Государственный комитет Российской Федерации
по высшему образованию

Ульяновский государственный технический университет

Факультет *Информационных систем и технологий*
Кафедра *Вычислительная техника*
Дисциплина *Функциональная организация электронных вычислительных машин*

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту

Студентов *третьего* курса, группы *ЭВМд-32*
Волкова С.В, Михайлова В.Н., Строганова К.С., Сидоренко В.А.

Руководитель_____(_____)

Ульяновск 1998

Ульяновский государственный технический университет

Кафедра *Вычислительная техника*

Задание на курсовой проект

По дисциплине *Организация электронных вычислительных машин*
Студентам *Волкову С.В, Михайлову В.Н., Строганову К.С., Сидоренко В.А.*
Тема *Возможность применения Sound Blaster'а для исследования электрических сигналов*

Технические условия:
Разработка програмного средства на языке высокого уровня Borland Pascal 7.0 для защищенного режима работы процессора с максимальным использованием языка Assembler. От программы требуется максимальное быстродействие, поэтому обмен данными с Sound Blaster'ом должен осуществляться через DMA канал. Программа должна работать как с 8-ми битными музыкальными картами, так и с 16-ю битными.

Объем работы:
Изучение работы Sound Blaster'ов. Изучение принципов работы с данными через DMA канал. Разработка концепции программы. Кодирование программы. Тестирование на различных Sound Blaster'ах. Результаты исследования.

Дата выдачи **Срок выполнения**
Зав. кафедрой
Руководитель проекта
Проект защищен с оценкой **Дата**

Содержание:

Цель работы.

Разработка програмного средства для исследования возможности применения Sound Blaster'ов современных мультимедийных компьютеров в качестве средств контроля за параметрами некоторых слабых (до 1.0В) переменных сигналов. Определение формы сигнала, максимального и минимального значений сигнала, его действующего значение и частоты на некотором промежутке времени.

Обоснование выбора Sound Blaster'а как устройства позволяющего использовать компьютер в качестве средства контроля за параметрами сигнала.

В данное время Sound Blaster является практически неотъемлемой частью современных компьютеров. Поэтому для исследования сигнала не требуется дополнительного оборудования. Современные Sound Blaster'ы имеют множество програмно регулируемых параметров. Таких как частота дискретизации (4 - 44кГц), громкость по линейному входу, установка фильтров высоких/низких частот по входу. А также, самое главное, SB16 (и лучше) позволяют 16-ти битную оцифровку сигнала, что абсолютно достаточно для точной оцифровки малых сигналов. Также основной особенностью является простота программирования Sound Blaster'а и легкость манипулирования его настраиваемыми параметрами. Возможность выполнения параллельных процессов: оцифровки очередной части сигнала и обработки ранее оцифрованного фрагмента или выполнение других действий.

Возможности программы Oscillo (version 3.00).

Система предназначена для мониторинга параметров некоторого исследуемого переменного сигнала (1-2В и частотой не более 22кГц), подаваемого на линейный вход Sound Blaster'а. Система позволяет определить форму сигнала, его частоту, максимальное и минимальное значения на некотором промежутке, а также действующее значение данного сигнала. Программа позволяет не только наблюдать значения данных параметров визуально в реальном времени, но и накапливать статистику в некотором файле, для последующей обработки полученных данных.

Система может работать в трех режимах:

1. Режим осциллографа, где исследуемый сигнал можно наблюдать визуально и определять его форму.
2. Режим вольтметра, где визуализируется только действующее значение сигнала.
3. Режим накопления статистики, где такие параметры, как максимальное, минимальное, действующее значения сигнала, его частота на некотором промежутке времени, накапливаются в файле.

Система имеет возможность настройки на требуемый режим работы, т.е. имеется множество регулируемых параметров, позволяющих выбрать оптимальный режим. Такими параметрами являются:

1. Настройка частоты оцифровки сигнала.
2. Изменение масштаба, полученных в ходе измерений значений.
3. Настройка точности подсчета действующего значения сигнала.
4. В режиме накопления статистики выбор требуемого интервала времени, выдачи рассчитанных данных.

Требования к аппаратному обеспечению.

Минимальные требования, необходимые для полноценной работы программы:

Компьютер IBM PC совместимый с процессором 386 DX, оснащенный Sound Blaster'ом Pro, видеокартой ISA 512Кб и монитором, поддерживающим разрешение 640x480x256 цветов.

Рекомендуемые требования, необходимые для полноценной работы программы:

Компьютер IBM PC совместимый с прцессором Pentium 100, оснащенный Sound Blaster'ом 16, видеокартой PCI 1Мб и монитором, поддерживающим разрешение 1024x768x256 цветов.

Обоснование выбора соответствующих компонентов компьютера:

Видеокарта PCI 1Мб и монитор, поддерживающий разрешение 1024x768x256 цветов необходим для полноценной работы в режиме вольтметра, где на экране отображается осциллограмма сигнала. Чем больше разрешение экрана, тем большую часть осциллограммы сигнала можно отобразить на экране за один кадр. Шестнадцати битный Sound Blaster необходим для более точной оцифровки сигнала, чем восьми битный.

Описание пользовательского интерфейса.

Программа имеет систему меню, с помощью которой и происходит все управление системой. Имеется шесть основных меню, рассмотрим их подробно.

Меню **File:**

DOS shell позволяет временно выйти в операционную систему.

Exit позволяет завершить работу.

Меню **Setup:**

Hardware позволяет выбрать тип Sound Blaster'а и установить IRQ и DMA присущие вашему Sound Blaster'у. Если данные установки проставлены неверно, то программа может зависнуть.

Frequence позволяет установить требуемую частоту оцифровки сигнала.

Input volume позволяет установить громкость линейного входа. Т.е. если сигнал в осциллографе зашкаливает, уменьшите громкость линейного входа.

GraphMode позволяет установить неоюходимое разрешение экрана при работе в режиме осциллографа.

Coefficient позволяет корректировать показания вольтметра. Для корректировки необходимо сравнить показания реального цифрового вольтметра с компьютерным и соответствующим образом изменить коэффициент. Для большей точности проверить показания для разных сигналов. Так как разные Sound Blaster'ы

могут иметь некоторые отклонения в оцифровке сигнала, то точность работы вольтметра желательно проверять каждый раз , как вы начинаете работать с неизвестным ранее Sound Blaster'ом.

Frequence correction позволяет корректировать значение частоты в режиме накопления статистики, если оно по вашему мнению не верно.

Calculate constant позволяет высчитывать постоянную составляющую сигнала или просто брать ее нулевой.

Garmonic number позволяет изменить количество гармоник, на которые разбивается исходный сигнал. Чем больше гармоник, тем точнее получается значение действующего напряжения, но тем менее скорость работы вольтметра. Максимальное количество гармоник – 30.

Clock разрешает или запрещает визуализацию часов.

Screen saver позволяет установить временной интервал, по истечении которого запускается гасилка экрана. Гасилка запускается только когда программа не находится ни в одном из режимов.

Current settings визуализирует окно с текущими установками, если оно ранее было закрыто.

Меню **Help:**

Help выдает на экран окно помощи.

About выдает информацию о разработчиках программы.

Меню **Oscillogaph:**

Выбор данного пункта переводит программу в режим осциллографа. В данном режиме на экране отображается осциллограмма сигнала и значение напряжения в последней точке осциллограммы.

Меню **Voltmetr:**

Выбор данного пункта переводит программу в режим вольтметра.

Меню **Statist:**

Выбор данного пункта переводит программу в режим накопления статистики.

Определение формы сигнала.

Перед началом работы необходимо убедиться в наличии Sound Blaster'а и видеокарты, удовлетворяющих требованиям программы. Запустить исполняемый файл программы. Через меню **Setup/Hardware** установить тип Sound Blaster'а, IRQ и DMA, используемые им. Подать на линейный вход Sound Blaster'а исследуемый переменный сигнал. Через меню **Setup/Frequence** установить требуемую частоту оцифровки сигнала. С помощью меню **Oscillograph** запустить осциллограф. На экране появиться осциллограмма сигнала. Если вы видите, что исследуемый сигнал превышает допустимое значение для Sound Blaster'а, т.е. осциллограмма сверху и (или) снизу отрубается, то нажав **Esc** выйдите из режима осциллографа. Через меню **Setup/Input volume** уменьшите значение «громкости сигнала» по входу. И попытайтесь снова войти в режим осциллографа. Если установка громкости в 1 не помогает и сигнал все равно зашкаливает, то данный сигнал не пригоден для

исследования. При нажатии клавиши **Space**, можно «заморозить» осциллограмму на экране. «Разморозка» происходит опять же по нажатию клавиши **Esc**. Посредством манипулирования навигационными клавишами (влево, вправо) при необходимости можно добиться более или менее «стоячей» осциллограммы на экране.

Определение действующего значения сигнала.

Перед началом работы необходимо убедиться в наличии Sound Blaster'a, удовлетворяющего требованиям программы. Запустить исполняемый файл программы. Через меню **Setup/Hardware** установить тип Sound Blaster'a, IRQ и DMA, используемые им. Подать на линейный вход Sound Blaster'a исследуемый переменный сигнал. Через меню **Setup/Frequence** установить требуемую частоту оцифровки сигнала. Запустите осциллограф (см. выше) и убедитесь, что исследуемый сигнал не зашкаливает. Действуйте по обстоятельствам, как описано выше. С помощью меню **Voltmetr** запустите вольтметр. Сравните показания програмного вольтметра с реальным цифровым вольтметром. Если значение програмного вольтметра вас не удовлетворяет, то его можно подкорректировать используя меню **Setup/Coefficient**. Измените коэффициент на требуемое значение. При подсчете действующего значения сигнала исходный исследуемый сигнал разбивается на гармоники. Поэтому чем больше гармоник, на которые разбивается сигнал, тем точнее полученное значение, но тем медленне работа. Изменить число гармоник, на которые необходимо разбивать требуемый сигнал можно посредством меню **Setup/Garmonic number**. Если вы знаете, что постоянная составляющая исследуемого сигнала равна нулю, то отмените подсчет постоянной составляющей с помощью меню **Setup/Calculate constant**. Если же вы не знаете постоянную составляющую, то установите ее подсчет. После удовлетворительной настройки всех выше перечисленных параметров запустите вольтметр. Выход из режима вольтметра осуществляется по нажатию клавиши **Esc**.

Внимание!

При изменении частоты оцифровки необходимо проверять точность программного вольтметра и при необходимости изменять все настраиваемые параметры.

Накопление статистики.

Измените настройки программы как указано в пунктах «**Определение формы сигнала»** и «**Определение действующего значения сигнала»**. Если все настройки вас удовлетворяют, то войдите в режим накопления статистики с помощью меню **Statist**. В появившемся диалоге введите требуемый интервал времени, по истечении которого необходимо выдавать рассчитанные значения. По нажатию кнопки **«Ok»** появится диалог выбора файла, в котором необходимо сохранять вычисляемые данные. Нажав опять кнопку **«Ok»** , программа начнет накопление статистики. В файле сохраняется дата и время, максимальное и минимальное значения сигнала на отрезке, его действующее значение и частота.

Тестирование и выявленные недостатки.

Работа программы тестировалась на следующих Sound Blaster'ах:

AW 35 (производитель - AOpen, chip - Crystal Semiconductor, совместима с SoundBlaster Pro);

SB16 (производитель - Creative Labs);

AWE 32 (производитель - Creative Labs).

При тестировании выявлено, что все Sound Blaster'ы имеют разные характеристики оцифровки сигнала. Поэтому в интерфейс системы введено множество настраиваемых коэффициентов, настройку которых необходимо производить при использовании разных Sound Blaster'ов. Программа обычно вылетает с выдачей сообщения об ошибке, если нет Sound Blaster'а или указаны неправильные настройки IRQ и DMA.

Результаты исследования.

В ходе исследования выявились следующие недостатки Sound Blaster'ов:

1. Оцифровка сигнала 8-ми битными Sound Blaster'ами недостаточна для удовлетворительного исследования сигнала, так как точность очень низка. Поэтому следует использовать 16-ти битные Sound Blaster'ы, точность которых абсолютно удовлетворяет условиям исследования.
2. Оцифровка одного и того же сигнала с разной частотой дает разные рассчитываемые результаты (частоту, действующее значение). Видимо сказывается неточность програмной регулировки оцифровки. Поэтому при изменении частоты оцифровки необходимо проверять точность работы и изменять при необходимости требуемые настройки в программе. Такие как **Frequence correction** и **Coefficient**.

При использовании SB16 и однократной точной настройке на требуемые значения выдаваемые вольтметром и статистом в дальнейшем не меняя настроек программу можно использовать многократно. При этом все вычисляемые значения удовлетворяют условиям точности, т.е. значение сигнала определяется с точностью не менее 1/10000 В.

Алгоритмы, используемые в программе.

Алгоритм получения действующего значения напряжения:

Оцифровка происходит через DMA канал, т.е. в фоновом режиме. Поэтому во время оцифровки каждого следующего блока данных и происходит подсчет действующего значения напряжения предыдущего блока данных.

Определение 1:

Действующее значение несинусоидального напряжения равно корню квадратному из суммы квадратов постоянной составляющей и действующих значений отдельных гармоник.

Из этого следует, что полученный цифровой набор необходимо разбить на некоторое число гармонических составляющих, найти их амплитуду и, исходя из этого, определить действующее значения каждой гармоники, а также определить постоянную составляющую.

И после этого воспользоваться определением 1.

Для определения постоянной и гармонических составляющих в программе использован аналитический метод определения гармоник ряда Фурье (Л.А.Бессонов "Теоретические основы электротехники").

Алгоритм подсчета частоты исследуемого сигнала:

Пусть размер буфера RB. Пусть FP первый переход через ноль в оцифрованной последовательности, а LP последний переход через ноль. Пусть NP – количество всех переходов через ноль в оцифрованной последовательности. Тогда расчитываем длину периода $T = \frac{LP - FP}{(NP-1)/2}$. И общее количество периодов в оцифрованной последовательности будет $NT = (NP-1)/2 + (RB - (LP - FP))/T$. Тогда частота будет $F = (Rate/RB) \cdot NT$, где *Rate*-частота оцифровки.

Программирование Sound Blaster'а (необходимые сведения).

Порты ввода/вывода SoundBlaster DSP

Чип DSP (Цифровой Звуковой Процессор) программируется через пять портов, которые определяются через базовый адрес Sound Blaster:

2x6h - DSP Сброс
2xAh - DSP Чтение
2xCh - DSP Запись (команды/данные) ,
состояние буфера записи DSP(Бит 7)
2xEh - Состояние буфер чтения DSP (Бит 7),
подтверждение прерывания DSP
Пятый порт только для Sound Blaster 16
2xFh - подтверждение прерывания DSP с 16 битами

Где x = 1 для базового адреса 210h.
x = 2 для базового адреса 220h.
.
.
x = 6 для базового адреса 260h.

Сброс DSP

Необходимо сбросить DSP прежде, чем начать работу с ним. Сброс осуществляется
по следующему алгоритму:

1) Запишите 1 в порт сброса (2x6)
2) Ждите 3 микросекунды
3) Запишите 0 в порт сброса (2x6)
4) Читайте порт состояния буфера чтения (2xE) пока бит 7 = 1

5) Опрашивайте порт данных чтения (2xA) пока вы не получите AAh.

Для сброса DSP требуется около 100 мкс. Если после этого вы не получили AAh. Значит, либо нет звуковой платы, или задан был неверный базовый адрес.

