Contents

1. Function

The SVisual software is designed to monitor the operation of the MK devices, debug the program, alert the user to the events that have occurred.

SVMonitor Features:

- connection to the MK via COM port (usb for arduino), over Ethernet or Wi-Fi protocol TCP;

- Interrogation of values of signals in real time with frequency before 100 Hz (default 10 Hz), the number of devices and signals is selected by the user;

- the permissible number of signals for recording 2048;

- output of the values of the selected signals to the monitor screen in real time;

- record the archive of signals to the hard disk of the PC;

- viewing the archive using additional SVViewer software;

- the ability to set alerts for an event (triggers) that occurred, start a user process when the trigger is triggered;

- adding a signal for viewing / recording only by the client, no additional movements are required.

2. System Requirements

OS: Windows-64 (7, 8, 10), Linux

CPU: >= Pentium 4

RAM: >= 512 Mb

3. Application Settings

From the main menu, select "Settings"

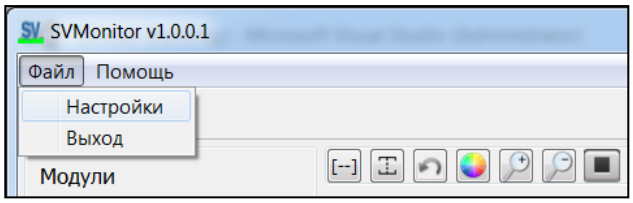


Fig.3.1. Selecting the menu item "Settings"

The Settings window opens.

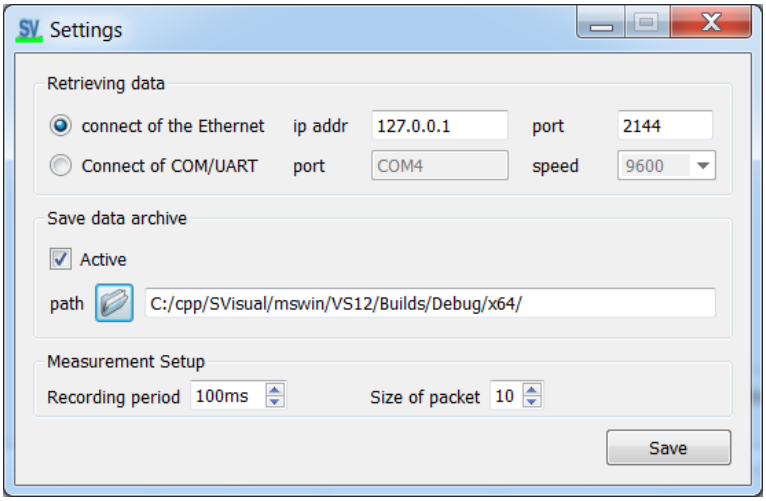


Fig.3.2 The Settings window.

When connecting to COM, select "COM connection", set the port and speed. The remaining transmission parameters are hardwired in the software:
ByteSize = 8, NoParity, NoFlowControl, StopBits::OneStop.

When connecting over Ethernet, select "Ethernet connection", set ip address and port.

To take effect of changes to the "Data Acquisition" parameters software reboot.

To view the archive of recorded signals in the "Save data" panel, activate the data saving, set the path to the folder.

The recording period is the cycle with which the values go into the buffer for sending, the packet size determines the period for sending the entire data buffer (see clause 5).

