МИНИСТЕРСТВО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

УРАЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

кафедра молекулярной физики

ПРОГРАММНЫЙ МОДУЛЬ УПРАВЛЕНИЯ МАСС-СПЕКТРОМЕТРОМ *МИ-1201 АГМ*. ВЕРСИЯ 2

ИНСТРУКЦИЯ ПО ЭКСПЛУАТАЦИИ

070500 Т00000 001 ИЭ

Разработчик _____ Александров О.Е.
к. ф.-м. н., программист 13 __.__.01
разряда

Екатеринбург 2003

РЕФЕРАТ

Александров О.Е. *ПРОГРАММНЫЙ МОДУЛЬ УПРАВЛЕНИЯ МАСС-СПЕКТРОМЕТРОМ МИ-1201 АГМ. ВЕРСИЯ 2*: Инструкция по эксплуатации. Екатеринбург: УГТУ, 2003. 54 с.

Библиогр. 0 назв. Рис. 2. Табл. 17. Прил. 2.

Издание третье, исправленное и дополненное.

ПРОГРАММА; МАСС-СПЕКТРОМЕТР; МИ 1201-АГМ.

Описаны изменения внесенные во вторую версию модуля.

Описывает структуру и принципы построения программного модуля управления масс-спектрометром МИ 1201-АГМ. Дает рекомендации по его использованию для написания пользовательских программ.

Язык программы — Borland Pascal 7.0. Модуль модифицирован, с сохранением DOS-совместимости на уровне исходного кода, и протестирован для работы в среде Borland Delphi 3÷5 в операционной системе Microsoft Windows 9x и Microsoft Windows NT.

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ СИМВОЛОВ, ЕДИНИЦ И ТЕРМИНОВ

[**значение**]	– в квадратных скобках приводится значение величины (параметра) по умолчанию.
сИмя	– имя константы.
exИмя	– имя исполняемого метода, т.е. метода осуществляющего непосредственное взаимодействие с оборудованием масс-спектрометра. Прочие методы только устанавливают значения параметров взаимодействия.
fИмя	– имя флага.
tИмя	– имя типа данных.
VCL	– Visual Component Library. Библиотека функций ввода/вывода Delphi для оконного интерфейса MS Windows.
Базовый контроллер	– программный модуль контроллера МИ 1201-АГМ для DOS.
БПГИ	– блок питания газового источника.
ВЭУ	– вторичный электронный умножитель.
Контроллер	– программный модуль управления аппаратной частью комплекса МИ 1201-АГМ.
мВ	– милливольт, $1мВ = 10^{-3}В$.
мкВ	– микровольт, $1мкВ = 10^{-6}В$.
Модуль	– см. **Модуль МИ 1201-АГМ**.
Модуль МИ 1201-АГМ	– совокупность программ, реализующих модуль управления масс-спектрометром МИ 1201-АГМ.
ОИК	– ошибка исполнения команды.
ОС	– операционная система.
ПНЧ	– преобразователь напряжение-частота.
ЭМУ	– электрометрический усилитель.

ВВЕДЕНИЕ

Инструкция описывает структуру и принципы работы программного модуля управления масс-спектрометром МИ 1201-АГМ (далее *модуль МИ 1201-АГМ* или *модуль*).

Приведено описание второй версии модуля. Основные изменения коснулись контроллера электромагнита, контроллера ПНЧ и основного контроллера МИ-1201 АГМ.

Модуль написан на языке программирования Borland Pascal и реализует функции управления и доступа к данным измерений в пределах, предусмотренных конструкцией МИ 1201-АГМ. Модуль дополнен изменениями и протестирован для работы в среде Delphi 5.

Совместимость с Borland Pascal 7.0 для реального и защищенного режима DOS сохранена на уровне исходного кода[1)].

Модуль не содержит обращений к функциям ввода/вывода на дисплей и/или в файлы[2)], и предназначен для использования в качестве базы для разработки программ управления измерениями в среде Borland Pascal 7.0 для реального и защищенного режима DOS или в среде Delphi 3÷5 для Windows 9x и Windows NT.

1. ОБЩАЯ СТРУКТУРА МОДУЛЯ

1.1. Цели создания модуля

Модуль МИ 1201-АГМ создан для обеспечения простого и надежного управления аппаратным комплексом МИ 1201-АГМ на уровне вызова процедур. Вызовы процедур, в идеале, соответствуют простейшим операциям человека в процессе получения масс-спектра.

Модуль МИ 1201-АГМ облегчает и упрощает создание программ управления и обработки данных измерений. Программист имеет возможность сосре-

---

[1]) Для использование в DOS необходима перекомпиляция с установленной {$DEFINE Seg16} опцией компилятора Borland Pascal 7.0.

[2]) ИСКЛЮЧЕНИЕ: в контроллере МИ1201 есть метод сохранения состояния контроллера в файл.

доточится на обработке данных, не задумываясь о тонкостях аппаратной и программной реализации управления масс-спектрометром.

Например, операция «*изменить текущую массу* (*значение магнитной индукции*)» соответствует вызову процедуры *JumpToMass* контроллера. Аргументом процедуры служит желаемое значение массы в единицах *аем*. Результатом вызова в программе пользователя процедуры *JumpToMass(100)*, будет изменение магнитного поля на значение соответствующее массе 100 *аем*.

Другой пример, операция «*прочитать значение ионного тока с ЭМУ 1*» соответствует вызову процедуры *SignalChannelSet(PNC1)*, устанавливающего канал измерения, и вызову функции *X*: = *exSignal*, возвращающей значение счетчика для ЭМУ 1. Результатом такого вызова в программе пользователя будет запись в переменную *X* значения ионного тока с *ЭМУ 1* на момент вызова функции *exSignal*.

Совокупность процедур и функций модуля дает полный контроль над аппаратным комплексом МИ 1201-АГМ в пределах, предусмотренных его конструкцией.

1.2. Принципы построения модуля

Для удобства использования процедуры и функции модуля (далее просто методы) сгруппированы. Группировка осуществлена в двух измерениях:

1) вертикальная (иерархическая) группировка по уровню сложности предоставляемого сервиса (уровни 0, 1, 2);
2) горизонтальная группировка по функциям частей аппаратного комплекса (только для уровня 1).

Схема вертикальной и горизонтальной группировки модуля приведены на рис. 1.1.

Модуль группирует методы управления в виде объектов — **контроллеров**. **Под контроллером здесь и далее подразумевается программа**, а не часть аппаратуры комплекса МИ 1201-АГМ.

Контроллер состоит из набора методов и структуры данных. Каждому сплошному прямоугольнику на рис. 1.1 соответствует отдельный контроллер.

Предполагается, что конечному пользователю достаточно использовать контроллер верхнего уровня — **контроллер МИ 1201-АГМ**.

Описание других контроллеров приводится для лучшего понимания работы модуля, модернизации и исправления ошибок в низкоуровневых контроллерах (см. прил. 1 и прил. 2).

ИЕРАРХИЯ ОБЪЕКТОВ МОДУЛЯ

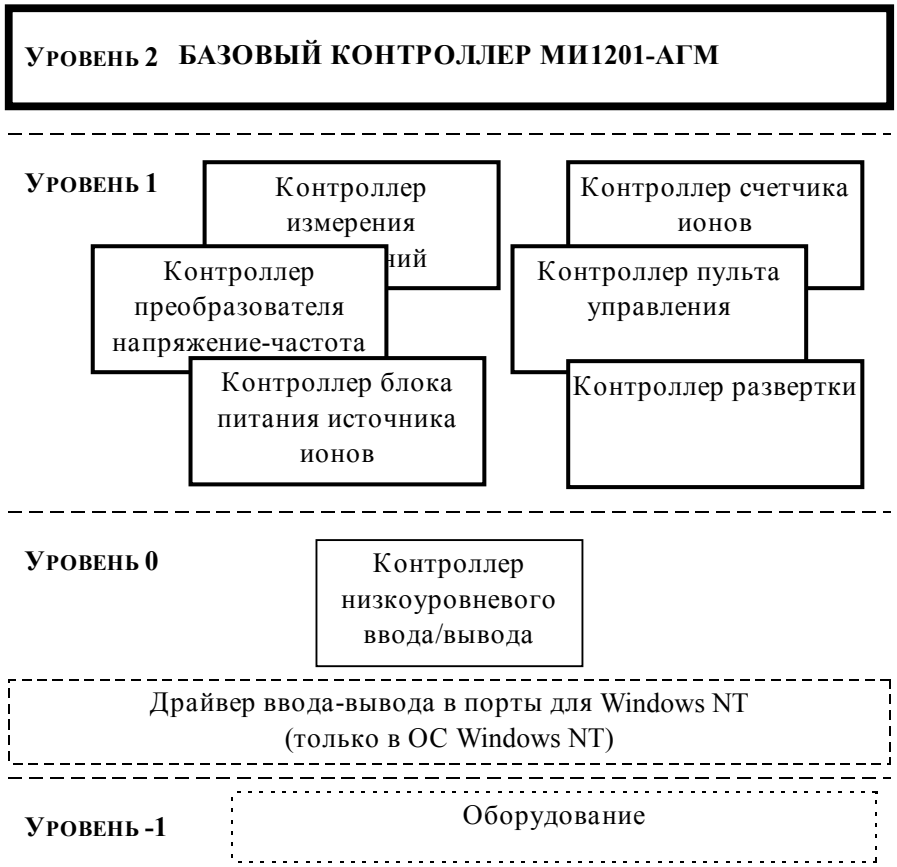

Рис. 1.1

1.3. ОБЩАЯ СТРУКТУРА ОБЪЕКТА «КОНТРОЛЛЕР»

Все контроллеры имеют унифицированную структуру и реализованы в форме *объектов* Borland Pascal 7.0 (см. табл. 1.1 и определение типа «*tCtrl*» в соответствующих файлах).

1. Каждый контроллер наследует свойства объекта *c_Ctrl.tCtrl*[3]. Объект *c_Ctrl.tCtrl* содержит общие для контроллеров структуры данных и методы (см. табл. 1.3).
2. Каждый контроллер имеет две группы методов:
   2.1. Неисполняемые методы. Осуществляют изменение внутренних данных контроллера, **без ввода/вывода в порты, т.е без изменения состояния аппаратуры**. Влияют на поведение исполняемых методов.

[3]) Поскольку все контроллеры имеют одинаковое имя «*tCtrl*», то для указания имени конкретного контроллера необходимо использование имени файла: «*имя_файла. tCtrl*». Имена файлов указаны в табл. 1.3. Например, имя контроллера развертки выглядит так: «*c_Roll. tCtrl*».