Запись в DSP

Значение можно записыть в DSP с следующей процедурой:

1) Читайте порт состояния буфера записи DSP (2xCh) пока бит 7 = 0
2) Запишите значение в порт записи (2xCh)

Чтение DSP

Значение можно прочитать из DSP с следующей процедурой:

1) Читайте порт состояния буфера чтения (2xE) пока бит 7 = 1
2) Читайте значение из порта чтения (2xA)

Прямой режим вывода на DAC

DAC - это та часть платы которая преобразовывает значение выборки амплитуды (то есть 0 - 255) к звуку. Чтобы генерировать квадратную звуковую волну с максимальной громкостью вы должны чередовать запись 0/255 на DAC. Только 8-разрядный DAC доступен в прямом режиме. В прямом режиме программа ответственна за выбор определенного времени между выборками, DAC может выводить звуковые выборки с такой скоростью с какой программа способна это делать.

Команды DSP для программирования DAC в прямом режиме:
10h - вывести данные на DAC без DMA 8 бит. Для использования команды необходимо:
1) Выдать 10h
2) Записать байт данных.
3) Продолжить через время T до окончания.
D1h - Включить воспроизведение звука.
D3h - Выключить воспроизведение звука.
20h - получить данные из ADC 8 бит.

Программирование DAC с DMA

DAC можно также запрограммировать, чтобы он принимал значения посланные ему через DMA.

Команды DAC:
14h - Включение вывода на DAC 8 Бит 4KHz - > 23 KHz. Затем младший байт длины, потом старший.
24h - Включение записи с ADC 8 Бит 4KHz - > 23 KHz
74h - Включение вывода на DAC 4 Бит ADPCM 4KHz - > 12 KHz

75h - Включение вывода на DAC 4 Бит ADPCM с 4KHz - > 12 KHz с байтом ссылки
76h - Включение вывода на DAC 2.6 Бит ADPCM 4KHz - > 13 KHz
77h - Включение вывода на DAC 2.6 Бит ADPCM с 4KHz - > 13 KHz с байтом ссылки
16h - Включение вывода на DAC 2 Бит ADPCM 4KHz - > 11 KHz
17h - Включение вывода на DAC 2 Бит ADPCM с 4KHz - > 11 KHz с байтом ссылки

ADPCM (Адаптивная Импульсно-кодовая Модуляция) - это звуковая методика сжатия, где различие между последовательными выборками сохраняется скорее чем их фактические значения. В режимах с байтами ссылки, первый байт - фактическое начальное значение. Наличие режимов с и без байтов ссылки значит что вы можете выводить последовательные блоки без наличия байта ссылки.

Bxh – программмирование режима DMA с 16 битным цифровым звуком.
(Только для SB16)
Командная последовательность:
Команда, Режим, Lo(Length-1), Hi(Length-1):

Первый байт команды состоит:

D7	D6	D5	D4	D3	D2	D1	D0
1	0	1	1	A/D	A/I	FIFO	0
				0=D/A 1=A/D	0=SC 1=AI	0=off 1=on	

Общие команды:
B8 - одиночный цикл с 16-битной записи звука
B0 - одиночного цикла с 16-битным воспроизведением
BE - автоинициализируемая 16-битная запись
B6 - автоинициализируемое 16-битной воспроизведение

Режим:

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0-mono 1-stereo	0-без знака 1-со знаком	0	0	0	0

Cxh - программирование режим DMA с 8-битным цифровым звуком.
(Только для SB16)
Те же самые команды, что для 16-бит.
C8 - одиночный цикл с 8-битной записи звука
C0 - одиночного цикла с 8-битным воспроизведением
CE - автоинициализируемая 8-битная запись
C6 - автоинициализируемое 8-битной воспроизведение

FIFO используется чтобы удалять несогласованности в тот период выборки, когда звуковая плата не способна получить DMA, когда это требуется. Без FIFO плата делает попытку захвата DMA в точно тот момент, когда требуется выборка. Если другое устройство с более высоким приоритетом обращается к

DMA, звуковая плата ожидает и скорость выборки может уменьшаться. FIFO позволяет во время выборки с DMA быть более гибким DSPбез потери звукового качества. FIFO очищается всякий раз, когда команда посылается DSP. В режиме одиночного цикла, DSP постоянно перепрограммируется. С FIFO DSP может еще содержать данные, который не были выданы, когда команда очистила DAC. Чтобы избежать этого, FIFO должен быть переключен в режим с одиночным циклом. Затем, снова переведен в автоинициализируемый режим, когда DSP не перепрограммируется.

1Ch - Включение вывода на DAC 8 Бит 4KHz - > 23 KHz с автоинциализацией
90h - Включение вывода на DAC 8 бит 4kHz - > 44 KHz с автоинициализацией
48h - Установить длину блока на пересылку перед посылкой 91h, 99h
сначала младший байт затем старший длину.
91h - Включение вывода на DAC 8 бит 4kHz - > 44 KHz стерео
99h - Включение записи с ADC 8 бит 4kHz - > 44 KHz стерео
D0h - остановить 8-битный DMA
D4h - возобновить 8-битный DMA
D5h - остановить 16-битный DMA
D6h - возобновить 16-битный DMA
Эти команды пригодны как и для автоинциализированного режима, так и для одиночных циклов.

D9h - Выход из авто инициализируемого режима DMA с 16 битами после окончания текущего блока.
DAh - Выход из авто инициализируемого режима DMA с 8 битами после окончания текущего блока.

E1h - Получить номер версии DSP. После посылки этой команды, прочитайте из DSP два байта. Первый байт - главный номер версии и второй байт - малый номер версии. Версия 4.00 - это SB16.

Версия	Стерео	Частота	FIFO	16 бит
<2.00	-	До 21379	-	-
>=2.00	-	До 21379	+	-
>=2.01	-	До 43478	+	-
>=3.00	+	До 43478	+	-
>=3.01	+	До 43478	+	+

Программирование DMA

Контроллер DMA (Прямого Доступа В память) управляет пересылками данных между устройствами ввода/вывода и памятью без использования центрального процессора. IBM совместимая ЭВМ имеет два контроллера DMA один для пересылок с 8 битами и другой для пересылок с 16 битами. Контроллер DMA, вместе с внешним регистром страницы, способен на перемещение блоков по 64 КБ. Ниже приведена информация по программированию DMA.

Адреса портов для адреса DMA и регистров счета.

Контроллер	Адрес	Функция
DMA1 с 8 битами Подчиненный	00 01 02 03 04 05 06 07	Канал 0 адреса Канал 0 счета Канал 1 адреса Канал 1 счета Канал 2 адреса Канал 2 счета Канал 3 адреса Канал 3 счета
DMA2 с 16 битами Ведущий	C0 C2 C4 C6 C8 CA CC CE	Канал 4 адреса Канал 4 счета Канал 5 адреса Канал 5 счета Канал 6 адреса Канал 6 счета Канал 7 адреса Канал 7 счета

Адреса портов для регистров управления

Адрес		Операция	Функция
DMAC1	DMAC2		
0A	D4	Запись	Регистр маски
0B	D6	Запсиь	Регистр режима
0C	D8	Запись	Регистр сброс байта flip-flop

Адреса портов для младших регистров страницы

Адрес	Функция
81	2 канал DMA с 8 битами
82	3 канал DMA с 8 битами
83	1 канал DMA с 8 битами
87	0 канал DMA с 8 битами
89	6 канал DMA с 16 битами
8A	7 канал DMA с 16 битами
8B	5 канал DMA с 16 битами

Бита регистра режима

Бит	Функция
Биты 7:6 00 01 10 11	Биты выбора режима Выбранный режим запроса Одиночный выбранный режим Выбранный блочный режим Каскадный выбранный режим
Бит 5 1 0	Бит приращения/декремента адреса Выбранный декремент адреса Выбранное приращение адреса
Бит 4 1 0	Автоинициализация Автоинициализация включена Одиночный
Биты 3:2 00 01 10 11 **	Биты пересылки Проверьте пересылку Запишите пересылку (к памяти) Читайте пересылку (из памяти) Запрещенный Игнорируется если биты 7:6 = 11
Биты 1:0 00 01 10 11	Биты выбора канала Выберите канал 0 (4) Выберите канал 1 (5) Выберите канал 2 (6) Выберите канал 3 (7)

Биты маски записи

Бит	Функция
Биты 7:3	Неиспользуемый (набор к 0)
Бит 2 1 0	Установить/сбросить бит маски Установить бит маски(Отключить канал) Сбросит бит маски (Доступен канал)
Биты 1:0 00 01 10 11	Биты выбора канала Канал 0 (4) Канал 1 (5) Канал 2 (6) Канал 3 (7)

DMAC2 используется для работы с 16 битами и DMAC1 используется для работы с 8 битами. Вот пример программирования DMA:

1) Вычислите абсолютный линейный адрес вашего буфера

```
LinearAddr = MK_SEG( Buf ) * 16L + MK_OFF ( Buf );
```

2) Отключите канал DMA звуковой платы установкой бита маски

```
outp(MaskPort, 1 + ( DMAChannel % 4 ));
```

3) Очистите указатель байта flip-flop

```
outp(ClrPort, DMAChannel );
```

4) Запишите режим DMA для пересылки
Биты выбора режима должны устанавливаться в 00h для режима запроса.
Адрес бита +/- должен устанавливаться в 0 для приращения адреса.
Бит автоинициализации должен устанавливаться соответственно.
Биты пересылки должны устанавливаться в 10h для воспроизведения и 01h для записи. Выбор канала должен устанавливаться так же как и на канал DMA звуковой платы.

```
outp(ModePort, Mode + ( DMAChannel % 4 ));
```

Некоторые часто используемые режимы:

48h + Канал - одиночный цикл воспроизведение
58h + Канал - автоинициализируемое воспроизведение
44h + Канал - запись одиночного цикла
54h + Канал - автоинициализируемая запись

5) Запишите смещение буфера, младший байт затем старший байт. Для шестнадцати разрядных данных, смещение должно быть в словах от начала 128k-байтной страницы, для 8-битных от 64K. Самый простой метод для вычисления смещения с 16 битами - это разделить линейный адрес на два перед вычислением смещения.

6) Запишите длину пересылки в соответствующий порт счета, младший байт затем старший байт. Для пересылки с 8 битами, запишите длину в байтах минус единица. Для пересылки с 16 битами, запишите номер длину в словах минус единица.

7) Запишите страницу буфера в регистр страницы DMA.

```
outp(PagePort, ( LinearAddr / 65536));
```

8) Включите DMA звуковой платы очистив соответствующий бит маски

```
outp(MaskPort, DMAChannel % 4);
```

Установка частоты выборки

Для версии Sound Blaster ниже 4.00 установка частоты выборки выполняется посылкой DSP команды 40h. При этом частота преобразуется к константе времени
по формуле:

4KHz - > 23 KHz:

```
Time Constant = 256 - (1,000,000 / sampling rate)
              = 256 - (1,000,000 / 8,000 )
              = 131
```

4KHz - > 44 KHz:

```
Time Constant = (MSByte of) 65536 - (256,000,000 / sampling rate)
              = (MSByte of) 65536 - (256,000,000 / 44,100)
              = (MSByte of) 59731
              = (MSByte of) 0E953h
              = 0E9h
```

В отличие от этого SB16 программируется фактической частотой выборки.
Команда 41h используется для воспроизведения, а 42h используется для записи.

```
if ( Play==1 )
        WriteSB ( 0x41 );
else    WriteSB ( 0x42 );
WriteSB ( hi( frequency ) );
WriteSB ( lo( frequency ) );
```

Алгоритм цифрового ввода/вывода звука

Чтобы записывать или воспроизводить звук, вы должны использовать следующую
последовательность:

1) Распределите буфер который не пересекает границу 64 Kb
2) Установите программу обработки прерывания.
3) Запрограммируйте контроллер DMA для фоновой пересылки
4) Установите частоту выборки
5) Запишите команду I/O в DSP
6) Запишите режим пересылки I/O в DSP
7) Запишите размер блока в DSP (Младший байт/Старший байт)

После этого сразу начнется запись или воспроизведение звука.

Ниже приведены примеры последовательностей для программирования SB с помощью DMA.
Нормальная частота, воспроизведение:

1) Записать D1h в 2xCh
2) Установить обработчик прерывания
3) Записать 40h в 2xCh
4) Записать константы времени в 2xCh
5) Запрограммировать DMA
6) Записать 14h в 2xCh
7) Записать длину выборки
8) Обслуживать прерывания, до окончания выборки
9) Восстановить старый обработчик
10) Записать D3h в 2xCh

При этом можно записывать любые команды в DSP, пока идет воспроизведение.

Повышенная частота, воспроизведение:

1) Записать D1h в 2xCh
2) Установить обработчик прерывания
3) Записать 40h в 2xCh
4) Записать константы времени в 2xCh
5) Запрограммировать DMA
6) Записать 48h в 2xCh
7) Записать длину выборки

8) Записать 91h в 2xCh
9) Обслуживать прерывания, до окончания выборки
10) Восстановить старый обработчик
11) Записать D3h в 2xCh

Нормальная частота, запись звука:

1) Установить обработчик прерывания
2) Записать 40h в 2xCh
3) Записать константы времени в 2xCh
4) Запрограммировать DMA
5) Записать 24h в 2xCh
6) Записать длину выборки
7) Обслуживать прерывания, до окончания выборки
8) Восстановить старый обработчик

При этом можно посылать любые команды в DSP, пока идет запись.

Повышенная частота, запись:

1) Установить обработчик прерывания
2) Записать 40h в 2xCh
3) Записать константы времени в 2xCh
4) Запрограммировать DMA
5) Записать 48h в 2xCh
6) Записать длину выборки
7) Записать 99h в 2xCh
8) Обслуживать прерывания, до окончания выборки
9) Восстановить старый обработчик

Конец цифрового ввода/вывода звука

Когда пересылка закончена генерируется прерывание. Фактический номер прерывания зависит от установки IRQ на плате Sound Blaster:

IRQ	Прерывание
2	0Ah
3	0Bh
5	0Dh
7	0Fh

Для обслуживания прерывания необходимо выполнить:

1) подтвердите прием прерывания от DSP прочитав порт (2xEh) один раз для 8-битного звука, или порт 2xF для 16-битного звука.
2) Вывод следующего буфера, если есть.
3) Выведите значение 20h (EOI) в порт контроллера прерывания 20h, а если IRQ8-15(прерывания 70h-77h), то записать 20h в A0h.

Стерео звук

При воспроизведении стерео звуков необходимо посылать 2 байта DSP, первый для левого канала, второй для правого. Необходимо так же указать SB, что вы воспроизводите стерео звук, через регистры миксера.

Миксер Sound Blaster

Ниже приведена информация для SbPro.
Порт 2x4h - индексный порт миксера, 2x5h - порт данных (чтения/записи).

Регистр Сброса Данных используется для инициализации миксера. Установите этот регистр в 0 перед изменением любого из других регистров миксера.

Регистр записи определяет источник звука и тип фильтра.

Индекс = 0Ch

7	6	5	4	3	2	1	0
		В фильтре 000-Низкие 001-Высокие 010-Нет фильтра			ADC источник 00-Микрофон1 01-CD 10-Микрофон2 11-Линейный вход		

Регистр воспроизведения служит для установки фильтра и стерео звука.