4. Description of the user interface

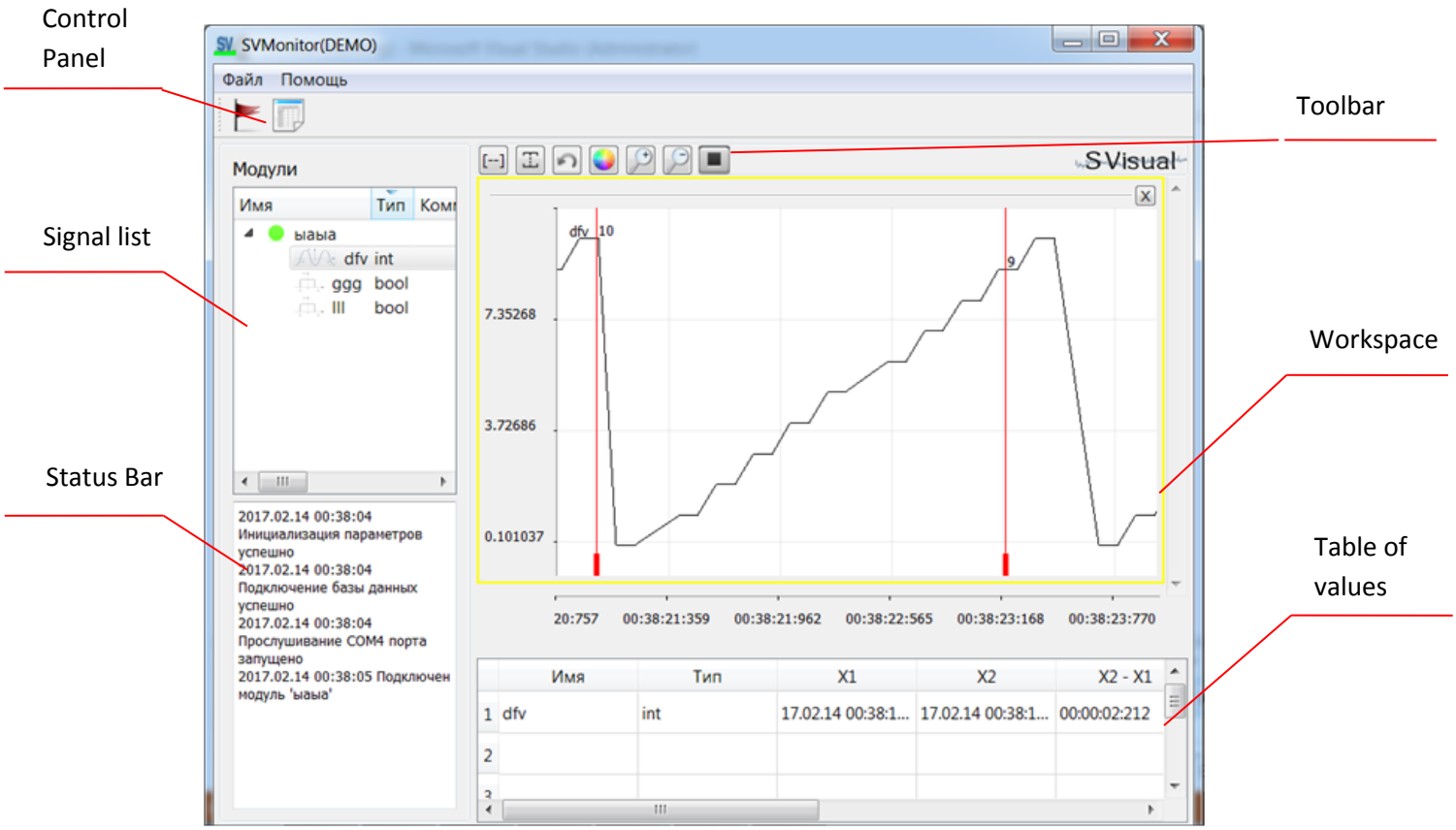


Fig.4.1. Main window of the program.

- The current signals of the connected devices are displayed in the signal list.
- The workspace is designed to display the values of the selected signals in real-time data acquisition and in a stable state.
- The value table shows the current values of the signal markers.
- The toolbar allows you to manipulate the graph of signals.
- The status bar displays system messages about the events that occurred: module connection, adding a new signal, etc.

4.1. List of signal modules

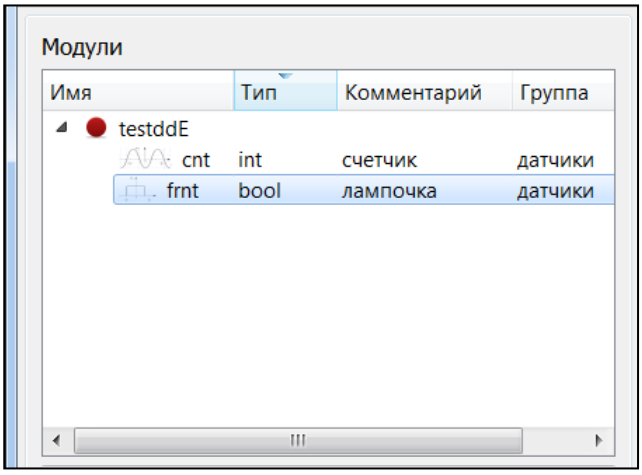


Fig.4.2. List of signal modules.

The list of signal modules contains the connected devices at a given time. Used to select the signal to display in the work area.

The indication to the left of the module shows the status of the data transmission.

Double clicking the LMB on the signal will display the signal in the work area. Also, the signal can be transferred to the work area without releasing the paintwork.

If the signal or the module is not currently in use, you can delete it: press RMB and select "Delete".

To delete the active module, the module must first be disconnected, then it will be possible to delete.

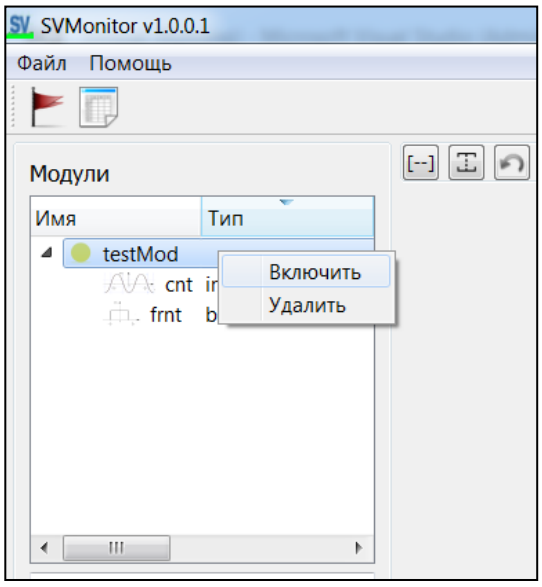


Fig.4.3. Removing Module.

The additional fields of the "Comment" and "Group" list are populated by the user and are used when viewing the recorded data archive.

4.2. Status Bar

The application status bar shows diagnostic messages:

- change the list of signal modules;
- triggering of signal triggers (see Section 6);
- physical connection-disconnection of modules.

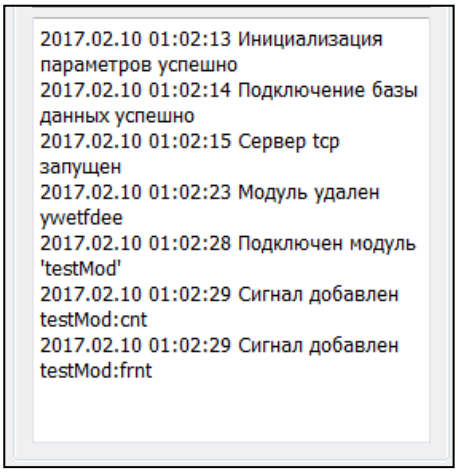


Fig.4.4. Status Bar.

4.3. Toolbar

The toolbar is used to work with the workspace windows.

