Программное обеспечение адаптера –

2. Программное обеспечение адаптера

В программное обеспечение, поставляемое вместе с адаптером, входят библиотеки подпрограмм **U217_10.lib**, **CyAPI.lib**, исполняемые модули **CAMAC.exe**, файл заголовка **217_10U.h**, папка с примером проекта **217_10U_Test**

2.1 Подключение драйвера CYUSB

2.2 Библиотека U217_10.lib включает набор подпрограмм, позволяющих иметь доступ к контроллеру 217.10 через адаптер USB-CAMAC, пользуясь средствами языка Visual C++ v6.0. В библиотеку входят подпрограммы: **Camac**, **CamInit**, **CamClose**, **CamC**, **CamZ**, **SequLoad**, **SequStop**, **SequWaitAndRead**, **SetLamTimeOut**, **GetLamTimeOut**. Подпрограммы декларированы в файле **217_10U.h**

2.2.1 Camac. Функция выполняет одну команду КАМАК.

Формат:

unsigned char Camac(

unsigned char N,

unsigned char A,

unsigned char F,

[int* D],

[BYTE* XQ]

);

Параметры:

N – номер станции

A – субадрес

F - функция

D – передаваемые данные (см. примечание)

XQ – содержит ответ по X и Q

XQ & 0x 80 = 0, до выполнения команды не было LAM,

XQ & 0x 02 = 0, если команда не имеет ответ по Q,

XQ & 0x 01 = 0, если команда не имеет ответ по X,

(XQ & 0x 76) >> 2 – номер станции выдавшей LAM

Возвращаемое значение:

1, если адресуемый модуль дал ответ по Q,

0 в противоположном случае.

Примечание:

Параметр D обязан быть переменной. В командах без передачи данных параметр D игнорируется.

Пример:

```
unsigned int D = 0xFFFFFF;

unsigned char Q;

Q = Camac(1,2,0,16,D);   // Write function

Q = Camac(1,2,0,16,0xFFFFFF); // Write function

Q = Camac(2,2,0,1,D); // Read Function
```

2.2.2 CamInit. Функция загружает в ЗУ адаптера прошивку, инициализирует адаптер и резервирует необходимые ресурсы системы

.

Формат:

char* CamInit()

Возвращаемое значение:

NULL – если инициализация прошла успешно.

Иначе - Указатель на текстовую строку с диагностикой ошибки

Примечание:

До первой операции с адаптером функция **CamInit** обязательно должна быть выполнена хотя бы один раз.

После завершения работы с адаптером, резервированные ресурсы должны быть освобождены функцией **CamClose** (см. описание).

Если возвращено ненулевое значение инициализацию следует повторить.

Пример:

```
char* c;

c = CamInit();

if (c) {

    printf ("%s \n",c);

    exit(1);

}
```

2.2.3 CamClose. Функция освобождает ресурсы, занятые при инициализации адаптера.

Формат:

void CamClose();

Параметры:

Нет.

Возвращаемое значение:

Нет.

Пример:

CamClose();

2.2.4 CamC, CamZ. Функции генерируют сигналы C и Z в магистрали крейта.

Формат:

void CamC();

void CamZ();

Параметры:

Нет.

Возвращаемое значение:

Нет.

2.1.5 **SequLoad**. Функция загружает в адаптер последовательность команд, которая будет выполняться секвенсором, и запускает процесс исполнения. После того, как последовательность выполнена и буфер прочитан, последовательность автоматически выполняется снова.

Формат:

int SequLoad(char* OutBuf, int Length);

Параметры:

OutBuf - Массив, содержащий команды,

Length – Число команд в массиве

Возвращаемое значение:

0, если загрузка прошла успешно,

1 в противоположном случае.

.

Примечание:

Массив **OutBuf** имеет длину **Length** записей, по 8 байтов каждая. Внутри каждой записи поля длиною один байт имеют следующее значение:

0x00 – Старший байт записываемых данных,

0x01 – Средний байт записываемых данных,

0x02 – Младший байт записываемых данных,

Если команда не записывает данных, вышеуказанные поля игнорируются.

0x03 – Субадрес,

0x04 – Функция,

0x05 – Номер модуля,

0x06 – Не используется,

0x07 – Ожидание LAM и/или ответа по Q.

Если старший бит этого поля равен 1, команда повторяется до тех пор, пока не поступит **LAM** в магистрали крейта, или не истечет предельное время ожидания.

Если младший бит этого поля равен 1, команда повторяется до тех пор, пока модуль не даст ответа по **Q**, или не истечет предельное время ожидания.

