

## Оглавление

1. Назначение .....	2
2. Системные требования .....	3
3. Параметры приложения .....	4
4. Описание пользовательского интерфейса.....	5
4.1. Список модулей сигналов .....	6
4.2. Панель состояния .....	7
4.4. Рабочая область .....	9
4.6. Таблица значений .....	13
5. Создание пользовательских уведомлений.....	14
5.1. Триггеры для модуля .....	16
5.2. Триггеры для сигнала.....	17
6. Отчет по активным событиям .....	23
7. Просмотр архива данных .....	24
7.1. Просмотр статистики по архивным данным.....	25
7.2. Экспорт данных в файл.....	26
8. Пользовательские скрипты – создание виртуальных сигналов.....	27
9. Просмотр в браузере .....	31
10. Поддержка Zabbix агента.....	32
11. Описание внутренней архитектуры ПО .....	33
Приложение 1. Клиент для МК Arduino .....	35
Приложение 2. Клиент для C/C++ .....	38
Приложение 3. Клиент для NET (C#).....	39
Приложение 4. Клиент для Python. ....	40
Приложение 5. Правила написания клиента для любых МК и устройств .....	41
Приложение 6. API для получения текущих значений .....	44
Приложение 7. Задание частоты получения данных.....	46
Приложение 8. Скрипт Python для отправки email .....	47
ЛИЦЕНЗИЯ.....	48

## 1. Назначение

Программное обеспечение SVisual предназначено для мониторинга работы МК устройств, отладки программы, оповещения пользователя о наступивших событиях.

Возможности SVMonitor:

- подключение к МК по COM порту (usb для arduino), по сети Ethernet или Wi-Fi протокол TCP;
- опрос значений сигналов в реальном времени с частотой до 100 Гц (по умолчанию 10 Гц), количество устройств и сигналов выбирается пользователем;
- допустимое количество сигналов для записи 2048;
- вывод значений выбранных сигналов на экран монитора в реальном времени;
- запись архива сигналов на жесткий диск ПК;
- просмотр архива с помощью дополнительного ПО SVViewer;
- возможность установки оповещений о возникшем событии (триггеров), запуск пользовательского процесса при срабатывании триггера;
- добавление сигнала для просмотра/записи только клиентом, никаких дополнительных движений не требуются.

## 2. Системные требования

Операционная система: Windows-64 (7, 8, 10), Linux

Процессор: не ниже Pentium 4

Оперативная память : не ниже 512 мб

### 3. Параметры приложения

В главном меню выберите «Настройки»

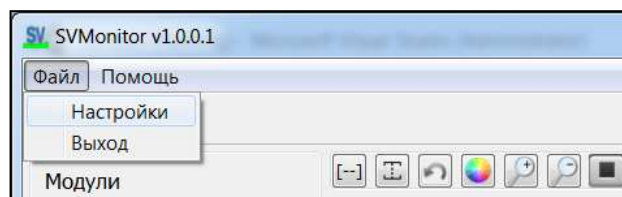


Рис.3.1. Выбор пункта меню «Настройки»

Откроется окно «Настройки»

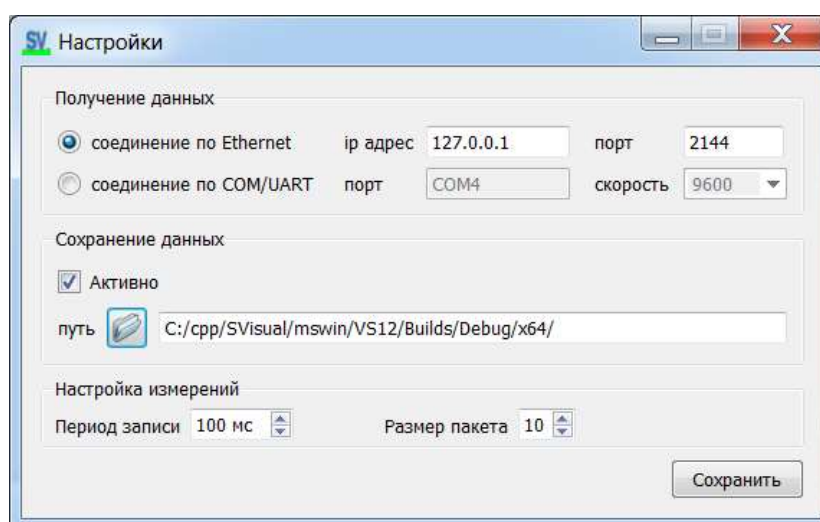


Рис.3.2 Окно «Настройки».

При соединении по COM выберите «соединение по COM», задайте порт и скорость. Остальные параметры передачи защиты жестко в ПО: ByteSize = 8, NoParity, NoFlowControl, StopBits::OneStop.

При соединении по Ethernet выберите «соединение по Ethernet», задайте IP адрес и порт.

Для вступления в силу изменений параметров «Получение данных» требуется перезагрузка ПО.

Для возможности просмотра архива записанных сигналов на панели «Сохранение данных» активируйте сохранение данных, задайте путь до папки.

Период записи – цикл, с которым значения попадают в буфер для отправки, размер пакета определяет период отправки всего буфера данных (см. пункт 5)

#### 4. Описание пользовательского интерфейса

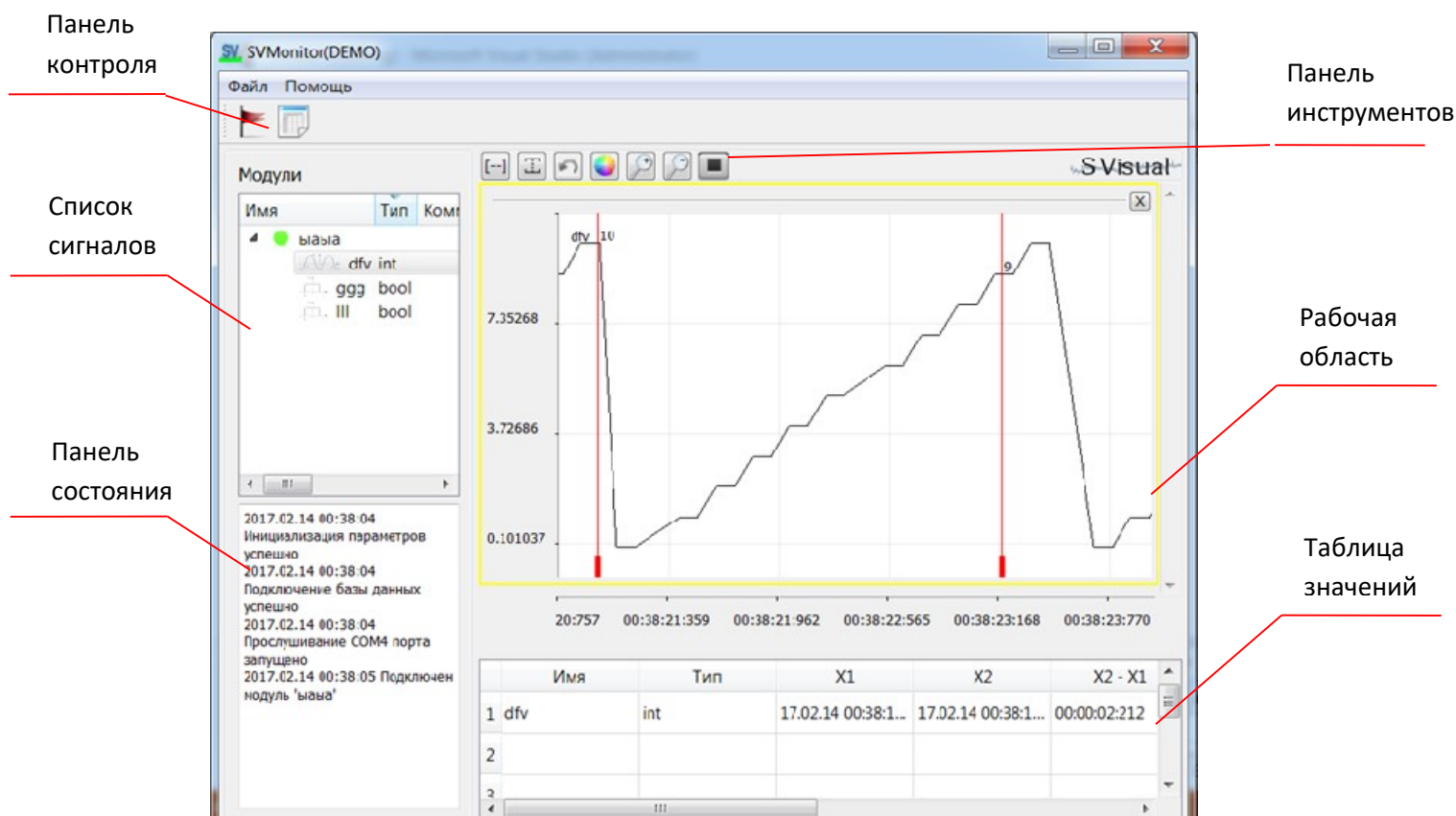


Рис.4.1. Главное окно программы.

- Текущие сигналы подключенных устройств отображаются в списке сигналов.
- Рабочая область предназначена для показа значений выбранных сигналов в реальном времени получения данных и в стабильном состоянии.
- Таблица значений показывает текущие значения маркеров сигналов.
- Панель инструментов позволяет производить манипуляции над графиком сигналов.
- Панель состояния показывает сообщения системы о наступивших событиях: подключение модуля, добавление нового сигнала и т.д.

#### 4.1. Список модулей сигналов

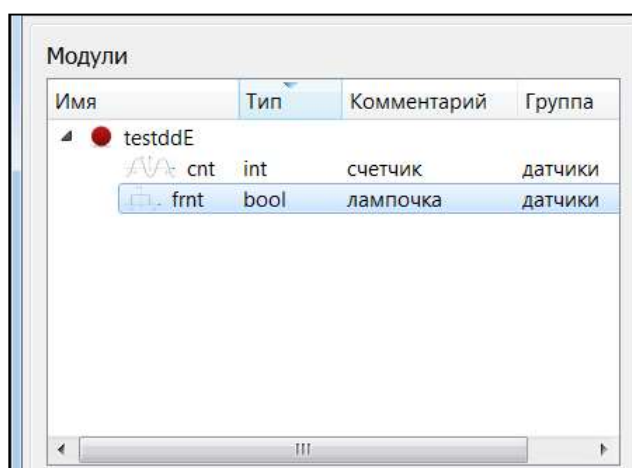


Рис.4.2. Список модулей сигналов.

Список модулей сигналов содержит подключенные устройства в данный момент времени. Используется для выбора сигнала для показа в рабочей области.

Индикация слева от модуля показывает состояние передачи данных.

Двойной щелчок ЛКМ на сигнале приводит к отображению сигнала в рабочей области. Также сигнал можно перенести в рабочую область не отпуская ЛКМ.

Если сигнал или модуль в данный момент не используется, его можно удалить: нажать ПКМ и выбрать пункт «Удалить».

Для удаления активного модуля, модуль нужно сначала отключить, затем появится возможность удаления.

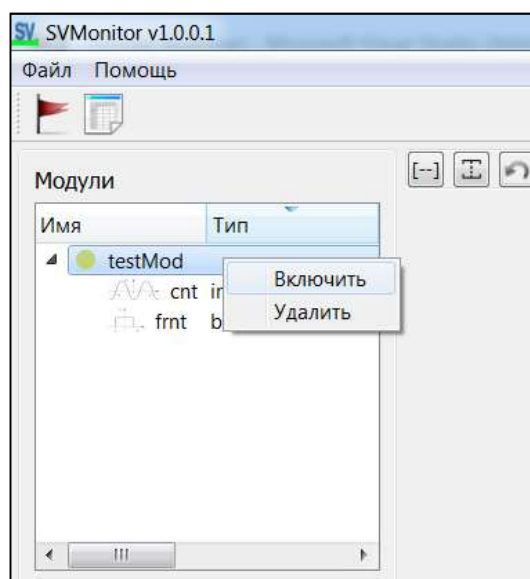


Рис.4.3. Удаление модуля.

Дополнительные поля списка «Комментарий» и «Группа» заполняются пользователем и используются при просмотре записанного архива данных.

## 4.2. Панель состояния

Панель состояния приложения показывает диагностические сообщения:

- изменение списка модулей сигналов;
- срабатывание триггеров сигналов (см. раздел 6);
- физическое подключение-отключение модулей.

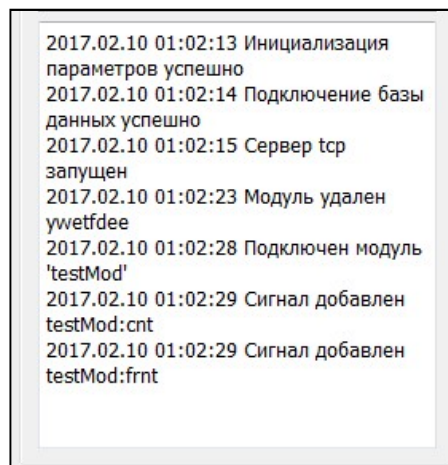


Рис.4.4. Панель состояния.

### 4.3. Панель инструментов

Панель инструментов используется в работе с окнами рабочей области.



Рис.4.5. Панель инструментов.

Команды с панели инструментов применяются к сигналам выбранного окна рабочей области. Выбранное окно обведено желтой рамкой.



#### 4.4. Рабочая область

Предназначена для отображения выбранных сигналов в реальном времени.

Легенда, слева сверху в рабочей области, содержит список сигналов типа «Наименование Комментарий», цвет легенды сигнала совпадает с цветом графика сигнала.

Детальный просмотр сигнала возможен при отключении активного вывода (отображение текущих полученных значений, - кнопка «старт-стоп» на панели инструментов ).

Используя мышь, вы можете:

- масштабировать сигнал, вращая колесико мыши,
- масштабировать отдельно оси координат,
- выделять область интереса,
- двигаться в рабочей области зажав ПКМ.