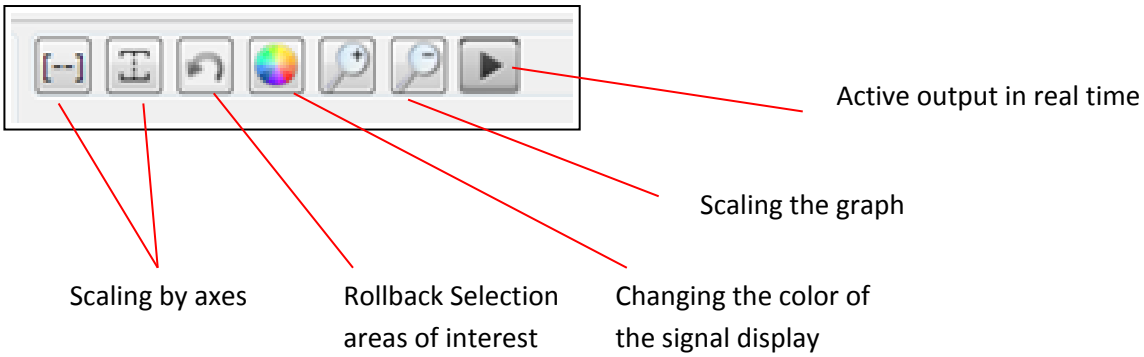


Fig.4.5. Toolbar.

Commands from the toolbar are applied to the signals of the selected workspace window. The selected window is circled with a yellow border.

4.4. Workspace

It is designed to display selected signals in real time.

Legend, top left in the workspace, contains a list of signals of the type "Name Comment", the color of the legend of the signal is the same as the color of the signal graph.

A detailed view of the signal is possible when the active output is disconnected (the current values are displayed, - the "start-stop" button on the toolbar).

Using the mouse, you can:
- scale the signal by rotating the mouse wheel,
- scales separately coordinate axes,
- allocate an area of interest,
- move in the work area holding the RBM.

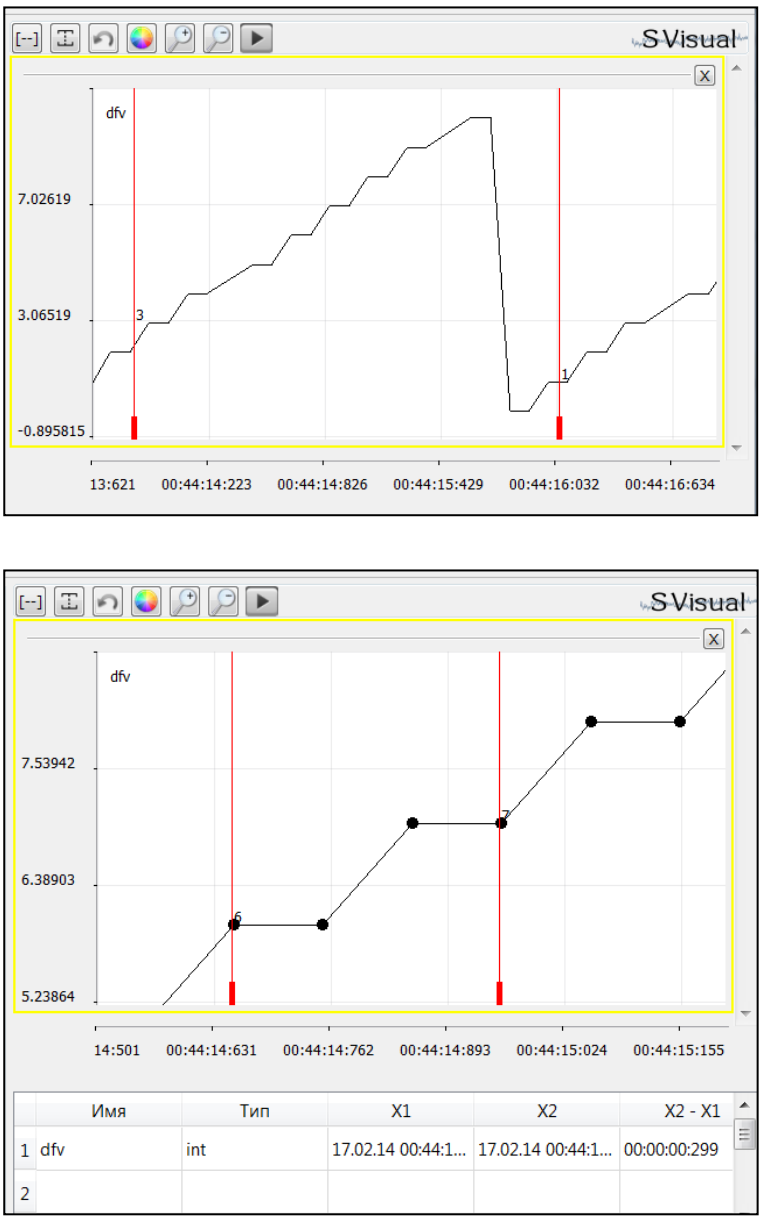


Fig.4.6. Selecting the area of interest of the signal.

Markers - red lines in the working area, designed to display the current value of the signal.

The signal value is updated when the marker is moved with the mouse and is displayed in the graph and in the table of values at the bottom of the working area.

Moving the mouse on the graph by pressing the LMB you can see the current signal values on the graph.

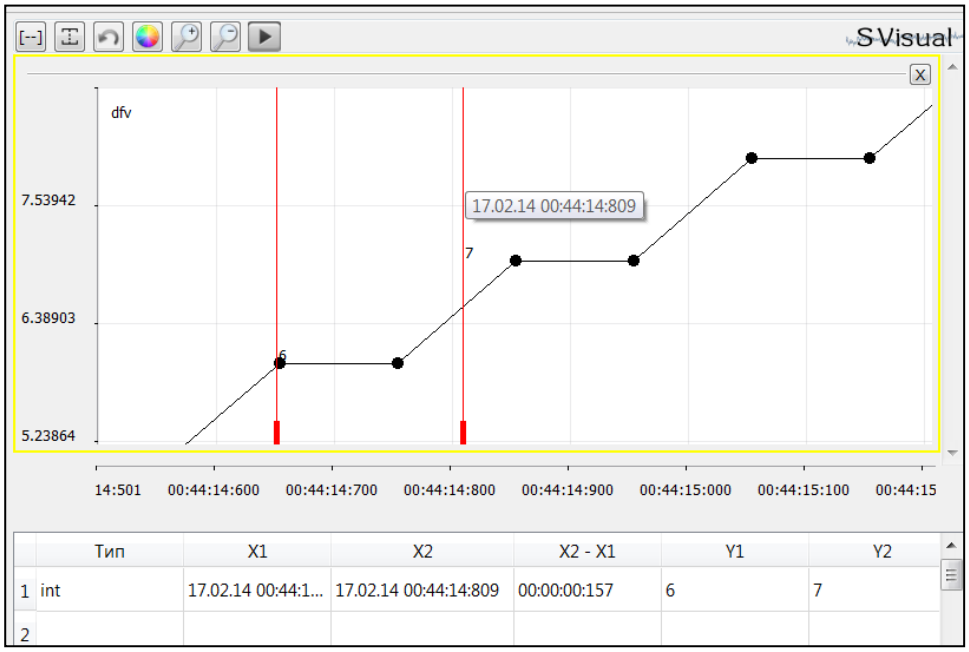


Fig.4.7. Marker in the workspace.