Индекс = 0Eh

7	6	5	4	3	2	1	0
		0-использовать фильтр 1-без фильтра				0-моно 1-стерео	

Регистр общей громкости:

Индекс = 22h

7	6	5	4	3	2	1	0
Громкость лево				Громкость право			

Регистр громкости DSP:

Индекс = 04h

7	6	5	4	3	2	1	0
Громкость лево				Громкость право			

Регистр громкости FM синтезатора:

Индекс= 26h

7	6	5	4	3	2	1	0
Громкость лево				Громкость право			

Регистр громкости CD:

Индекс = 28h

7	6	5	4	3	2	1	0
Громкость лево				Громкость право			

Регистр громкости линейного входа:

Индекс = 2Eh

7	6	5	4	3	2	1	0
Громкость лево				Громкость право			

Регистр громкости микрофона:

Индекс = 0Ah

7	6	5	4	3	2	1	0
					Громкость микрофона		

Описание модулей программы.

Программа написана с использование объектно-ориентированного программирования и стандартной библиотеки Turbo Vision. Исходные тексты программы содержатся в 16 модулях. Рассмотрим их подробно.

Модуль aboutosc.

В модуле определен объект TaboutDialog наследник объекта Dialog.

```
TAboutDialog = object(TDialog)
   constructor Init;
  end;
```

Служит для визуализации информации о разработчиках программы.

Модуль clock.

В модуле определен объект TclockView и процедура GetTime.

```
TClockView = Object(TView)
   Refresh: Byte;
   LastSec: Byte;
   TimeStr: String[10];
   SaverTime:Word;
   TimeCount:Word;
   Button:Word;
   Constructor Init(Var Bounds: TRect);
   Procedure Draw; virtual;
   Function ByteToStr(AByte:Byte):String;
   Procedure Update; Virtual;
   Procedure SetSaverTime;
  End;

Procedure GetTime(Var Hour,Min,Sec:Byte);
```

TclockView – объект визуализирующий часы на экране. Процедура GetTime возвращает системное время.

Модуль helpfile.

В модуле определены объекты, процедуры и функции для работы с файлом помощи. Данный модуль является стандартным и поставляется вместе с Borland Pascal 7.0.

Модуль oscconst.

В модуле описаны все константы использующиеся в других модулях программы.

Константы служащие для управления событиями от пунктов меню:

```
cm_SetupHardwareIrq2=201;
 cm_SetupHardwareIrq3=202;
 cm_SetupHardwareIrq5=203;
 cm_SetupHardwareIrq7=204;
 cm_SetupHardwareDMA0=205;
 cm_SetupHardwareDMA1=206;
 cm_SetupHardwareDMA3=207;
 cm_SetupHardwareDMA5=208;
 cm_SetupHardwareDMA6=209;
 cm_SetupHardwareDMA7=210;
 cm_SetupFrequence6024=211;
 cm_SetupFrequence8000=212;
 cm_SetupFrequence12048=213;
 cm_SetupFrequence16129=214;
 cm_SetupFrequence22222=215;
 cm_SetupFrequence30303=216;
 cm_SetupFrequence41667=217;
 cm_SetupFrequence45454=218;
 cm_SetupGraphMode101=220;
 cm_SetupGraphMode103=221;
 cm_SetupGraphMode105=222;
 cm_HelpAbout=223;
 cm_Oscillograph=224;
 cm_Help=225;
 cm_SetupCoefficient=226;
 cm_VoltMetr=227;
 cm_SetupGarmonicNumber=228;
 cm_ShowScreenSaver=229;
 cm_ScreenSaver=230;
 cm_Clock=231;
 cm_Information=232;
 cm_FindCurSetWindow=233;
 cm_RedrawCurSetWindow=234;
 cm_SetupHardwareDeviceSB=236;
 cm_SetupHardwareDeviceSB16=237;
 cm_Statist=238;
 cm_Volume1=239;
 cm_Volume2=240;
 cm_Volume3=241;
```

```
  cm_Volume4=242;
  cm_Volume5=243;
  cm_Volume6=244;
  cm_Volume7=245;
  cm_Volume8=246;
  cm_Volume9=247;
  cm_Volume10=248;
  cm_Volume11=249;
  cm_Volume12=250;
  cm_Volume13=251;
  cm_Volume14=252;
  cm_Volume15=253;
  cm_SetupFreqCorrection=254;
  cm_CalcConstSost=255;
```

Константы имен файлов, используемых программой:

```
  ConfigFileStr='oscillo.cfg';  - конфигурационный файл
  FontFileStr='oscillo.chr'; - файл со шрифтом
  HelpFileStr='oscillo.hlp'; - файл помощи
  SpriteFileStr='oscillo.spr'; - файл с картинками
  SaverFileStr='oscillo.ssv'; - файл-гасилка экрана
```

Коэффициенты и др.:

```
Divider:Real=21846;
  Irq:Word=5;
  DMA:Word=1;
  Rate:Word=45454;
```

Константы палитры:

```
  CNewColor = CAppColor + CHelpColor;
  CNewBlackWhite = CAppBlackWhite + CHelpBlackWhite;
  CNewMonochrome = CAppMonochrome + CHelpMonochrome;
  MainPalette: Array[apColor..apMonochrome] Of String[Length(CNewColor)] =
  (CNewColor, CNewBlackWhite, CNewMonochrome);
```

Модуль oscgraph.

В модуле определены следующие процедуры и функции:

```
Function Oscillgraph(X1,Y1,X2,Y2:Word):Boolean;
```

Ииициализация осциллографа и его дальнейшая работа.

```
Procedure OscillogrammaSB(X1,Y1,X2,Y2:Word);
```

Оцифровка сигнала и вывод осциллограммы на экран.

```
Procedure OscillogrammaWithAxis(X1,Y1,X2,Y2:Word);
```

Вывод осциллограммы на экран с осями кооординат.

Модуль oscillo.

В модуле определены следующие обекты:

```
TOscilloApplication=Object(TApplication)
    ClockView:PClockView;
```

```
   Constructor Init;
   Procedure HandleEvent(Var Event: TEvent);Virtual; обработчик событий
   Procedure WriteShellMsg;Virtual; выводит сообщение при временном выходе в
DOS
   Procedure InitDeskTop;Virtual;
   Procedure OutOfMemory;Virtual;
   Procedure InitMenuBar;Virtual;
   Procedure InitStatusLine;Virtual;
   Function GetPalette:PPalette;Virtual;
   Procedure GetEvent(Var Event:TEvent);Virtual;
   Procedure ShowScreenSaver; запуск гасилки экрана
   Procedure Idle;Virtual;
   Destructor Done;Virtual;
 End;

 PBaseDeskTop=^TBaseDeskTop;
 TBaseDeskTop=Object(TDeskTop)
  Procedure InitBackGround;Virtual;
 End;
```

Так как эти объекты являются наследниками стандартных объектов Turbo Vision, то опустим их описание. Описание процедур и функций можно найти в учебнике по Turbo Vision.

Модуль oscillo1.

В модуле определены константы используемые для визуализации помощи.

Модуль saver.

В модуле определен объект TscreensavDialog.

```
TScreenSavDialog = object(TDialog)
   constructor Init;
 end;
```

Служит для визуализации диалога выбора времени, по истечении которого необходимо запускать ScreenSaver.

Модуль sbmixer.

В модуле определены следующие процедуры, функции, константы и переменные:

Откуда брать сигнал:

```
 Source_Mic1=      0;  с микрофона 1
 Source_CD=        2;  с CD ROM'a
 Source_Mic2=      4;  с микрофона 2
 Source_Line=      6;  с линейного входа
```

Наличие фильтра:

```
 Filtr_Low=   0;        фильтр нижних частот
 Filtr_High=  8;        фильтр высоких частот
 Filtr_None=  16;       отсутствие фильтра
```

Громкость по входу:

```
 Volume:Byte=  15;
```

Procedure WriteMixerSB(Index,Value:Byte); запись команды-байта в порт миксера
Function ReadMixerSB(Index:Byte):Byte; чтение состояния порта миксера
Procedure ResetMixerSB; инициализация миксера
Procedure InputMixerSB(Source,Filtr:Byte); настройка миксера
Procedure LineVolumeSB(Left,Right:Byte); настройка громкости входа

Модуль sbprodma.

Модуль содержит функции и процедуры работы с Sound Blaster'ом.
Управляющие команды:

```
WRITE_DATA_SB   =       $10; команда посылки байта на DSP
ON_SOUND_SB     =       $D1;включение звука
OFF_SOUND_SB    =      $D3;   выключение звука
HALT_DMA_8      =       $D0; прерывание 8-ми битной пересылки
TIME_CONSTANT =         $40;  команда установки частоты для SB Pro
FREQ_CONSTANT =     $42;     команда установки частоты для SB16
DMA_8_BIT_ADC_HI=       $99;  команда начала 8-ми битной оцифровки
SET_LEN_DMA_8_BIT=      $48;  длина оцифровки
```

Количество базовых портов для Sound Blaster'а:

```
MAX_BASE_SB         =5;
```

Базовые порты Sound Blaster'а:

```
BasesSB:Array [0..MAX_BASE_SB-1] of Word=( $220, $230, $240, $250, $260 );
```

Текущий базовый адрес Sound Blaster'а:

```
baseAddrSB:Word=$220;
```

Текущий DMA канал:

```
DMAChannel:Byte=1;
```

Текущее IRQ Sound Blaster'а:

```
SbIRQ:Byte=5;
```

Версия Sound Blaster'а:

```
VerSB:Word=$201;
Fifo:Boolean=False;
```

Поддержка 16-и битной оцифровки:

```
SixteenBits:Boolean=False;
DeviceSB16:Boolean=False;
```

Поддерживает ли повышенную частоту оцифровки до 44кГц

```
MaxFrequence:Boolean=False;
```

Поддерживает ли стерео оцифровку:

```
Stereo:Boolean=False;
```

Type

TArray=Array[0..8191] of ShortInt; тип данных, получаемых с Sound Blaster'a

Var
Data:^TArray; буфер для записи
DosSelector:Word; селектор адреса буфера в real-режиме работы процессора
DPMISelector:Word; селектор адреса буфера в protected-режиме работы процессора
DMA_PAGE:Byte; физическая страница памяти, в которой находится буфер
DMA_POFF:Word; смещение в этой странице
OldIRQ:Pointer; указатель на предыдущий обработчик прерывания
DMA_complete:Boolean; флаг завершения оцифровки

Procedure SBHandler;Interrupt; обработчик прерывания конца оцифровки
Procedure RateSB(Rate:Word); установка частоты дискретизации
Function CheckSB:Boolean; проверка наличия и инициализация SB
Function ReadSB:Byte; чтение байта из порта SB
Procedure WriteSB(Value:Byte); запись байта в порт SB
Function SetupSB(Irq,Dma:Integer):Boolean; инициализация SB
Procedure ResetSB; сброс SB
Procedure RecordSB(Len:Word); начать оцифровку
Procedure VersionSB; возвращает версию SB

Модуль setwin.

В модуле определен объект TcurSetWindow.

```
TCurSetWindow=Object(TWindow)
   Constructor Init;
   Procedure HandleEvent(Var Event:TEvent);Virtual;
   Procedure Draw;Virtual;
 End;
```

Служит для визуализации текущих установок программы.

Модуль sprlib.

В модуле реализованы функции для работы с библиотекой рисунков.
Var LibPointer:Pointer; указатель на загруженную библиотеку
Function LoadLibrary(Path:String):Pointer; загрузка библиотеки
Function FreeLibrary(LibPtr:Pointer):Pointer; освобождение памяти из-под библиотеки
Function GetSpriteOffset(LibPtr:Pointer;N:Word):Pointer; возвращает указатель на начало N-го спрайта в библиотеке.

Модуль statint.

Модуль реализует диалог настройки временного интервала, по истечении которого следует выдавать рассчитанные данные в режиме накопления статистики.

```
type
  StatIntDataRec = record
    Hours : LongInt;
    Minutes : LongInt;
    Seconds : LongInt;
    end;
```

```
 PStatIntDataRec = ^StatIntDataRec;

 PStatIntDialog = ^TStatIntDialog;
 TStatIntDialog = object(TDialog)
  constructor Init;
end;
```

Модуль statist.

В модуле реализованы процедуры и функции режима накопления статистики.
Function Frequence(Var Data:TWorkWords):Single; подсчет частоты сигнала
Function StatMaker:Boolean; главная функция режима накопления статистики
Procedure StatistSB; накопление статистики
Function CalculateTime(Hours,Minutes,Seconds:LongInt):LongInt; перевод времени в LongInt-число
Procedure MakeOutString(Hours,Minutes,Seconds:Byte;MaxVolt,MinVolt,RealVolt,Frequence: Single;Stroka:PChar); формирование выходной строки

Модуль svgadrv1.

Модуль предназначен для работы с графикой. Практически все процедуры и функции модуля написаны на языке Assembler для наибольшей скорости работы с графикой.

Модуль voltmetr.

Модуль реализует работу вольтметра.

```
Type
   TWorkWords=Array [0..4095] Of Integer; рабочий массив

Var
   Sequence:TWorkWords;

Const
   NumGarmonic : Integer = 30; количество гармоник, на которые следует разбивать входной сигнал
   CalcConstSost: Boolean = True; вычисление или нет постоянной составляющей

Function VoltoMetr:Boolean;
Procedure OutVoltMetr; вывод на экран подсчитанного значения
Function RealVoltage(Var YourData:TWorkWords):Single; подсчет действующего значения
Procedure DigitalizationSB; оцифровка
Procedure WriteVoltMetrImage; изображение всяких рамочек
```

Исследование возможностей контроля качества сетевого питания с помощью Sound Blaster’a.

В ходе исследований выявились некоторые недостатки работы Sound Blaster’ов, которые описаны в пункте **«Результаты исследования»** .

Исходя из выше перечисленных результатов, можно сделать следующие выводы:

Использование 8-ми битных Sound Blaster’ов для контроля качества сетевого питания является нецелесообразным из-за низкой точности оцифроки сигнала. В отличие от использования 8-ми битных Sound Blaster’ов, 16-ти битные дают значительно большую точность оцифровки сигнала. Поэтому 16-ти битные Sound Blaster’ы возможны для использования в качестве средства контроля за параметрами исходного сигнала. А учитывая, что множество параметров можно програмно регулировать, то использование возможно. Так как полноценные исследования не были проведены, ввиду малых сроков разработки и исследования программы, то желательно провести более детальную проверку возможностей Sound Blaster’ов и конкретно проверку по точности получаемых и рассчитываемых параметров. Положителной возможностью является то, что можно вычислять любые из возможных параметров сигналов, которые (параметры) можно вычислять на основе массива оцифрованных данных.