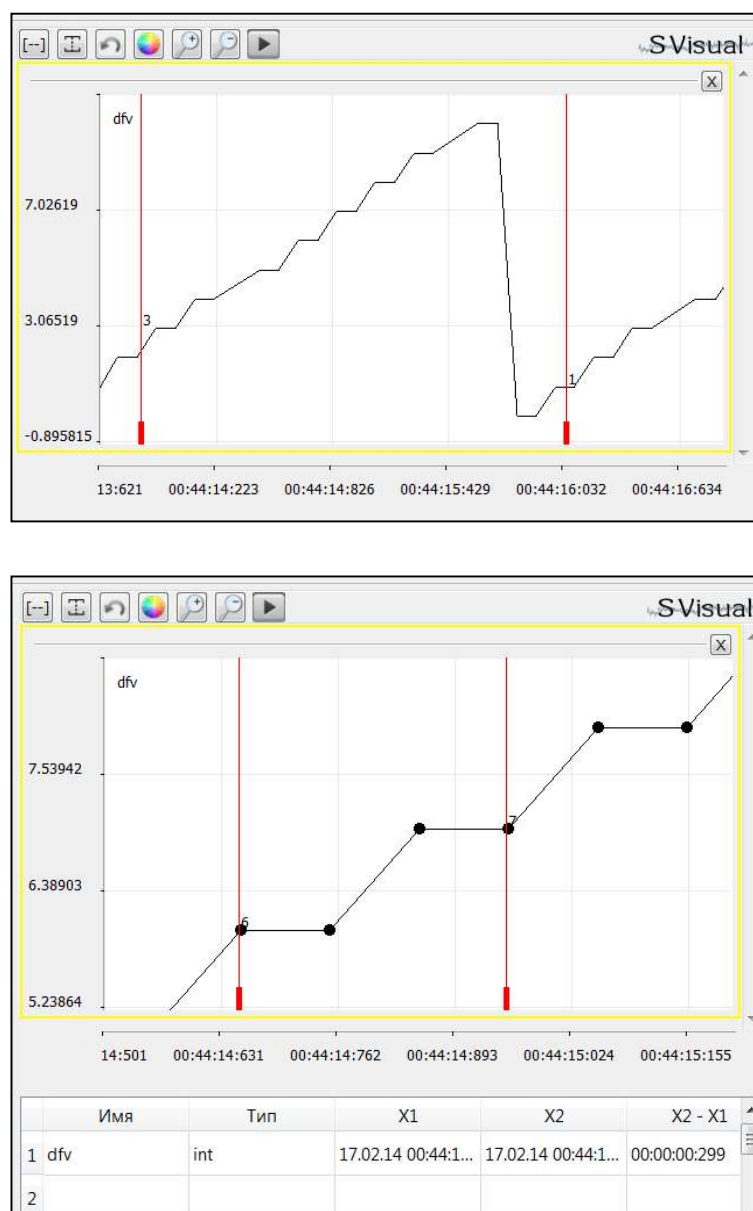


Рис.4.6. Выделение области интереса сигнала.

Маркеры – красные линии в рабочей области, предназначены для показа текущего значения сигнала.

Значение сигнала обновляется при перемещении маркера мышью и отображается на графике и в таблице значений снизу рабочей области.

Двигая мышью по графику, нажав ЛКМ можно видеть текущие значения сигнала на графике.

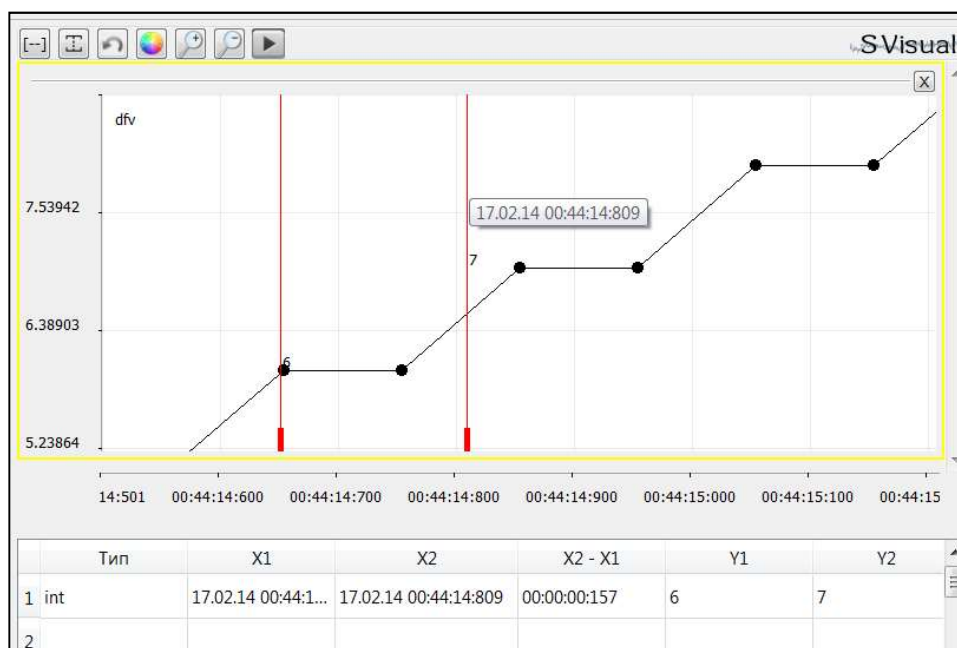


Рис.4.7. Маркер в рабочей области.

В одно окно рабочей области может быть добавлено неограниченное количество сигналов, при этом все сигналы масштабируются по одной оси.

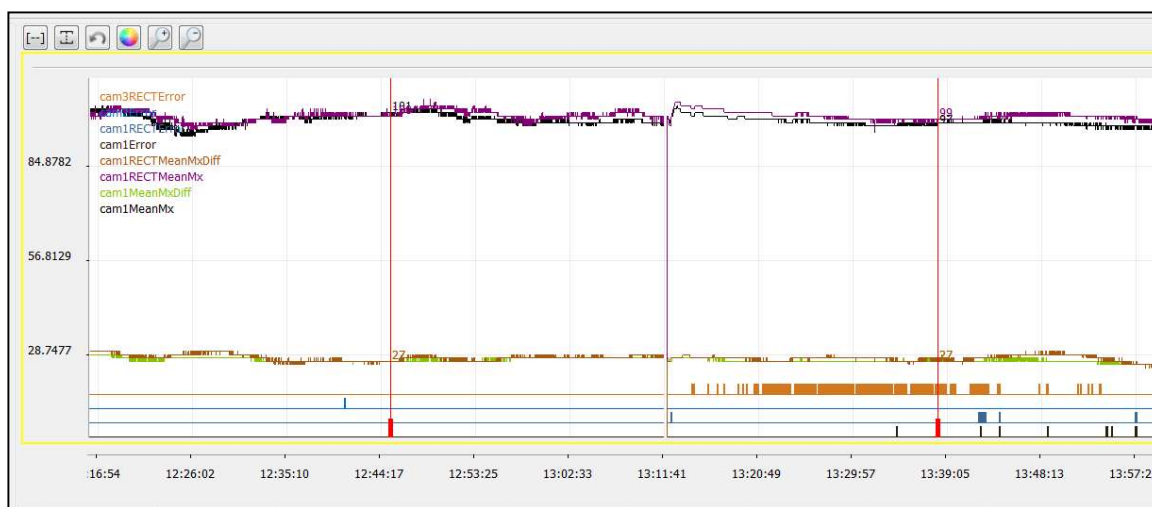


Рис. 4.8. Множество сигналов в окне рабочей области.

Для удаления сигнала из рабочего окна нажмите ПКМ на легенде сигнала.

Если ваши сигналы имеют разную размерность и вы хотите сопоставить сигналы, вы можете добавить сигнал в другое окно рабочей области.

Для этого достаточно выбрать сигнал в таблице, и не отпуская ЛКМ переместить сигнал ниже текущего окна рабочей области. Будет создано другое окно для этого сигнала.



Рис.4.9. Дополнительные окна в рабочей области.

Также сигнал можно перенести на другую ось в этом же окне, для этого достаточно выбрать сигнал в окне, и не отпуская ЛКМ переместить сигнал в это же окно. При этом сигнал всегда будет максимально отмасштабирован по оси ординат и не связан с другими сигналами окна.

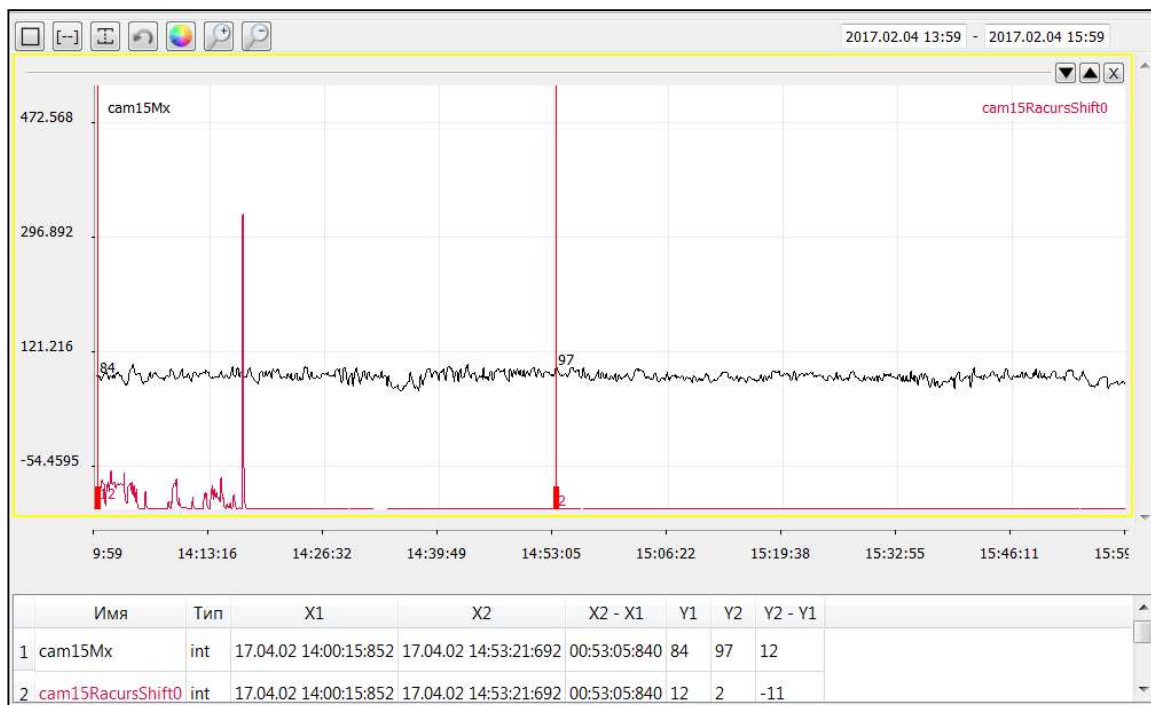


Рис.4.10. Альтернативная ось окна.

## 4.6. Таблица значений

Таблица значений служит для показа текущего значения сигнала в точке нахождения маркера.

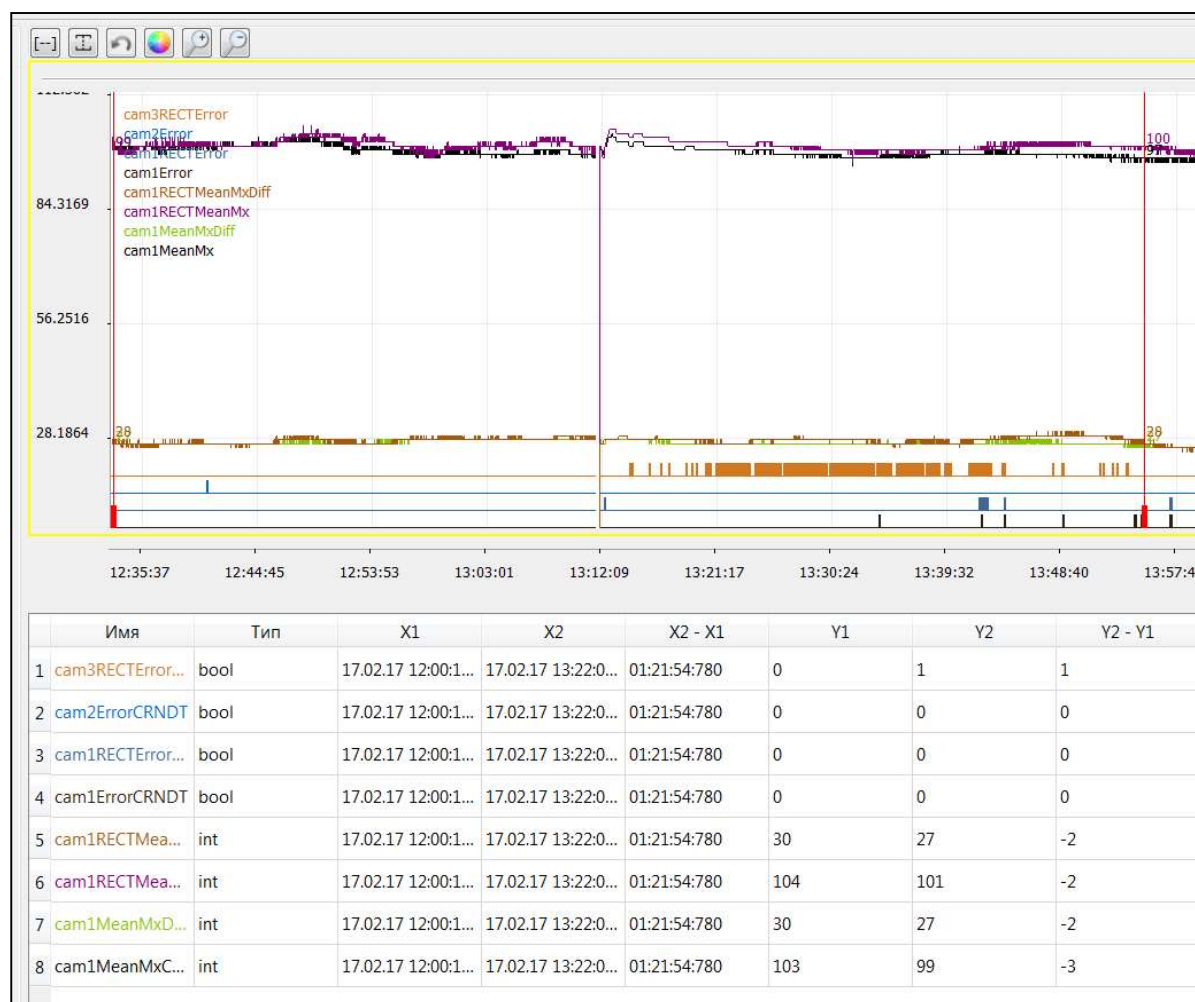


Рис.4.11. Таблица значений.