An unlimited number of signals can be added to one window of the working area, with all signals scaled in one axis.

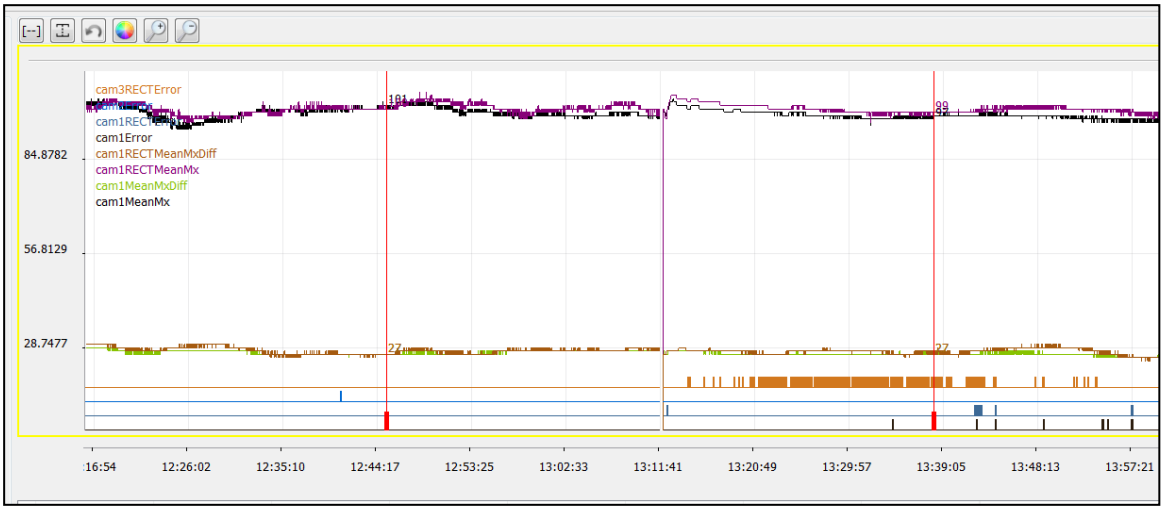
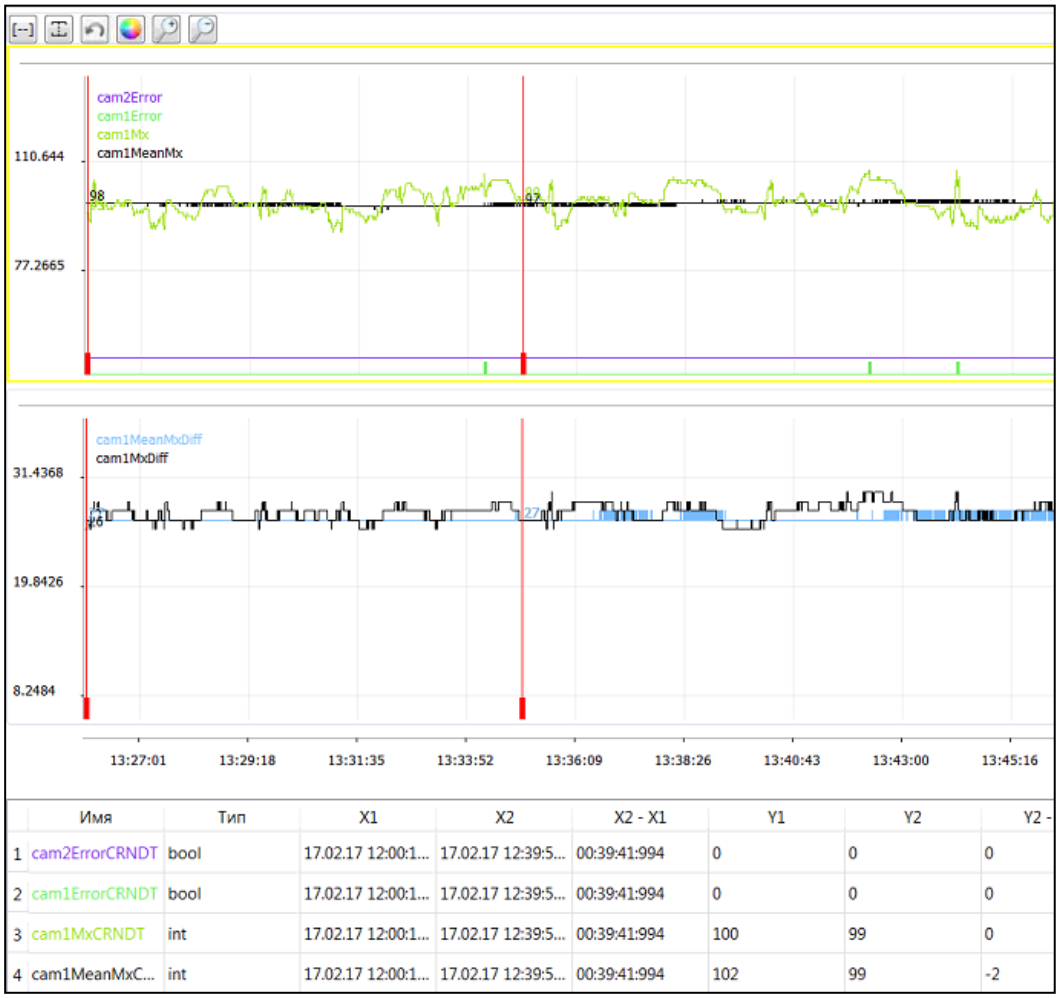


Fig. 4.8. Multiple signals in the workspace window.

