invoked by A4 with result false
          | Automa A0 is in state 2
          > x0_1 invoked by A0 with result false
          > x0_1 invoked by A0 with result false
```

```
          < z0_3 invoked by A0
| Automa A7 is in state 2
> x7_3 invoked by A7 with result false
> x7_4 invoked by A7 with result false
     | Automa A4 is in state 2
     > x4_1 invoked by A4 with result true
     > x4_2 invoked by A4 with result false
     > x4_2 invoked by A4 with result false
     > x4_1 invoked by A4 with result true
     < z4_3 invoked by A4
| Automa A7 is in state 2
> x7_3 invoked by A7 with result false
> x7_4 invoked by A7 with result false
     | Automa A4 is in state 3
     > x4_1 invoked by A4 with result false
     > x4_2 invoked by A4 with result false
          | Automa A0 is in state 0
          < z0_1 invoked by A0
| Automa A7 is in state 2
> x7_3 invoked by A7 with result false
> x7_4 invoked by A7 with result false
     | Automa A4 is in state 3
     > x4_1 invoked by A4 with result false
     > x4_2 invoked by A4 with result false
          | Automa A0 is in state 1
          < z0_2 invoked by A0
| Automa A7 is in state 2
> x7_3 invoked by A7 with result false
> x7_4 invoked by A7 with result false
     | Automa A4 is in state 3
     > x4_1 invoked by A4 with result false
     > x4_2 invoked by A4 with result false
          | Automa A0 is in state 2
          > x0_1 invoked by A0 with result false
          > x0_1 invoked by A0 with result false
          < z0_3 invoked by A0
| Automa A7 is in state 2
> x7_3 invoked by A7 with result false
> x7_4 invoked by A7 with result false
     | Automa A4 is in state 3
     > x4_1 invoked by A4 with result true
     > x4_2 invoked by A4 with result false
     > x4_2 invoked by A4 with result false
     > x4_1 invoked by A4 with result true
     < z4_4 invoked by A4
| Automa A7 is in state 2
> x7_3 invoked by A7 with result false
> x7_4 invoked by A7 with result false
     | Automa A4 is in state 4
     > x4_6 invoked by A4 with result true
     > x4_5 invoked by A4 with result false
     > x4_5 invoked by A4 with result false
     < z4_5 invoked by A4
```

```
| Automa A7 is in state 2
> x7_3 invoked by A7 with result false
> x7_4 invoked by A7 with result false
     | Automa A4 is in state 5
     > x4_1 invoked by A4 with result false
     > x4_2 invoked by A4 with result false
          | Automa A0 is in state 0
          < z0_1 invoked by A0
| Automa A7 is in state 2
> x7_3 invoked by A7 with result false
> x7_4 invoked by A7 with result false
     | Automa A4 is in state 5
     > x4_1 invoked by A4 with result false
     > x4_2 invoked by A4 with result false
          | Automa A0 is in state 1
          < z0_2 invoked by A0
| Automa A7 is in state 2
> x7_3 invoked by A7 with result false
> x7_4 invoked by A7 with result false
     | Automa A4 is in state 5
     > x4_1 invoked by A4 with result false
     > x4_2 invoked by A4 with result false
          | Automa A0 is in state 2
          > x0_1 invoked by A0 with result false
          > x0_1 invoked by A0 with result false
          < z0_3 invoked by A0
| Automa A7 is in state 2
> x7_3 invoked by A7 with result false
> x7_4 invoked by A7 with result false
     | Automa A4 is in state 5
     > x4_1 invoked by A4 with result true
     > x4_2 invoked by A4 with result false
     > x4_2 invoked by A4 with result false
     > x4_1 invoked by A4 with result true
     < z4_6 invoked by A4
| Automa A7 is in state 2
> x7_3 invoked by A7 with result false
> x7_4 invoked by A7 with result false
     | Automa A4 is in state 6
     > x4_3 invoked by A4 with result false
     > x4_4 invoked by A4 with result false
          | Automa A1 is in state 0
          < z1_1 invoked by A1
| Automa A7 is in state 2
> x7_3 invoked by A7 with result false
> x7_4 invoked by A7 with result false
     | Automa A4 is in state 6
     > x4_3 invoked by A4 with result false
     > x4_4 invoked by A4 with result false
          | Automa A1 is in state 1
          < z1_2 invoked by A1
| Automa A7 is in state 2
> x7_3 invoked by A7 with result false
```