Пример: Последовательность, включающая одну команду

```
int D;

int Reslt;

int N = 1;
```

```
int A = 0;

int F = 16;

char          Command [1][8];

Command[1][0] = BYTE(D >> 16);

Command[1][1] = BYTE(D >> 8);

Command[1][2] = BYTE(D);

Command[1][3] = BYTE(A);

Command[1][4] = BYTE(F);

Command[1][5] = BYTE(N);

Command[1][7] = BYTE(0x01);

Reslt = SequLoad(&Command,1);
```

2.2.6 SequWaitAndRead. Функция ожидает окончания выполнения секвенсором последовательности команд или окончания предельного времени ожидания, после чего считывает входной буфер. После прочтения буфера, последовательность команд автоматически выполняется повторно и следующее выполнение **SequWaitAndRead** считывает новое значение буфера и снова запускает исполнение последовательности команд и т.д.

Формат:

```
int SequWaitAndRead(char*  InBuf , int  Length);
```

Параметры:

InBuf - Массив, содержащий результаты выполнения команд,

Length – Число команд в последовательности, которая выполнялась

Возвращаемое значение:

0, если последовательность выполнена успешно,

1 истекло предельное время ожидания.

Примечание:

Массив InBuf имеет длину Length записей, по 8 байтов каждая. Внутри каждой записи поля длиною один байт имеют следующее значение:

0x00 – Субадрес соответствующей команды,

0x01 – Функция соответствующей команды,

0x02 – Номер модуля соответствующей команды,

0x03 – LAM, X Q,

0x04 – Старший байт считываемых данных,

0x05 – Средний байт считываемых данных,

0x06 – Младший байт записываемых данных,

0x07 – Состояние завершения команды,

Если младший бит этого поля равен 1, команда завершена, 0 - команда не завершена из-за истечения предельного времени ожидания.

Пример:

```
int D;

int Reslt;

int N;

int A;

int F;

char          Command [1][8];

Reslt = SequWaitAndRead (&Command,1);

if (!Reslt){

    D = Command[1][4] << 16;

    D += Command[1][5] << 8;

    D += Command[1][6];

    A = Command[1][0];
```

```
    F = Command[1][1];

    N = Command[1][2];

}
```

2.1.7 SequStop. Функция останавливает выполнение последовательности инструкций

Формат:

void SequStop();

Параметры:

Нет

Возвращаемое значение:

Нет

2.1.8 SetLamTimeOut. Функция устанавливает предельное время ожидания LAM.

Формат:

int SetLamTimeOut(int Time);

Параметр:

Time – новое значение предельного времени ожидания LAM

Возвращаемое значение:

0, если функция выполнена успешно,
1 - в противоположном случае

2.1.9 GetLamTimeOut. Функция проверяет предельное время ожидания LAM.

Формат:

int GetLamTimeOut(int* Time);

Параметр:

Time – адрес переменной, которая будет содержать значение предельного времени ожидания LAM

Возвращаемое значение:

0, если функция выполнена успешно,
1 - в противоположном случае

2.2.8 Включение библиотеки в проект пользователя

На панели инструментов Visual C++ кликните кнопку **“View”** затем ”**Workspace**”□”**FileView**”. Затем подведите курсор к пункту “**Source Files**”, щелкните правой кнопкой мыши и в открывшемся локальном меню выберите пункт “**Add Files To Folder**”. В появившемся диалоговом окне “**Incert Files into Project**” выберите нужную папку и файл “**U217_10.lib**” и дважды кликните левой кнопкой мыши по этому имени этого файла, После этого библиотека будет включена в проект пользователя.

2.2 Библиотека **CyAPI.lib** включает набор подпрограмм для связи библиотечных подпрограмм с контроллером CY7C68013A. Библиотека не документирована.

2.3 Приложение **CAMAC.exe**. Предназначено для отладки логики сбора данных, проверки модулей КАМАК и организации простых систем сбора и записи информации.

Главное окно приложения (рис 1.,2.) включает две закладки, соответствующие режимам тестер и секвенсор.