To remove a signal from the working window, click on the RBM on the signal legend.

If your signals have different dimensions and you want to match the signals, you can add the signal to another window in the workspace.

To do this, it is enough to select the signal in the table, and not letting the LMC move the signal below the current window of the working area. Another window will be created for this signal.



	Имя	Тип	X1	X2	X2 - X1	Y1	Y2	Y2 -
1	cam2ErrorCRNDT	bool	17.02.17 12:00:1...	17.02.17 12:39:5...	00:39:41:994	0	0	0
2	cam1ErrorCRNDT	bool	17.02.17 12:00:1...	17.02.17 12:39:5...	00:39:41:994	0	0	0
3	cam1MxCRNDT	int	17.02.17 12:00:1...	17.02.17 12:39:5...	00:39:41:994	100	99	0
4	cam1MeanMxC...	int	17.02.17 12:00:1...	17.02.17 12:39:5...	00:39:41:994	102	99	-2

Fig.4.9. Additional windows in the workspace.

Also, the signal can be transferred to another axis in the same window, it is enough to select the signal in the window, and not letting the LMC move the signal to the same window. In this case, the signal will always be maximally scaled along the ordinate axis and not associated with other window signals.

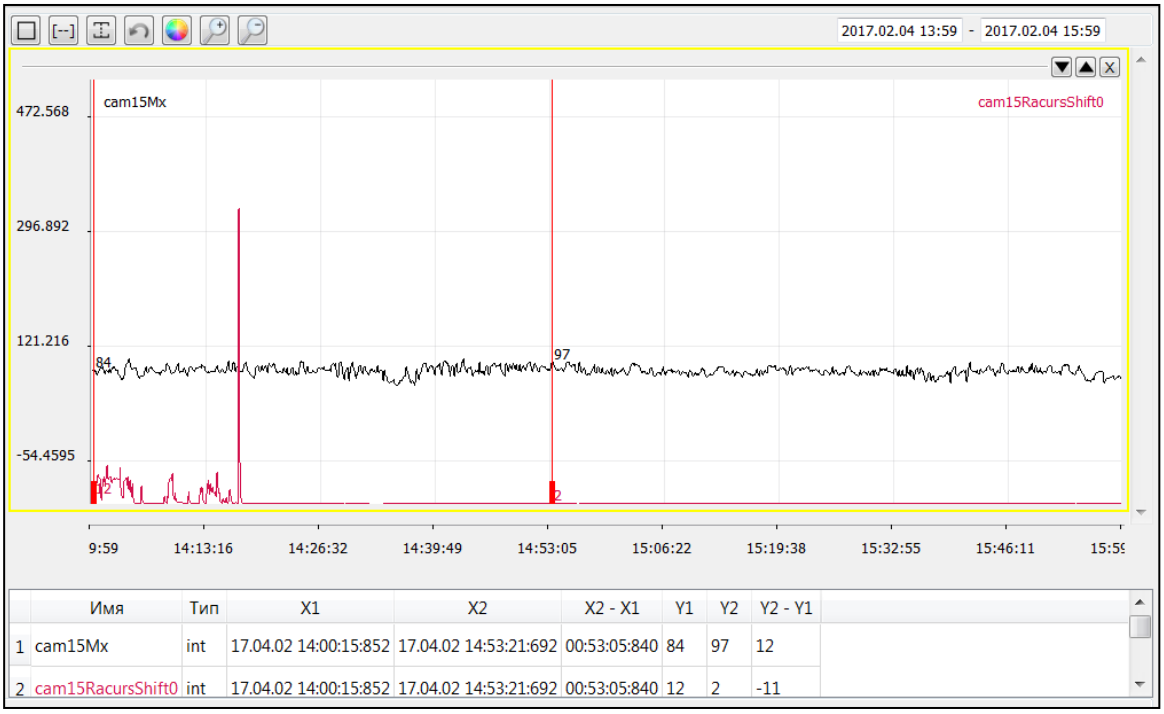
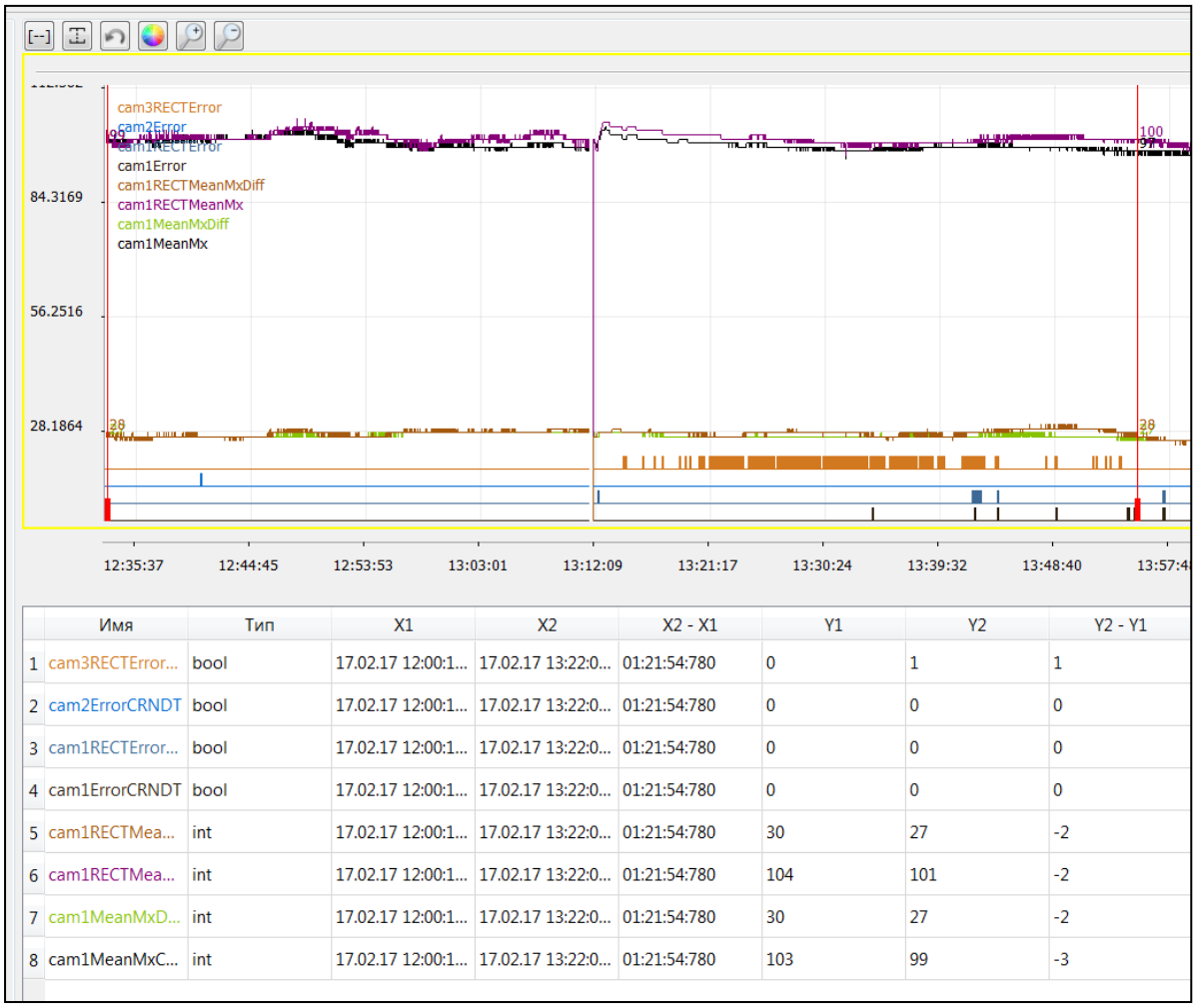