2.2. Исполняемые методы. Осуществляют **ввод/вывод в порты и изменение состояния аппаратуры**.

3. Исполняемые методы. Для удобства имена исполняемых методов отличаются от имен неисполняемых методов префиксом «*ex*». Например, метод «*Reset*» — неисполняемый, а «*exReset*» — исполняемый. Исполняемые методы различных контроллеров различаются (см. прил. 1).
4. По смыслу выполняемых операций все методы делятся на следующие четыре группы:
   4.1. Инициализационные. Выполняют первоначальную установку параметров контроллера, для подготовки контроллера к работе (см. табл. 1.4). Вызываются однократно, в начале работы программы. Повторный вызов может быть необходим при сбоях в работе контроллера.
   4.2. Установка параметров. Выполняют изменения внутренних данных контроллера (зависят от типа контроллера).
   4.3. Чтение текущих значений параметров. Возвращают текущие значения внутренних данных контроллера (зависят от типа контроллера).
   4.4. Информационные (см. табл. 1.5). Это в основном неисполняемые методы.
5. Доступ к данным контроллеров осуществляется только через методы. Поэтому структуры внутренних данных контроллеров далее не рассматриваются.
6. Исполняемые методы могут вызвать ошибку исполнения команды контроллера (ОИК).

Таблица 1.1

ИМЕНА И НАЗНАЧЕНИЕ ФАЙЛОВ МОДУЛЯ

Контроллер	Обозначение	Файл	Уровень контроллера
1. Некоторые общие типы данных	—	*MITypes.pas*	—
2. Типы данных для обмена с драйвером Windows NT	—	*xDDK.pas*	—
3. Вспомогательные функции	—	*MiscFunc.pas*	—
4. Калибровка шкалы масс	—	*MassClbr.pas*	—
5. Запись/чтение данных для контроллеров	—	*DataSave.pas*	—
6. Взаимодействие с драйвером устройства Windows NT	—	*PortsIO.pas*	—
7. Общий предок контроллеров	—	*c_Ctrl.pas*	—
8. Расширенный общий предок контроллеров (используется некоторыми контроллерами вместо *c_Ctrl.pas*). Содержит дополнение в виде процедур измерения интервалов времени.	—	*c_Ctrl*1.*pas*	—
9. Ввод/вывод в порты	—	*c_Bus.pas*	0
10. Блок питания	АК2	*c_ISSB.pas*	1
11. Пульт управления	АК3	*c_Panel.pas*	1
12. Измерение напряжений	АК5	*c_Volts.pas*	1
13. Счетчик ионов	АК7	*c_Count.pas*	1
14. Развертка (управление магнитом)	АК8	*c_Roll.pas*	1
15. Преобразователь напряжение-частота (ПНЧ, он же ЭМУ)	АК9	*c_CVF.pas*	1
16. МИ 1201-АГМ	—	*c_MI1201.pas*	2

1.4. ОБРАБОТКА ОШИБОК КОНТРОЛЛЕРА

1. Исполняемые методы могут вызвать ошибку исполнения команды контроллера (ОИК).
2. Каждый контроллер имеет собственную ячейку данных для запоминания кода ОИК.

3. Если возникает ОИК, то код ОИК запоминается и выполнение любых исполняемых методов данного контроллера прекращается — исполняемый метод при вызове просто ничего не делает. Неисполняемые методы продолжают работать.
4. Возобновление работы исполняемых методов происходит ТОЛЬКО после явной отмены ОИК.
5. Для отмены ОИК необходим вызов метода: *SetNoError*.
6. Вызов метода *SetNoError* отменяет состояние ошибки в данном контроллере и во всех контроллерах, от которых зависит данный контроллер.
7. Вызов метода *SetNoError* не гарантирует полного восстановления работоспособности контроллера, это зависит от типа ошибки. В некоторых случаях необходима повторная инициализация контроллера.
8. Неисполняемые методы практически никогда не вызывают ОИК и всегда продолжают работать. Исключения: см. табл. 1.2.
9. Все общедоступные исполняемые методы контроллеров построены по общей схеме обработки ошибок (см. рис. 1.2).
10. Схема обработки ошибок может не соблюдаться в скрытых исполняемых методах (см. раздел «*private*» в описании объектов).
11. Последовательной вызов нескольких исполняемых методов, например, *exМетод1*; *exМетод2*; … *exМетодN*; выполняется только до момента возникновения первой ошибки.

СХЕМА ОБРАБОТКИ ОШИБОК В ИСПОЛНЯЕМЫХ МЕТОДАХ КОНТРОЛЛЕРОВ

```
procedure tCtrl.exМетод(…аргументы…);
begin
    If ErrorCode=0 then begin
        … исполнение …
        If Ошибка_исполнения then
        SetErrorCode(код_ошибки);
    end;
end;
```

Рис. 1.2

1.5. СПЕЦИАЛЬНАЯ ОШИБКА — *ECABORT*

Для поддержки многопотоковых программ в модуле определен специальный код ошибки «*ecAbort*». Эта ошибка никогда не возбуждается в модуле самостоятельно, но может быть возбуждена вызовом *SetErrorCode*(*ecAbort*).

Предназначен для остановки контроллера во время выполнения длительных операций, например: ожидание окончания интегрирования сигнала при длительном времени или развертка по шкале масс на значительный интервал.

При вызове метода *SetErrorCode*(*ecAbort*) контроллера код ошибки устанавливается в контроллере, если в нем не была уже возбуждена ошибка.

При вызове метода *SetErrorCode*(*ecAbort*) контроллера MI1201 код ошибки устанавливается во всех контроллерах 1-го уровня и контроллере шины, если в них не была уже возбуждена ошибка.

Таблица 1.2

ИСКЛЮЧЕНИЯ ИЗ СХЕМЫ ОБРАБОТКИ ОШИБОК КОНТРОЛЛЕРОВ — ВОЗБУЖДЕНИЕ ОШИБОК НЕИСПОЛНЯЕМЫМИ МЕТОДАМИ

Контроллер	Имя метода	Описание
Все контроллеры	*Save(var DataPtr*: *pointer);*	Возбуждает ОИК *ecDataSaveFail* при ошибке сохранения данных контроллера.
Все контроллеры	*RestoreData(var DataPtr*: *pointer);*	Возбуждает ОИК *ecDataRestoreFail* при ошибке чтения данных контроллера.
Все контроллеры уровня 1	*InitDefault(var Bus*: *c_Bus.tCtrl);* *Init(var Bus*: *c_Bus.tCtrl; Ports*: *tPorts);*	Возбуждает ОИК *ecBadController* при передаче в качестве аргумента *Bus* контроллера ввода/вывода с состоянием *Bus.ErrorCode* <> 0.

Таблица 1.3

ОСНОВНЫЕ МЕТОДЫ БАЗОВОГО ОБЪЕКТА *C_CTRL.TCTRL*

Имя метода	Назначение
function *ErrorCode*: *word;*	Возвращает код ошибки. НОЛЬ означает отсутствие ошибок.
procedure *SetErrorCode(ec*: *tErrorCode);*	Устанавливает код ошибки. Коды зависят от контроллера (см. определение типа *tError* в соответствующем файле), общие коды: • *ecOK* (=0) — нет ошибки; • *ecAbort* (=1) — принудительная остановка выполнения команды (зарезервировано для использования в многозадачном расширении, никогда не возбуждается контроллером самостоятельно). • *ecNotInitialized* (=2) — не инициализирован контроллер. • *ecBadBus* (=3) — не инициализирован контроллер шины (возбуждается только контроллерами уровня 1). • *ecDataSaveFail* — ошибка сохранения состояния конт-

Имя метода	Назначение
	роллера; • *ecDataRestoreFail* — ошибка восстановления состояния контроллера; Вызов метода не устанавливает нового кода ошибки, если в контроллере уже возбуждена ошибка. Для изменения кода ошибки *AErrorCode* следует выполнить *SetErrorCode(ecOK); SetErrorCode(AErrorCode)*.
procedure *SetNoError;*	Устанавливает код ошибки *ecOK* в данном контроллере. Для контроллера MI1201 устанавливает код ошибки *ecOK* и во ВСЕХ контроллерах 0÷1 уровня.

Таблица 1.4

МЕТОДЫ ИНИЦИАЛИЗАЦИИ КОНТРОЛЛЕРОВ

Имя метода	Назначение
constructor *Init(…);*	Инициализирует контроллер, захватывает необходимые ресурсы ЭВМ и устанавливает значения параметров контроллера «*по умолчанию*», ЗА ИСКЛЮЧЕНИЕМ ПЕРЕДАННЫХ ЯВНЫХ ЗНАЧЕНИЙ.
constructor *InitDefault;*	Инициализирует контроллер, захватывая необходимые ресурсы и устанавливает ВСЕ значения параметров «*по умолчанию*».
destructor *Done;*	Деинициализирует контроллер, освобождая занятые контроллером ресурсы ЭВМ.
procedure *exInit;*	Инициализирует аппаратную часть контроллера в соответствии с текущими значениями параметров. Не вызывается автоматически.
procedure *exDone;*	Деинициализирует контроллер, устанавливая безопасное состояние аппаратуры. Не вызывается автоматически.
function *DataSize*: *word;*	Возвращает размер блока памяти (в байтах), необходимый для сохранения состояния контроллера.
procedure *Save(var DataPtr*: *pointer);*	Сохраняет состояние контроллера в блоке памяти, на который указывает *DataPtr*. Возвращает измененный *DataPtr,* устанавливая указатель на байт, следующий за сохраненными данными.

Имя метода	Назначение
procedure *exRestore(var DataPtr*: *pointer);*	НЕ рекомендуется. Восстанавливает состояние контроллера из ранее записанного блоке памяти, на который указывает *DataPtr*. Возвращает измененный *DataPtr,* устанавливая указатель на байт, следующий за сохраненными данными.
procedure *Restore(var DataPtr*: *pointer);*	Восстанавливает состояние контроллера из ранее записанного блоке памяти, на который указывает *DataPtr*. Возвращает измененный *DataPtr,* устанавливая указатель на байт, следующий за сохраненными данными. В ОТЛИЧИЕ ОТ *exRestore* должна вызываться после *Init*, но ДО *exInit*.