```
> x7_4 invoked by A7 with result false
     | Automa A4 is in state 6
     > x4_3 invoked by A4 with result false
     > x4_4 invoked by A4 with result false
          | Automa A1 is in state 2
          > x1_1 invoked by A1 with result false
          > x1_1 invoked by A1 with result false
          < z1_3 invoked by A1
| Automa A7 is in state 2
> x7_3 invoked by A7 with result false
> x7_4 invoked by A7 with result false
     | Automa A4 is in state 6
     > x4_3 invoked by A4 with result true
     > x4_4 invoked by A4 with result false
     > x4_4 invoked by A4 with result false
     > x4_3 invoked by A4 with result true
     < z4_7 invoked by A4
| Automa A7 is in state 2
> x7_3 invoked by A7 with result false
> x7_4 invoked by A7 with result false
     | Automa A4 is in state 7
     > x4_3 invoked by A4 with result false
     > x4_4 invoked by A4 with result false
          | Automa A1 is in state 0
          < z1_1 invoked by A1
| Automa A7 is in state 2
> x7_3 invoked by A7 with result false
> x7_4 invoked by A7 with result false
     | Automa A4 is in state 7
     > x4_3 invoked by A4 with result false
     > x4_4 invoked by A4 with result false
          | Automa A1 is in state 1
          < z1_2 invoked by A1
| Automa A7 is in state 2
> x7_3 invoked by A7 with result false
> x7_4 invoked by A7 with result false
     | Automa A4 is in state 7
     > x4_3 invoked by A4 with result false
     > x4_4 invoked by A4 with result false
          | Automa A1 is in state 2
          > x1_1 invoked by A1 with result false
          > x1_1 invoked by A1 with result false
          < z1_3 invoked by A1
| Automa A7 is in state 2
> x7_3 invoked by A7 with result false
> x7_4 invoked by A7 with result false
     | Automa A4 is in state 7
     > x4_3 invoked by A4 with result true
     > x4_4 invoked by A4 with result false
     > x4_4 invoked by A4 with result false
     > x4_3 invoked by A4 with result true
     < z4_8 invoked by A4
| Automa A7 is in state 2
```

```
> x7_3 invoked by A7 with result false
> x7_4 invoked by A7 with result false
     | Automa A4 is in state 8
     > x4_7 invoked by A4 with result true
     > x4_7 invoked by A4 with result true
| Automa A7 is in state 2
> x7_3 invoked by A7 with result false
> x7_4 invoked by A7 with result false
     | Automa A4 is in state 9
     > x4_8 invoked by A4 with result true
     > x4_8 invoked by A4 with result true
| Automa A7 is in state 2
> x7_3 invoked by A7 with result true
> x7_4 invoked by A7 with result false
> x7_4 invoked by A7 with result false
> x7_3 invoked by A7 with result true
< z7_3 invoked by A7
| Automa A7 is in state 3

/*............*/

| Automa A7 is in state 4
> x7_9 invoked by A7 with result false
> x7_10 invoked by A7 with result false
| Automa A7 is in state 4
> x7_9 invoked by A7 with result true
> x7_10 invoked by A7 with result false
> x7_10 invoked by A7 with result false
< z7_5 invoked by A7
| Automa A7 is in state 5

/*............*/

| Automa A7 is in state 6
> x7_11 invoked by A7 with result true
> x7_12 invoked by A7 with result false
> x7_11 invoked by A7 with result true
```