Fig.4.10. Alternative Window Axis.

4.6. Table of values

The value table serves to display the current signal value at the marker location.



	Имя	Тип	X1	X2	X2 - X1	Y1	Y2	Y2 - Y1
1	cam3RECTError...	bool	17.02.17 12:00:1...	17.02.17 13:22:0...	01:21:54:780	0	1	1
2	cam2ErrorCRNDT	bool	17.02.17 12:00:1...	17.02.17 13:22:0...	01:21:54:780	0	0	0
3	cam1RECTError...	bool	17.02.17 12:00:1...	17.02.17 13:22:0...	01:21:54:780	0	0	0
4	cam1ErrorCRNDT	bool	17.02.17 12:00:1...	17.02.17 13:22:0...	01:21:54:780	0	0	0
5	cam1RECTMea...	int	17.02.17 12:00:1...	17.02.17 13:22:0...	01:21:54:780	30	27	-2
6	cam1RECTMea...	int	17.02.17 12:00:1...	17.02.17 13:22:0...	01:21:54:780	104	101	-2
7	cam1MeanMxD...	int	17.02.17 12:00:1...	17.02.17 13:22:0...	01:21:54:780	30	27	-2
8	cam1MeanMxC...	int	17.02.17 12:00:1...	17.02.17 13:22:0...	01:21:54:780	103	99	-3

Fig.4.11. Table of values.

5. Create custom notifications

A trigger is a condition that can be met, for example, triggering the sensor when the water level is exceeded.

A notification is a reaction to an event triggered by a trigger.

As notifications, a call to a third-party process of the computer running SVMonitor is used. This can be any application OS, - console program, python script, any other program.

On the control panel, click the checkbox.

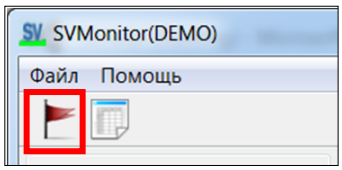


Fig.5.1. Open the Custom Notification Settings window.

The notification settings window will open.