Таблица 1.5

ИНФОРМАЦИОННЫЕ МЕТОДЫ КОНТРОЛЛЕРОВ

Имя метода	Назначение
function *Present*: *tCtrlAlive;*	Возвращает результат автообнаружения контроллера методом *Init*: *aYes* — да присутствует; *aMayBeYes* — скорее присутствует, чем отсутствует; *aMayBeNo* — скорее отсутствует, чем присутствует; *aNo* — отсутствует. Только для контроллеров уровня 1 и выше.
function *ErrorMessage(en*: *tErrorCodes)*: *string;*	Возвращает развернутый текст сообщения об ошибке с кодом *en*.
function *CurErrorMessage*: *string;*	Возвращает развернутый текст сообщения о текущей ошибке.
function *Name*: *tName;*	Возвращает имя контроллера (длиной до 16 символов).

1.6. Принцип «одна программа — один контроллер»

Чтобы избежать ошибок в программе следует:

1) Использовать только один экземпляр контроллера уровня 2 в программе. Этим контроллером должен быть *контроллер МИ 1201*.
2) Контроллеры уровня 1 лучше не использовать, либо использовать только те экземпляры, которые доступны в составе *контроллера МИ 1201* (см. главу 2).

3) Контроллер уровня 0 не использовать.
4) Программа должна БЛОКИРОВАТЬ ОДНОВРЕМЕННЫЙ ЗАПУСК 2-х И БОЛЕЕ ЭКЗЕМПЛЯРОВ ЗАДАЧИ. В идеале это должен обеспечивать драйвер портов ввода/вывода, но он пока написан только для Windows NT и не обеспечивает данной функции.

1.7. Особенности применения контроллера в среде Delphi

Инициализацию контроллеров рекомендуется проводить только после запуска основной программы. Инициализация в разделе «*begin … end.*» основной программы или модуля допустима, но нежелательна.

Инициализацию контроллера рекомендуется проводить в обработчике события «*OnCreate*» главного окна (*Form*) программы.

В рекомендуется использовать модуль совместно с отдельным «витком» («thread») Windows.

Контроллер МИ 1201-АГМ и контроллеры уровней 1 и 0 используют язык Borland Pascal 7.0. В языке Borland Pascal 7.0 отсутствуют многие возможности языка Object Pascal, используемого в Delphi. Рекомендуется не расширять реализацию базовых контроллеров возможностями Object Pascal для сохранения совместимости с DOS-программами.

1.8. Особенности применения модуля в ОС Windows NT

Настоящая реализация модуля совместима с Windows NT. Для работы модуля в Windows NT необходимо установить драйвер ввода/вывода в порты. Описание драйвера и его установки — см. инструкцию [1].

Взаимодействие модуля с драйвером сосредоточено в *PortsIO* (см. файл PortsIO.pas). Модуль автоматически распознает Windows NT и пытается подключиться к драйверу ввода/вывода в порты. При отсутствии драйвера возбуждается соответствующая ОИК.

При работе в Windows 9x модуль использует прямой доступ к портам и драйвер не нужен.

2. КОНТРОЛЛЕР МИ 1201-АГМ

2.1. Общие замечания

Предполагается, что контроллер МИ 1201-АГМ реализует все необходимые функции для управления масс-спектрометром. Рекомендуется избегать

прямых вызовов контроллеров уровня 1. Вызов контроллера ввода/вывода (уровень 0) невозможен.

Объект *контроллер МИ 1201-АГМ*, реализован в файле «*c_MI1201.pas*». Поддерживаемые им методы и их назначение описаны ниже.

Контроллер МИ 1201-АГМ должен быть **ЕДИНСТВЕННЫМ**. Это означает, что в программе пользователя только ОДИН РАЗ декларируется переменная типа *Контроллер МИ 1201-АГМ*, т.е. текст определения переменной типа *c_MI1201.tCtrl*:

var x: *c_MI1201.tCtrl;*

должен присутствовать в программе только один раз. Все части программы должны использовать эту переменную для обращения к методам контроллера.

Контроллер МИ 1201-АГМ сам размещает и инициализирует (в процессе выполнения методов *Init* и *exInit*) по одному экземпляру всех контроллеров уровня 1 и уровня 0.

2.3. Структура данных контроллера МИ 1201-АГМ

Контроллер МИ 1201-АГМ имеет доступные для пользователя данные. В качестве доступных данных выступают контроллеры уровня 1.

Недоступные данные контроллеров не описаны в данном документе и не обсуждаются.

Обращение к контроллеру уровня 1 может быть осуществлено так:

*переменная_контроллера_МИ 1201.**Имя_Контроллера_Уровня_1***;

Имена для контроллеров уровня 1 приведены в табл. 2.1. Обращение к методам контроллеров уровня 1:

*переменная_контроллера_МИ 1201.Имя_Контроллера_Уровня_1.**Метод***;

Непосредственное обращение к методам контроллеров уровня 1 не рекомендуется. В недоступной части данных контроллера МИ 1201-АГМ имеет, помимо прочего, контроллер ввода/вывода, но прямой доступ к нему невозможен.

2.3. Методы контроллера МИ 1201-АГМ

Поскольку большинство методов контроллера МИ 1201-АГМ являются исполняемыми, то ниже они сгруппированы по выполняемым функциям.

Таблица 2.1

ИМЕНА КОНТРОЛЛЕРОВ УРОВНЯ 1 ВНУТРИ СТРУКТУРЫ ДАННЫХ КОНТРОЛЛЕРА МИ 1201-АГМ

Контроллер	Обозначение	Имя для доступа к контроллеру
1. Блок питания	АК2	*ctrlISSB*
2. Пульт управления	АК3	*ctrlPanel*
3. Измерение напряжений	АК5	*ctrlVolts*
4. Счетчик ионов	АК7	*ctrlCount*
5. Развертка магнитного поля	АК8	*ctrlRoll*
6. Преобразователь напряжение-частота (ПНЧ)	АК9	*ctrlCVF*

2.3.1. Инициализация и деинициализация

constructor *InitDefault;*

Инициализирует контроллер и все низкоуровневые контроллеры, используя значения параметров по умолчанию. Инициализируются только внутренние структуры данных без обращения к оборудованию.

constructor *Init(Ports*: *tPorts);*

Инициализирует контроллер и все низкоуровневые контроллеры, используя значения параметров по умолчанию. Инициализируются только внутренние структуры данных без обращения к оборудованию. Аргумент *Ports*: *tPorts* не используется. Зарезервировано для дальнейшего развития.

destructor *Done;*

Деинициализирует контроллер и все низкоуровневые контроллеры.

procedure *exInit;*

Инициализирует аппаратную часть контроллера и всех низкоуровневых контроллеров. НЕ ВЫЗЫВАЕТСЯ АВТОМАТИЧЕСКИ методами *InitDefault* и *Init*. Последнее сделано, чтобы дать возможность изменить параметры контроллеров до инициализации аппаратной части.

procedure *exDone;*

Деинициализирует аппаратную часть контроллера и всех низкоуровневых контроллеров. Автоматически вызывается методом *Done*.

function *ComplitelyInitiated*: *boolean;*

Возвращает TRUE если контроллер полностью и без ошибок прошел инициализацию (*Init* и *exInit*).

2.3.2. Методы сохранения и восстановления рабочих параметров контроллера

Для возможности полного сохранения и восстановления рабочих параметров контроллер снабжен специальными методами.

Методы позволяют сохранить и восстановить **все** текущие установки контроллера и **точное**[4)] текущее состояние оборудования.

Данные методы не могут контролировать размеры памяти, передаваемой для записи параметров. Контроль за достаточностью размера областей памяти возлагается на программиста.

function *DataSize*: *word;*

Возвращает размер памяти в байтах, необходимый для сохранения параметров контроллера.

procedure *Save(var DataPtr*: *pointer);*

Записывает параметры контроллера в область памяти, на которую указывает *DataPtr*. Предполагает, **но не проверяет и не может проверить**, что область памяти имеет размер не менее *DataSize*.

Возвращает измененный *DataPtr* так, что он указывает на байт, следующий за последним байтом записанных данных.

procedure *Restore(var DataPtr*: *pointer);*

Восстанавливает состояние контроллера из ранее записанного блоке памяти, на который указывает *DataPtr*. Возвращает измененный *DataPtr,* устанавливая указатель на байт, следующий за сохраненными данными. В ОТЛИЧИЕ ОТ *exRestore* должна вызываться после *Init*, но ДО *exInit*.

НЕ рекомендуется. Восстанавливает ранее записанные параметры контроллера из области памяти, на которую указывает *DataPtr* и соответствующее состояние оборудования. Предполагает, но **не проверяет и не может проверить**, что область памяти имеет размер не менее *DataSize*. Про-

---

[4)] В доступных пределах, например, точно восстановить значение магнитной индукции невозможно из-за конструктивных особенностей контроллера развертки.

веряет, что переданные данные действительно являются ранее записанными методом *Save* параметрами контроллера.

Возвращает измененный *DataPtr* так, что он указывает на байт, следующий за последним байтом данных.

procedure *exRestore(var DataPtr*: *pointer);*

Аналогична последовательности: *Restore*; *exInit*

procedure *SaveToFile(const FName*: *String);*

Записывает параметры контроллера файл *FName*.

procedure *RestoreFromFile(const FName*: *String);*

Восстанавливает ранее записанные параметры контроллера из файла *FName*.

procedure *exRestoreFromFile(const FName*: *String);*

Восстанавливает ранее записанные параметры контроллера из файла *FName* и соответствующее состояние оборудования. Аналогична последовательности: *RestoreFromFile; exInit*.

function *DataSize*: *word;*

Возвращает размер блока памяти (в байтах), необходимый для сохранения состояния контроллера.

procedure *Save(var DataPtr*: *pointer);*

procedure *SaveEx(var DataPtr*: *pointer):boolean;*

Сохраняет состояние контроллера в блоке памяти, на который указывает *DataPtr*. Возвращает измененный *DataPtr,* устанавливая указатель на байт, следующий за сохраненными данными.

Save возбуждает ОИК, если сохранение не прошло успешно.

SaveEx возвращает *TRUE* если сохранение прошло успешно и FALSE иначе, но не возбуждает ОИК.

2.3.3. Управление инициализацией оборудования

Поскольку полная инициализация требует значительного времени и/или иногда необходима повторная инициализация отдельных частей оборудования, то в контроллер МИ 1201 включены методы управления процессом инициализации.

Методы управления влияют только на исполнение *exInit*. Методы должны быть вызваны после метода *Init*, но перед вызовом метода *exInit*. Повторный вызов метода *Init* переписывает установки управления инициализацией значениями по умолчанию (полная инициализация и полное тестирование).