## 5. Создание пользовательских уведомлений

Триггер – это условие, которое может быть выполнено, например, срабатывание датчика при превышении уровня воды.

Уведомление – это реакция на событие сработавшего триггера.

В качестве уведомлений используется вызов стороннего процесса компьютера, на котором запущен SVMonitor. Это может быть любое приложение ОС, - консольная программа, python скрипт, любая другая программа.

На панели контроля нажмите кнопку с флажком.

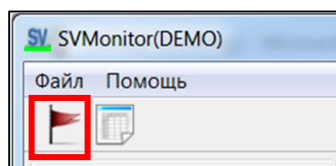


Рис.5.1. Открытие окна настройки пользовательских уведомлений.

Откроется окно настройки уведомлений.

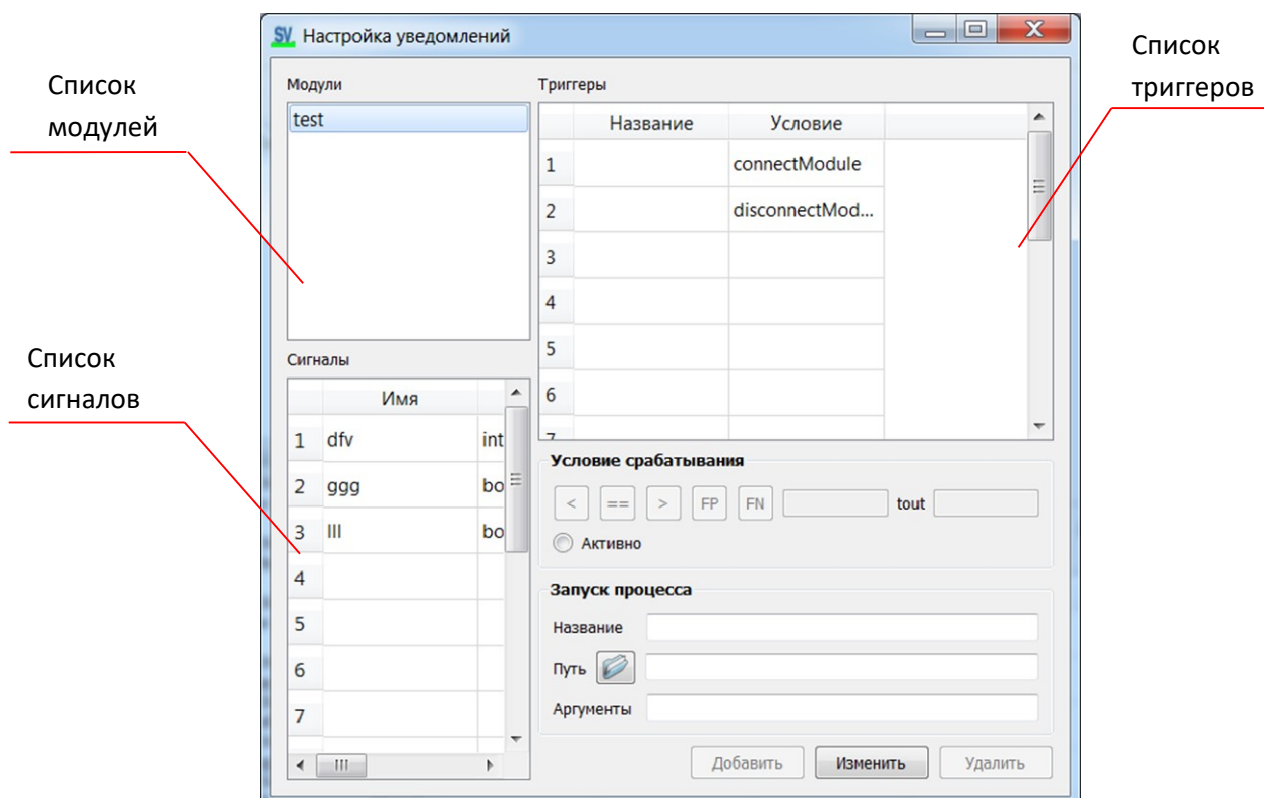


Рис.5.2. Окно настройки уведомлений.

На панели «Условие срабатывания» вы выбираете условие, при котором произойдет срабатывание триггера и соответственно вызов внешнего процесса.

На панели «Запуск процесса» вы задаете название триггера, путь к процессу который требуется запустить при срабатывании условия, аргументы процесса задаются через пробел.

## 5.1. Триггеры для модуля

При выборе модуля из списка модулей, в таблице триггеров появляется два стандартных триггера: «подключение модуля» и «отключение модуля». Эти триггеры создаются автоматически при подключении нового модуля.

Для модулей нельзя удалять стандартные триггеры, и добавлять пользовательские триггеры.

Триггер «подключение модуля» (connectModule в столбце «Условие» ) срабатывает когда известный модуль вновь подключается к сети.

Триггер «отключение модуля» (disconnectModule в столбце «Условие» ) срабатывает когда модуль отключается от сети.

Если вы хотите обработать события подключения/отключения модуля, вы должны:

- выбрать условие в таблице триггеров;
- в секции «Условие срабатывания» нажать кнопку «Активно»;
- в секции «Запуск процесса» придумать и написать название триггера, выбрать путь к нужному процессу;
- нажать кнопку «Изменить».

	Название	Условие
1		connectModule
2		disconnectMod...
3		
4		
5		
6		
7		

**Условие срабатывания**

< == > FP FN tout

☒ Активно

**Запуск процесса**

Название: Блокнот

Путь: C:/Program Files (x86)/Notepad++/notepad++.exe

Аргументы:

\* Добавить Изменить Удалить

Рис.5.3. Создание стандартного триггера.



## 5.2. Триггеры для сигнала

Для любого записываемого сигнала можно создать триггер, выбрав условие срабатывания:

- выберите сигнал из списка сигналов;
- выберите условие срабатывания;

Для сигналов типа int и real условия формулируются так:

«значение сигнала больше (>)/меньше (<)/равно (==) значению заданного порога».

Для сигналов типа bool условия формулируются так:

«положительный(FP)/отрицательный(FN) фронт сигнала».

(подробное описание п.5.3)

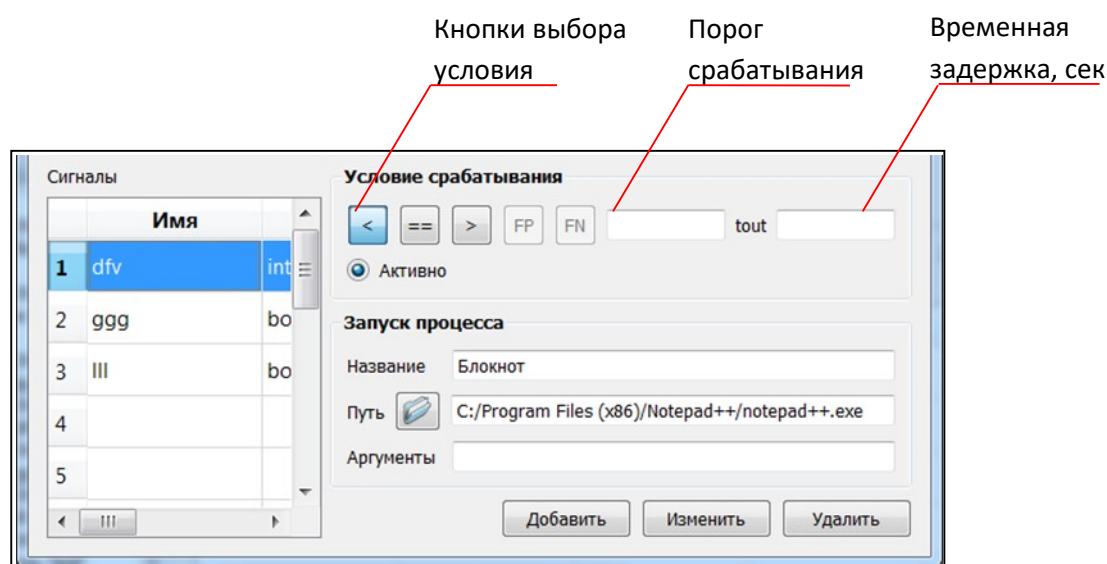


Рис.5.4. Создание триггера для сигнала.

- в секции «Условие срабатывания» задайте порог срабатывания и временную задержку (при необходимости);
- нажмите кнопку «Активно»;
- в секции «Запуск процесса» написать название триггера, выбрать путь к нужному процессу;
- нажать кнопку «Добавить».

В таблице триггеров вы увидите созданный триггер с заданным именем и условием.

Созданные триггеры можно редактировать, деактивировать, удалять. Для этого нужно выбрать триггер в таблице триггеров и нажать кнопку снизу «Изменить» либо «Удалить».

### 5.3. Порядок работы триггеров

Условие срабатывания триггера – выход значения целевого сигнала за пределы заданного порога с учетом установленной задержки (задается в секундах).

Пример 1. Сигнал типа integer, условие «значение > порога, задержка > 0».

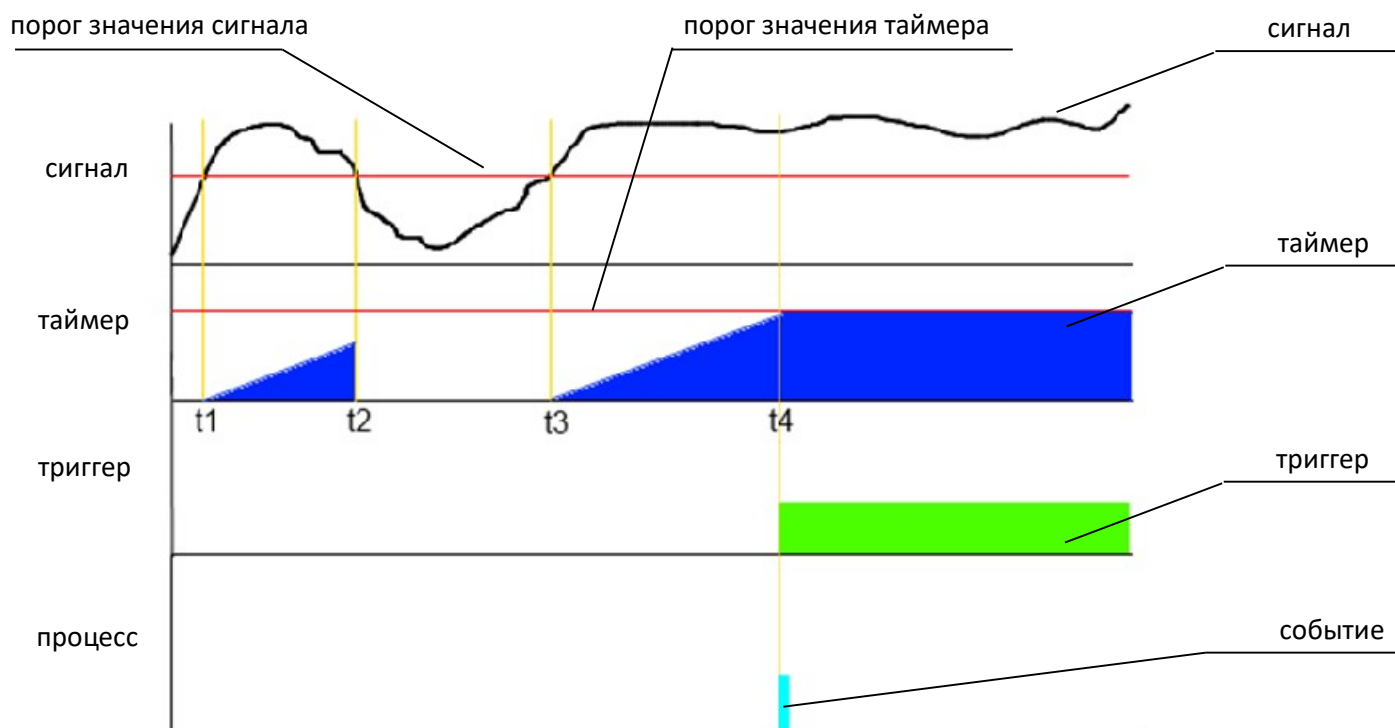


Рис.5.5. Пример №1.

В момент  $t_1$  значение сигнала превысило порог, начался отсчет времени таймера.

В момент  $t_2$  значение сигнала стало меньше порога, таймер не успел отсчитать заданный интервал времени и сбросился в ноль.

В момент  $t_3$  значение сигнала вновь превысило порог, снова начался отсчет времени таймера.

В момент времени  $t_4$  таймер отсчитал заданную паузу, и сработал триггер – запустился пользовательский процесс.

Пример 2. Сигнал типа integer, условие «значение > порога, задержка > 0».