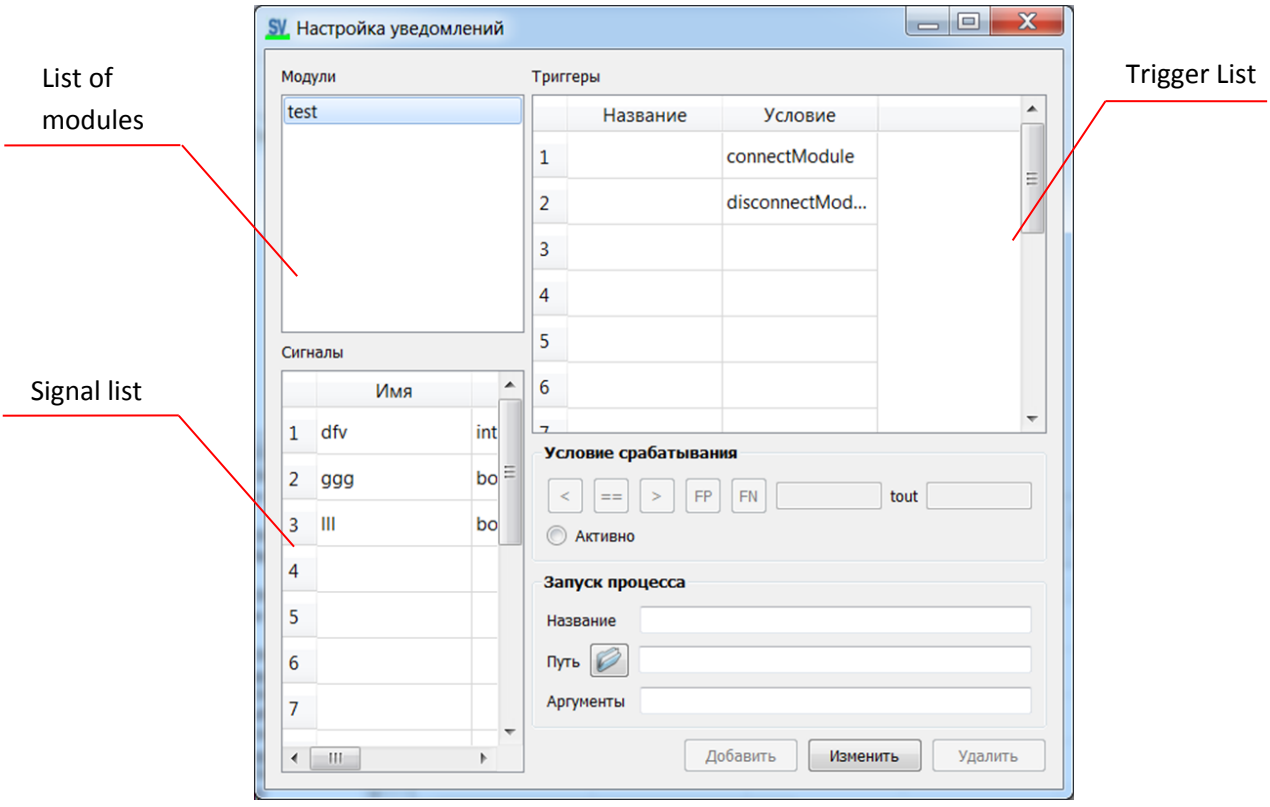


Fig.5.2. Notification settings window.

In the "Trigger Condition" panel, you select the condition under which the trigger will fire and, accordingly, call the external process.

In the "Run process" panel, you specify the name of the trigger, the path to the process that you want to start when the condition is triggered, the process arguments are specified with a space.

5.1. Triggers for the module

When selecting a module from the list of modules, two standard triggers appear in the trigger table: "module connection" and "module shutdown". These triggers are created automatically when a new module is connected.

For modules, you can not remove standard triggers, and add custom triggers.

The "connect module" trigger (connectModule in the "Condition" column) is triggered when the known module is reconnected to the network.

The "module shutdown" trigger (disconnectModule in the "Condition" column) is triggered when the module is disconnected from the network.

If you want to handle module connection / outage events, you must:

- select a condition in the trigger table;

- in the "Trigger condition" section, click the "Active" button;

- in the section "Running the process" to think up and write the name of the trigger, choose the path to the desired process;

- click the "Edit" button.

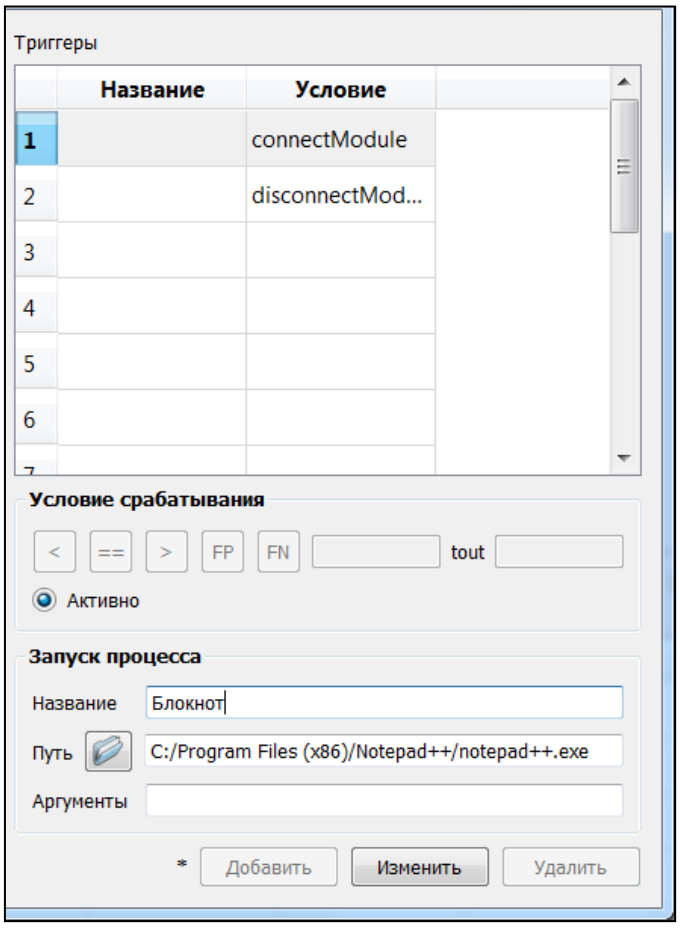


Fig.5.3. Creating a Standard Trigger.

5.2. Triggers for signal

For any recorded signal, you can create a trigger by selecting the trigger condition:

- select an alarm from the signal list;

- select trigger condition;

For signals of type int and real, the conditions are formulated as follows:

"The value of the signal is greater (>) / less (<) / equal (==) to the value of the set threshold".

For signals of type bool, the conditions are formulated as follows:

"Positive (FP) / negative (FN) signal edge".