Таблица 2.2

Список флагов управления инициализацией

Флаг	Контроллер	Описание действия
ffRollFastExInit	не используется	не оказывает действия из-за изменения в контроллере эл. магнита. Сохранен для совместимости.
ffSkipValvesExInit	пульт управления (АК3)	Отключает процедуру закрытия всех клапанов при инициализации.
ffSkipValvesExDone	пульт управления (АК3)	Отключает процедуру закрытия всех клапанов при деинициализации.
ffSkipExDoneAtAll	МИ 1201-АГМ	Отключает процедуру *exDone* — не выключает оборудование при закрытии программы.
ffFastVoltsInit	измерение напряжений (АК5)	Быстрее инициализировать контроллер измерения напряжений (ускорение незначительно, не проводится контрольное измерение для проверки работоспособности).
ffUseEmulator	контроллер шины	Использовать эмулятор масс-спектрометра. В настоящее время не работает.

procedure *SkipMaskSetForExInit(x: tControllers);*

procedure *SkipMaskSetForTest(x: tControllers);*

Устанавливают список отключения контроллеров от выполнения процедуры *exInit* (*SkipMaskSetForExInit*) и процедуры выполнения тестирования (*SkipMaskSetForTest*). Список контроллеров может включать имена: *Bus, CVF, Roll, Volts, Panel, ISSB, Count*.

После вызова *Init* устанавливается: *SkipMaskSetForExInit([])* и *SkipMaskSetForTest([])* — инициализировать и тестировать ВСЕ контроллеры.

procedure *SpecialFeaturesSet(x: tFeatures);*

procedure *SpecialFeaturesGet(var x: tFeatures);*

Устанавливает и возвращает флаги ускорения и/или отключения некоторых процедур при инициализации/деинициализации контроллеров. Список флагов приведен в табл. 2.2.

2.3.4. Включение и выключения блоков и устройств

Устройства, допускающие включение и выключение по команде ЭВМ перечислены в табл. 2.3. В этой же таблице приведены имена устройств, которые должны использоваться в качестве аргументов при вызове методов.

2.3.5. Методы контроля включения блоков и устройств

procedure *exSwitchesSet(x: tSwitches);*

Включает все устройства (несколько одновременно), присутствующие в наборе *x* и выключает все остальные.

procedure *exSwitchesGet(var x: tSwitches);*

Возвращает набор включенных устройств в переменной *x*.

procedure *exSwitchTurnON(x: tSwitch);*

Включает устройство (только одно), определенное аргументом *x*.

procedure *exSwitchTurnOFF(x: tSwitch);*

Выключает устройство (только одно), определенное аргументом *x*.

function *exSwitchIsON(x: tSwitch): boolean;*

Таблица 2.3

УСТРОЙСТВА МИ 1201-АГМ, ДОПУСКАЮЩИЕ ВКЛЮЧЕНИЕ И ВЫКЛЮЧЕНИЕ

Устройство	Имя для доступа
Контроллер ПНЧ (АК9)	
Луч	*fCVF_Beam*
Опорное напряжение ПНЧ равно 0 *В* (иначе =-9 *В*)	*fCVF_Base0*
Подключение на вход ПНЧ сигнала с ЭМУ (иначе на вход подано опорное напряжение)	*fCVF_InputEMU*
Не инвертировать сигнал на входе в ПНЧ (иначе сигнал инвертируется)	*fCVF_DoNotInvertInput*
Контроллер пульта управления (АК3)	
Блок питания газового источника	*fBPGI*
Блок питания высокого напряжения	*fHighVoltageSupplay*
ВЭУ	*fSEM*
Разрешение управления клапанами	*fValvesControl*
Прочие (виртуальные переключатели**)**	
Разрешение одновременного включения блока питания высокого напряжения и ВЭУ.	*fAllowHighVoltageAndSEM*

Возвращает состояние устройства *x*. Возврат *exSwitchIsON(x)* значения *TRUE* означает, что устройство включено.

2.3.6. Включение и выключения клапанов. Получение информация о состоянии клапанов

Клапаны, допускающие включение и выключение по команде ЭВМ перечислены в табл. 2.3. В этой же таблице приведены имена клапанов, которые должны использоваться в качестве аргументов при вызове методов.

Открытие клапанов магистралей в данной версии контроллера МИ 1201-АГМ не реализовано. Открытие клапанов магистралей доступно через метод *exInOut* контроллера пульта управления (АК3).

Система клапанов масс-спектрометра МИ 1201-АГМ устроена так, что при открытии клапана сначала автоматически закрывается другой открытый клапан (если таковой был). Исключение составляют клапаны магистралей.

2.3.7. Методы контроля включения клапанов

procedure *exSourceSet(x*: *tSource);*

Открывает клапан *x*. Для закрытия всех клапанов следует вызвать метод с аргументом *x = sCloseAll;*

Таблица 2.4

Клапаны МИ 1201-АГМ

Клапан	Имя для доступа
Закрыть все клапаны	*sCloseAll*
Открыть напуск пробы 1	*sSample*1
Открыть напуск пробы 2	*sSample*2
Открыть напуск стандартной пробы 1	*sStandard*1
Открыть напуск стандартной пробы 2	*sStandard*2
Открыть напуск стандартной пробы 3	*sStandard*3
Открыть напуск стандартной пробы 4	*sStandard*4
Открыть откачку	*sPumping*
Ошибка открытия[5)]	*sBad*
Клапаны магистралей[6)]	
Открыть выход магистрали 1	*fLine*1*Out*
Открыть вход магистрали 1	*fLine*1*In*
Открыть выход магистрали 2	*fLine*2*Out*
Открыть вход магистрали 2	*fLine*2*In*

function *exSource*: *tSource;*

Возвращает имя открытого клапана, если все закрыты возвращает *sCloseAll.*

2.3.8. Массовая шкала

procedure *tCtrl.MassCalibration*(var *MassCalibr*:*tMassCalibration*);

Возвращает калибровку шкалы масс в виде объекта, см. определение *tMassCalibration* в файле «*MassClbr.pas*».

---

[5]) Вызов метода *exSetSource(sBad)* приводит к возбуждению ОИК. Метод *exCurSource* возвращает *sBad*, если контроллер находится в состоянии ошибки.

[6]) Открытие клапанов магистралей в данной версии контроллера МИ1201-АГМ не реализовано. Открытие клапанов магистралей доступно через метод *exInOut* контроллера пульта управления (АК3).

procedure *MassCalibrationSet*(*M0*, *K*: *tMass*);

Устанавливает соответствие [счетчик импульсов] ↔ [масса]. Аргументы: *M0* и *K* — значения коэффициентов в формуле преобразования $M = M0 + K \cdot C^2$, где *M* - масса иона и *C* - значение счетчика импульсов развертки [*M*=0*; K=1*].

procedure *MassCalibrationGet*(var *M0*, *K*: *tMass*);

Возвращает данные о соответствии [счетчик импульсов] ↔ [масса] (см. *MassCalibrationSet*).

function *Mass2Counter(x*: *tMass)*: *longint;*

Преобразует массу *x* в значение счетчика импульсов контроллера электромагнита, согласно текущей калибровке (см. *MassCalibrationSet*).

function *Counter2Mass(x*: *longint)*: *tMass;*

Преобразует значение счетчика импульсов контроллера электромагнита *x* в массу, согласно текущей калибровке (см. *MassCalibrationSet*).

2.3.9. Методы управления магнитным полем

procedure *exJumpToMass(M*: *tMass);*

function *exJumpToMassEx(M*: *tMass): boolean;*

Изменяет магнитное поле для установки на массу иона *M*. Требует задания калибровки, см. *MassCalibrationSet*. Функция *exJumpToMass* дополнительно проверяет допустимость значения *M* и возвращает TRUE, если магн. поле успешно изменено, иначе (в том числе и при ОИК) возвращает FALSE.

procedure *exJumpTo(C: longint);*

function *exJumpToEx(C: longint): boolean;*

Изменяет магнитное поле и устанавливает его на значение счетчика импульсов *C*. Функция *exJumpToEx* дополнительно проверяет допустимость значения *C* и возвращает TRUE, если магн. поле успешно изменено, иначе (в том числе и при ОИК) возвращает FALSE.

procedure *exScroll(Shift*: *longint);*

function *exScrollEx(Shift*: *longint): boolean;*

Изменяет магнитное поле на *Shift* импульсов контроллера магнита с учетом знака. Функция *exScrollEx* дополнительно проверяет допустимость зна-

чения *Shift* и возвращает TRUE, если магн. поле успешно изменено, иначе (в том числе и при ОИК) возвращает FALSE.

function *Counter*: *longint;*

Возвращает текущее значение счетчика импульсов.

function *Mass*: *tMass;*

function *MassMin*: *tMass;*

function *MassMax*: *tMass;*

Возвращает массу иона для установленного магнитного поля, минимальную допустимую массу и максимальную допустимую массу на массовой шкале. Требует задания калибровки, см. *MassCalibrationSet*.

function *MassValid*(*m*: *tMass*):*boolean;*

Возвращает TRUE, если значение массы допустимо (можно преобразовать в разумное значение счетчика).

2.3.10. Методы измерения напряжений

procedure *VoltageChannelSet(x*: *tVoltageChannel);*

Устанавливает канал измерения напряжения. Допустимые имена точек измерения приведены в табл. 2.5.

function *VoltageChannel*: *tVoltageChannel;*

Возвращает текущий канал измерения напряжения.

procedure *VoltageReadParametersSet(Time*: *word; Cnt*: *word);*

Устанавливает параметры чтения показаний вольтметра с ожиданием стабильности показаний за период *Time* (мс) с повтором попытки не более *Cnt* раз.

procedure *VoltageReadParametersGet(var Time*: *word; var Cnt*: *word);*

Возвращает текущие параметры чтения показаний вольтметра с ожиданием стабильности показаний за период *Time* (*мс*) с повтором попытки не более *Cnt* раз.

function *exVoltage*: *longint;*

Таблица 2.5

ТОЧКИ ИЗМЕРЕНИЯ НАПРЯЖЕНИЙ

Точка измерения	Имя для доступа
Напряжение ИМЧ	*IMCh*
Напряжение ускоряющее	*Acceleration*
Ток электромагнита	*Magnet*
Напряжение умножителя	*SEM*
Напряжение антидинатрона	*AntiDinatron*
Напряжение опоры ПНЧ	*BaseUPT*
Напряжение УПТ 1÷8	*UPT1, UPT2, UPT3, UPT4, UPT5, UPT6, UPT7, UPT8*
Напряжение УПТ-У	*UPTU*
Напряжение линзы (фокусирующее)	*Lens*
Нет выбранного канала - ОШИБКА	*NoChannelSelected*

Возвращает значение напряжения с текущего канала измерения, (*мкВ*).