Исходные тексты модулей программы.

```
unit ABOUTOSC;

interface

uses Drivers, Objects, Views, Dialogs;

type

  PAboutDialog = ^TAboutDialog;
  TAboutDialog = object(TDialog)
    constructor Init;
  end;

implementation

constructor TAboutDialog.Init;
var
  R: TRect;
  Control : PView;
begin
R.Assign(20, 2, 59, 21);
inherited Init(R, 'About');
Options := Options or ofCenterX or ofCenterY;

R.Assign(4, 2, 35, 15);
Control := New(PStaticText, Init(R, '       PC Oscillograph'^M+
    '         Version 3.0'^M+
    "^M+
    ' Idea by Serge Volkov (SerVo)'^M+
    "^M+
    '          Coded by:'^M+
    '        Serge Volkov'^M+
    '     Konstantin Stroganov'^M+
    '       Vladimir Mihajlov'^M+
    '       Vitaliy Sidorenko'^M+
    "^M+
    '     Copyright (c) 1998  by'^M+
    '          Serge Volkov'));
Insert(Control);

R.Assign(14, 16, 24, 18);
Control := New(PButton, Init(R, 'O~K~', cmOK, bfDefault));
Insert(Control);

SelectNext(False);
end;
```

```
end.

Unit Clock;
Interface
Uses App,Views,Objects,Drivers,OscConst,Dialogs,Saver;

Type
  PClockView = ^TClockView;
  TClockView = Object(TView)
    Refresh: Byte;
    LastSec: Byte;
    TimeStr: String[10];
    SaverTime:Word;
    TimeCount:Word;
    Button:Word;
    Constructor Init(Var Bounds: TRect);
    Procedure Draw; virtual;
    Function ByteToStr(AByte:Byte):String;
    Procedure Update; Virtual;
    Procedure SetSaverTime;
  End;

Procedure GetTime(Var Hour,Min,Sec:Byte);

Implementation

Procedure GetTime(Var Hour,Min,Sec:Byte);Assembler;
Asm
    xor ax,ax
    mov ah,02h
    int 1ah
    mov bl,16
    mov bh,10

    mov al,ch
    div bl
    mov dl,ah
    mul bh
    add al,dl
    les di,Hour
    stosb

    mov al,cl
    div bl
    mov dl,ah
    mul bh
    add al,dl
    les di,Min
    stosb
```

```
    mov al,dh
    div bl
    mov dl,ah
    mul bh
    add al,dl
    les di,Sec
    stosb
End;

Procedure TClockView.SetSaverTime;
Var ResFile:TResourceFile;
Begin
  If
Application^.ExecuteDialog(New(PScreenSavDialog,Init),@Button)<>cmCan
cel Then
  Begin
    TimeCount:=0;
    Case Button Of
      0:SaverTime:=0;
      1:SaverTime:=60;
      2:SaverTime:=180;
      3:SaverTime:=300;
      4:SaverTime:=600;
      Else SaverTime:=0;
    End;
  End;
End;

Constructor TClockView.Init(Var Bounds: TRect);
Begin
  Inherited Init(Bounds);
  TimeStr := '';
  Refresh := 1;
  SaverTime:=0;
  TimeCount:=0;
  Button:=0;
End;

Procedure TClockView.Draw;
Var
  B: TDrawBuffer;
  C: Byte;
Begin
  C := GetColor(2);
  MoveChar(B, ' ', C, Size.X);
  MoveStr(B, TimeStr, C);
  WriteLine(0, 0, Size.X, 1, B);
End;

Procedure TClockView.Update;
```

```
Var
  H,M,S: Byte;
  Stroka:String;
Begin
  GetTime(H,M,S);
  If Abs(S - LastSec) >= Refresh Then
  Begin
    If (SaverTime<>0) And (TimeCount>SaverTime) Then
    Begin
      TimeCount:=0;
      Message(Application,evCommand,cm_ShowScreenSaver,Nil);
    End;
    LastSec:=S;
    Inc(TimeCount,Refresh);
    TimeStr:=' '+ByteToStr(H)+':'+ByteToStr(M)+':'+ByteToStr(S);
    DrawView;
  End;
End;

Function TClockView.ByteToStr(AByte:Byte):String;
Var Stroka:String[3];
Begin
  Str(AByte:2,Stroka);
  If Stroka[1]=' ' Then Stroka[1]:='0';
  ByteToStr:=Stroka;
End;

End.

Unit OscConst;
Interface
Uses App,HelpFile;

Const
  cm_SetupHardwareIrq2=201;
  cm_SetupHardwareIrq3=202;
  cm_SetupHardwareIrq5=203;
  cm_SetupHardwareIrq7=204;
  cm_SetupHardwareDMA0=205;
  cm_SetupHardwareDMA1=206;
  cm_SetupHardwareDMA3=207;
  cm_SetupHardwareDMA5=208;
  cm_SetupHardwareDMA6=209;
  cm_SetupHardwareDMA7=210;
  cm_SetupFrequence6024=211;
  cm_SetupFrequence8000=212;
  cm_SetupFrequence12048=213;
  cm_SetupFrequence16129=214;
  cm_SetupFrequence22222=215;
  cm_SetupFrequence30303=216;
```

```
cm_SetupFrequence41667=217;
cm_SetupFrequence45454=218;
cm_SetupGraphMode101=220;
cm_SetupGraphMode103=221;
cm_SetupGraphMode105=222;
cm_HelpAbout=223;
cm_Oscillograph=224;
cm_Help=225;
cm_SetupCoefficient=226;
cm_VoltMetr=227;
cm_SetupGarmonicNumber=228;
cm_ShowScreenSaver=229;
cm_ScreenSaver=230;
cm_Clock=231;
cm_Information=232;
cm_FindCurSetWindow=233;
cm_RedrawCurSetWindow=234;
cm_SetupHardwareDeviceSB=236;
cm_SetupHardwareDeviceSB16=237;
cm_Statist=238;
cm_Volume1=239;
cm_Volume2=240;
cm_Volume3=241;
cm_Volume4=242;
cm_Volume5=243;
cm_Volume6=244;
cm_Volume7=245;
cm_Volume8=246;
cm_Volume9=247;
cm_Volume10=248;
cm_Volume11=249;
cm_Volume12=250;
cm_Volume13=251;
cm_Volume14=252;
cm_Volume15=253;
cm_SetupFreqCorrection=254;
cm_CalcConstSost=255;
ConfigFileStr='oscillo.cfg';
FontFileStr='oscillo.chr';
HelpFileStr='oscillo.hlp';
SpriteFileStr='oscillo.spr';
SaverFileStr='oscillo.ssv';
Divider:Real=21846;
Irq:Word=5;
DMA:Word=1;
Rate:Word=45454;
CNewColor = CAppColor + CHelpColor;
CNewBlackWhite = CAppBlackWhite + CHelpBlackWhite;
CNewMonochrome = CAppMonochrome + CHelpMonochrome;
```

```
  MainPalette:                Array[apColor..apMonochrome]                Of
String[Length(CNewColor)] =
  (CNewColor, CNewBlackWhite, CNewMonochrome);

Implementation

End.

Unit OscGraph;
Interface
Uses
SVGADrv1,SBProDMA,SBMixer,WinAPI,Crt,OscConst,SprLib,VoltMetr;

Function Oscillgraph(X1,Y1,X2,Y2:Word):Boolean;

Implementation

Procedure OscillogrammaSB(X1,Y1,X2,Y2:Word);
Var Stroka:String;
    Power:Real;
    R:Integer;
Begin
  ClearRectBuffer(X1,Y1,X2,Y2);
  Asm
    les di,Data
    mov cx,1024
    mov ax,8080h
    rep stosw
  End;
  InputMixerSB(Source_Line,Filtr_None);
  LineVolumeSB(Volume,Volume);
  RateSB(Rate);
  Repeat
    If SixteenBits Then
    Asm
      mov bx,ds
      mov ax,Seg Sequence
      mov es,ax
      mov di,Offset Sequence
      lds si,Data
      mov cx,1024
@@1:  lodsw
      sub ax,32768
      stosw
      loop @@1
      mov ds,bx
    End
    Else
    Asm
      mov bx,ds
```

```
    mov ax,Seg Sequence
    mov es,ax
    mov di,Offset Sequence
    lds si,Data
    mov cx,1024
@@1:  xor ax,ax
    lodsb
    sub al,128
    cbw
    shl ax,8
    stosw
    loop @@1
    mov ds,bx
  End;

  Asm
    mov ax,X2
    sub ax,X1
    inc ax
    push ax
    call RecordSB
  End;
  ClearRectBuffer(X1,Y1,X2,Y2);
  Asm
    push X1
    mov bx,128
    mov ax,Y1
    add ax,bx
    push ax
    push X2
    push ax
    push 12
    call Line
  End;
  Asm
    mov cx,X2
    sub cx,X1
    dec cx                  {устранение глюков для SB16}

@@lp1:push cx

    mov dx,cx
    add dx,X1
    push dx

    mov bx,cx
    mov cx,Seg Sequence
    mov es,cx
    mov di,Offset Sequence
    inc di                  {устранение глюков для SB16}
```

```
  inc di

  shl bx,1
  add di,bx
  mov ax,es:[di]
  sar ax,8
  neg ax

  add ax,Y1
  add ax,128
  push ax

  dec dx
  push dx

  dec di
  dec di
  mov ax,es:[di]
  sar ax,8
  neg ax

  add ax,Y1
  add ax,128
  push ax

  push 10

  call Line
  pop cx
  loop @@lp1
End;
UpdateRectScreen(X1,Y1,X2,Y2);
Asm
  mov ax,X2
  sub ax,X1
  mov bx,Seg Sequence
  mov es,bx
  mov di,Offset Sequence
  shl ax,1
  add di,ax
  xor ax,ax
  mov ax,es:[di]
  mov R,ax
End;

Power:=R/Divider;
Str(Power:0:5,Stroka);
ClearRectBuffer(180,300,243,307);
OutBufferTextXY(180,300,15,Stroka);
UpdateRectScreen(180,300,243,307);
```

```
    Asm
@@1:  or DMA_complete,0
      jz @@1
    End;
  Until Keypressed;
End;

Procedure OscillogrammaWithAxis(X1,Y1,X2,Y2:Word);
Begin
  OutBufferTextXY(X1+10,Y1,12,'U,V');
  OutBufferTextXY(X2-7,Y1+128+16,12,'t');
  Line(X1+3,Y1,X1+3,Y2,12);
  Line(X1,Y1+7,X1+3,Y1,12);
  Line(X1+6,Y1+7,X1+3,Y1,12);
  Line(X2-8,Y1+8+128,X2,Y1+8+128,12);
  Line(X2-7,Y1+8+128-3,X2,Y1+8+128,12);
  Line(X2-7,Y1+8+128+4,X2,Y1+8+128,12);
  UpdateRectScreen(X1,Y1,X2,Y2);
  OscillogrammaSB(X1+4,Y1+8,X2-8,Y2-4);
End;

Function Oscillgraph(X1,Y1,X2,Y2:Word):Boolean;
Var C:Char;
Begin
  Oscillgraph:=False;
  If SetUpSB(Irq,DMA) Then
  Begin
    Oscillgraph:=True;
    ClearBuffer;
    OutTextXY(MaxX Shr 1 - 56,0,12,'PC Oscillograph');
    OutTextXY(50,300,15,'Voltage value :');
    OutTextXY(251,300,15,'V');
    OutTextXY(50,MaxY-80,11,'Use ARROW KEYS for change sequence
synchronization.');
    OutTextXY(50,MaxY-70,11,'Use SPACE KEY for freeze sequence
outputing.');
    OutTextXY(50,MaxY-60,11,'Use ESC KEY for exit.');
    OutTextXY(MaxX-145,MaxY-55,10,'ULYANOVSK STATE');
    OutTextXY(MaxX-165,MaxY-45,10,'TECHNICAL UNIVERSITY');
    PutSprite(MaxX-160,MaxY-30,GetSpriteOffset(LibPointer,1));
    PutSprite(MaxX-148,MaxY-200,GetSpriteOffset(LibPointer,2));
    UpdateScreen;
    Repeat
      OscillogrammaWithAxis(X1,Y1,X2,Y2);
      While KeyPressed Do
      Begin
        C:=Readkey;
        If C=#32 Then Repeat Until Readkey=#32;
        ClearRectBuffer(X1,Y1+8,X2,Y2);
        UpdateRectScreen(X1,Y1+8,X2,Y2);
```

```
      If C=#0 Then C:=Readkey;
      If C=#75 Then If X2>X1+16 Then Dec(X2,4);
      If C=#77 Then If X2<MaxX Then Inc(X2,4);
     End;
   Until C=#27;
   ResetSB;
  End;
End;

End.

{$A+,B-,D+,E-,F-,G+,I-,L+,N+,P-,Q+,R-,S+,T-,V+,X+,Y+}
{$M 32768,0}

Program Oscillograph;
Uses
Dialogs,SBProDMA,SVGADrv1,MsgBox,Objects,SprLib,Dos,Clock,SetWin,

Drivers,Views,Menus,App,OscConst,Oscgraph,VoltMetr,HelpFile,AboutOsc,
    Statist,StatInt,StdDlg,SBMixer;

Type
  TOscilloApplication=Object(TApplication)
    ClockView:PClockView;
    Constructor Init;
    Procedure HandleEvent(Var Event: TEvent);Virtual;
    Procedure WriteShellMsg;Virtual;
    Procedure InitDeskTop;Virtual;
    Procedure OutOfMemory;Virtual;
    Procedure InitMenuBar;Virtual;
    Procedure InitStatusLine;Virtual;
    Function GetPalette:PPalette;Virtual;
    Procedure GetEvent(Var Event:TEvent);Virtual;
    Procedure ShowScreenSaver;
    Procedure Idle;Virtual;
    Destructor Done;Virtual;
  End;

  PBaseDeskTop=^TBaseDeskTop;
  TBaseDeskTop=Object(TDeskTop)
    Procedure InitBackGround;Virtual;
  End;

Procedure TBaseDesktop.InitBackground;
Var R: TRect;
Begin
  GetExtent(R);
  New(Background, Init(R, '-'));
End;
```

```
Procedure TOscilloApplication.InitDeskTop;
Var R: TRect;
Begin
  GetExtent(R);
  Inc(R.A.Y);
  Dec(R.B.Y);
  DeskTop:=New(PBaseDesktop, Init(R));
End;

Procedure TOscilloApplication.WriteShellMsg;
Begin

PrintStr('r==========================================================
==¬'#$0A#$0D+
         '¦ Type EXIT to return in PC Oscillograph... :) ... ¦'#$0A#$0D+

'L==========================================================-
'#$0A#$0D);
End;

Constructor TOscilloApplication.Init;
Var S:TBufStream;
    R:TRect;
    ResFile:TResourceFile;
    P:PDialog;
Begin
  Inherited Init;
  LoadFont(FontFileStr);
  RegisterHelpFile;
  If CheckSB Then
  Begin
    If                Not                DeviceSB16                Then
DisableCommands([cm_SetupHardwareDeviceSB16,

cm_SetupHardwareDMA5,cm_SetupHardwareDMA6,cm_SetupHardwareDM
A7]);
  End Else
  Begin
    MessageBox('Sound Blaster not found.',Nil,mfError or mfOkButton);
  End;

  LibPointer:=LoadLibrary(SpriteFileStr);
  If LibPointer=Nil Then OutOfMemory;
  GetExtent(R);
  R.A.X := R.B.X - 10;
  R.B.Y := R.A.Y + 1;
  ClockView := New(PClockView, Init(R));
  If ValidView(ClockView)<>Nil Then Insert(ClockView);

  S.Init(ConfigFileStr,stOpen,1024);
```

```
  If S.Status=stOk Then
  Begin
    S.Read(Irq,SizeOf(Irq));
    S.Read(DMA,SizeOf(DMA));
    S.Read(BaseAddrSB,SizeOf(BaseAddrSB));
    S.Read(Rate,SizeOf(Rate));
    S.Read(VideoMode,SizeOf(VideoMode));
    S.Read(Divider,SizeOf(Divider));
    S.Read(NumGarmonic,SizeOf(NumGarmonic));
    S.Read(ClockView^.State,SizeOf(ClockView^.State));