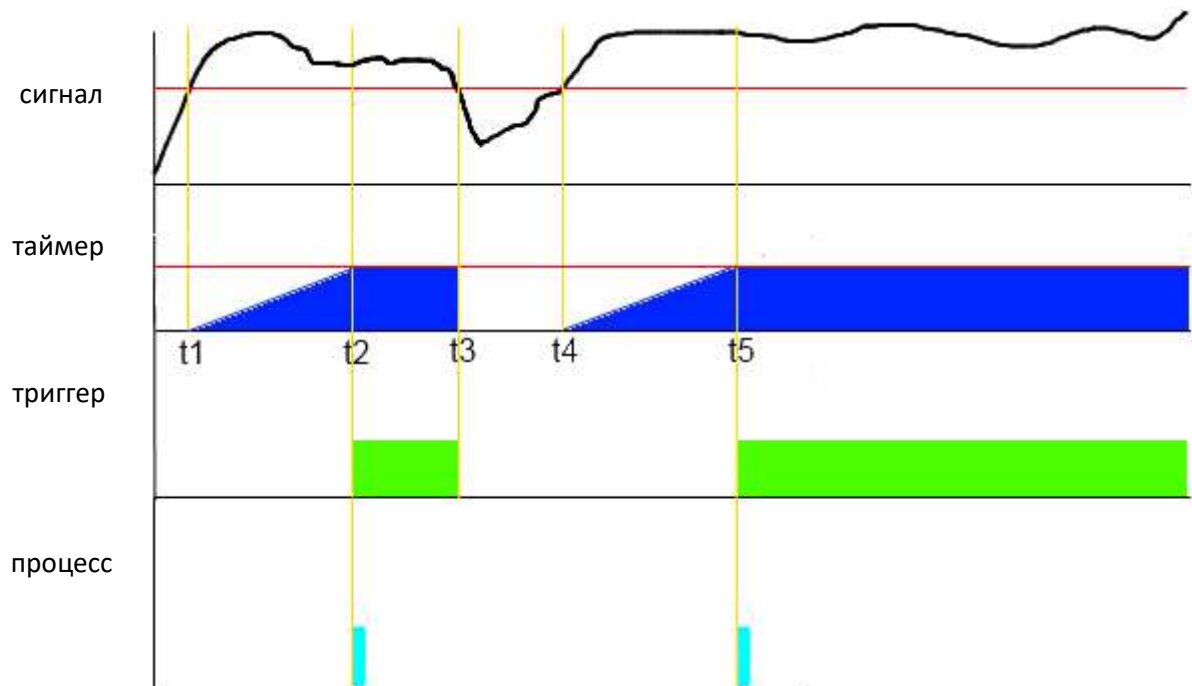


Рис.5.6. Пример №2.

В момент t1 значение сигнала превысило порог, начался отсчет времени таймера.

В момент t2 таймер отсчитал заданную паузу, и сработал триггер – запустился пользовательский процесс.

В момент t3 значение сигнала опустилось ниже порога, таймер и триггер сбросились.

В момент t4 значение сигнала вновь превысило порог, начался отсчет времени таймера.

В момент t5 таймер снова отсчитал заданную паузу, и снова сработал триггер – запустился пользовательский процесс.

Пример 3. Сигнал типа integer, условие «значение < порога, задержка > 0».

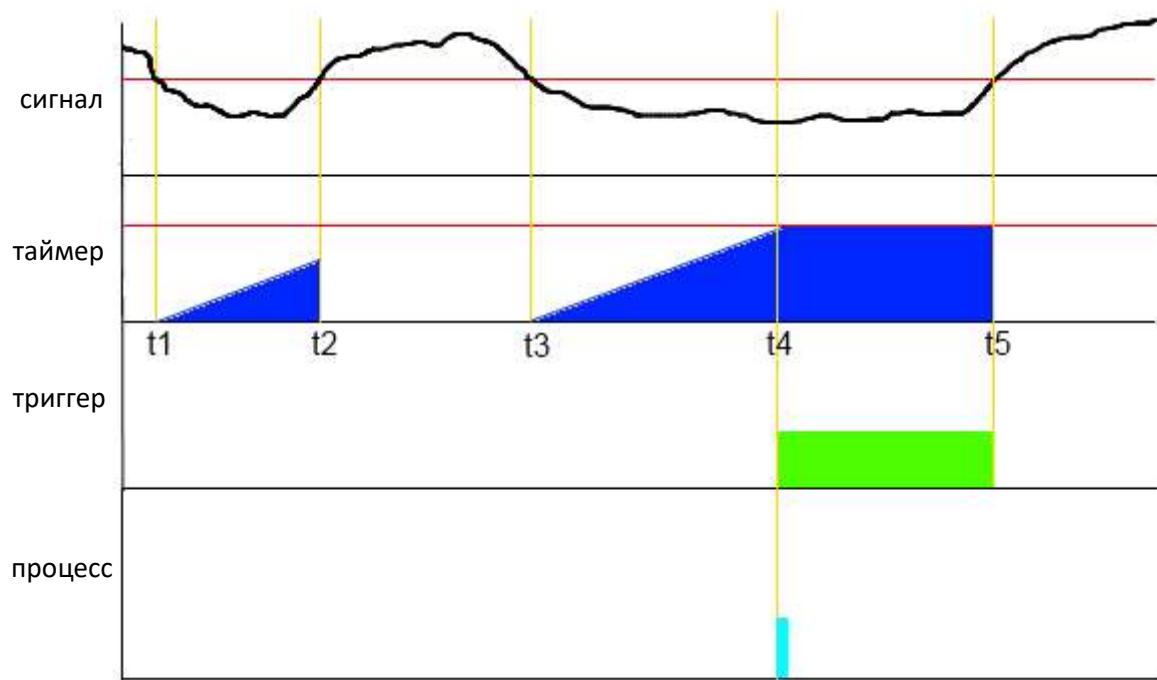


Рис.5.7. Пример №3.

В момент t1 значение сигнала стало ниже порога, начался отсчет времени таймера.

В момент t2 значение сигнала стало выше порога, таймер не успел отсчитать заданный интервал времени и сбросился в ноль.

В момент t3 значение сигнала вновь ниже порога, снова начался отсчет времени таймера.

В момент времени t4 таймер отсчитал заданную паузу, и сработал триггер – запустился пользовательский процесс.

В момент t5 значение сигнала стало выше порога, таймер и триггер сбросились.

Пример 4. Сигнал типа bool, условие «значение FP (положительный фронт), задержка > 0».

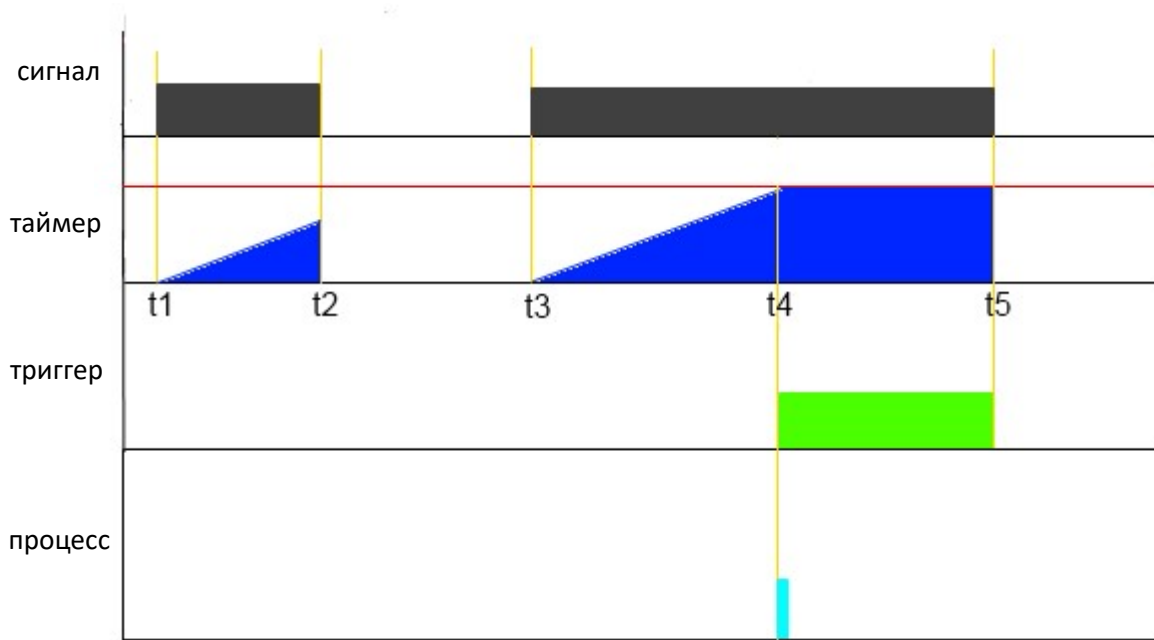


Рис.5.8. Пример №4.

В момент t1 значение сигнала стало TRUE, начался отсчет времени таймера.

В момент t2 значение сигнала стало FALSE, таймер не успел отсчитать заданный интервал времени и сбросился в ноль.

В момент t3 значение сигнала вновь TRUE, снова начался отсчет времени таймера.

В момент времени t4 таймер отсчитал заданную паузу, и сработал триггер – запустился пользовательский процесс.

В момент t5 значение сигнала стало FALSE, таймер и триггер сбросились.

Пример 5. Сигнал типа bool, условие «значение FN (отрицательный фронт), задержка == 0».

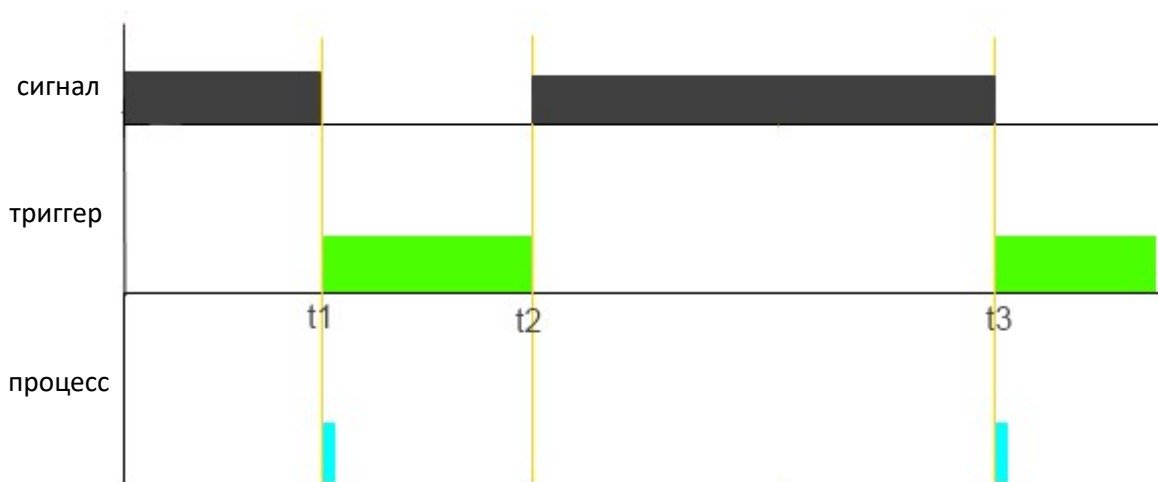


Рис.5.9. Пример №5.

В момент времени t1 значение сигнала стало FALSE, сработал триггер – запустился пользовательский процесс.

В момент t2 значение сигнала стало TRUE, триггер сбросился.

В момент t2 значение сигнала вновь стало FALSE, сработал триггер – запустился пользовательский процесс.

## 6. Отчет по активным событиям

На панели контроля нажмите кнопку с таблицей.

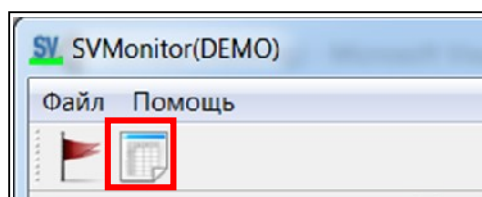
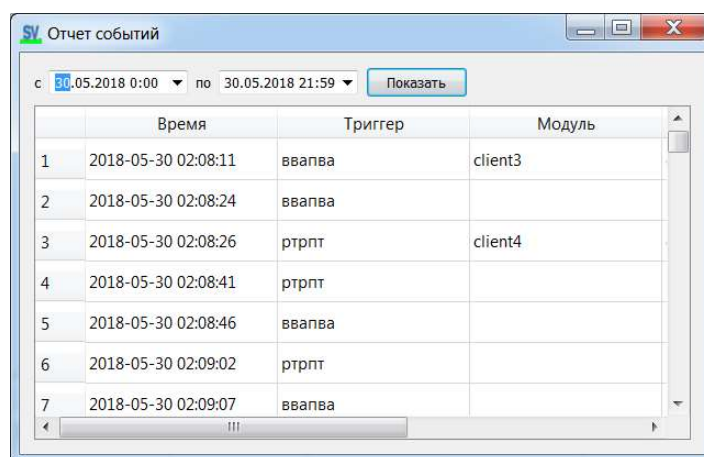


Рис.6.1. Открытие окна отчета.

Откроется окно отчета.



	Время	Триггер	Модуль
1	2018-05-30 02:08:11	ввапва	client3
2	2018-05-30 02:08:24	ввапва	
3	2018-05-30 02:08:26	ртрпт	client4
4	2018-05-30 02:08:41	ртрпт	
5	2018-05-30 02:08:46	ввапва	
6	2018-05-30 02:09:02	ртрпт	
7	2018-05-30 02:09:07	ввапва	

Рис.6.2. Окно отчета.

Отчет содержит время срабатывания триггера, условие срабатывания.

## 7. Просмотр архива данных

Для просмотра записанных файлов данных необходимо запустить приложение SVViewer.exe.