2.3.11. Методы получения данных измерения ионного тока

procedure *SignalChannelSet(u*: *tChannel);*

function *SignalChannel*: *tChannel;*

Устанавливает/возвращает канал для считывания данных по ионному току. Допустимые имена каналов: (*PNC1, PNC2, PNC3, PNC4, PNC5, PNC6, SEM, IonCounter*).

function *exSignal*: *longint;*

Считывание данных с канала *SignalChannel*. Проводит запуск измерения, ожидает завершения измерения и возвращает данные без преобразования.

function *exSignalV*: *double;*

Считывание данных с канала *SignalChannel*. Проводит запуск измерения, ожидает завершения измерения и возвращает данные в *Вольтах*, кроме счетчика ионов, последний возвращает число импульсов за 1 *мс*.

procedure *exSignalsStart*;

Запуск измерения на ВСЕХ каналах.

procedure *SignalsWait*;

Ожидание готовности ВСЕХ каналов — задержка выполнения программы на время интегрирования (для разгрузки процессора) и проверка готовности.

procedure *exSignalsVRead*(*var Signals*: *tSignals*);

Ожидание готовности, считывание данных со ВСЕХ каналов и возврат данных в *Вольтах*, кроме счетчика ионов, последний возвращает число импульсов за 1 *мс*. Структура данных *tSignalsV*:

```
tSignalsV=record
  Count:byte; { число каналов }
  Time:word; { время интегрирования сигнала }
  Mass:tMass; { значение массы для которой измерен сигнал }
  Signals:tSignalsVArray; { значение сигналов }
end;
tSignalsVArray=array[tChannel] of double;
```

procedure *exSignalsRead*(*var Signals*: *tSignals*);

Ожидание готовности, считывание данных со ВСЕХ каналов и возврат данных БЕЗ ПРЕОБРАЗОВАНИЯ. Структура данных *tSignals*

```
tSignal=record
    Signal: longint;    { значение сигнала }
    Time: word;         { время интегрирования сигнала }
end;
tSignals=record
    Count: word;        { число каналов }
    Signals: array[tChannel] of tSignal; { данные для каналов }
end;
```

procedure *exSignalsGet*(*var Signals*: *tSignals*);

Запуск измерения, ожидание готовности, считывание данных со ВСЕХ каналов и возврат данных БЕЗ ПРЕОБРАЗОВАНИЯ.

procedure *exSignalsVGet*(*var Signals*: *tSignals*);

Запуск измерения, ожидание готовности, считывание данных со ВСЕХ каналов и возврат данных в *Вольтах*, кроме счетчика ионов, последний возвращает число импульсов за 1 *мс*.

procedure *SignalsGet*(*var Signals*: *tSignals*);

Возврат данных последнего измерения для ВСЕХ каналов БЕЗ ПРЕОБРАЗОВАНИЯ.

procedure *SignalsVGet*(*var Signals*: *tSignals*);

Возврат данных последнего измерения для ВСЕХ каналов в *Вольтах*, кроме счетчика ионов, последний возвращает число импульсов за 1 *мс*.

2.3.12. Методы управления измерением ионного тока

procedure *IntegrationTimeSet(t*: *word);*

function *IntegrationTime*: *word;*

Установка/возврат времени интегрирования (мс) для всех каналов.

procedure *IntegrationTimeForChannelSet* (*t*: *word*);

function *IntegrationTimeForChannel*: *word*;

Установка/возврат времени интегрирования (мс) для канала. Параметр сохраняется для счетчика ионов *IonCounter* и для всех остальных каналов.

procedure *exCalibrate*;

procedure *exCalibrateFast*;

procedure *exCalibrateEx*(*ChannelSet*: *tChannels*);

Проводит калибровку каналов ПНЧ для преобразования числа импульсов счетчика в *Вольты*. Для этого на вход ПНЧ подается фиксированное напряжение 0 *В* и 9 *В*, и проводится запуск счета с текущим временем интегрирования. По разнице показаний вычисляются калибровочные коффициенты. Доступ к данным калибровки — см. П.1.5.

Разница между функциями: *exCalibrate* — проводит полную калибровку всех каналов с замером подаваемых напряжений вольтметром (если он доступен); *exCalibrateFast* — проводит калибровку всех каналов, полагая напряжения равными 0 *В* и 9 *В*; *exCalibrateEx* — проводит полную калибровку как *exCalibrate*, но только для указанных в *ChannelSet* каналов.

procedure*CalibrateSetDelayTime (x*: *word);*

procedure *CalibrateSetRetryCount(x*: *word);*

function *CalibrateDelayTime*: *word);*

function *CalibrateRetryCount*: *word;*

Устанавливают/возвращают параметры калибровки ПНЧ: *DelayTime* — время ожидания стабильности показаний вольтметра [300 мс]; *RetryCount* — максимальное число повторов ожидания стабильности [100].

Используются для калибровки ПНЧ. Калибровка НЕ ВЫЗЫВАЕТСЯ АВТОМАТИЧЕСКИ.

Операция перекалибровки занимает значительное время от 4×*CalibrateDelayTime* миллисекунд до 4×*CalibrateDelayTime*×*CalibrateRetryCount* миллисекунд.

```
procedure exCalibrate;
  procedure exCalibrateEx(ChannelSet:tChannels);
  procedure exCalibrateFast;
  procedure CalibrateSetDelayTime(x:word);
  procedure CalibrateSetRetryCount(x:word);
  function  CalibrateDelayTime:word;
  function  CalibrateRetryCount:word;
```

2.3.13. Методы регулировки напряжений

procedure *exDeviceUSet (Device*: *tDevice; Value*: *longint);*

Устанавливает для устройства *Device* значение величины *Value* (обычно это напряжение, *Value* может изменяться от минимального *DeviceUMin* до максимального значения *DeviceUMax* с дискретностью *DeviceUStep*.

Если методу *exDeviceUSet* передано значение не совпадающее с допустимым, но в пределах диапазона [*DeviceUMin*.. *DeviceUMax*], то значение округляется до ближайшей возможной величины.

Имена устройств приведены в табл. 2.6.

function *DeviceU(Device*: *tDevice)*: *longint;*

Возвращает для устройства *Device* текущее значение величины *Value*.

function *DeviceUMax(Device*: *tDevice)*: *longint;*

Возвращает для устройства *Device* максимальное значение величины.

function *DeviceUMin(Device*: *tDevice)*: *longint;*

Таблица 2.6

ИМЕНА УСТРОЙСТВ ДЛЯ МЕТОДОВ РЕГУЛИРОВКИ НАПРЯЖЕНИЙ

Устройство	Имя для доступа	Единицы измерения	Диапазон изменения, min..max
Напряжение ионизации	*IonizationVoltage*	*В*	30..100
Ток эмиссии	*EmissionCurrent*	*мкА*	0..100
Вытягивающее напряжение	*ExtractingVoltage*	*усл. единицы*	0..99
Фокусирующее напряжение	*FocusingVoltage*	*усл. единицы*	0..99
Коррекция X	*CorrectionX*	*усл. единицы*	0..99
Коррекция Z	*CorrectionZ*	*усл. единицы*	0..99
Напряжение ВЭУ	*SEM Voltage*	*В*	0..5000

Возвращает для устройства *Device* минимальное значение величины.

function *DeviceUStep(Device*: *tDevice)*: *longint;*

Возвращает для устройства *Device* текущее значение шага изменения величины.

function *DeviceCounterMax (Device*: *tDevice)*: *word;*

Возвращает для устройства *Device* максимальное значение счетчика шагов при изменении величины.

function *DeviceCounter(Device*: *tDevice)*: *word;*

Возвращает для устройства *Device* текущее значение счетчика числа шагов. Счетчик может изменяется в диапазоне [0.. *DeviceCounterMax*].

2.3.14. Методы получения аварийных сигналов

procedure *exEmergencyFlagsGet(var x*: *tEmergencyFlags);*

Возвращает набор аварийных флагов: *tEmergencyFlag= (efFlagsUnavailable, efCatodBurnOUT, efOverload)*.

procedure *exCatodBurnOUT*: *boolean;*

Возвращает *TRUE* если КАТОД СГОРЕЛ.

procedure *exOverload*: *boolean;*

Возвращает *TRUE* если ПЕРЕГРУЗКА ИСТОЧНИКА ПИТАНИЯ.

ЗАКЛЮЧЕНИЕ

Модуль управления для масс-спектрометра МИ 1201-АГМ может применяться как основа для написания пользовательских программ. Основные преимущества использования модуля, по сравнению с прямым программированием управления:

1) компактный и удобный код (в идеале, после отладки, без ошибок);
2) отделение процедур управления аппаратурой от интерфейса программы;
3) обеспечение управления МИ 1201-АГМ в операционных системах Windows 9x/NT.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Драйвер Windows NT для МИ 1201-AGM: Инструкция по эксплуатации. УГТУ, 2000.

ПРИЛОЖЕНИЕ 1

КОНТРОЛЛЕРЫ УРОВНЯ 1

Содержание

П.1.1. Соответствие базовых функций управления МИ 1201-АГМ и контроллеров уровня 1

Базовые функции управления и получения данных измерений сосредоточены в контроллерах 1-го уровня (см. рис. 1.1). Перечень операций и распределение операций по контроллерам приведен в табл. П.1.1.

Таблица П.1.1

Соответствие основных операций и контроллеров

Операция	Контроллер	Метод[7)]
Установка напряжений и токов		
Установка напряжения ионизации	АК2	*exSetValue(IonizationVoltage, величина)*
Установка тока эмиссии	АК2	*exSetValue(EmissionCurrent, величина)*
Установка вытягивающего напряжения	АК2	*exSetValue(ExtractingVoltage, величина)*
Установка фокусирующего напряжения	АК2	*exSetValue(FocusingVoltage, величина)*
Установка коррекции X	АК2	*exSetValue(CorrectionX, величина)*
Установка коррекции Z		*exSetValue(CorrectionZ, величина)*
Управление		
Управление магнитным полем	АК8	*exScroll(изменение в относительных единицах);*
Получение данных измерений		
Чтение значения ионного тока с ЭМУ 1 ÷ ЭМУ 5	АК9	*exGetData; Channel(1..5);*
Чтение значения ионного тока с ВЭУ	АК9	*exGetData; Channel(9);*
Чтение счетчика ионов	АК7	*Count;*

[7]) Приводится только один метод прямого доступа к операции в полном объеме, хотя могут существовать методы поэтапного выполнения данной операции (см. описание соответствующего контроллера).