    S.Read(ScreenMode,SizeOf(ScreenMode));
    S.Read(ClockView^.Button,SizeOf(ClockView^.Button));
    S.Read(ClockView^.SaverTime,SizeOf(ClockView^.SaverTime));
    S.Read(SixteenBits,SizeOf(SixteenBits));
    S.Read(Volume,SizeOf(Volume));
    S.Read(FreqCorrection,SizeOf(FreqCorrection));
    S.Read(CalcConstSost,SizeOf(CalcConstSost));
  End;
  S.Done;
  Message(@Self, evCommand, cm_Information,Nil);
End;

Procedure TOscilloApplication.OutOfMemory;
Begin
  MessageBox('Not enough memory to complete operation.',
    Nil, mfError + mfOKButton);
  Message(@Self,evCommand,cmQuit,Nil);
End;

Function TOscilloApplication.GetPalette: PPalette;
Begin
  GetPalette := @MainPalette[AppPalette];
End;

Procedure TOscilloApplication.Idle;
Begin
  Inherited Idle;
  ClockView^.UpDate;
  If (MouseWhere.X>=ScreenWidth-2) And
     (MouseWhere.Y=0) Then ShowScreenSaver;
End;

Procedure TOscilloApplication.ShowScreenSaver;
Begin
  DoneEvents;
  DoneVideo;
  Exec(SaverFileStr,'');
  DosError:=0;
  InitVideo;
  InitEvents;
```

```
  ReDraw;
End;

procedure TOscilloApplication.GetEvent(var Event: TEvent);
var
  W: PWindow;
  HFile: PHelpFile;
  HelpStrm: PDosStream;
const
  HelpInUse: Boolean = False;
begin
  inherited GetEvent(Event);

  If ((Event.What And evMouse) <>0) Or
     ((Event.What And evKeyboard)<>0) Then
     ClockView^.TimeCount:=0;

  case Event.What of
    evCommand:
      if (Event.Command = cm_Help) and not HelpInUse then
      begin
        HelpInUse := True;
        HelpStrm := New(PDosStream, Init(HelpFileStr, stOpenRead));
        HFile := New(PHelpFile, Init(HelpStrm));
        if HelpStrm^.Status <> stOk then
        begin
          MessageBox('Could not open help file.', nil, mfError + mfOkButton);
          Dispose(HFile, Done);
        end
        else
        begin
          W := New(PHelpWindow,Init(HFile, GetHelpCtx));
          if ValidView(W) <> nil then
          begin
            DeskTop^.ExecView(W);
            Dispose(W,Done);
          end;
          ClearEvent(Event);
        end;
        HelpInUse := False;
      end;
  end;
end;

Destructor TOscilloApplication.Done;
Var S:TBufStream;
Begin
  S.Init(ConfigFileStr,stCreate,1024);
  S.Write(Irq,SizeOf(Irq));
  S.Write(DMA,SizeOf(DMA));
```

```
  S.Write(BaseAddrSB,SizeOf(BaseAddrSB));
  S.Write(Rate,SizeOf(Rate));
  S.Write(VideoMode,SizeOf(VideoMode));
  S.Write(Divider,SizeOf(Divider));
  S.Write(NumGarmonic,SizeOf(NumGarmonic));
  S.Write(ClockView^.State,SizeOf(ClockView^.State));
  S.Write(ScreenMode,SizeOf(ScreenMode));
  S.Write(ClockView^.Button,SizeOf(ClockView^.Button));
  S.Write(ClockView^.SaverTime,SizeOf(ClockView^.SaverTime));
  S.Write(SixteenBits,SizeOf(SixteenBits));
  S.Write(Volume,SizeOf(Volume));
  S.Write(FreqCorrection,SizeOf(FreqCorrection));
  S.Write(CalcConstSost,SizeOf(CalcConstSost));
  S.Done;
  UnloadFont;
  FreeLibrary(LibPointer);
  Inherited Done;
End;

Procedure TOscilloApplication.HandleEvent(Var Event:TEvent);

Procedure StartOscillograph;
Var Result:Boolean;
Begin
  If Not InitGraphMode(VideoMode) Then
  Begin
    MessageBox('Video Mode not supported or low memory.',Nil,mfError or
mfOkButton);
    Exit;
  End;
  SetAllPalette(Ptr(Seg(LibPointer^),10));
  DoneEvents;
  DoneVideo;
  Result:=Oscillgraph(0,0,MaxX,255+12);
  CloseGraphMode;
  InitVideo;
  InitEvents;
  ReDraw;
  If Not Result Then MessageBox('Sound Blaster not found.',Nil,mfError or
mfOkButton);
End;

Procedure StartVoltMetr;
Var Result:Boolean;
Begin
  DoneEvents;
  DoneVideo;
  Result:=VoltoMetr;
  InitVideo;
  InitEvents;
```

```
  ReDraw;
  If Not Result Then MessageBox('Sound Blaster not found.',Nil,mfError or
mfOkButton);
End;

Procedure StartStatist;
Var Result:Boolean;
Begin
  If
Application^.ExecuteDialog(New(PStatIntDialog,Init),@Values)=cmCancel
Then Exit;
  FileName:='*.ost';
  If Application^.ExecuteDialog(New(PFileDialog,
    Init('*.ost','Save File as...','File
name',fdOkButton,1)),@FileName)=cmCancel Then Exit;
  DoneEvents;
  DoneVideo;
  Result:=StatMaker;
  InitVideo;
  InitEvents;
  ReDraw;
  If Not Result Then MessageBox('Sound Blaster not found.',Nil,mfError or
mfOkButton);
End;

Procedure About;
Var Dlg:PDialog;
Begin
  Dlg:=New(PAboutDialog,Init);
  ExecuteDialog(Dlg,Nil);
End;

Procedure SetupCoefficient;
Var Stroka:String;
    Error:Integer;
    Res:Real;
Begin
  Str(Divider:0:5,Stroka);
  If InputBox('Setup Coefficient', 'Please read help before changing!', Stroka,
15)=cmCancel Then Exit;
  Val(Stroka,Res,Error);
  If Error<>0 Then MessageBox('Incorrect input!',Nil,mfError Or
mfOkButton)
  Else Divider:=Res;
End;

Procedure SetupFreqCorrection;
Var Stroka:String;
    Error:Integer;
    Res:Real;
```

```
Begin
  Str(FreqCorrection:0:5,Stroka);
  If InputBox('Setup frequence correction', 'Please read help before changing!',
Stroka, 15)=cmCancel Then Exit;
  Val(Stroka,Res,Error);
  If Error<>0 Then MessageBox('Incorrect input!',Nil,mfError Or
mfOkButton)
  Else FreqCorrection:=Res;
End;

Procedure SetupGarmonicNumber;
Var Stroka:String;
    Error:Integer;
    Res:Integer;
Begin
  Str(NumGarmonic,Stroka);
  If InputBox('Setup number of garmonic', 'Please read help before changing!',
Stroka, 2)=cmCancel Then Exit;
  Val(Stroka,Res,Error);
  If (Error<>0) Or (Res<1) Or (Res>30) Then MessageBox('Incorrect
input!',Nil,mfError Or mfOkButton)
  Else NumGarmonic:=Res;
End;

Procedure ShowCurSetWindow;
Var AreYouThere:PView;
Begin
  AreYouThere := Message(DeskTop, evBroadcast,
cm_FindCurSetWindow,Nil);
  If AreYouThere = Nil Then
  Begin
    InsertWindow(New(PCurSetWindow,Init));
  End
  Else AreYouThere^.Select;
End;

Begin
  Inherited HandleEvent(Event);
  If Event.What=evCommand Then
  Begin
    Case Event.Command Of
      cm_Oscillograph: StartOscillograph;
      cm_VoltMetr: StartVoltMetr;
      cm_Statist: StartStatist;
      cm_SetupHardwareDeviceSB: SixteenBits:=False;
      cm_SetupHardwareDeviceSB16: SixteenBits:=True;
      cm_SetupHardwareIrq2: Irq:=2;
      cm_SetupHardwareIrq3: Irq:=3;
      cm_SetupHardwareIrq5: Irq:=5;
      cm_SetupHardwareIrq7: Irq:=7;
```

```
      cm_SetupHardwareDMA0: DMA:=0;
      cm_SetupHardwareDMA1: DMA:=1;
      cm_SetupHardwareDMA3: DMA:=3;
      cm_SetupHardwareDMA5: DMA:=5;
      cm_SetupHardwareDMA6: DMA:=6;
      cm_SetupHardwareDMA7: DMA:=7;
      cm_SetupFrequence6024: Rate:=6024;
      cm_SetupFrequence8000: Rate:=8000;
      cm_SetupFrequence12048: Rate:=12048;
      cm_SetupFrequence16129: Rate:=16129;
      cm_SetupFrequence22222: Rate:=22222;
      cm_SetupFrequence30303: Rate:=30303;
      cm_SetupFrequence41667: Rate:=41667;
      cm_SetupFrequence45454: Rate:=45454;
      cm_SetupGraphMode101: VideoMode:=$101;
      cm_SetupGraphMode103: VideoMode:=$103;
      cm_SetupGraphMode105: VideoMode:=$105;
      cm_Volume1: Volume:=1;
      cm_Volume2: Volume:=2;
      cm_Volume3: Volume:=3;
      cm_Volume4: Volume:=4;
      cm_Volume5: Volume:=5;
      cm_Volume6: Volume:=6;
      cm_Volume7: Volume:=7;
      cm_Volume8: Volume:=8;
      cm_Volume9: Volume:=9;
      cm_Volume10: Volume:=10;
      cm_Volume11: Volume:=11;
      cm_Volume12: Volume:=12;
      cm_Volume13: Volume:=13;
      cm_Volume14: Volume:=14;
      cm_Volume15: Volume:=15;
      cm_CalcConstSost: CalcConstSost := Not CalcConstSost;
      cm_SetupCoefficient: SetupCoefficient;
      cm_SetupFreqCorrection: SetupFreqCorrection;
      cm_SetupGarmonicNumber: SetupGarmonicNumber;
      cm_HelpAbout: About;
      cm_ScreenSaver: ClockView^.SetSaverTime;
      cm_Clock :    Begin
                      ClockView^.State:=ClockView^.State Xor sfVisible;
                      ReDraw;
                    End;
      cm_ShowScreenSaver:ShowScreenSaver;
      cm_Information:ShowCurSetWindow;
    End;
    Message(DeskTop, evCommand, cm_RedrawCurSetWindow,Nil);
  End;
  ClearEvent(Event);
End;
```

```
Procedure TOscilloApplication.InitMenuBar;
Var R:TRect;
Begin
  GetExtent(R);
  R.B.Y := R.A.Y + 1;
  MenuBar := New(PMenuBar, Init(R, NewMenu(
    NewSubMenu('~F~ile', hcNoContext, NewMenu(
      NewItem('~D~OS shell',",kbNoKey, cmDosShell, hcNoContext,
      NewItem('E~x~it','Alt-X',kbAltX, cmQuit, hcNoContext,
      Nil))),
    NewSubMenu('~S~etup', hcNoContext, NewMenu(
      NewSubMenu('~H~ardware', hcNoContext, NewMenu(
        NewSubMenu('  ~D~evice ', hcNoContext, NewMenu(
          NewItem('          SB 2.0 or SB Pro          ', '', kbNoKey,
cm_SetupHardwareDeviceSB, hcNoContext,
          NewItem('SB16 or AWE32 or AWE64', '', kbNoKey,
cm_SetupHardwareDeviceSB16, hcNoContext,
          Nil))),
        NewSubMenu('   ~I~RQ   ', hcNoContext, NewMenu(
          NewItem('          ~2~          ', '', kbNokey, cm_SetupHardwareIrq2,
hcNoContext,
          NewItem('          ~3~          ', '', kbNokey, cm_SetupHardwareIrq3,
hcNoContext,
          NewItem('          ~5~          ', '', kbNokey, cm_SetupHardwareIrq5,
hcNoContext,
          NewItem('          ~7~          ', '', kbNokey, cm_SetupHardwareIrq7,
hcNoContext,
          Nil))))),
        NewSubMenu('   ~D~MA   ', hcNoContext, NewMenu(
          NewItem('          ~0~          ', '', kbNoKey, cm_SetupHardwareDMA0,
hcNoContext,
          NewItem('          ~1~          ', '', kbNoKey, cm_SetupHardwareDMA1,
hcNoContext,
          NewItem('          ~3~          ', '', kbNoKey, cm_SetupHardwareDMA3,
hcNoContext,
          NewItem('          ~5~          ', '', kbNoKey, cm_SetupHardwareDMA5,
hcNoContext,
          NewItem('          ~6~          ', '', kbNoKey, cm_SetupHardwareDMA6,
hcNoContext,
          NewItem('          ~7~          ', '', kbNoKey, cm_SetupHardwareDMA7,
hcNoContext,
          Nil))))))),
        Nil)))),
      NewSubMenu('~F~requence', hcNoContext, NewMenu(
        NewItem('6024 Hz', '', kbNoKey, cm_SetupFrequence6024,
hcNoContext,
        NewItem('8000 Hz', '', kbNokey, cm_SetupFrequence8000, hcNoContext,
        NewItem('12048 Hz', '', kbNokey, cm_SetupFrequence12048,
hcNoContext,
```

```
      NewItem('16129 Hz', '', kbNoKey, cm_SetupFrequence16129,
hcNoContext,
      NewItem('22222 Hz', '', kbNoKey, cm_SetupFrequence22222,
hcNoContext,
      NewItem('30303 Hz', '', kbNoKey, cm_SetupFrequence30303,
hcNoContext,
      NewItem('41667 Hz', '', kbNoKey, cm_SetupFrequence41667,
hcNoContext,
      NewItem('45454 Hz', '', kbNoKey, cm_SetupFrequence45454,
hcNoContext,
      Nil))))))))),
    NewSubMenu('~I~nput Volume', hcNoContext, NewMenu(
      NewItem('1', '', kbNoKey, cm_Volume1, hcNoContext,
      NewItem('2', '', kbNoKey, cm_Volume2, hcNoContext,
      NewItem('3', '', kbNoKey, cm_Volume3, hcNoContext,
      NewItem('4', '', kbNoKey, cm_Volume4, hcNoContext,
      NewItem('5', '', kbNoKey, cm_Volume5, hcNoContext,
      NewItem('6', '', kbNoKey, cm_Volume6, hcNoContext,
      NewItem('7', '', kbNoKey, cm_Volume7, hcNoContext,
      NewItem('8', '', kbNoKey, cm_Volume8, hcNoContext,
      NewItem('9', '', kbNoKey, cm_Volume9, hcNoContext,
      NewItem('10', '', kbNoKey, cm_Volume10, hcNoContext,
      NewItem('11', '', kbNoKey, cm_Volume11, hcNoContext,
      NewItem('12', '', kbNoKey, cm_Volume12, hcNoContext,
      NewItem('13', '', kbNoKey, cm_Volume13, hcNoContext,
      NewItem('14', '', kbNoKey, cm_Volume14, hcNoContext,
      NewItem('15', '', kbNoKey, cm_Volume15, hcNoContext,
      Nil))))))))))))))),
    NewSubMenu('~G~raphMode', hcNoContext, NewMenu(