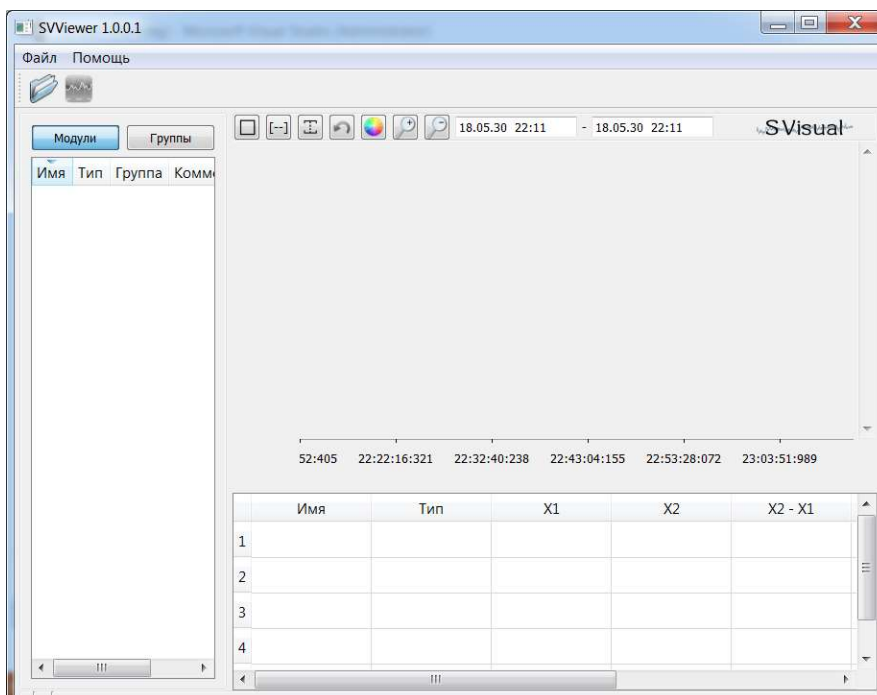


Рис.7.1. Окно просмотрщика архива.

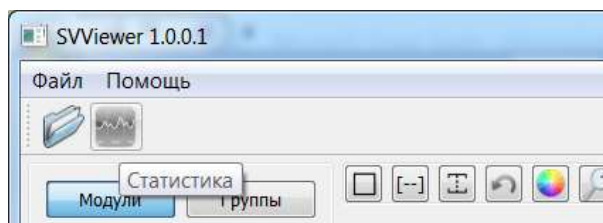
Все операции в окне просмотра аналогичны описанным выше для SVMonitor.

Для добавления файлов данных, нажмите на кнопку «открыть» в панели управления и выберите файлы архива.



## 7.1. Просмотр статистики по архивным данным

Для показа гистограммы изменения сигнала нажмите кнопку «статистика» на панели управления.



Откроется окно для построения гистограммы. Перетащите выбранный сигнал в окно.

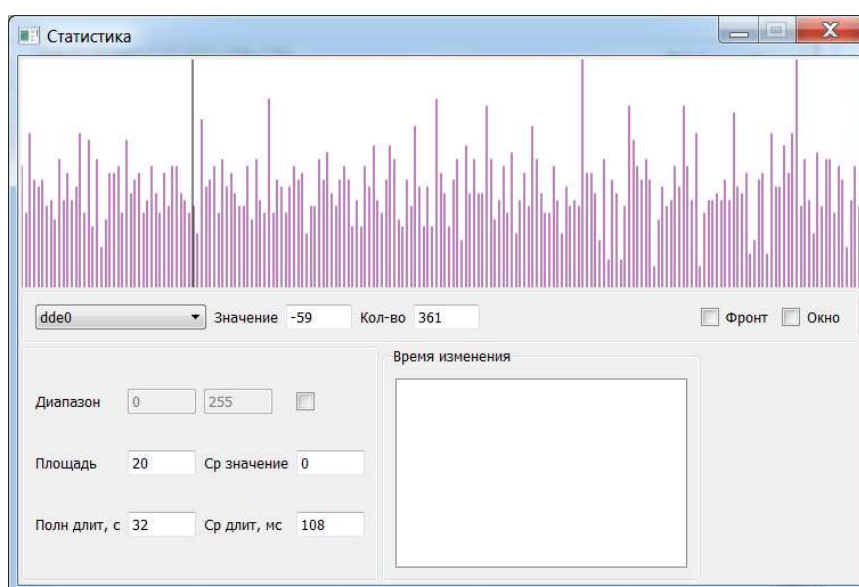


Рис.7.2. Окно статистики.

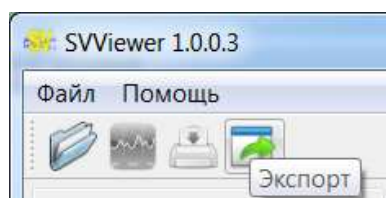
В окне отображается гистограмма значений сигнала, площадь гистограммы до маркера слева, среднее значение, длительность текущего значения сигнала.

ПКМ на графике выбирает конкретное значение сигнала.

## 7.2. Экспорт данных в файл

Экспорт данных возможен в текстовый файл, файл json и в файл xlsx.

Для открытия окна экспорта нажмите кнопку «Экспорт» на панели управления.



Откроется окно для экспорта данных.

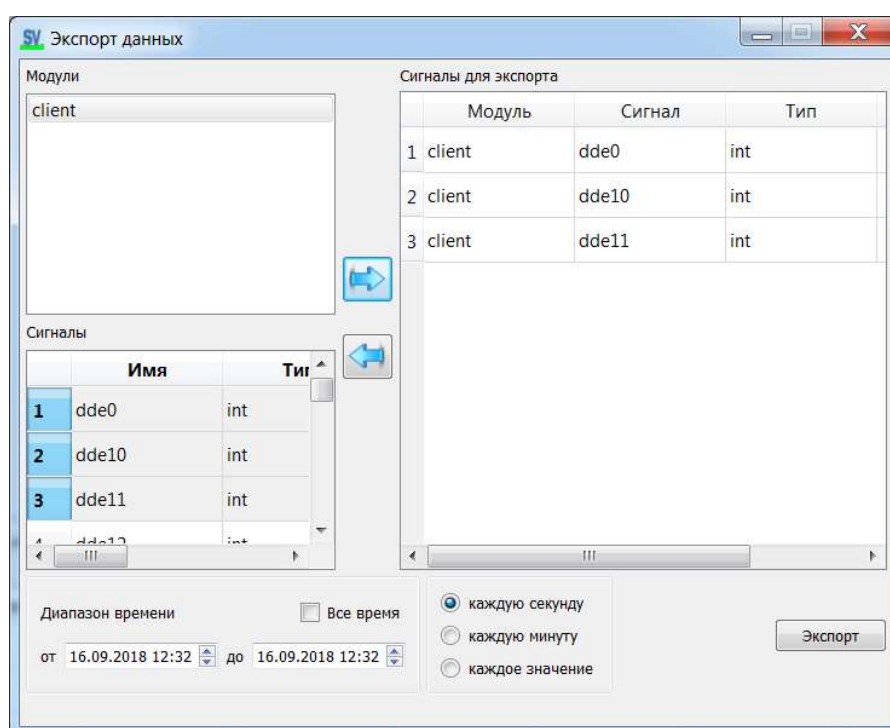


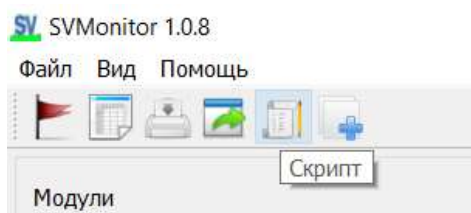
Рис.7.3. Окно экспорта.

Выберите сигналы для экспорта, выберите время и нажмите кнопку "Экспорт".

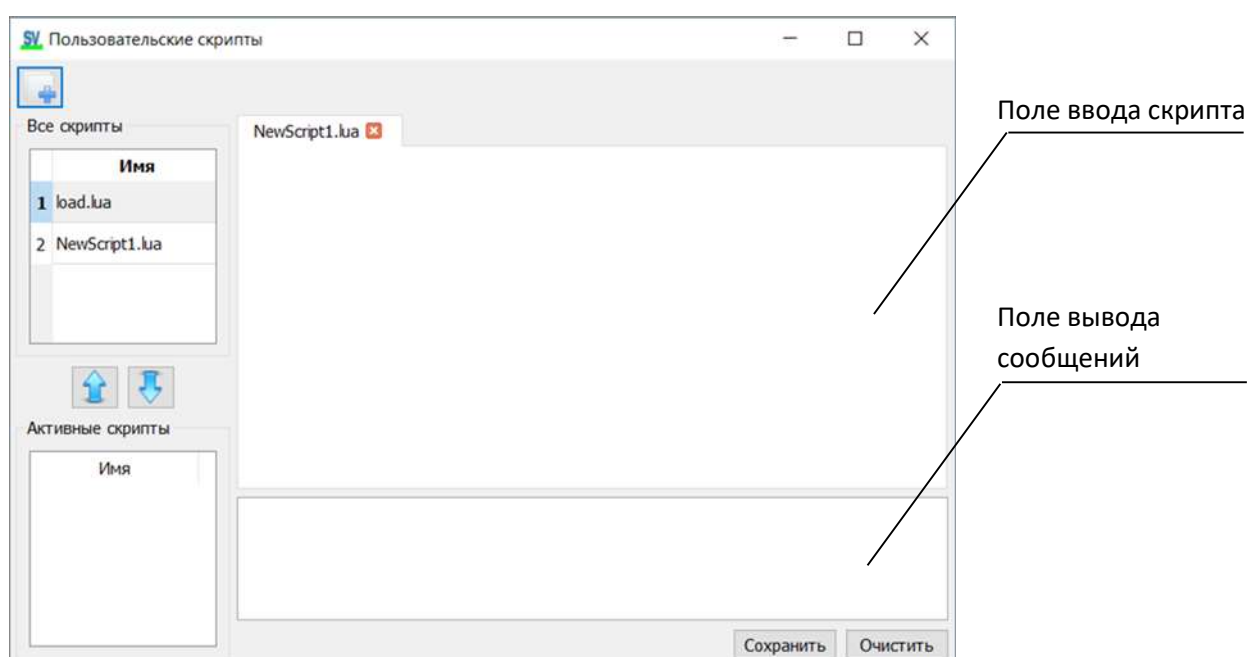
**Замечание:** для SVMonitor экспорт возможен только просматриваемых в данный момент сигналов.

## 8. Пользовательские скрипты – создание виртуальных сигналов

Для открытия окна скриптов нажмите кнопку «Скрипт» на панели управления.



Откроется окно для создания скриптов.



Все написанные скрипты подаются на вход интерпретатора языка [Lua](#).

Для создания нового файла скрипта нажмите кнопку сверху «Создать скрипт».

В списке «Все скрипты» двойной щелчок ЛКМ на названии скрипта откроет скрипт для редактирования.

Скрипт выполняется циклически на каждое значение сигнала.

Чтобы скрипт начал выполняться, выберите скрипт в списке «Все скрипты» и нажмите стрелочку «Добавить в активные».

В поле вывода сообщений помимо пользовательских сообщений (функция `printMess`) попадают сообщения об ошибках в скрипте.

Скрипт с названием «load.lua» выполняется только один раз при добавлении его в активные скрипты, это позволяет вам создать и инициализировать в нем глобальные переменные, которые могут быть использованы в других скриптах.

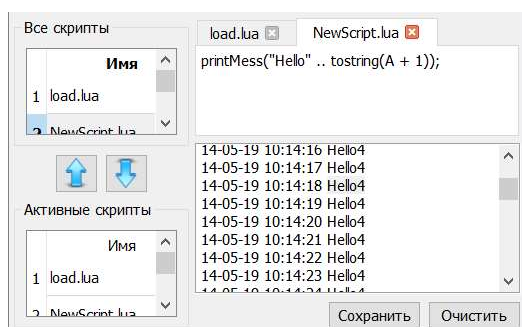
Чтобы удалить скрипт, нужно нажать ПКМ на выбранном в списке «Все скрипты».

Помимо стандартных функций языка Lua доступны следующие функции SV:

- вывод сообщения  
`void printMess(const std::string& mess);`
- получение времени сигнала, вмс  
`uint64_t getTimeValue(const std::string& module, const std::string& signal);`
- получение значения сигнала типа Bool  
`bool getBoolValue(const std::string& module, const std::string& signal);`
- получение значения сигнала типа Int  
`int getIntValue(const std::string& module, const std::string& signal);`
- получение значения сигнала типа Float  
`float getFloatValue(const std::string& module, const std::string& signal);`
- задание значения сигнала типа Bool  
`void setBoolValue(const std::string& signal, bool value, uint64_t time);`
- задание значения сигнала типа Int  
`void setIntValue(const std::string& signal, int value, uint64_t time);`
- задание значения сигнала типа Float  
`void setFloatValue(const std::string& signal, float value, uint64_t time);`

Пример 1. Вывод текстового сообщения.