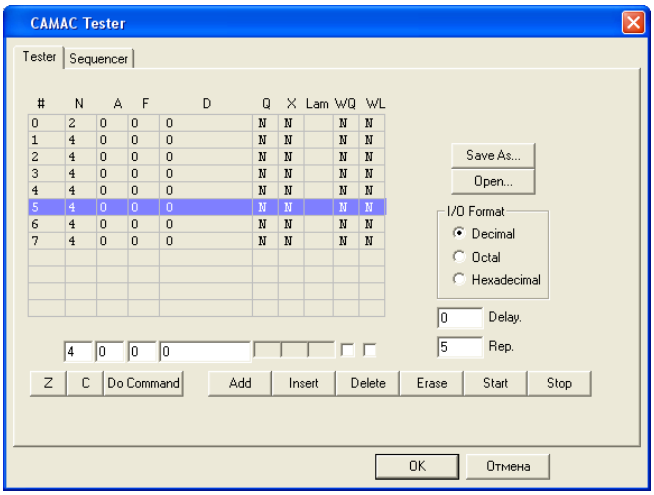

Рис 1. Главное окно приложения (Режим Tester)

Режим **Tester** позволяет выполнять команды поодиночке и в списке, сохранять, восстанавливать и редактировать параметры команд, создавать, сохранять редактировать и выполнять список команд. В этом режиме число команд ограничено только объемом свободной памяти компьютера и может быть весьма велико. Этот же список используется для выполнения в режиме **Sequensor**

Окно ввода команд предназначено для ввода параметров команды. В поля N,A,F,D вводятся номера модуля, субадрес, функция и данные, соответственно. В полях WQ, WL устанавливаются биты ожидания Q и LAM.

Окно списка позволяет контролировать выполнение команд в списке. Если все команды не помещаются в этом окне, появляется линейка прокрутки, позволяющая быстро перейти к нужной команда с помощью клавиш навигации или «мыши». Двойной клик «мышью» в поле отдельной команды списка вызывает выполнение этой команды.

Кнопки **Z** и **C** генерируют сигналы Z и C в крейте.

Кнопка **Do Command** вызывает выполнение набранной команды.

Кнопка **Add** добавляет набранную команду в конец списка команд секвенсора.
Кнопка **Insert** вставляет набранную команду в список команд секвенсора, перед командой, выделенной курсором.
Кнопка **Delete** удаляет команду, выделенную курсором.
Кнопка **Erase** удаляет, после подтверждения, все команды из списка.
Кнопки **Start** и **Stop** запускают/останавливают выполнение набранной последовательности в режиме эмуляции секвенсора. Выполнение останавливается также после исчерпания заказанного в поле **Rep**. числа выполнений.

В поле **Delay** устанавливается задержка в миллисекундах, после каждого выполнения списка команд.

Панель **I/O Format** позволяет выбрать один из трех форматов представления данных - десятичного, восьмеричного и шестнадцатеричного.

Кнопки **Save As…** и **Open..** позволяют сохранить в выбранном файле и восстановить конфигурацию приложения. Конфигурация, также, сохраняется при выходе из приложения с помощью кнопки **OK**. Сохраненная конфигурация автоматически восстанавливается при входе в программу.

Режим **Sequenser** (Рис.2) позволяет многократно выполнять последовательность

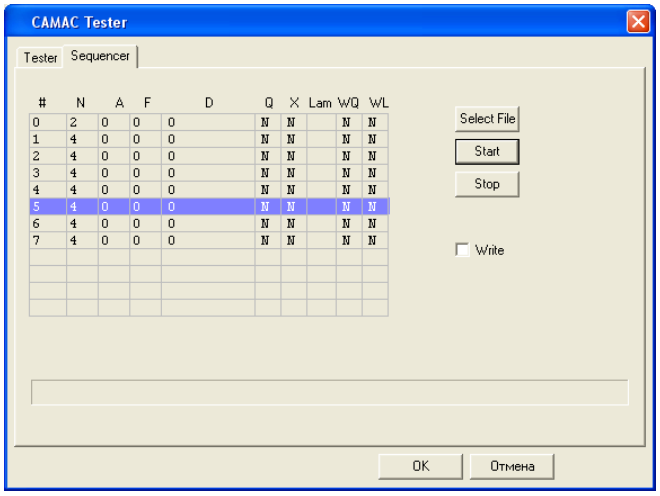

Рис 2. Главное окно приложения (Режим Sequencer)

команд, записывая, при необходимости, результаты в выбранный файл. В этом режиме числа команд в списке не должно превышать **64**.

Кнопка **Select File** позволяет выбрать файл, в который буде производиться запись результатов.

Кнопки **Start** и **Stop** запускают/останавливают выполнение набранной последовательности секвенсора. Выполнение останавливается также после исчерпания заказанного в поле **Rep** закладки Tester числа выполнений.

Поле **Write** позволяет запретить/разрешить записи в файл.

Данные записываются в файл в коде ASCII. Одна строка соответствует одному событию (одному выполнения всех команд списка). Поля внутри события располагаются в следующим порядке:

1) Номер события, отделенный символом «**;**»
2) По восемь полей атрибутов каждой команды (см. описание подпрограммы SequWaitAndRead) отделенных друг от друга символом «**,**» Команды отделяются друг от друга символом «**;**»