      NewItem(' 640x480 ', '', kbNoKey, cm_SetupGraphMode101,
hcNoContext,
      NewItem(' 800x600 ', '', kbNoKey, cm_SetupGraphMode103,
hcNoContext,
      NewItem(' 1024x768 ', '', kbNoKey, cm_SetupGraphMode105,
hcNoContext,
      Nil))),
    NewItem('~C~oefficient', '', kbNoKey, cm_SetupCoefficient,
hcNoContext,
    NewItem('F~r~equence correction', '', kbNoKey, cm_SetupFreqCorrection,
hcNoContext,
    NewItem('C~a~lculate constant', '', kbNoKey, cm_CalcConstSost,
hcNoContext,
    NewItem('~G~armonic number', '', kbNoKey, cm_SetupGarmonicNumber,
hcNoContext,
    NewItem('C~l~ock', '', kbNoKey, cm_Clock, hcNoContext,
    NewItem('~S~creen saver', '', kbNoKey, cm_ScreenSaver, hcNoContext,
    NewItem('C~u~rrent settings', '', kbNoKey, cm_Information,
hcNoContext,
    Nil))))))))))),
  NewSubMenu('~H~elp', hcNoContext, NewMenu(
```

```
      NewItem('~H~elp', 'F1', kbF1, cm_Help, hcNoContext,
      NewItem('~A~bout', '', kbNoKey, cm_HelpAbout, hcNoContext,
      Nil))),
    NewItem('~O~scillograph', '', kbNoKey, cm_Oscillograph, hcNoContext,
    NewItem('~V~oltMetr', '', kbNoKey, cm_VoltMetr, hcNoContext,
    NewItem('S~t~atist', '', kbNoKey, cm_Statist, hcNoContext,
  Nil)))))))));
  Insert(MenuBar);
End;

Procedure TOscilloApplication.InitStatusLine;
Var R:TRect;
Begin
  GetExtent(R);
  R.A.Y := R.B.Y - 1;
  StatusLine := New(PStatusLine, Init(R,
    NewStatusDef(0, $FFFF,
      StdStatusKeys(
      NewStatusKey('~Alt-X~ Exit', kbAltX, cmQuit,
      NewStatusKey('~F1~ Help', kbF1, cm_Help,
      NewStatusKey('', kbF10, cmMenu,
    Nil)))),
  Nil)));
  Insert(StatusLine);
End;

Var Oscillo:TOscilloApplication;
Begin
  Oscillo.Init;
  Oscillo.Run;
  Oscillo.Done;
  Writeln('The PC Oscillograph, Version 3.00 shareware');
  Writeln('Copyright (C) 1998 by SerVo (Serge Volkov), Ulyanovsk, Russia');
End.

unit OSCILLO1;

interface

const

  hcAlgorithm           = 12;
  hcConflict            = 11;
  hcInstruction         = 10;
  hcMenu                = 2;
  hcMenuFile            = 3;
  hcMenuHelp            = 5;
  hcMenuOscillograph    = 6;
  hcMenuSetup           = 4;
  hcMenuStatist         = 8;
```

```
  hcMenuVoltMetr        = 7;
  hcNoContext           = 0;
  hcTechnic             = 9;

implementation

end.

unit SAVER;

interface

uses Drivers, Objects, Views, Dialogs;

type

  PScreenSavDialog = ^TScreenSavDialog;
  TScreenSavDialog = object(TDialog)
    constructor Init;
  end;

implementation

constructor TScreenSavDialog.Init;
var
  R: TRect;
  Control : PView;
begin
R.Assign(26, 6, 54, 17);
inherited Init(R, 'Screen saver');
Options := Options or ofCenterX or ofCenterY;

R.Assign(6, 2, 22, 7);
Control := New(PRadioButtons, Init(R,
  NewSItem('Off',
  NewSItem('1 minute',
  NewSItem('3 minutes',
  NewSItem('5 minutes',
  NewSItem('10 minutes', Nil)))))));
Insert(Control);

R.Assign(3, 8, 13, 10);
Control := New(PButton, Init(R, 'O~K~', cmOK, bfDefault));
Insert(Control);

R.Assign(15, 8, 25, 10);
Control := New(PButton, Init(R, 'Cancel', cmCancel, bfNormal));
Insert(Control);

SelectNext(False);
```

```
end;

end.

Unit SBMixer;
Interface
Uses SBProDMA;

Const
  Source_Mic1=        0;
  Source_CD= 2;
  Source_Mic2=        4;
  Source_Line=        6;
  Filtr_Low=    0;
  Filtr_High=   8;
  Filtr_None=   16;
  Volume:Byte=  15;

  Procedure WriteMixerSB(Index,Value:Byte);
  Function ReadMixerSB(Index:Byte):Byte;
  Procedure ResetMixerSB;
  Procedure InputMixerSB(Source,Filtr:Byte);
  Procedure LineVolumeSB(Left,Right:Byte);

Implementation

Procedure WriteMixerSB(Index,Value:Byte);
Begin
  Port[BaseAddrSB+4]:=Index;
  Port[BaseAddrSB+5]:=Value;
End;

Function ReadMixerSB(Index:Byte):Byte;
Begin
  Port[BaseAddrSB+4]:=Index;
  ReadMixerSB:=Port[BaseAddrSB+5];
End;

Procedure ResetMixerSB;
Begin
  WriteMixerSB(0,0);
End;

Procedure InputMixerSB(Source,Filtr:Byte);
Var Value:Byte;
Begin
  Value:=Source Or Filtr;
  WriteMixerSB($C,Value);
End;
```

```
Procedure LineVolumeSB(Left,Right:Byte);
Var Value:Byte;
Begin
  Value:=(Left Shl 4) Or Right;
  WriteMixerSB($2E,Value);
End;

End.

Unit SBProDMA;
Interface
Uses Crt,WinApi,Dos;

Const
  WRITE_DATA_SB   =       $10;
  ON_SOUND_SB     =       $D1;
  OFF_SOUND_SB    =      $D3;
  HALT_DMA_8      =       $D0;
  TIME_CONSTANT =         $40;
  FREQ_CONSTANT =       $42;
  DMA_8_BIT_ADC_HI=     $99;
  SET_LEN_DMA_8_BIT=    $48;
  MAX_BASE_SB          =5;
  BasesSB:Array [0..MAX_BASE_SB-1] of Word=( $220, $230, $240, $250,
$260 );
  baseAddrSB:Word=$220;
  DMAChannel:Byte=1;
  SbIRQ:Byte=5;
  VerSB:Word=$201;
  Fifo:Boolean=False;
  SixteenBits:Boolean=False;
  DeviceSB16:Boolean=False;
  MaxFrequence:Boolean=False;
  Stereo:Boolean=False;

Type
  TArray=Array[0..8191] of ShortInt;

Var
  Data:^TArray;
  DosSelector:Word;
  DPMISelector:Word;
  DMA_PAGE:Byte;
  DMA_POFF:Word;
  OldIRQ:Pointer;
  DMA_complete:Boolean;

Procedure SBHandler;Interrupt;
Procedure RateSB(Rate:Word);
Function CheckSB:Boolean;
```

```
Function ReadSB:Byte;
Procedure WriteSB(Value:Byte);
Function SetupSB(Irq,Dma:Integer):Boolean;
Procedure ResetSB;
Procedure RecordSB(Len:Word);
Procedure VersionSB;

Implementation

Procedure VersionSB;
Var B1,B2:Byte;
Begin
  WriteSB($E1);
  B1:=ReadSB;
  B2:=ReadSB;
  VerSB:=(B1 Shl 8) Or B2;
  If VerSB>=$200 Then Fifo:=True;
  If VerSB>=$201 Then MaxFrequence:=True;
  If VerSB>=$300 Then Stereo:=True;
  If VerSB>=$400 Then DeviceSB16:=True;
End;


Procedure RateSB(Rate:Word);      {Процедура установки частоты          }
Var Tc:Byte;                      {выборки                              }
Begin
  If SixteenBits Then
  Begin
    WriteSB(FREQ_CONSTANT);
    WriteSB(hi(Rate));
    WriteSB(lo(Rate));
  End
  Else
  If MaxFrequence Then
  Begin
    Tc := 256 - (1000000 div Rate); {Rate и есть частота выборки          }
    WriteSB(TIME_CONSTANT);               {Команда установки временной
константы }
    WriteSB(Tc);                  {Временная константа                  }
  End;
End;

Function ReadSB:Byte;             {Чтение байта с DSP                   }
Begin
  Repeat                          {Ждать пока 7 бит порта статуса чтения }
  Until (Port[BaseAddrSB+$E] And $80) <> 0;{2xE не будет 1               }
  ReadSB:=Port[BaseAddrSB+$A];      {Читать порт данных 2xA               }
End;
```

```
Function CheckSB:Boolean;          {Выяснение базового порта SoundBlaster
}
Var I,J:Integer;                   {и сброс DSP при обнаружении          }
Begin
  For J:=0 To Max_Base_SB-1 Do
  Begin
    BaseAddrSB:=BasesSB[J];
    Port[BaseAddrSB+6]:=1;
    Delay(10);
    Port[BaseAddrSB+6]:=0;
    Delay(10);
    For I:=0 To 30000 Do
      If ReadSB=$AA Then
      Begin
        CheckSB:=True;
        VersionSB;
        Exit;
      End;
  End;
  CheckSB:=False;
End;

Procedure WriteSB(Value:Byte);     {Посылка байта в DSP                  }
Begin
  Repeat                           {Ждать пока 7 бит порта статуса записи }
  Until (Port[BaseAddrSB+$C] And $80) = 0;{не будет 1                    }
  Port[BaseAddrSB+$C]:=Value;      {Послать байт в порт 2xC              }
End;

Procedure SBHandler;               {Обработчик пррывания, которое генери- }
Var I:Byte;                        {руется, когда закончена пресылка через}
Begin                              {DMA                                   }
  Asm
    sti
  End;
  DMA_complete:=True;              {Установка флага конца пересылки      }
  If SixteenBits Then
    I:=Port[BaseAddrSB+$F]         {Подтверждение конца записи           }
  Else
    I:=Port[BaseAddrSB+$E];
  Port[$20]:=$20;
End;

Function SetupSB(Irq,Dma:Integer):Boolean;
Var Im,Tm:Byte;
    I:Word;
    Physical:LongInt;
Begin
    DMAChannel:=Dma;
```

```
    If (Irq<>2) And (Irq<>3) And (Irq<>5) And (Irq<>7) Then SBIrq:=5
    Else SBIrq:=Irq;
    If Not CheckSB Then Begin SetupSb:=False; Exit; End;

    Physical:=GlobalDosAlloc(8192);
    DosSelector:=HiWord(Physical);
    DPMISelector:=LoWord(Physical);
    Data:=GlobalLock(DPMISelector);
    If Data=Nil Then Begin SetupSb:=False; Exit; End;
    Asm
      mov cx,16
      mov ax,DosSelector
      mul cx
      mov DMA_PAGE,dl
      or SixteenBits,0
      jz @@m1
      mov cx,2
      div cx
@@m1: mov DMA_POFF,ax
    End;
  Asm
    cli
  End;
  GetIntVec($8+SbIrq,OldIrq);
  SetIntVec($8+SbIrq,@SBHandler);
  DMA_Complete:=False;
  Im:=Port[$21];
  Tm:=Not (1 Shl SBIrq);
  Port[$21]:=Im And Tm;
  Asm
    sti
  End;
  WriteSB(ON_SOUND_SB);
  SetUpSB:=True;
End;

Procedure ResetSB;
Var J:Byte;
    I:Word;
Begin
  If SixteenBits Then J:=Port[BaseAddrSB+$F]
  Else J:=Port[BaseAddrSB+$E];
  Port[$20]:=$20;
  For I:=0 To 1 Do
  Begin
    GlobalUnlock(DPMISelector);
    GlobalDosFree(DPMISelector);
  End;
  WriteSB(OFF_SOUND_SB);
  If SixteenBits Then WriteSB($D5)
```

```
  Else WriteSB(HALT_DMA_8);
  Asm
    cli
  End;
  J:=Port[$21];
  Port[$21]:=J Or (1 Shl SBIrq);
  SetIntVec($8+SBIrq,OldIRQ);
  Asm
    sti
  End;
End;

Procedure RecordSB(Len:Word);
Var ModeDMA,CountPort, PagePort, BaseAddrPort:Word;
Const
    PagePorts:Array [0..7] Of Word=($87, $83, $81, $82, 0, $8B, $89, $8A);
    CountPorts:Array [0..7] Of Word=($01, $03, $05, $07, $C2, $C6, $CA,
$CE);
    BaseAddrPorts:Array [0..7] Of Word=($00, $02, $04, $06, $C0, $C4, $C8,
$CC);
Var MaskPort:Byte;
    ClrPort:Byte;
    ModePort:Byte;
Begin
  If SixteenBits Then
  Begin
    MaskPort:=$D4;
    ClrPort:=$D8;
    ModePort:=$D6;
  End
  Else
  Begin
    MaskPort:=$A;
    ClrPort:=$C;
    ModePort:=$B;
  End;

  DMA_complete:=False;
  ModeDMA:=$44+DMAChannel And Not 4;
  CountPort:=CountPorts[DMAChannel];
  BaseAddrPort:=BaseAddrPorts[DMAChannel];
  PagePort:=PagePorts[DMAChannel];

  Port[MaskPort]:=(DMAChannel And Not 4) Or 4;
  Port[ClrPort]:=DMAChannel And Not 4;
  Port[ModePort]:=ModeDMA;

  Port[BaseAddrPort]:=Lo(DMA_POFF);
  Port[BaseAddrPort]:=Hi(DMA_POFF);
  Port[PagePort]:=DMA_PAGE;
```

```
  Port[CountPort]:=lo(len-1);
  Port[CountPort]:=hi(len-1);
  Port[MaskPort]:=DMAChannel And Not 4;

  If SixteenBits Then
  Begin
    WriteSB($BA);           {fifo No - $B8 fifo Yes - $BA}
    WriteSB(0);
    WriteSB(lo(len-1));
    WriteSB(hi(len-1));
  End
  Else
  Begin
    WriteSB(SET_LEN_DMA_8_BIT);
    WriteSB(lo(len-1));
    WriteSB(hi(len-1));
    WriteSB(DMA_8_BIT_ADC_HI);
  End;
End;

End.

Unit SetWin;
Interface
Uses
Views,Objects,App,Drivers,OscConst,SBProDMA,SVGADrv1,VoltMetr,SB
Mixer,Statist;

Type
  PCurSetWindow=^TCurSetWindow;
  TCurSetWindow=Object(TWindow)
    Constructor Init;
    Procedure HandleEvent(Var Event:TEvent);Virtual;
    Procedure Draw;Virtual;
  End;

Implementation

Constructor TCurSetWindow.Init;
Var R:TRect;
Begin
  DeskTop^.GetExtent(R);
  Inc(R.A.Y,ScreenHeight-14);
  R.B.X:=34;
  Inherited Init(R,'Current settings',wnNoNumber);
  Flags:=Flags And Not wfGrow And Not wfZoom;
End;

Procedure TCurSetWindow.HandleEvent(Var Event:TEvent);
```

```
Begin
  Case Event.Command Of
    cm_FindCurSetWindow:ClearEvent(Event);
    cm_RedrawCurSetWindow:DrawView;
  End;
  Inherited HandleEvent(Event);
End;

Procedure TCurSetWindow.Draw;
Var Stroka:String;
Begin
  Inherited Draw;
  Dec(Size.Y);
  Dec(Size.X);
  If SixteenBits Then WriteStr(2,1,'Device: SB16 or AWE32 or AWE64',6)
  Else WriteStr(2,1,'Device: SB 2.0 or SB Pro',6);
  Str(Irq,Stroka);
  WriteStr(2,2,'IRQ: '+Stroka,6);
  Str(DMA,Stroka);
  WriteStr(2,3,'DMA: '+Stroka,6);
  Str(Rate,Stroka);
  WriteStr(2,4,'Frequence: '+Stroka+' Hz',6);