1. Откройте скрипт load.lua в поле ввода и наберите:  
`A = 3; -- создадим и инициализируем переменную типа int`
2. Откройте скрипт NewScript.lua в поле ввода и наберите:  
`printMess("Hello" .. tostring(A + 1));`
3. Оба скрипта добавьте в активные.
4. В поле вывода вы должны увидеть:  
Hello4



## Пример 2. Преобразование значений сигнала.

1. Для примера, пусть дан синусоидальный сигнал:

```
int cp = 0;
while(true){

    SV_Cln::svAddFloatValue("sin", sin(cp *M_PI/ 180.0)* 100);

    cp += 1;if(cp > 359) cp = 0;

    Sleep(100);
}
```

2. Откройте скрипт NewScript.lua в поле ввода и наберите:

```
-- получаем время сигнала
tm = getTimeValue("client", "sin");

-- получаем значение сигнала
val = getFloatValue("client", "sin");

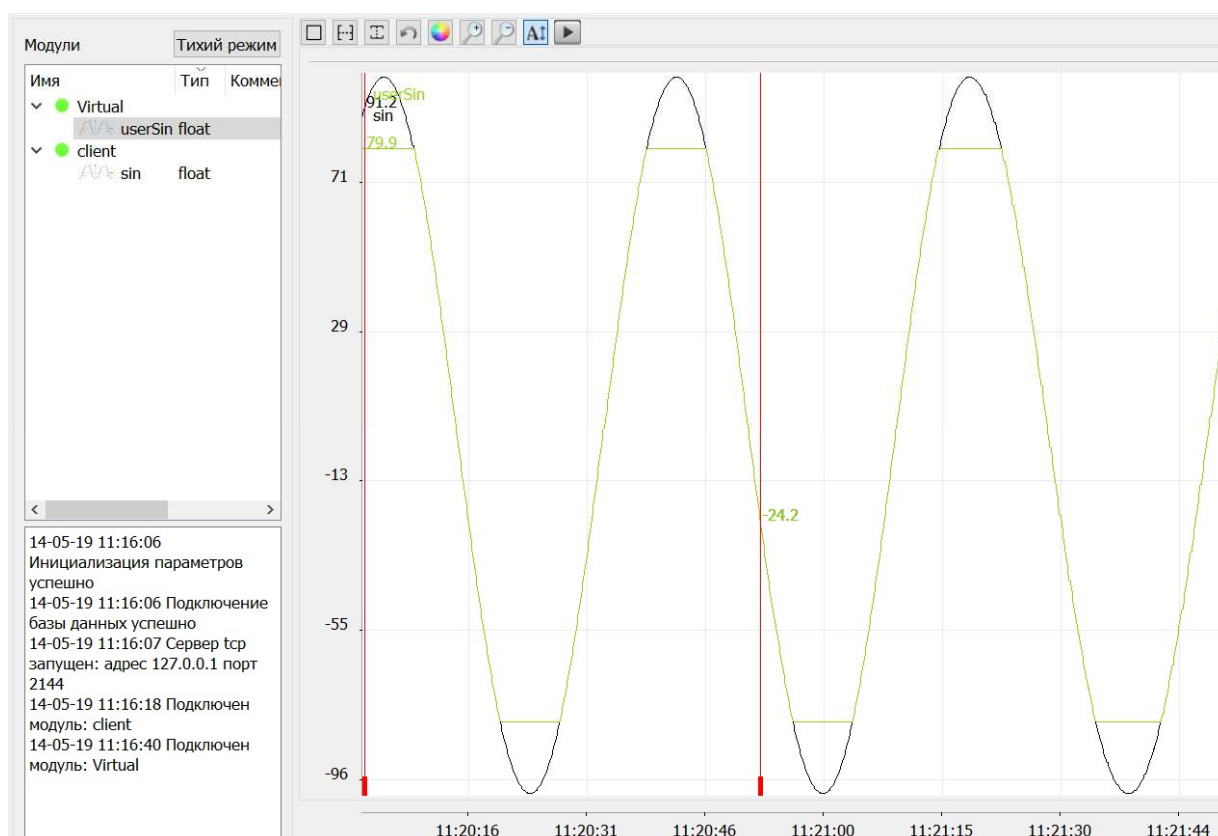
-- обрезаем "верх" синусоиды
usVal = (val > 80) and 80 or val;

-- обрезаем "низ" синусоиды
usVal = (usVal < -80) and -80 or usVal;

-- создаем новый виртуальный сигнал "userSin"
setFloatValue("userSin", usVal, tm);
```

3. Сделайте скрипт активным.

4. Будет подключен модуль «Virtual», на графике вы должны увидеть следующее:



## Пример 3. Выделение значений сигнала.

1. Для примера, пусть дан синусоидальный сигнал:

```
int cp = 0;
while(true){

    SV_Cln::svAddFloatValue("sin", sin(cp *M_PI/ 180.0)* 100);

    cp += 1;if(cp > 359) cp = 0;

    Sleep(100);
}
```

2. Откройте скрипт NewScript.lua в поле ввода и наберите:

```
-- получаем время сигнала
tm = getTimeValue("client", "sin");

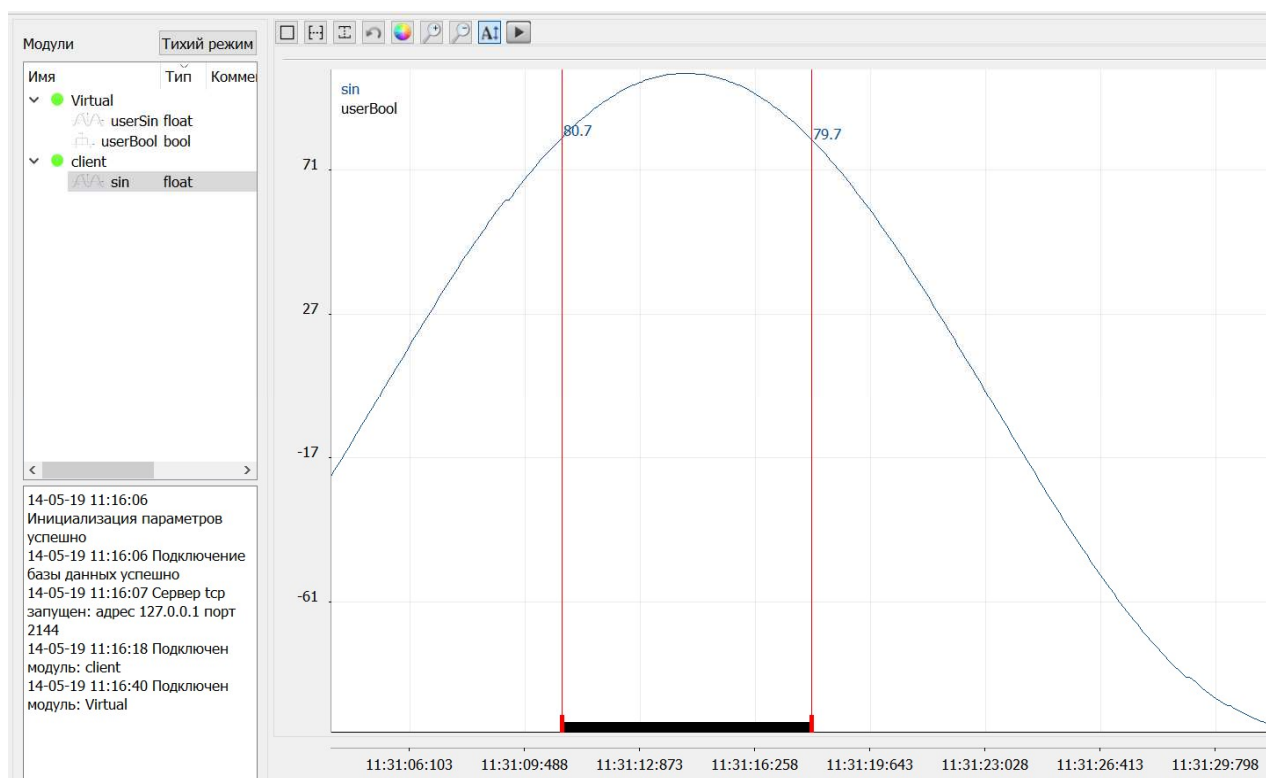
-- получаем значение сигнала
val = getFloatValue("client", "sin");

-- получаем булевый сигнал
usVal = (val> 80);

-- создаем новый виртуальный сигнал "userBool"
setBoolValue("userBool", usVal, tm);
```

3. Сделайте скрипт активным.

4. Будет подключен модуль «Virtual», на графике вы должны увидеть следующее:



## 9. Просмотр в браузере

В браузере возможен только просмотр активных в данный момент сигналов.

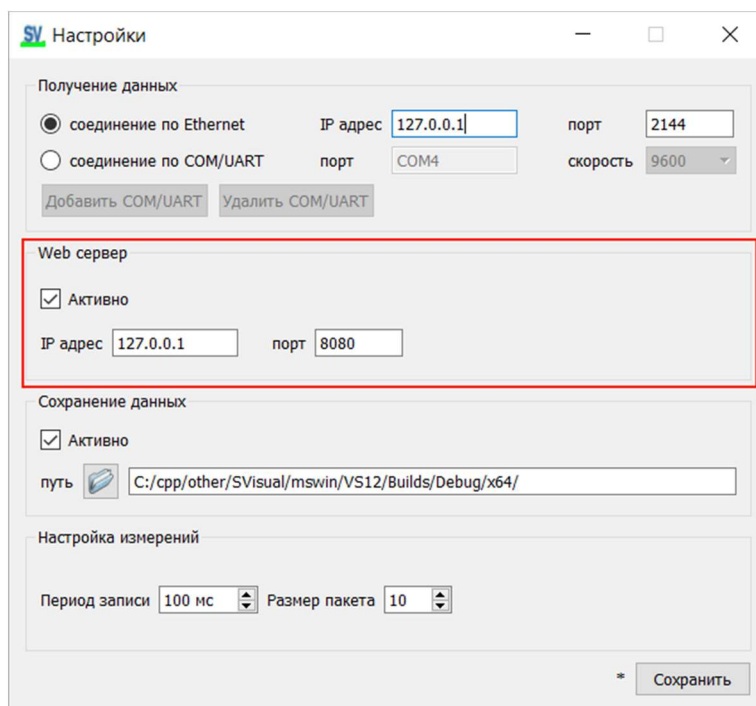


Рис. 9.1 Окно «Настройки».

В окне настроек задайте IP адрес и порт для веб-сервера, перезагрузите SVMonitor.

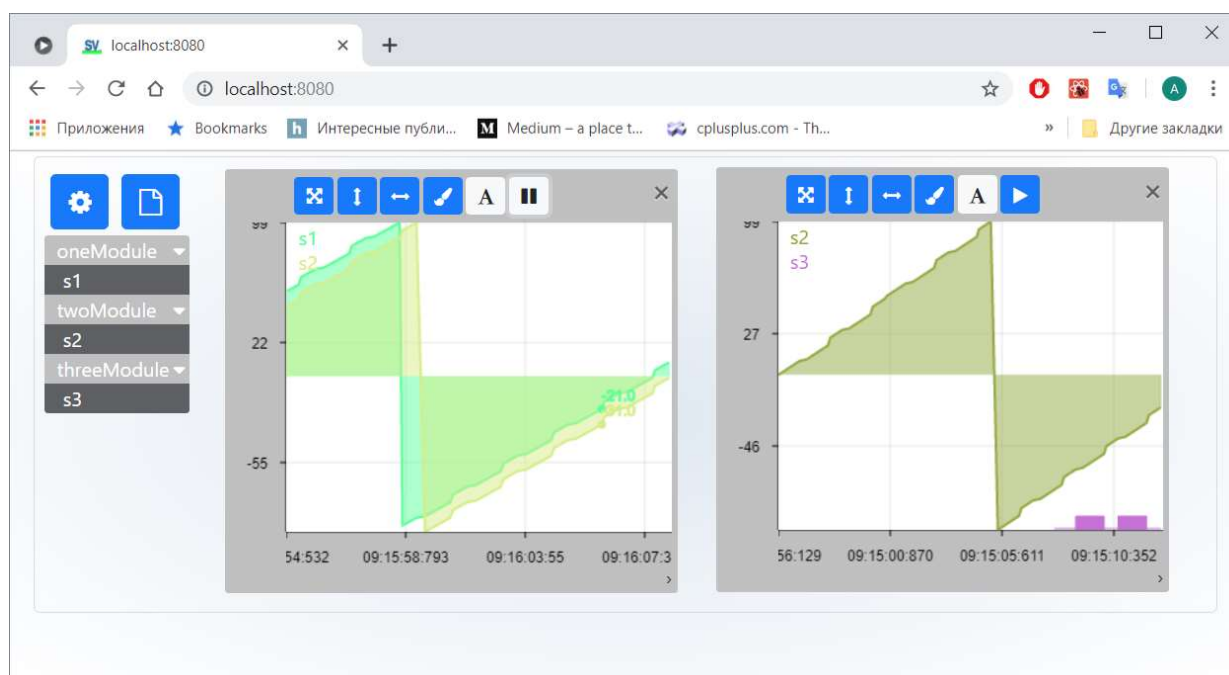


Рис. 9.2 Окно браузера.

Откройте браузер, в адресной строке напишите адрес, который вы указали в окне настроек.

Все операции просмотра на странице браузера аналогичны описанным выше для SVMonitor.

## 10. Поддержка Zabbix агента

В окне настроек задайте IP адрес и порт для Zabbix-агента, перезагрузите SVMonitor.

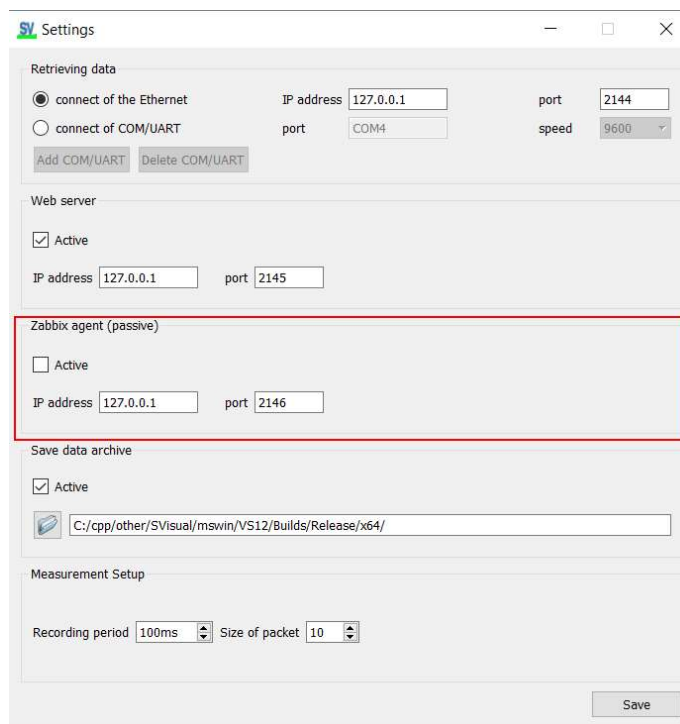


Рис. 10.1 Окно «Настройки».

Для передачи выбранного сигнала на Zabbix-сервер:

- Zabbix-агент должен послать на адрес заданный в настройках имя сигнала в формате: «ИмяСигналаИмяМодуля» (слитно);

- в ответ получит последнее на данный момент значение сигнала.

Для этого в ini-файле Zabbix-агента добавьте запись пользовательского параметра.