Продолжение табл. П.1.1

Операция	Контроллер	Метод[7)]
Включение/Выключение		
Включение луча стабилизатора	АК2	*exBeamON(True);*
Включение блоков: 1. ВЭУ. 2. 10кВ. 3. БПГИ.	АК3	*exBlocksON([блоки]);*
Включение разрешения управлять клапанами	АК3	*exValvesControlEnable(True);*
Открытие клапанов магистралей		*exInOut([клапаны]);*
Открытие клапанов: 1. Проб. 2. Стандартных проб.		*exSources(клапан);*
Информационные		
Чтение информационных сигналов: 1. «Катод цел». 2. «Блок питания газового источника (БПГИ) ВКЛючен». 3. «10кВ ВКЛючен». 4. «Перегрузка БПГИ». 5. «Луч ВЫКЛючен».	АК2	*exCurFlags(var флаги состояния);*
Чтение значений напряжений: 1. ИМЧ. 2. Ускоряющее. 3. Электромагнит (ток). 4. Умножитель. 5. Антидинатрон. 6. Опоры ПНЧ. 7. УПТ 1 … 14.УПТ 8. 15.УПТ-У. 16.Линзы (фокусирующее).	АК5	*exVoltage(1..15);*

П.1.2. Общие методы

Данные методы реализованы во всех контроллерах уровня 1 и имеют смысл, описанный ниже.

П.1.2.1. Методы инициализации

constructor *Init(var Bus*: *c_Bus.tCtrl; Ports*: *tPorts);*

Инициализирует контроллер с использованием контроллера ввода/вывода *Bus* и значений портов ввода/вывода *Ports*. Структура данных *tPorts* уникальна для каждого контроллера уровня 1, см. исходные тексты модуля.

Контроллер ввода/вывода должен быть инициализирован перед использованием в качестве аргумента метода *Init*.

constructor *InitDefault(var Bus*: *c_Bus.tCtrl);*

Инициализирует контроллер с использованием контроллера ввода/вывода *Bus* и стандартных значений портов ввода/вывода. Контроллер ввода/вывода должен быть инициализирован перед использованием в качестве аргумента метода *InitDefault*.

procedure *exInit;*

Инициализирует аппаратную часть контроллера. Аппаратная часть контроллера должна быть инициализирована перед использованием любого исполняемого метода *exMemod*.

procedure *exDone;*

Деинициализирует аппаратную часть контроллера, приводя в безопасное состояние.

destructor *Done;*

Деинициализирует контроллер. Автоматически вызывает *exDone.*

П.1.2.2. Методы установки параметров

procedure *TimeOut(x*: *word);*

Установка времени (мс) ожидания события в портах [зависит от контроллера, обычно ~100÷500 мс].

function *CurTimeOut*: *word;*

Значение текущей установки времени (мс) ожидания события в портах.

П.1.2.3. Методы контроля обработки ошибок

procedure *SetNoError;*

Отмена состояния ошибки для контроллера и всех контроллеров, от которых зависит данный контроллер[8)].

function *ErrorMessage(ec*: *tErrorCode);*

Возвращает текст сообщения об ошибке с кодом *ec*.

П.1.2.4. Методы сохранения и восстановления состояния котроллера

Данные методы не могут контролировать размеры памяти, передаваемой для записи параметров. Контроль за достаточностью размера областей памяти возлагается на программиста.

function *DataSize*: *word;*

Возвращает размер памяти в байтах, необходимый для сохранения параметров контроллера.

procedure *Save(var DataPtr*: *pointer);*

Записывает параметры контроллера в область памяти, на которую указывает *DataPtr*. Предполагает, **но не проверяет и не может проверить**, что область памяти имеет размер не менее *DataSize*.

Возвращает измененный *DataPtr* так, что он указывает на байт, следующий за последним байтом записанных данных.

procedure *exRestore(var DataPtr*: *pointer);*

Восстанавливает ранее записанные параметры контроллера из области памяти, на которую указывает *DataPtr* и соответствующее состояние оборудования. Предполагает, но **не проверяет и не может проверить**, что область памяти имеет размер не менее *DataSize*. Проверяет, что переданные данные действительно являются ранее записанными методом *Save* параметрами контроллера.

Возвращает измененный *DataPtr* так, что он указывает на байт, следующий за последним байтом данных.

[8)] Все контроллеры уровня 1 зависят от контроллера ввода/вывода.

П.1.3. Контроллер пульта управления

П.1.3.1. Назначение контроллера пульта управления

Контроллер АК3 пульта управления предназначен для управления включением блоков и открытием клапанов масс-спектрометра.

Объект *контроллер пульта управления* реализован в файле «*c_Panel.pas*». Поддерживаемые им методы и их назначение приведены ниже.

П.1.3.2. Методы контроллера пульта управления

П.1.3.2.1. Неисполняемые

Общие методы: см. таблицы 1.3 — 1.5

function *AllowHighVoltageAndSEM*: *boolean;*

Возвращает разрешение включения блока питания высокого напряжения и ВЭУ одновременно [*FALSE*]. При запрете включения попытка одновременно включить эти блоки возбуждает ошибку.

function *CurSwitchDelay*: *word;*

procedure *SetSwitchDelay(x*: *word);*

function *CurMinSwitchDelay*: *word;*

procedure *SetMinSwitchDelay(x*: *word);*

Возвращают и устанавливают время задержки (*SwitchDelay* [500 *мс*]) и минимальное время задержки (*MinSwitchDelay* [0 *мс*]) перед повторным выводом в порты управления включением блоков (см. *exBlocksON*) и клапанов (см. *exSource* и *exInOut*). Необходимо для того, чтобы реле и клапаны успели сработать, иначе при быстрых последовательных вызовах методов случаются несрабатывания.

function *SEMMax*: *word;*

function *SEMMin*: *word;*

function *SEMStep*: *word;*

Возвращают для ВЭУ максимальное (*В*), минимальное (*В*) значение величины напряжения и шаг (м*В*) изменения напряжения.

function *SEM_Value*: *word;*

Возвращает для ВЭУ текущее значение величины напряжения (*В*).

function *SEMCountMax*: *word;*

Возвращает для ВЭУ максимальное значение счетчика шагов. Минимальное значение всех счетчиков — НОЛЬ. Полное число шагов *SEMCountMax* +1.

function *SEMCount*: *word;*

Возвращает для ВЭУ текущее значение счетчика шагов.

procedure *SEM_CoeffSet*(*coeff*: *longint*);

function *SEM_CoeffGet*: *longint;*

Устанавливает и возвращает для ВЭУ текущее значение коэффициента преобразования число в ЦАП ВЭУ ↔ напряжение на ВЭУ в вольтах [98500]. Коэффициент возвращается умноженный на 10000. Формула преобразования:

(*напряжение_на_ВЭУ _в_вольтах***GetSEM_coeff*) div 10000 = *число_в_ЦАП_ВЭУ*.

П.1.3.2.2. Исполняемые

procedure *exInit;*

Инициализирует контроллер: включает все блоки (кроме ВЭУ) и закрывает все клапаны.

procedure *exAllowHighVoltageAndSEMSet(x*: *boolean);*

Устанавливает разрешение(*x* = *TRUE*) или запрет (*x* = *FALSE*) включения блока питания высокого напряжения и ВЭУ одновременно [*FALSE*]. Исполняемая частично — если при вызове *exSetAllowHighVoltageAndSEM(FALSE)* были включены блок 10кВ и ВЭУ, то ВЭУ выключается.

procedure *exBlocksONSet(x*: *tBlocks);*

Управляет включением (выключение) блоков. Управление производится с помощью переключателя типа *SET*. Допустимые значения аргумента определены в табл. П.1.2.

Наличие флага в наборе *x* приводит к включению, а отсутствие — к выключению, соответствующего блока. Например, вызов *exBlocksONSet([fBPGI, fHighVoltageSupplay])* — включает (или оставляет включенными) блок БПГИ и блок питания высокого напряжения, и отключает все остальные блоки.

procedure *exSourceSet(x: tSource);*

Управляет клапанами источников. Значения переключателя *x* приведены в табл. П.1.3. При открывании клапана другой открытый клапан закрывается автоматически (это функция оборудования).

procedure *exInOutSet(x: tValves);*

Управляет клапанами подключения к магистралям. Допустимые значения набора *x* (тип *SET*) приведены в табл. П.1.4.

procedure *exBlocksONGet(var x: tBlocks);*

Возвращает включенные блоки в переменной типа *SET* (см. *exBlocks*).

function *exSource: tSource;*

Возвращает текущий открытый клапан (см. *exSource*).

procedure *exInOutGet(var x: tValves);*

Возвращает включенные клапаны магистралей в переменной типа *SET* (см. *exInOut*).

procedure *exSEM_ValueSet(x: tValue);*

Устанавливает для ВЭУ значение величины напряжения *x* (*B*), *x* может изменяться от минимального *SEMMin* до максимального значения *SEMMax* с дискретностью *SEMStep*.

Если методу *exSEMSet* передано значение не совпадающее с допустимым, но в пределах диапазона [*SEMMin*.. *SEMMax*], то значение округляется до ближайшей возможной величины.

procedure *exSEM_CountSet(c: tCount);*

Устанавливает для ВЭУ значение величины счетчика шагов *c*, *c* может изменяться от 0 до максимального значения *SEMCountMax* с дискретностью 1.

Таблица П.1.2

УСТРОЙСТВА, ДОПУСКАЮЩИЕ ВКЛЮЧЕНИЕ И ВЫКЛЮЧЕНИЕ

Устройство	Имя для доступа
Блок питания газового источника	*fBPGI*
Блок питания высокого напряжения	*fHighVoltageSupplay*
ВЭУ	*fSEM*
Разрешение управления клапанами	*fValvesControl*

Таблица П.1.3

Клапаны источников пробы

Клапан	Имя для доступа
Закрыть все клапаны	*sCloseAll*
Открыть напуск пробы 1	*sSample*1
Открыть напуск пробы 2	*sSample*2
Открыть напуск стандартной пробы 1	*sStandard*1
Открыть напуск стандартной пробы 2	*sStandard*2
Открыть напуск стандартной пробы 3	*sStandard*3
Открыть напуск стандартной пробы 4	*sStandard*4
Открыть откачку	*sPumping*
Ошибка	*sBad*

Таблица П.1.4

Клапаны магистралей[9)]

Клапан	Имя для доступа
Открыть вход магистрали 1	*fLine*1*In*
Открыть выход магистрали 1	*fLine*1*Out*
Открыть вход магистрали 2	*fLine*2*In*
Открыть выход магистрали 2	*fLine*2*Out*

П.1.4. Контроллер развертки (электромагнита)

П.1.4.1. Назначение контроллера развертки

Контроллер развертки АК8 предназначен для управления током электромагнита масс-спектрометра.