  Str(VideoMode,Stroka);
  WriteStr(2,5,'GraphMode: '+Stroka,6);
  Str(Divider:0:5,Stroka);
  WriteStr(2,6,'Coefficient: '+Stroka,6);
  Str(NumGarmonic,Stroka);
  WriteStr(2,7,'Garmonic number: '+Stroka,6);
  Str(Volume,Stroka);
  WriteStr(2,8,'Input volume: '+Stroka,6);
  Str(FreqCorrection:0:5,Stroka);
  WriteStr(2,9,'Frequence correction: '+Stroka,6);
  If CalcConstSost Then
    WriteStr(2,10,'Calculate constant: Yes',6)
  Else
    WriteStr(2,10,'Calculate constant: No',6);
  Inc(Size.Y);
  Inc(Size.X);
End;

End.

Unit SprLib;
Interface
Uses WinAPI,OscConst,SVGADrv1;

Var LibPointer:Pointer;

Function LoadLibrary(Path:String):Pointer;
Function FreeLibrary(LibPtr:Pointer):Pointer;
```

```
Function GetSpriteOffset(LibPtr:Pointer;N:Word):Pointer;

Implementation

Function LoadLibrary(Path:String):Pointer;
Var OffMH:LongInt;
    LP,LibPtr:Pointer;
    H,L:Word;
    SprFile:File;
    I:Integer;
Begin
    Assign(SprFile,Path);
    Reset(SprFile,1);
    If MaxAvail>FileSize(SprFile) Then
    Begin
      LibPtr:=GlobalAllocPtr(GMem_Fixed,FileSize(SprFile)+1024);
      OffMH:=0;
      LP:=LibPtr;
      While Not Eof(SprFile) Do
      Begin
        BlockRead(SprFile,LP^,32768);
        Inc(OffMH,32768);
        H:=OffMH shr 16;
        L:=OffMH- H shl 16;
        LP:=Ptr(Seg(LibPtr^)+H*SelectorInc,Ofs(LibPtr^)+L);
      End;
      I:=IOResult;
      LoadLibrary:=LibPtr;
    End Else LoadLibrary:=Nil;
    Close(SprFile);
End;

Function FreeLibrary(LibPtr:Pointer):Pointer;
Begin
  If GlobalFreePtr(LibPtr)=0 Then FreeLibrary:=Nil;
End;

Function GetSpriteOffset(LibPtr:Pointer;N:Word):Pointer;
Var Disp:LongInt;
    Sme:Longint;
    H,L:Word;
Begin
  Sme:=(N-1)*16+12+12+768;
  H:=Sme shr 16;
  L:=Sme-H shl 16;
  Disp:=LongInt(Ptr(Seg(LibPtr^)+H*SelectorInc,L)^);
  H:=Disp shr 16;
  L:=Disp-H shl 16;
  GetSpriteOffset:=Ptr(Seg(LibPtr^)+H*SelectorInc,L);
End;
```

```
End.

unit STATINT;

interface

uses Drivers, Objects, Views, Dialogs, Validate;

type
  StatIntDataRec = record
    Hours : LongInt;
    Minutes : LongInt;
    Seconds : LongInt;
    end;
  PStatIntDataRec = ^StatIntDataRec;

  PStatIntDialog = ^TStatIntDialog;
  TStatIntDialog = object(TDialog)
    constructor Init;
  end;

implementation

constructor TStatIntDialog.Init;
var
  R: TRect;
  Control : PView;
begin
R.Assign(24, 6, 55, 17);
inherited Init(R, 'Statistic interval');
Options := Options or ofCenterX or ofCenterY;

R.Assign(16, 2, 25, 3);
Control := New(PInputLine, Init(R, 10));
Insert(Control);
  PInputLine(Control)^.Validator := New(PRangeValidator, Init(0, 100));
  PInputLine(Control)^.Validator^.Options := voTransfer;

  R.Assign(6, 2, 15, 3);
  Insert(New(PLabel, Init(R, 'Hours:', Control)));

R.Assign(16, 4, 25, 5);
Control := New(PInputLine, Init(R, 7));
Insert(Control);
  PInputLine(Control)^.Validator := New(PRangeValidator, Init(0, 59));
  PInputLine(Control)^.Validator^.Options := voTransfer;

  R.Assign(6, 4, 15, 5);
  Insert(New(PLabel, Init(R, 'Minutes:', Control)));
```

```
R.Assign(16, 6, 25, 7);
Control := New(PInputLine, Init(R, 7));
Insert(Control);
  PInputLine(Control)^.Validator := New(PRangeValidator, Init(0, 59));
  PInputLine(Control)^.Validator^.Options := voTransfer;

  R.Assign(6, 6, 15, 7);
  Insert(New(PLabel, Init(R, 'Seconds:', Control)));

R.Assign(3, 8, 13, 10);
Control := New(PButton, Init(R, 'O~K~', cmOK, bfDefault));
Insert(Control);

R.Assign(18, 8, 28, 10);
Control := New(PButton, Init(R, '~C~ancel', cmCancel, bfNormal));
Insert(Control);

SelectNext(False);
end;

end.

Unit Statist;
Interface
Uses Crt,StatInt,Dialogs,App,Views,Objects,StdDlg,Dos,Clock,
    Strings,MsgBox,VoltMetr,SBMixer,OscConst,SBProDMA;

Const
  OstHeader:String=
  '{      This file was created by PC Oscillograph 2.0       }'#13#10+
  '{-----------------------------------------------------------}'#13#10+
  '{ Date  Time   MaxVolt  MinVolt  RealVolt  Frequence}'#13#10+
  '{-----------------------------------------------------------}'#13#10#13#10;
  FreqCorrection:Single=1;

Function Frequence(Var Data:TWorkWords):Single;
Function StatMaker:Boolean;
Procedure StatistSB;
Function CalculateTime(Hours,Minutes,Seconds:LongInt):LongInt;
Procedure
MakeOutString(Hours,Minutes,Seconds:Byte;MaxVolt,MinVolt,RealVolt,Fre
quence:Single;Stroka:PChar);

Var Values:StatIntDataRec;
    FileName:FNameStr;
    RateDiv4096:Single;

Implementation
```

```
Procedure
MakeOutString(Hours,Minutes,Seconds:Byte;MaxVolt,MinVolt,RealVolt,Fre
quence:Single;Stroka:PChar);
Var S:String;
    Year,Month,Day,DayOfWeek:Word;
Begin
  GetDate(Year,Month,Day,DayOfWeek);
  StrCopy(Stroka,'');
  Str(Day:2,S);
  StrPCopy(@S,S);
  StrCat(Stroka,@S);
  StrCat(Stroka,'.');
  Str(Month:2,S);
  StrPCopy(@S,S);
  StrCat(Stroka,@S);
  StrCat(Stroka,'.');
  Str(Year:4,S);
  StrPCopy(@S,S);
  StrCat(Stroka,@S);
  StrCat(Stroka,'  ');
  Str(Hours:2,S);
  StrPCopy(@S,S);
  StrCat(Stroka,@S);
  StrCat(Stroka,':');
  Str(Minutes:2,S);
  StrPCopy(@S,S);
  StrCat(Stroka,@S);
  StrCat(Stroka,':');
  Str(Seconds:2,S);
  StrPCopy(@S,S);
  StrCat(Stroka,@S);
  StrCat(Stroka,'  ');
  Str(MaxVolt:2:5,S);
  StrPCopy(@S,S);
  StrCat(Stroka,@S);
  StrCat(Stroka,'  ');
  Str(MinVolt:2:5,S);
  StrPCopy(@S,S);
  StrCat(Stroka,@S);
  StrCat(Stroka,'  ');
  Str(RealVolt:2:5,S);
  StrPCopy(@S,S);
  StrCat(Stroka,@S);
  StrCat(Stroka,'  ');
  Str(Frequence:5:5,S);
  StrPCopy(@S,S);
  StrCat(Stroka,@S);
  StrCat(Stroka,#13#10);
End;
```

```
Function FindMaxVolt(Data:TWorkWords):Single;
Var I,Max:Integer;
Begin
  Max:=Data[I];
  For I:=1 To 4095 Do If Data[I]>Max Then Max:=Data[I];
  FindMaxVolt:=Max/Divider;
End;

Function FindMinVolt(Data:TWorkWords):Single;
Var I,Min:Integer;
Begin
  Min:=Data[I];
  For I:=1 To 4095 Do If Data[I]<Min Then Min:=Data[I];
  FindMinVolt:=Min/Divider;
End;

Function Frequence(Var Data:TWorkWords):Single;
  Function Sgn(A:Single):Integer;
  Begin
      If A>=0 Then Sgn:=+1 Else
      If A<0 Then Sgn:=-1;
  End;
Var
    Summa,PeriodLen:Real;
    F:Text;
    FP,LP,I,J:Integer;
Begin
  J:=0;
  While ((Sgn(Data[J])=Sgn(Data[J+1])) And (J<=4094)) Do Inc(J);
  FP:=J;
  Summa:=1;
  While J<=4094 Do
  Begin
    If Sgn(Data[J])<>Sgn(Data[J+1]) Then
    Begin
      Summa:=Summa+1;
      LP:=J;
    End;
    Inc(J);
  End;
  If Summa>0 Then
  Begin
    Summa:=((Summa-1)/2)*RateDiv4096*FreqCorrection;
    PeriodLen:=(LP-FP)/((Summa-1)/2);
    Summa:=Summa+(FP+4095-LP)/PeriodLen;
  End;
  Frequence:=Summa;
End;

Function CalculateTime(Hours,Minutes,Seconds:LongInt):LongInt;
```

```
Begin
  CalculateTime:=Hours*3600+Minutes*60+Seconds;
End;

Procedure StatistSB;
Var S:TBufStream;
    Previous,Current,Interval:LongInt;
    MaxVolt,MinVolt,Swap:Single;
    RealVolt,SumRealVolt:Single;
    SumFreq,Freq:Single;
    Hours,Minutes,Seconds:Byte;
    Stroka:Array[0..79] Of Char;
    J:Word;
    R:Integer;
    KolvoSred:Single;
Begin
  S.Init(FileName,stCreate,4096);
  If S.Status<>stOk Then
  Begin
    S.Done;
    MessageBox('Can not create file.',Nil,mfError Or mfOkButton);
  End;
  S.Write(OstHeader[1],Length(OstHeader));
  GetTime(Hours,Minutes,Seconds);
  Previous:=CalculateTime(Hours,Minutes,Seconds);
  Interval:=CalculateTime(Values.Hours,Values.Minutes,Values.Seconds);
  KolvoSred:=0;
  SumRealVolt:=0;
  SumFreq:=0;
  RateDiv4096:=Rate/4096;
  Asm
    les di,Data
    mov cx,4096
    mov ax,8080h
    rep stosw
  End;
  InputMixerSB(Source_Line,Filtr_None);
  LineVolumeSB(Volume,Volume);
  RateSB(Rate);
  Repeat
{-------------------------------------------------------------------}
    If SixteenBits Then
    Asm
      mov bx,ds
      mov ax,Seg Sequence
      mov es,ax
      mov di,Offset Sequence
      lds si,Data
      mov cx,4096
@@1:  lodsw
```

```
      sub ax,32768
      stosw
      loop @@1
      mov ds,bx
    End
    Else
    Asm
      mov bx,ds
      mov ax,Seg Sequence
      mov es,ax
      mov di,Offset Sequence
      lds si,Data
      mov cx,4096
@@1:  xor ax,ax
      lodsb
      sub al,128
      cbw
      shl ax,8
      stosw
      loop @@1
      mov ds,bx
    End;
    RecordSB(4096);

    MaxVolt:=FindMaxVolt(Sequence);
    MinVolt:=FindMinVolt(Sequence);

    KolvoSred:=KolvoSred+1;
    GetTime(Hours,Minutes,Seconds);
    Current:=CalculateTime(Hours,Minutes,Seconds);
    RealVolt:=RealVoltage(Sequence)/Divider;
    SumRealVolt:=SumRealVolt+RealVolt;
    Freq:=Frequence(Sequence);
    SumFreq:=SumFreq+Freq;
    If Abs(Current-Previous)>=Interval Then
    Begin
      Previous:=Current;
      RealVolt:=SumRealVolt/KolvoSred;
      Freq:=SumFreq/KolvoSred;

MakeOutString(Hours,Minutes,Seconds,MaxVolt,MinVolt,RealVolt,Freq,Stro
ka);
      S.Write(Stroka,StrLen(Stroka));
      Write(Stroka);
      MaxVolt:=-1E10;
      MinVolt:=1E10;
      KolvoSred:=0;
      SumRealVolt:=0;
      SumFreq:=0;
    End;
```

```
    Asm
@@1:  or DMA_complete,0
      jz @@1
    End;

  Until KeyPressed;
  S.Done;
End;

Function StatMaker:Boolean;
Var Dlg:PDialog;
Begin
  StatMaker:=False;
  If SetUpSB(Irq,DMA) Then
  Begin
    StatMaker:=True;
    StatistSB;
    ResetSB;
  End;
End;

End.

Unit SVGADrv1;
Interface
Uses WinAPI;

Const
  drvstOk=0;
  drvstLowMemory=1;
  CharWidth:Word=8;
  CharHeight:Word=8;
  VideoMode:Word=$101;

Var
  SVGADrvBuffer:Pointer;
  SVGADrvStatus:Word;
  OldVideoMode:Word;
  MaxX:Word;
  MaxY:Word;
  NumBanks:Word;
  FontPointer:Pointer;
  SVGAModeInfoRec:Record
    rModeAttrs:Word;
    rWinAAttrs:Byte;
    rWinBAttrs:Byte;
    wWinGran  :Word;
    wWinSize  :Word;
    wWinASeg  :Word;
```

```
    wWinBSeg  :Word;
    pfWinFnPtr:Pointer;
    wScanLineSize:Word;
    wHorisRes :Word;
    wVertRes  :Word;
    bCharWide :Byte;
    bCharHigh :Byte;
    bPlaneCnt :Byte;
    bBitsPerPel:Byte;
    bBankCnt  :Byte;
    bMemModel :Byte;
    bBankSize :Byte;
    Reserved  :Array [0..226] Of Byte;
  End;

  Function InitGraphMode(Mode:Word):Boolean;
  Procedure SetVideoMode(Mode:Word);
  Function GetVideoMode:Word;
  Procedure PutPixel(X,Y,Color:Word);
  Procedure UpdateRectScreen(X1,Y1,X2,Y2:Word);
  Procedure UpdateScreen;
  Procedure ClearBuffer;
  Procedure ClearRectBuffer(X1,Y1,X2,Y2:Word);
  Procedure Line(a,b,c,d:integer;col:byte);
  Procedure OutTextXY(X,Y:Word;Color:Byte;Text:String);
  Procedure OutBufferTextXY(X,Y:Word;Color:Byte;Text:String);
  Procedure LoadFont(FileName:String);
  Procedure UnLoadFont;
  Function  VesaModeInfo(Mode:Word):Boolean;
  Procedure CloseGraphMode;
  Procedure SetAllPalette(PalBuf:Pointer);
  Procedure PutSprite(X,Y:Word;SprPtr:Pointer);

Implementation

Procedure PutSprite(X,Y:Word;SprPtr:Pointer);
Var Width,Height,I,J,Pixel:Word;
Begin
  Width:=Word(Ptr(Seg(SprPtr^),Ofs(SprPtr^)+2)^);
  Height:=Word(Ptr(Seg(SprPtr^),Ofs(SprPtr^)+4)^);
  SprPtr:=Ptr(Seg(SprPtr^),Ofs(SprPtr^)+6);
  For J:=Y To Y+Height-1 Do
  Begin
    For I:=X To X+Width-1 Do
    Begin
      Pixel:=Word(SprPtr^);
      If Byte(Pixel)<>0 Then PutPixel(I,J,Pixel);
      Asm
        inc word ptr [SprPtr]
        jnc @@ok
```

```
        mov ax,[SelectorInc]
        add word ptr [SprPtr+2],ax
@@ok:
      End;
    End;
  End;
End;

Procedure SetAllPalette(PalBuf:Pointer);Assembler;
Asm
  les dx,PalBuf
  mov ax,1012h
  xor bx,bx
  mov cx,256
  int 10h
End;

Function  VesaModeInfo(Mode:Word):Boolean;Assembler;
Asm
        mov ax,Seg SVGAModeInfoRec
        mov es,ax
        mov di,Offset SVGAModeInfoRec