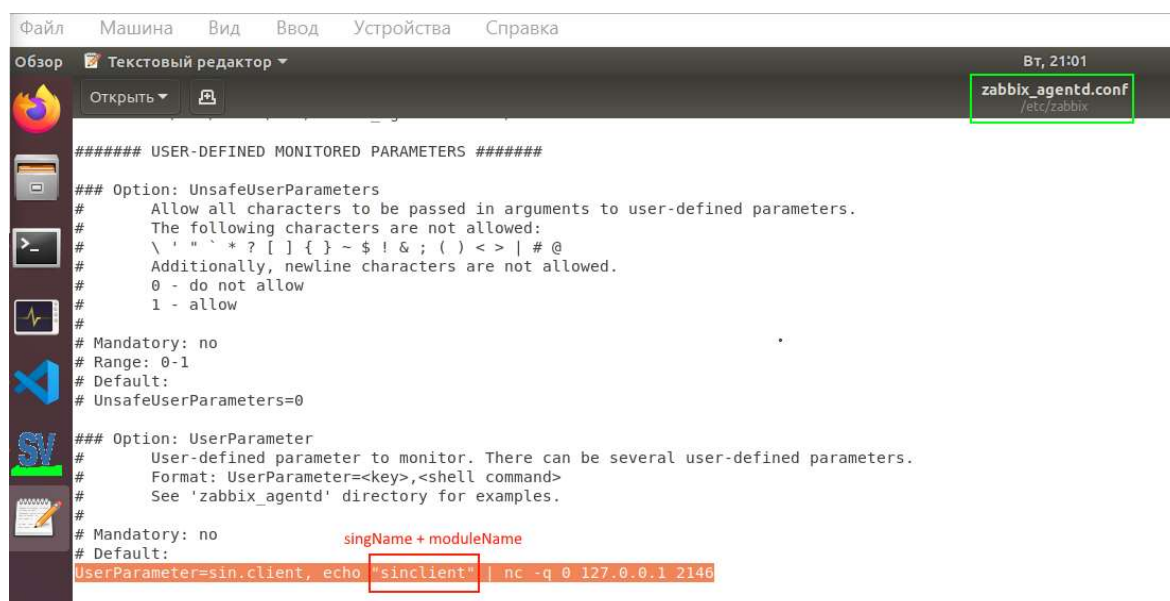


Рис. 10.2 etc/zabbix/zabbix\_agentd.conf



## 11. Описание внутренней архитектуры ПО

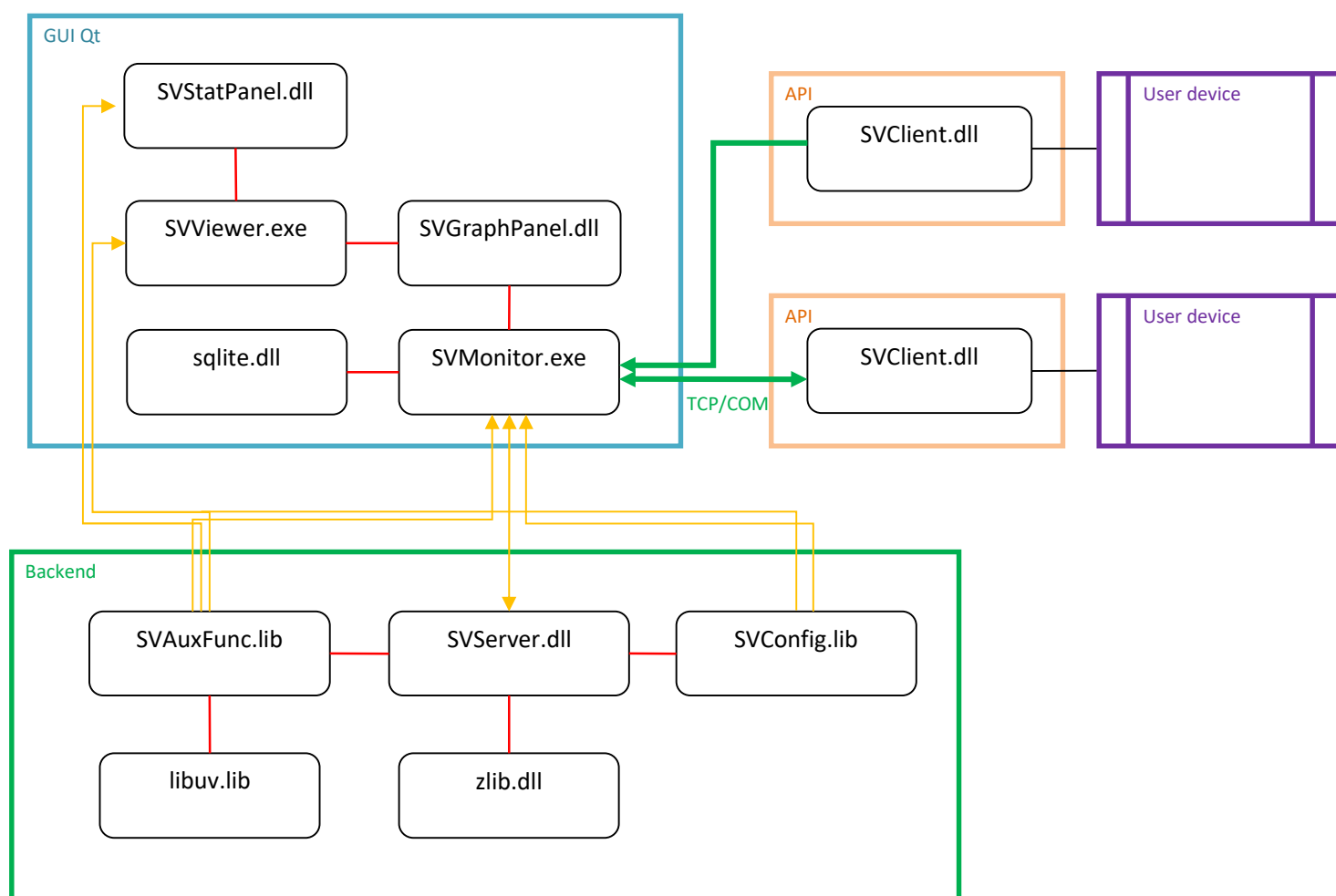


Рис.11.1. Архитектура ПО.

- SVClient.dll – кроссплатформенная библиотека для передачи данных от клиента, написана на C++, только STL. Нативные сокеты из SVAuxFunc.lib. Имеет C-интерфейс.

- SVMonitor.exe – главная графическая часть ПО, написана на C++, Qt + STL. Для получения данных от клиента использует QSerial для COM и libuv (в SVAuxFunc.lib) для TCP. Полученные данные передает серверной библиотеке SVServer.dll. Для отображения графиков используется библиотека SVGraphPanel.dll. БД sqlite.dll используется для хранения названий сигналов, триггеров, статистики событий.

- SVServer.dll – серверная библиотека, написана на C++, только STL. Буферизует переданные данные для отображения графика (для SVMonitor), записывает архив данных на диск, отслеживает цикл отработки триггеров. Архив сжимает с помощью zlib.dll. Непосредственно получением данных не занимается. Может быть использована отдельно от SVMonitor в консольном приложении, как сервис для записи архива.

-SVConfig.lib – хранит конфигурацию системы: ограничения на объем получаемых данных (кол-во сигналов, количество модулей, триггеров), глобальные типы структур.

-SVViewer.exe – просмотр архива сигналов, C++ Qt. Использует библиотеку SVGraphPanel.dll.

**Описание в свободном стиле.**

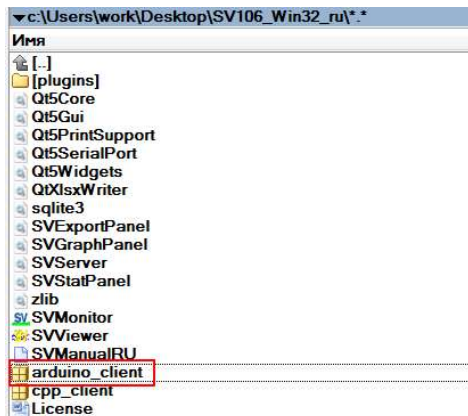
SVMonitor получает данные по TCP или COM, передает данные в SVServer для обработки.

Все сигналы добавляет динамически SVServer, все коллекции данных находятся в SVServer.

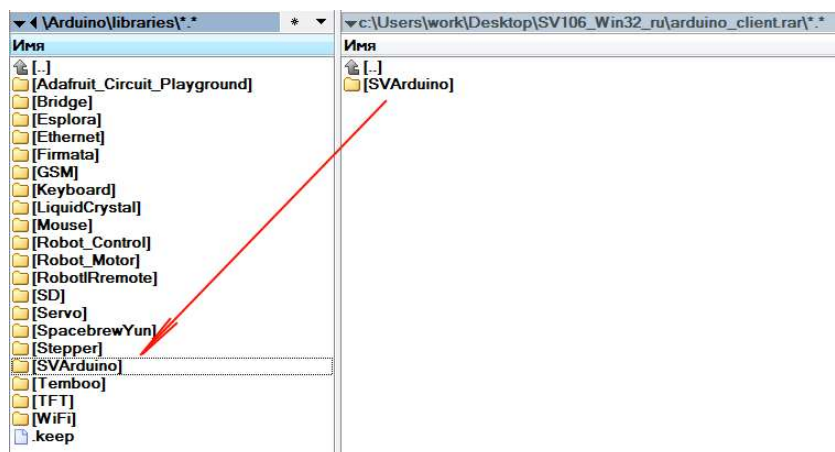
Принцип работы SVServer: циклический буфер, клиент записывает сигналы в текущую позицию буфера, одновременно с этим по буферу «бежит» читатель буфера в отдельном потоке. Читатель копит для каждого сигнала архив, размер архива 10 мин. По прошествии 10 мин архив сбрасывается на диск, то есть данные сжимаются и записываются в файл посылками по 10 минут. Еще один поток отвечает за обработку пользовательских триггеров. Триггеры добавляются/удаляются с SVMonitor.

## Приложение 1. Клиент для МК Arduino

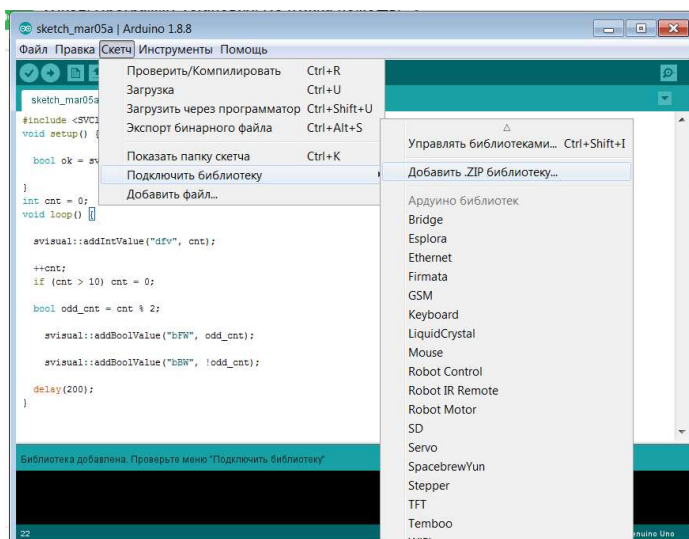
Клиент находится в папке с установленным ПО:



Разархивируйте туда, где у вас лежат библиотеки для Arduino:



Подключите библиотеку в Arduino IDE:



Вам не придется заботиться о передаче данных, передача данных происходит автоматически по таймерным прерываниям, используется таймер 2.

Необходимо лишь обновлять переменные для записи.

**При соединении по COM вы не должны использовать вывод сообщений по COM порту (Serial.Print(), Serial.Write()).**

Интерфейс

```
namespace svisual{

    // connect to server of ethernet
    // module - name module [24max]
    // macAddrModule - mac addr module
    // ipAddrModule - ipaddr module
    // ipAddrServ - ipaddr server
    bool connectOfEthernet(const char* module, const char* macAddrModule, const char*
ipAddrModule, const char* ipAddrServ, int portServ);

    // connect to server of wi-fi
    // module - name module [24max]
    // ssid - your network SSID
    // pass - secret password network
    // ipAddrServ - ipaddr server
    // portServ - port server
    bool connectOfWiFi(const char* module, const char* ssid, const char* pass, const char*
ipAddrServ, int portServ);

    // connect to server of COM
    // module - name module[24max]
    // ipAddrServ - ipaddr server
    // portServ - port server
    bool connectOfCOM(const char* module, int speed = 9600);

    // add bool value for rec
    // name [24max]
    bool addBoolValue(const char* name, bool value, bool onlyPosFront = false);

    // add int value for rec
    bool addIntValue(const char* name, int value);

    // add float value for rec
    bool addFloatValue(const char* name, float value);

};
```

connectOfEthernet - функция для соединения по Ethernet. Принимает IP адрес модуля Arduino, адрес сервера SVMonitor. Возвращает TRUE, если соединение установлено.

connectOfWiFi - функция для соединения по WiFi. Принимает название сети и пароль, адрес сервера SVMonitor. Возвращает TRUE, если соединение установлено.

connectOfCOM - функция для соединения по COM порту. Принимает название модуля и скорость передачи. Возвращает TRUE, если соединение установлено.

`addBoolValue(const char* name, bool value, bool onlyPosFront = false)` – функция используется для записи переменной типа `bool`.

`name` – имя переменной, не должно содержать сочетаний «=end=» и «=begin=» и превышать длину 24 символа;

`value` – значение переменной;

`onlyPosFront` – если требуется запись только положительного фронта сигнала.

`addIntValue(const char* name, int value)` – функция используется для записи переменной типа `int`.