Объект *контроллер развертки* реализован в файле «*c_Roll.pas*». Поддерживаемые им методы и их назначение перечислены ниже.

[9)] Открытие клапанов магистралей в данной версии контроллера МИ1201-АГМ не реализовано. Открытие клапанов магистралей доступно только через метод *exInOut* контроллера пульта управления (АК3).

П.1.4.2. Методы контроллера развертки

П.1.4.2.1. Неисполняемые
Общие методы: см. таблицы 1.3 — 1.5

function *CurTotalCounter*: *longint;*

Возвращает текущее значение счетчика импульсов, выданных на контроллер**.**

function *MaxTotalCounter*: *longint;*

Максимальное число импульсов (минимум равен 0). Возвращает правильное значение только после вызова *exInit*.

function *CurAutoTuning*: *boolean;*

Значение текущей установки автоподстройки магнитного поля [*False*].

procedure *LoBound(x*: *word);*

Установка нижней границы безопасной прокрутки магнита для программной проверки, $x = 0$ отключает этот механизм [10000]. Границы программного «STOP» устанавливаются так: минимум — *CurLoBound*; максимум — (*MaxTotalCounter - CurUpBound*).

При отключении программной проверки границ действует аппаратная.

procedure *UpBound(x*: *word);*

Установка верхней границы безопасной прокрутки магнита для программной проверки, $x = 0$ отключает этот механизм [50000]. Границы программного «STOP» устанавливаются так: минимум — *CurLoBoundt*; максимум — (*MaxTotalCounter - CurUpBound*).

function *CurLoBound*: *word;*

Текущее значение нижней границы безопасной прокрутки магнита для программной проверки.

function *CurUpBound*: *word;*

Текущее значение верхней границы безопасной прокрутки магнита для программной проверки.

function *LimitReached*: *boolean;*

Возвращает TRUE если достигнут предел интенсивности магнитного поля (верхний, либо нижний). Правильно работает только непосредственно после вызова *exScroll*.

function *LowLimit*: *boolean;*

Возвращает TRUE если достигнут нижний предел интенсивности магнитного поля. Правильно работает только, если *LimitReached* = TRUE.

П.1.4.2.2. Исполняемые

procedure *exScroll(x*: *tCounter);*

Изменение магнитного поля на *x* импульсов, с учетом знака.

function *exFlags*: *byte;*

Возвращает битовые флаги состояния контроллера.

procedure *exAutoTuning(x*: *boolean);*

Включение, если *x=True* или выключение *x=False* автоподстройки магнитного поля [*False*].

П.1.5. Контроллер ПНЧ

П.1.5.1. Назначение контроллера ПНЧ

Контроллер АК9 преобразователя напряжение-частота (ПНЧ) предназначен для управления и чтения данных с канала регистрации ионного тока методом интегрирования тока.

Объект *контроллер ПНЧ* реализован в файле «*c_CVF.pas*». Поддерживаемые им методы и их назначение приведены ниже.

П.1.5.2. Методы контроллера ПНЧ

П.1.5.2.1. Неисполняемые

Общие методы: см. таблицы 1.3 — 1.5

procedure *Regime(x*: *tRegime);*

Установка режима работы ПНЧ. Аргумент *x* типа *SET*. Переключатели режимов перечислены в табл. П.1.5. Физическое переключение режима производится при вызове любой исполняемой функции или метода *exRegime*.

procedure *IntegrationTime(x*: *word);*

Устанавливает время интегрирования сигнала в миллисекундах [100].

procedure *ActiveChannels(x*: *tChannels);*

Устанавливает набор рабочих каналов ПНЧ [1..5, 9]. Все операции проводятся только для рабочих каналов, остальные каналы игнорируются.

procedure *FastMode(x: boolean);*

Разрешает (*x = TRUE*) процедуру быстрого запуска измерения [*False*].

function *Channel(x: tChannel): longint;*

Возвращает значение последнего считывания данных с канала *x*. Без преобразований.

function *AverageOverActiveChannels: longint;*

Возвращает значение средней суммы всех счетчиков последнего считывания данных.

function *AverageOverChannels(x: tChannels): longint;*

Возвращает значение средней суммы всех счетчиков пересечения множеств *x* и *CurActiveChannels*. Результат для последнего считывания данных методом *exGetData*.

function *ChannelU(x: tChannel): longint;*

Возвращает значение напряжения (вольт·10^{-6}) последнего считывания данных с канала *x*. Требует калибровки, см. *exCalibrate*.

procedure *CurActiveChannels(x: tChannels);*

Возвращает в *x* набор активных каналов.

procedure *CurRegime(var x: tRegime);*

Возвращает в *x* набор текущий режим.

function *CurIntegrationTime: word;*

Возвращает текущее время интегрирования.

function *CurFastMode: boolean;*

Возвращает текущее состояние разрешения быстрого запуска измерения.

Таблица П.1.5

РЕЖИМЫ ПНЧ

Переключатель	Значение, если присутствует	Значение, если отсутствует
frsBaseIsZero	Напряжение на шине опоры ПНЧ равно нулю.	Напряжение на шине опоры ПНЧ равно -9 *В*.
frsInputIsEMU	На вход ПНЧ подключен сигнал с ЭМУ.	На вход ПНЧ подключена шина опоры.
frsDoNotInvertSignal	Сигнал не инвертируется.	Сигнал инвертируется.

Примечание.

1. Рабочий режим ПНЧ (измерение) соответствует *x* = [*frsBaseIsZero, frsInputIsEMU, frsDoNotInvertSignal*].
2. Другие режимы используются для специальных задач.

П.1.5.2.2. Исполняемые

procedure *exRegime;*

Физическое переключение режима работы ПНЧ на текущее значение, см. *Regime*.

procedure *exCalibrate(U0, U1*: *longint);*

Калибровка счетчиков ПНЧ для преобразования (*число импульсов*) → (*напряжение*). Напряжения для режимов калибровки *U0*, *U1* (микровольты[10)]) должны быть заданы извне. См. также *ChannelU*.

Режим калибровки 0: [*frsBaseIsZero, frsDoNotInvertSignal*]. Режим калибровки 1: [*frsDoNotInvertSignal*]. См. также *Regime* и *exRegime*.

Калибровка проводится подсчетом импульсов при подключении на вход ПНЧ нуля (*U0*, режим 0) и опорного напряжения (*U1*, режим 1). Поскольку контроллер ПНЧ не может самостоятельно замерить реальные величины этих напряжений, то они должны быть ему переданы.

procedure *exGetData;*

Запуск измерения и чтение данных во внутренний буфер. Доступ см. *Channel*.

[10]) Значения *U*0, *U*1 могут быть переданы в любых единицах, но и возвращаемое значение для *ChannelU* будет в тех же самых единицах.

procedure *exStart;*

Запуск измерения[11)] с текущими значениями параметров.

function *exReady*: *boolean;*

Возвращает *TRUE*, если измерение закончено.

procedure *exRead;*

Чтение данных (с ожиданием завершения измерения) во внутренний буфер. Доступ к данным см. *Channel*.

П.1.6. Контроллер счетчика ионов

П.1.6.1. Назначение контроллера счетчика ионов

Контроллер счетчика ионов АК7 предназначен для управления и чтения данных с канала регистрации ионного тока методом счета импульсов.

Объект *контроллер счетчика ионов*, реализован в файле «*c_Count.pas*». Поддерживаемые им методы и их назначение приведены ниже.

П.1.6.2. Методы контроллера счетчика ионов

П.1.6.2.1. Неисполняемые

Общие методы: см. таблицы 1.3 — 1.5

procedure *DivCoeff(x*: *word);*

Устанавливает коэффициент деления для таймера [1000] (*x=1000* соответствует единице измерения в миллисекундах).

procedure *IntegrationTime(x*: *word);*

Устанавливает время счета импульсов в миллисекундах, независимо от текущего коэффициента деления таймера.

procedure *ResetCtrl(x*: *boolean);*

Устанавливает применение аппаратного «сброса» (*x=True*) при каждом запуске измерения.

function *CurDivCoeff*: *word;*

Возвращает коэффициент деления для таймера (*x=1000* соответствует единице измерения в миллисекундах).

[11)] Методы *exStart*, *exReady* и *exRead* реализованы для возможности использования времени измерения для более полезных действий, чем ожидание в цикле. Время измерения легко установить, оно равно *CurIntegrationTime*.

function *CurIntegrationTime*: *word;*

Возвращает время счета импульсов (мс).

function *CurResetCtrl*: *boolean;*

Возвращает установку применения аппаратного «сброса» (*x=True*) при каждом запуске измерения.

П.1.6.2.2. Исполняемые

function *exGetData*: *longint;*

Запускает измерение, ожидает завершения измерения и возвращает результаты измерения.

procedure *exStart;*

Запуск измерения[12)] с текущими значениями параметров.

function *exReady*: *boolean;*

Возвращает *TRUE*, если измерение закончено.

function *exWaitReady*: *boolean*;

Ожидает (время интегрирования+*CurTimeOut*) завершения измерения и возвращает *TRUE*, если измерение закончено успешно (удалось дождаться).

function *exRead*: *longint;*

Ожидает завершения измерения и возвращает результаты измерения.

П.1.7. Контроллер измерения напряжений

П.1.7.1. Назначение контроллера измерения напряжений

Контроллер АК5 измерения напряжений предназначен для измерения напряжений в узлах масс-спектрометра.

Объект *контроллер измерения напряжений* реализован в файле «*c_Volts.pas*». Поддерживаемые им методы и их назначение приведены ниже.

П.1.7.2. Методы контроллера измерения напряжений

П.1.7.2.1. Неисполняемые

Общие методы: см. таблицы 1.3 — 1.5

[12)] Методы *exStart*, *exReady* и *exRead* реализованы для возможности использования времени измерения для более полезных действий, чем ожидание в цикле. Время измерения легко установить, оно равно *CurIntegrationTime*.

procedure *Channel(x: tChannel);*

Устанавливает канал измерения. Допустимые имена каналов приведены в табл. 2.5

procedure *RetryCount(x: word);*

Устанавливает число *x* повторов попыток ожидания стабильности показаний[13)] [10].

procedure *RetryDelay(x: word);*

Устанавливает время *x* (мс) ожидания стабильности показаний[13)] [100].

procedure *RetryMask(x: tDigits);*

Устанавливает маску *x* проверки стабильности показаний[13)] для метода *exCurVoltage*. Маска задается по десятичным разрядам [[1..4]]. Разряд 1 соответствует старшему разряду индикатора вольтметра, а 4 — младшему разряду.

function *CurChannel: tChannel;*

Возвращает текущий канал измерения. Допустимые имена каналов приведены в табл. 2.5.

function *CurRetryCount: word;*

Возвращает текущее число *x* повторов попыток ожидания стабильности показаний[13)] [10].

function *CurRetryDelay: word;*

Возвращает текущее время *x* (мс) ожидания стабильности показаний[13)] [100].

procedure *CurRetryMask(x: tDigits);*

Возвращает текущую маску *x* стабильности показаний[13)]. Маска задается по десятичным разрядам [[1..4]]. Цифра 1 соответствует старшему разряду индикатора вольтметра, а 4 — младшему разряду.