        mov ax,4f01h
        mov cx,[Mode]
        int 10h
        or ah,ah
        jnz @@error
        mov ax,1
        jmp @@end
@@error:xor ax,ax
@@end:
End;

Procedure ClearBuffer;Assembler;
Asm
        mov cx,[NumBanks]
        mov dx,[SelectorInc]
        les di,[SVGADrvBuffer]
        mov ax,es

@@lp1:  push cx
        mov es,ax
        mov cx,16384
        db 66h; xor ax,ax
        rep; db 66h; stosw
        mov ax,es
        add ax,dx
        pop cx
        loop @@lp1
End;
```

```
Procedure ClearRectBuffer(X1,Y1,X2,Y2:Word);Assembler;
Asm
      mov bx,X2
      sub bx,X1
      mov ax,[MaxX]
      mov dx,ax
      sub dx,bx
      push dx
      push bx
      mov bx,word ptr [SVGADrvBuffer+2]
      inc ax
      mul [Y1]
      add ax,[X1]
      adc dx,0
      mov di,ax
      mov ax,[SelectorInc]
      push ax
      mul dx
      add bx,ax
      mov es,bx

      pop bx
      pop ax
      pop dx
      inc ax
      shr ax,2
      mov cx,[Y2]
      sub cx,[Y1]
      inc cx

@@lp1:  push cx
      mov cx,ax
      push cx

      shl ax,2
      add ax,di
      jnc @@2

      push ax
      xor cx,cx
      sub cx,di
      xor al,al
      rep stosb
      mov ax,es
      add ax,bx
      mov es,ax
      pop cx
      xor al,al
      rep stosb
```

```
        jmp @@3

@@2:    db 66h; xor ax,ax
        rep; db 66h; stosw
@@3:    add di,dx
        jnc @@1

        mov ax,es
        add ax,bx
        mov es,ax

@@1:    pop ax
        pop cx
        loop @@lp1
End;

Procedure UpdateScreen;Assembler;
Asm
        cli
        push ds

        mov ax,[SegA000]
        mov bx,[SelectorInc]
        mov cx,[NumBanks]
        lds si,[SVGADrvBuffer]
        mov es,ax
        xor di,di
        xor dx,dx
        mov ax,ds

@@lp1:  push cx
        mov ds,ax
        push bx
        mov ax,4f05h
        xor bx,bx
        int 10h
        pop bx
        mov cx,16384
        rep; db 66h; movsw
        mov ax,ds
        add ax,bx
        inc dx
        pop cx
        loop @@lp1

        pop ds
        sti
End;

Procedure UpdateRectScreen(X1,Y1,X2,Y2:Word);
```

```
Var Propusk:Word;
Begin
Asm
        cli
        push ds

        mov bx,X2
        sub bx,X1
        mov ax,[MaxX]
        mov dx,ax
        sub dx,bx
        mov Propusk,dx
        push bx
        mov bx,word ptr [SVGADrvBuffer+2]
        inc ax
        mul [Y1]
        add ax,[X1]
        adc dx,0
        mov si,ax
        mov di,ax
        mov ax,[SegA000]
        mov es,ax
        mov ax,[SelectorInc]
        push ax
        push dx
        mul dx
        add bx,ax
        mov ds,bx

        pop dx
        mov ax,4f05h
        xor bx,bx
        int 10h

        pop bx
        pop ax
        inc ax
        shr ax,2
        mov cx,[Y2]
        sub cx,[Y1]
        inc cx


@@lp1:  push cx
        mov cx,ax
        push cx

        shl ax,2
        add ax,di
        jnc @@2
```

```
        push ax
        xor cx,cx
        sub cx,di
        rep movsb
        inc dx
        mov ax,ds
        add ax,bx
        mov ds,ax
        push bx
        mov ax,4f05h
        xor bx,bx
        int 10h
        pop bx
        pop cx
        rep movsb
        jmp @@3

@@2:    rep; db 66h; movsw
@@3:    mov ax,[Propusk]
        add si,ax
        add di,ax
        jnc @@1

        adc dx,0
        push bx
        mov ax,4f05h
        xor bx,bx
        int 10h
        pop bx
        mov ax,ds
        add ax,bx
        mov ds,ax

@@1:    pop ax
        pop cx
        loop @@lp1

        pop ds
        sti
End;
End;

Procedure PutPixel(X,Y,Color:Word);Assembler;
Asm
        mov ax,[MaxX]
        mov bx,word ptr [SVGADrvBuffer+2]
        inc ax
        mul [Y]
        add ax,[X]
```

```
      adc dx,0
      mov di,ax

      mov ax,[SelectorInc]
      mul dx
      add bx,ax
      mov es,bx

      mov ax,[Color]
      mov es:[di],al
End;

Function InitGraphMode(Mode:Word):Boolean;
Begin
 If SVGADrvBuffer<>Nil Then CloseGraphMode;
 OldVideoMode:=GetVideoMode;
 If Not VESAModeInfo(Mode) Then Begin InitGraphMode:=False;Exit;End;
 SetVideoMode(Mode);
 Case Mode Of
  $101: Begin
          MaxX:=639;
          MaxY:=479;
        End;
  $103: Begin
          MaxX:=799;
          MaxY:=599;
        End;
  $105: Begin
          MaxX:=1023;
          MaxY:=767;
        End;
  Else  Begin
          InitGraphMode:=False;
          Exit;
        End;
 End;
 Asm
    mov ax,[MaxX]
    mul [MaxY]
    inc dx
    mov NumBanks,dx
 End;
 SVGADrvBuffer:=GlobalAllocPtr(GMem_Fixed,NumBanks*65536);
 If SVGADrvBuffer=Nil Then
 Begin
  SVGADrvStatus:=drvstLowMemory;
  InitGraphMode:=False;
  Exit;
 End;
 InitGraphMode:=True;
```

```
End;

Procedure SetVideoMode(Mode:Word);Assembler;
Asm
      mov ax,4f02h
      mov bx,[Mode]
      int 10h
End;

Function GetVideoMode:Word;Assembler;
Asm
      mov ax,4f03h
      int 10h
      mov ax,bx
End;

Procedure CloseGraphMode;
Begin
  GlobalFreePtr(SVGADrvBuffer);
  SVGADrvBuffer:=Nil;
  SetVideoMode(OldVideoMode);
End;

Procedure Line(a,b,c,d:integer;col:byte);
  function sgn(a:Integer):integer;
  begin
      if a>0 then sgn:=+1 else
      if a<0 then sgn:=-1 else
      if a=0 then sgn:=0;
  end;
var i,s,d1x,d1y,d2x,d2y,u,v,m,n:integer;
label lp1;
begin
    u:= c - a;
    v:= d - b;
    d1x:= SGN(u);
    d1y:= SGN(v);
    d2x:= d1x;
    d2y:= 0;
    m:= ABS(u);
    n := ABS(v);
    IF NOT (M>N) then
    BEGIN
        d2x := 0 ;
        d2y := d1y;
        Asm
          mov ax,n
          mov bx,m
          mov m,ax
          mov n,bx
```

```
        End;
    END;
    s := m shr 1;
    FOR i := 0 TO m DO
    Asm
      mov cx,0
  lp1: push cx
    End;
    BEGIN
        putpixel(a,b,col);
        Inc(s,n);
        IF not (s<m) THEN
        BEGIN
            Dec(s,m);
            Inc(a,d1x);
            Inc(b,d1y);
        END
        ELSE
        BEGIN
            Inc(a,d2x);
            Inc(b,d2y);
        END;
    End;
    Asm
      pop cx
      inc cx
      cmp m,cx
      jnc lp1
    End;
END;

Procedure LoadFont(FileName:String);
Var Fp:File;
   I:Word;
Begin
  FontPointer:=GlobalAllocPtr(GMem_Fixed,2048);
  If FontPointer=Nil Then Exit;
  Assign(Fp,FileName);
  Reset(Fp,2048);
  BlockRead(Fp,FontPointer^,1);
  Close(Fp);
End;

Procedure UnLoadFont;
Begin
  If FontPointer<>Nil Then GlobalFreePtr(FontPointer);
End;

Procedure OutBufferTextXY(X,Y:Word;Color:Byte;Text:String);
Var I:Word;
```

```
Begin
  For I:=1 To Byte(Text[0]) Do
  Begin
       Asm
         lea di,Text
         mov ax,ss
         mov es,ax
         add di,[I]
         xor bx,bx
         mov bl,byte ptr es:[di]
         les di,[FontPointer]
         shl bx,3
         add di,bx
         mov cx,8

@@lp2:    mov bl,byte ptr es:[di]
         push cx
         mov cx,8
@@lp1:    shl bl,1
         jnc @@1
         push bx
         push di
         push es
         push [X]
         push [Y]
         push word ptr [Color]
         call PutPixel
         pop es
         pop di
         pop bx
@@1:      inc [X]
         loop @@lp1
         sub [X],8
         inc [Y]
         inc di
         pop cx
         loop @@lp2
         add [X],8
         sub [Y],8
       End;
  End;
End;

Procedure OutTextXY(X,Y:Word;Color:Byte;Text:String);
Begin
  OutBufferTextXY(X,Y,Color,Text);
  UpdateRectScreen(X,Y,X+(Byte(Text[0]) shl 3)-1,Y+7);
End;

End.
```

```
Unit VoltMetr;
Interface
Uses Crt,OscConst,SBProDMA,WinAPI,SBMixer,SVGADrv1;

Type
   TWorkWords=Array [0..4095] Of Integer;

Var
   Sequence:TWorkWords;

Const
   NumGarmonic : Integer = 30;
   CalcConstSost: Boolean = True;

Function VoltoMetr:Boolean;
Procedure OutVoltMetr;
Function RealVoltage(Var YourData:TWorkWords):Single;
Procedure DigitalizationSB;
Procedure WriteVoltMetrImage;

Implementation

Procedure FindPeriod(Var Data:TWorkWords;Var
FirstPoint,LastPoint:Integer);
  Function Sgn(A:Integer):Integer;
  Begin
     If A>=0 Then Sgn:=+1 Else
     If A<0 Then Sgn:=-1;
  End;
Var I:Integer;
Begin
  I:=1;
  While ((Sgn(Data[I])=Sgn(Data[I+1])) And (I<4094)) Do Inc(I);
  FirstPoint:=I;
  Inc(I);
  While ((Sgn(Data[I])=Sgn(Data[I+1])) And (I<4094)) Do Inc(I);
  Inc(I);
  While ((Sgn(Data[I])=Sgn(Data[I+1])) And (I<4094)) Do Inc(I);
  LastPoint:=I;
End;

Function RealVoltage(Var YourData:TWorkWords):Single;
Var
   CosSost,SinSost,UMax:Array [0..50] Of Single;
   I,J:Integer;
   FirstPoint,LastPoint,NumPoints:Integer;

ConstSost,TwoDivNumPoints,Summa,SinSumma,CosSumma,DeltaX:Single;
Begin
```

```
  FindPeriod(YourData,FirstPoint,LastPoint);
  NumPoints:=LastPoint-FirstPoint;
  TwoDivNumPoints:=2/NumPoints;

  DeltaX:=Pi*TwoDivNumPoints;

  For I:=1 To NumGarmonic Do
  Begin
    SinSumma:=0;
    CosSumma:=0;
    For J:=1 To NumPoints Do
    Begin
      SinSumma:=SinSumma+YourData[J+FirstPoint]*Sin(I*J*DeltaX);
      CosSumma:=CosSumma+YourData[J+FirstPoint]*Cos(I*J*DeltaX);
    End;
    SinSost[I]:=SinSumma*TwoDivNumPoints;
    CosSost[I]:=CosSumma*TwoDivNumPoints;
  End;

  If CalcConstSost Then
  Begin
    Summa:=0;
    For J:=1 To NumPoints Do
    Begin
      Summa:=Summa+YourData[J+FirstPoint];
    End;
    ConstSost:=Summa/NumPoints;
  End
  Else ConstSost:=0;

  For I:=1 To NumGarmonic Do
  Begin
    UMax[I]:=Sqrt(SinSost[I]*SinSost[I]+CosSost[I]*CosSost[I]);
  End;

  Summa:=ConstSost*ConstSost;
  For I:=1 To NumGarmonic Do
  Begin
    Summa:=Summa+UMax[I]*UMax[I]/2;
  End;
  Summa:=Sqrt(Summa);
  If (Summa>1E6) Or (Summa<1E-6) Then RealVoltage:=0
  Else RealVoltage:=Summa;
End;

Procedure DigitalizationSB;
Begin
  Asm
    les di,Data
    mov cx,4096
```

```
    mov ax,8080h
    rep stosw
  End;
  InputMixerSB(Source_Line,Filtr_None);
  LineVolumeSB(Volume,Volume);
  RateSB(Rate);
  Repeat
    If SixteenBits Then
    Asm
      mov bx,ds
      mov ax,Seg Sequence
      mov es,ax
      mov di,Offset Sequence
      lds si,Data
      mov cx,4096
@@1:  lodsw
      sub ax,32768
      stosw
      loop @@1
      mov ds,bx
    End
    Else
    Asm
      mov bx,ds
      mov ax,Seg Sequence
      mov es,ax
      mov di,Offset Sequence
      lds si,Data
      mov cx,4096
@@1:  xor ax,ax
      lodsb
      sub al,128
      cbw
      shl ax,8
      stosw
      loop @@1
      mov ds,bx
    End;
    RecordSB(4096);
    OutVoltMetr;
    Asm
@@1:  or DMA_complete,0
      jz @@1
    End;
  Until Keypressed;
End;

Procedure OutVoltMetr;
Var Power:Single;
    Stroka:String;
```

```
Begin
  Power:=RealVoltage(Sequence)/Divider;
  Str(Power:0:5,Stroka);
  Asm
    mov ah,02h
    xor bx,bx
    mov dx,090Dh
    int 10h
  End;
  Write(Stroka,'   ');
  Asm
    mov ah,02h
    xor bx,bx
    mov dx,1900h
    int 10h
  End;
End;

Procedure WriteVoltMetrImage;
Var OldTextAttr:Byte;
Begin
  OldTextAttr:=TextAttr;
  TextAttr:=12;
  GoToXY(14,7);
  Write('PC VoltMetr');
  GoToXY(7,24);
  Write('CAUTION: Watch the voltage!');
  TextAttr:=11;
  GoToXY(12,9);
  Write('r==============¬');
  GoToXY(12,10);
  Write('¦              ¦');
  GoToXY(12,11);
  Write('L==============-');
  TextAttr:=OldTextAttr;
  Asm
    mov ah,02h
    xor bx,bx
    mov dx,1900h
    int 10h
  End;
End;

Function VoltoMetr:Boolean;
Var OldVideoMode:Word;
    OldTextAttr:Byte;
Begin
  VoltoMetr:=False;
  OldVideoMode:=GetVideoMode;
  SetVideoMode(0);
```

```
  OldTextAttr:=TextAttr;
  TextAttr:=10;
  WriteVoltMetrImage;
  If SetUpSB(Irq,DMA) Then
  Begin
    VoltoMetr:=True;
    DigitalizationSB;
    ResetSB;
  End;
  SetVideoMode(OldVideoMode);
  TextAttr:=OldTextAttr;
End;

End.
```