`name` – имя переменной, не должно содержать сочетаний «=end=» и «=begin=» и превышать длину 24 символа;

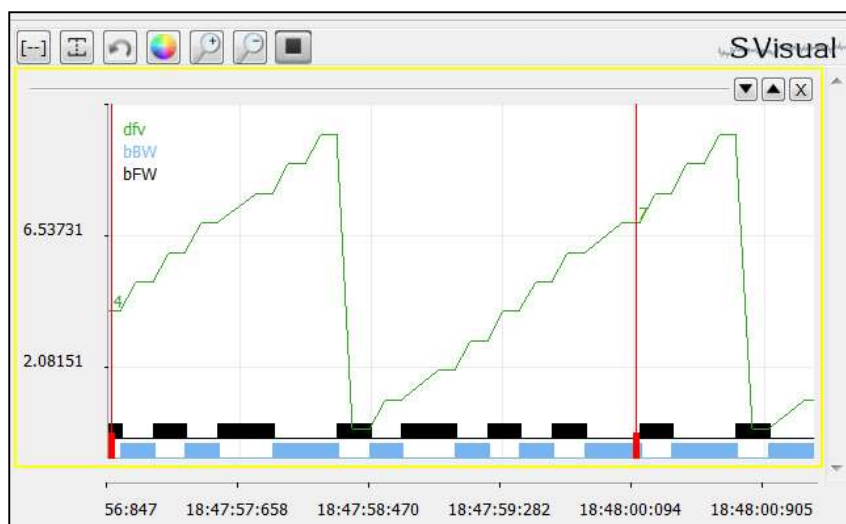
`value` – значение переменной;

## Пример скетча

```
#include <SVClient.h>

void setup() {
    bool ok = svisual::connectOfCOM("test");
}
int cnt = 0;
void loop() {
    svisual::addIntValue("dfv", cnt);
    ++cnt;
    if (cnt > 10) cnt = 0;
    bool isOdd = cnt % 2;
    svisual::addBoolValue("bFW", isOdd);
    svisual::addBoolValue("bBW", !isOdd);
    delay(200);
}
```

В результате вы увидите:



## Приложение 2. Клиент для C/C++.

Написан на C++. Может использоваться в многопоточных приложениях.

```
#pragma once

#ifdef SVCLDLL_EXPORTS
#define SV_API extern "C" __declspec(dllexport)
#else
#define SV_API extern "C" __declspec(dllimport)
#endif

// соедин-ся с сервером
// objName - имя модуля клиента записи
// ipAddrServ - адрес сервера
SV_API bool svConnect(const char* objName, const char* ipAddrServ, int portServ);

// разъед с сервером
SV_API void svDisconnect();

// добавить значение сигнала для записи
SV_API bool svAddBoolValue(const char* name, bool value, bool onlyPosFront = false);

// добавить значение сигнала для записи
SV_API bool svAddIntValue(const char* name, int value);

// добавить значение сигнала для записи
SV_API bool svAddFloatValue(const char* name, float value);

// изменение частоты записи сигналов
SV_API bool svSetParam(int cycleRecMs, int packetSz);
```

### Пример

```
#include "SVClient.h"

#include <windows.h>

#pragma comment(lib, "SVClient.lib")

int _tmain(int argc, _TCHAR* argv[])
{
    if (svConnect("testMod", "127.0.0.1", 1001)){
        std::cout << "connect ok";
    }
    else { std::cout << "connect err"; std::cin.get(); return -1;}

    int cp = 0;
    while (true){
        svAddIntValue("cnt", cp);
        svAddBoolValue("frnt", cp % 2 == 0);

        ++cp; if (cp > 10) cp = 0;

        Sleep(300);
    }
    return 0;
}
```

## Приложение 3. Клиент для NET (C#).

Является оберткой клиентской библиотеки C++.

```
namespace SVisual
{
    public class Client
    {
        [DllImport("SVClient.dll", CharSet = CharSet.Ansi, CallingConvention =
CallingConvention.Cdecl)]
        static extern bool svisual_connect([MarshalAs(UnmanagedType.LPStr)]string objName,
[MarshalAs(UnmanagedType.LPStr)]string ipAddrServ, int portServ);
        [DllImport("SVClient.dll", CharSet = CharSet.Ansi, CallingConvention =
CallingConvention.Cdecl)]
        static extern bool svisual_disconnect();
        [DllImport("SVClient.dll", CharSet = CharSet.Ansi, CallingConvention =
CallingConvention.Cdecl)]
        static extern bool svisual_addBoolValue([MarshalAs(UnmanagedType.LPStr)]string name,

.....
        public bool connect(string objName, string ipAddrServ, int portServ)
        {
            return sv_connect(objName, ipAddrServ, portServ);
        }
        public void disconnect()
        {
            sv_disconnect();
        }
        public bool addBoolValue(string name, bool value, bool onlyPosFront = false)
        {
            return sv_addBoolValue(name, value, onlyPosFront);
        }
        public bool addIntValue(string name, int value)
        {
            return sv_addIntValue(name, value);
        }
        public bool addFloatValue(string name, float value)
        {
            return sv_addFloatValue(name, value);
        }
    }
}
```

## Пример использования

```
static void Main(string[] args)
{
    Client clt = new Client();

    clt.connect("sss", "127.0.0.1", 1111);

    // clt.addBoolValue("sdd", true);
    clt.addIntValue("cnt", n);
}
```

## Приложение 4. Клиент для Python.

Является оберткой клиентской библиотеки C++.

```

from ctypes import *
import sys

lb = 0

def connect(moduleName, ipAddr, port):
    global lb
    lb = CDLL(sys.path[0] + "\SVClient.dll")

    return lb.sv_connect(moduleName, ipAddr, port)

def disconnect(moduleName, ipAddr, port):
    global lb
    lb = CDLL(sys.path[0] + "\SVClient.dll")

    return lb.sv_disconnect()

def addIntValue(valueName, value):
    global lb
    if (lb != 0):
        return lb.sv_addIntValue(valueName, value)

    return False

def addBoolValue(valueName, value, onlyFront = False):
    global lb
    if (lb != 0):
        return lb.sv_addBoolValue(valueName, value, onlyFront)

    return False

def addFloatValue(valueName, value):
    global lb
    if (lb != 0):
        return lb.sv_addFloatValue(valueName, value)

    return False

```

## Пример использования

```

from SVClient import SVClient

ok = SVClient.connect(b"sdsxxd", b"127.0.0.1", 1111)

while(ok):
    SVClient.addIntValue(b"sds", 22)

```



## Приложение 5. Правила написания клиента для любых МК и устройств

Для каждой записываемой переменной вы должны создать массив (пусть) на 10 значений. И обновлять переменную для текущего внутреннего цикла – прерывание (пусть) 100 мс (см. Приложение 7).

```
bool sv_client::addValue(const char* name, valueType type, value val, bool onlyPosFront){
    if (!isConnect_) return false;

    valueRec* vr = values_.find(name);
    if (!vr){
        int sz = values_.size();
        if ((sz + 1) > SV_VALS_MAX_CNT) return false;

        int len = strlen(name);
        if ((len == 0) || (len >= SV_NAMESZ)) return false;

        if (strstr(name, "=end=") || strstr(name, "=begin=")) return false;

        vr = new valueRec();
        strcpy(vr->name, name);
        vr->type = type;
        memset(vr->vals, 0, sizeof(value) * SV_PACKETSZ);

        values_.insert(vr->name, vr);
    }

    vr->vals[curCycCnt_] = val;
    vr->isActive = true;
    vr->vals[curCycCnt_] = val; // if happen interrupt
    vr->isOnlyFront = onlyPosFront;

    return true;
}
```

Каждые 100 мс (см. Приложение 7) по прерываниям вы должны проверять была ли обновлена записываемая переменная в текущем цикле и обновлять ее предыдущим значением, если не была обновлена:

```
// check active value in current cycle
int prevCyc = curCycCnt_ - 1; if (prevCyc < 0) prevCyc = SV_PACKETSZ - 1;
int sz = values_.size(); valueRec* vr;
for (int i = 0; i < sz; ++i){
    vr = (valueRec*)values_.at(i);

    if (!vr->isActive){
        vr->vals[curCycCnt_] = vr->vals[prevCyc];

        if ((vr->type == valueType::tBool) && vr->isOnlyFront)
            vr->vals[curCycCnt_].tBool = false;
    }

    vr->isActive = false;
}
```

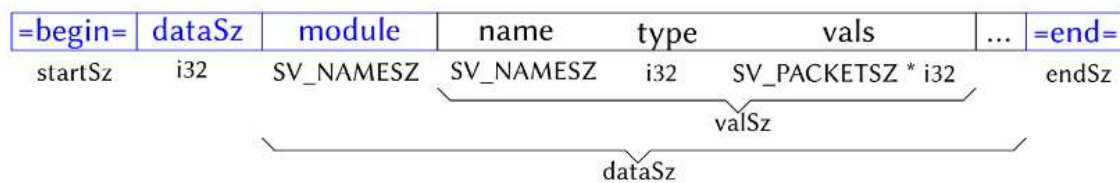
```

int next = curCycCnt_ + 1;

// send data
if (next >= SV_PACKETSZ) {
    isConnected_ = (isConnect_) ? sendData() : false;
    curCycCnt_ = 0;
}
else curCycCnt_++;

```

Каждую секунду (см. Приложение 7) по сокету или COM необходимо отправлять следующую структуру данных:



Маркер начала пакета

```

char* start = "=begin=";
send(start, strlen(start));

```

Общий размер данных

```

#define SV_NAMESZ 24
#define SV_PACKETSZ 10 (см. Приложение 7)

// кол-во записываемых переменных в данный момент
int sz = values_.size();

struct dataRef{
    char name[SV_NAMESZ];
    long int type; // tBool = 0, tInt = 1, tFloat = 2
    long int vals[SV_PACKETSZ]; // values data of 1 sec
};

int v1Sz = sizeof(dataRef); long int allSz = SV_NAMESZ + v1Sz * sz;
send((char*)&allSz, 4);

```

Название модуля

```
send(module, SV_NAMESZ);
```

Данные записанные за прошедшую секунду

```

dataRef * data;
for (int i = 0; i < sz; ++i){
    data = (dataRef *)values_.at(i);
}

```

```
    send(data, v1Sz);  
}
```

Маркер окончания пакета

```
char* end = "=end=";  
send(end, strlen(end));
```

Подробнее смотрите скетч для МК Arduino.

## Приложение 6. API для получения текущих значений

На адрес SVMonitor по сокету вы можете послать следующие команды в формате JSON:

- Вернуть все сигналы

```
{ "Command": "getAllSignals" }
```

Ответ

```
{
  "Command": "allSignals",
  "Signals": {
    "Name": "...",
    "Module": "...",
    "Group": "...",
    "Comment": "...",
    "Type": "...",          // int, bool, float
    "State": "...",        // состояние: isActive, noActive, isDelete
  },
  "SignCnt": "..."       // кол-во сигналов
}
```

- Вернуть все триггеры

```
{ "Command": "getAllTriggers" }
```

Ответ

```
{
  "Command": "allTriggers",
  "Triggers": {
    "Name": "...",
    "Signal": "...",
    "Module": "...",
    "CondType": "...",      // тип условия: connectModule, disconnectModule,
                           // less, equals, more, posFront, negFront
    "CondValue": "...",    // порог
    "CondToutSec": "...",  // таймаут сек
    "TrgType": "...",      // тип: isSignal, isModule
    "State": "...",        // состояние: isActive, noActive
  },
  "TrgCnt": "..."        // кол-во триггеров
}
```

- Вернуть текущее значение сигнала

```
{"Command":"getSignalData","Signal":"...", "Module":"..."}
```

**Signal** – наименование сигнала

**Module** – наименование модуля

Ответ

```
{  
  "Command":"signalData",  
  "Signal":"...",  
  "Module":"...",  
  "ValueTime":"...",  
  "Value":"..."  
}
```

Если сигнал не найден, ответ будет таким:

```
{  
  "Command":"signalData",  
  "Signal":"",  
  "Module":"",  
  "ValueTime":"","  
  "Value":""  
}
```

## Приложение 7. Задание частоты получения данных

Частота записи и получения данных задается параметрами в SVMonitor.ini, файл находится в папке установки ПО.

cycleRecMs = 100 мс – цикл записи сигнала – максимальная частота записи  
packetSz = 10 – размер пакета для передачи данных

Параметрам по умолчанию соответствует частота записи 10 Гц, отправка пакета каждую секунду, размер пакета 10 значений.

Например, вы хотите отправлять данные каждую минуту, с частотой 1 Гц (значение в секунду), значит вы устанавливаете :

cycleRecMs = 1000 мс  
packetSz = 60

**cycleRecMs не должен быть задан меньше 10 мс.**

**Если вы изменили эти параметры в SVMonitor.ini, то изменения надо провести в:**

- ини просмотрщика записей SViewer.ini;
- для клиентской библиотеке sv\_client.dll перед использованием вызвать метод:

```
bool sv_setParam(int cycleRecMs, int packetSz)
```

- для клиента Arduino поправить в файле sv\_structurs.h:

```
#define SV_CYCLEREC_MS 100  
#define SV_PACKETSZ 10
```

## Приложение 8. Скрипт Python для отправки email

Может быть использован для обработки триггера события.

```
import sys
import os
import smtplib
from email.mime.text import MIMEText
import configparser

##### init
cng = configparser.ConfigParser()
cng.read(sys.path[0] + "/sendMess.ini")
recvAddr = cng["Param"]["recvAddr"]
recvPassw = cng["Param"]["recvPassw"]
sendAddr = cng["Param"]["sendAddr"].split()

##### sendMess
def sendMess(modName, subject):
    msg = MIMEText("")
    msg['Subject'] = modName + ' ' + subject

    s = smtplib.SMTP("smtp.gmail.com", 587)
    s.starttls()
    s.login(recvAddr, recvPassw)
    s.sendmail(recvAddr, sendAddr, msg.as_string())
    s.quit()

sendMess('Stk', 'P0. High level')
```

### Файл ини

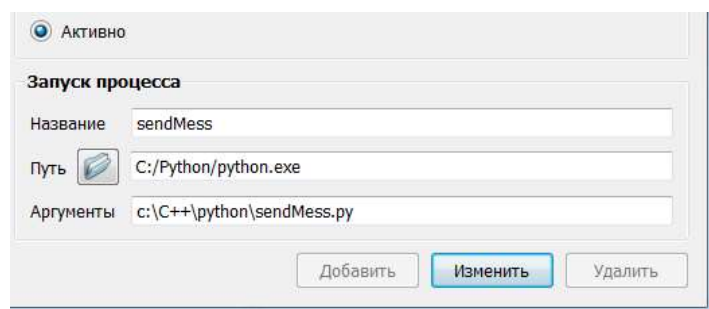
[Param]

recvAddr = [recv@mail.ru](mailto:recv@mail.ru)

recvPassw = 12345

sendAddr = [user1@mail.ru](mailto:user1@mail.ru) [user2@mail.ru](mailto:user2@mail.ru)

### Пример использования



## ЛИЦЕНЗИЯ

The MIT  
License

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:  
The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.  
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.