[13)] Полное максимальное время ожидания равно произведению *CurRetryCount* на *CurRetryDelay*. Ожидание прекращается и возвращается результат, если за интервал *CurRetryDelay* показания в десятичных разрядах, определенных *CurRetryMask* не изменились.

П.1.7.2.2. Исполняемые

function *exCurVoltage*: *longint;*

Возвращает текущее значение напряжения канала измерения С ОЖИДАНИЕМ СТАБИЛЬНОСТИ[13)], см. *RetryCount*, *RetryDelay* и *RetryMask*.

function *exCurVoltageFast*: *longint;*

Возвращает текущее значение напряжения канала измерения БЕЗ ОЖИДАНИЯ СТАБИЛЬНОСТИ[13)].

П.1.8. Контроллер блока питания источников ионов

П.1.8.1. Назначение контроллера блока питания источников ионов

Контроллер АК2 блока питания источников ионов предназначен для управления напряжениями источника ионов и включения (выключения) некоторых блоков. Кроме того блок питания источников генерирует аварийные и информационные сигналы о состоянии масс-спектрометра.

Объект *контроллер блока питания* реализован в файле «*c_ISSB.pas*». Поддерживаемые им методы и их назначение приведены ниже.

П.1.8.2. Методы контроллера блока питания источников ионов

Неисполняемые

Общие методы: см. таблицы 1.3 — 1.5

function *MaxValue(u*: *tUnit)*: *tValue;*

Возвращает для устройства *u* максимальное значение величины (обычно напряжения), в единицах $\times 10^{-6}$ (т.е. для напряжений в *микроВольтах*, $1мкВ = 10^{-6}В$ и для тока в *микроАмперах*).

Имена устройств приведены в табл. 2.6.

function *MinValue(u*: *tUnit)*: *tValue;*

Возвращает для устройства *u* минимальное значение величины (обычно напряжения), в единицах $\times 10^{-6}$ (т.е. для напряжений в *микроВольтах*, $1мкВ = 10^{-6}В$ и для тока в *микроАмперах*).

function *Step(u*: *tUnit)*: *tValue;*

Возвращает для устройства *u* значение шага изменения величины (обычно напряжения), в единицах $\times 10^{-6}$ (т.е. для напряжений в *микроВольтах*, $1мкВ = 10^{-6}В$ и для тока в *микроАмперах*).

function *CurValue(u*: *tUnit)*: *tValue;*

Возвращает для устройства *u* текущее значение величины (обычно напряжения), в единицах $\times 10^{-6}$ (т.е. для напряжений в *микроВольтах*, $1мкВ = 10^{-6}В$ и для тока в *микроАмперах*).

function *MaxCount(u*: *tUnit)*: *tCount;*

Возвращает для устройства *u* максимальное значение счетчика шагов. Минимальное значение всех счетчиков НОЛЬ. Полное число шагов *MaxCount*+1.

function *CurCount(u*: *tUnit)*: *tCount;*

Возвращает для устройства *u* текущее значение счетчика шагов.

Исполняемые

procedure *exSetValue(u*: *tUnit; x*: *tValue);*

Устанавливает для устройства *u* значение величины *x* (обычно напряжения), в физических единицах (т.е. для напряжений в *Вольтах*, и для тока в *Амперах*), *x* может изменяться от минимального *MinValue* до максимального значения *MaxValue* с дискретностью *StepValue*.

Если методу *exSetValue* передано значение не совпадающее с допустимым, но в пределах диапазона [*MinValue*.. *MaxValue*], то значение округляется до ближайшей возможной величины.

Имена устройств приведены в табл. 2.6.

procedure *exSetCount(u*: *tUnit; c*: *tCount);*

Устанавливает для устройства *u* значение величины счетчика шагов *c*, *c* может изменяться от 0 до максимального значения *MaxCount* с дискретностью 1.

procedure *exResetAllValues;*

Устанавливает заново (поворачивает моторы гарантированно до нуля и устанавливает заново значение) для ВСЕХ устройств значение величины в *CurValue*.

procedure *exBeamON(x*: *boolean);*

Включает (*x* = *True*) луч источника ионов.

function *exCurBeamON*: *boolean;*

Возвращает состояние луча источника ионов (*True* — включен).

procedure *exCurFlags(var x*: *tCtrlFlags);*

Возвращает информационные флаги. Значения флагов приведены в табл. П.1.6.

Таблица П.1.6

ЗНАЧЕНИЯ ИНФОРМАЦИОННЫХ ФЛАГОВ

Значение	Имя флага
Катод цел	*fCatodOK*
БПГИ включен	*fBPGI_On*
Высокое напряжение включено	*fHighVoltageSupplay_On*
Перегрузка	*fOverload*
Луч выключен	*fBeam_Off*

ПРИЛОЖЕНИЕ 2

КОНТРОЛЛЕР УРОВНЯ 0

Содержание

П.2.1. Контроллер ввода/вывода в порты

П.2.1.1. Назначение контроллера ввода/вывода

Контроллер ввода/вывода предназначен для организации и управления вводом/выводом в порты.

Дополнительно контроллер ввода/вывода предоставляет методы для ожидания в течение заданного интервала времени события в порту и задержки исполнения на заданный интервал времени.

НЕ РЕКОМЕНДУЕТСЯ ИСПОЛЬЗОВАТЬ НЕПОСРЕДСТВЕННО в программах пользователя.

Объект *контроллер ввода/вывода* реализован в файле «*c_Bus.pas*». Поддерживаемые им методы и их назначение приведены ниже.

П.2.1.2. Методы контроллера ввода/вывода в порты

П.2.1.2.1. Неисполняемые

Общие методы: см. таблицы 1.3 — 1.5

constructor *Init(BasePort*: *word);*

BasePort = базовый порт;

constructor *InitDefault*;

Инициализация с базовым портом = $ED00;

procedure *OutDelay(x: word);*

Число циклов задержки между последовательным исполнением команды вывода в порт (OUT AL, DX) [10 циклов]. Один цикл приблизительно равен 10÷20 тактам процессора[14)].

procedure *InDelay(x: word);*

Число циклов задержки между последовательным исполнением команды ввода из порта (IN AL, DX) [10 циклов].

procedure *SingleOutDelay(x: boolean);*

x=True — применять задержку вывода для одиночных байтов *exOutByte* [*False*].

procedure *SingleInDelay(x: boolean);*

x=True — применять задержку ввода для одиночных байтов *exInByte* [*False*].

function *CurOutDelay: word;*

Значение текущей установки числа циклов задержки между последовательным исполнением команды вывода в порт (OUT AL, DX) [10 циклов].

function *CurInDelay: word;*

Значение текущей установки числа циклов задержки между последовательным исполнением команды ввода из порта (IN AL, DX) [10 циклов].

function *CurSingleOutDelay: boolean;*

Если *True* — применяется задержка после вывода для одиночных байтов *exInByte* [*False*].

function *CurSingleInDelay: boolean;*

Если *True* — применяется задержка после ввода для одиночных байтов *exOutByte* [*False*].

procedure *Delay(Time: longint);*

Задержка на *Time* миллисекунд.

---

[14]) Цикл задержки для методов *exOutBytes* и *exInBytes*:

```
test es:tCtrl([bx]).prFlags,(1 shl fSingleOutDelay); jnz @SkipDelay
 mov ax,es:tCtrl([bx]).prOutDelay
 or ax,ax; jz @SkipDelay
 @DelayLoop: dec ax; jnz @DelayLoop {собственно цикл}
@SkipDelay:
```

function *ErrorMessage(ec*: *tCtrlErrorCodes)*: *string;*

Текстовое описание кода ошибки *ec*.

П.2.1.2.2. Исполняемые

function *exInByte(PortShift*: *word)*: *Byte;*

Ввод байта из порта: "базовый порт"+*PortShift*.

function *exInByteX(Port*: *word)*: *Byte;*

Ввод байта из порта *Port*.

procedure *exOutByte(PortShift*: *word; B*: *Byte);*

Вывод байта в порт: "базовый порт"+*PortShift*.

procedure *exOutByteX(Port*: *word; B*: *Byte);*

Вывод байта в порт *Port*.

procedure *exInBytes(PortShift*: *word; var Data; Size*: *word);*

Ввод строки байтов *Data* длиной *Size* из порта: "базовый порт"+*PortShift*.

procedure *exInBytesX(Port*: *word; var Data; Size*: *word);*

Ввод строки байтов *Data* длиной *Size* из порта *Port*.

procedure *exOutBytes(PortShift*: *word; var Data; Size*: *word);*

Вывод строки байтов *Data* длиной *Size* в порт: "базовый порт"+*PortShift*.

procedure *exOutBytesX(Port*: *word; var Data; Size*: *word);*

Вывод строки байтов *Data* длиной *Size* в порт *Port*.

function *exWait(PortShift*: *word; Mask, Patt*: *Byte; Time*: *longint)*: *tErrorCode;*

Ожидание в течении *Time* миллисекунд события: [(значение в порту "базовый порт"+*PortShift*) AND *Mask*] = *Patt*.

Если дождаться события не удалось, то возбуждается ошибка *ecTimeOut*.

При *Mask*=0 ошибка *ecTimeOut* не возбуждается.

function *exWaitX(Port*: *word; Mask, Patt*: *Byte; Time*: *longint)*: *tErrorCode;*

Ожидание в течении *Time* миллисекунд события: [(значение в порту *Port*) AND *Mask*] = *Patt*.

Если дождаться события не удалось, то возбуждается ошибка *ecTimeOut*.

При *Mask*=0 ошибка *ecTimeOut* не возбуждается.

function *exLazyWait(Port*: *word; Mask, Patt*: *Byte; Time*: *longint)*: *tErrorCode;*

Для DOS то же самое, что и *exWaitX*. Для Windows (Delphi) цикл ожидания устроен иначе: считывается порт, проверяется событие, если нет, то вызывается системная задержка 1 мс и все повторяется. В *exWaitX* происходит постоянное чтение порта — это сильно нагружает систему и для некритических ожиданий в многозадачном режиме выгоднее использовать *exLazyWait*.