(detailed description of clause 5.3)

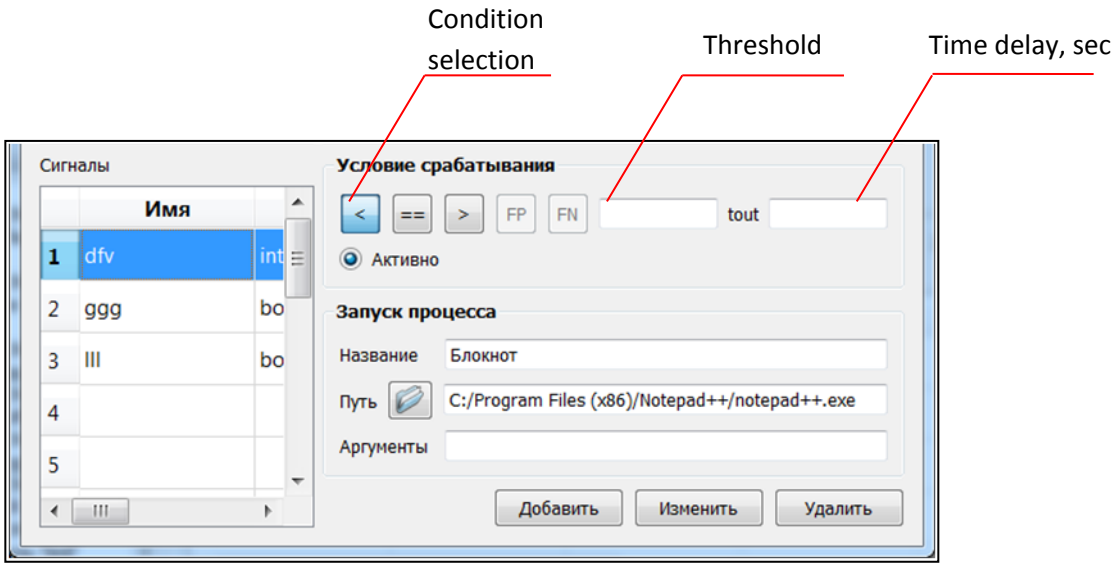


Fig.5.4. Creating a Trigger for a Signal.

- in the "Trigger Condition" section, set the trigger threshold and the time delay (if necessary);

- click the "Activate" button;

- in the section "Start the process" write the name of the trigger, select the path to the desired process;

- click the "Add" button.

In the table of triggers, you will see the created trigger with the given name and condition.

The created triggers can be edited, deactivated, deleted. To do this, select the trigger in the trigger table and click the button below "Edit" or "Delete.

5.3. How triggers work

The triggering condition is the output of the value of the target signal beyond the specified threshold, taking into account the set delay (specified in seconds).

Example 1. Signal of type integer, condition "value> threshold, delay> 0".

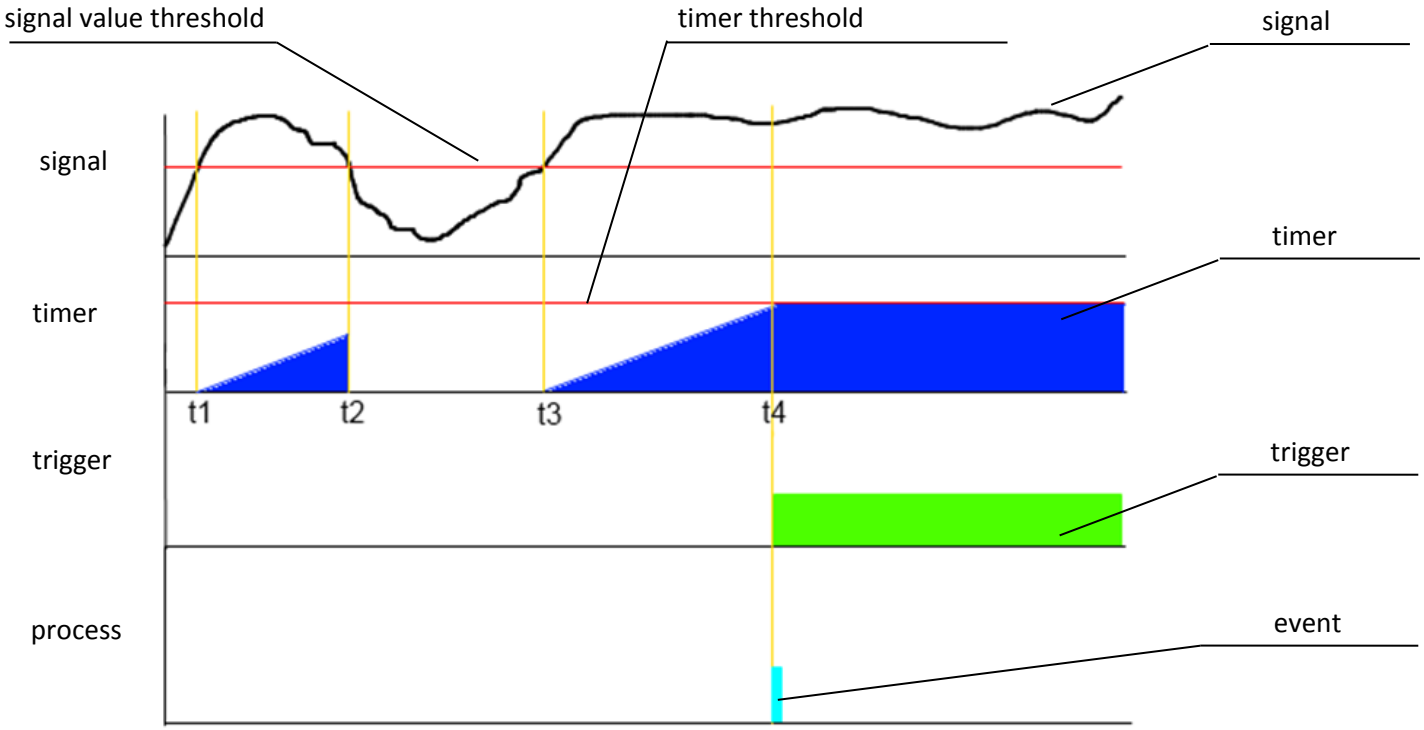


Fig.5.5. Example №1.

At time t1, the signal value exceeded the threshold, the countdown timer started.

At time t2, the value of the signal became less than the threshold, the timer did not have time to count the specified time interval and was reset to zero.

At time t3, the signal value again exceeded the threshold, the countdown timer started again.

At time t4, the timer counted the set pause, and the trigger triggered - the user process started.

Example 2. Signal of type integer, condition "value> threshold, delay> 0".

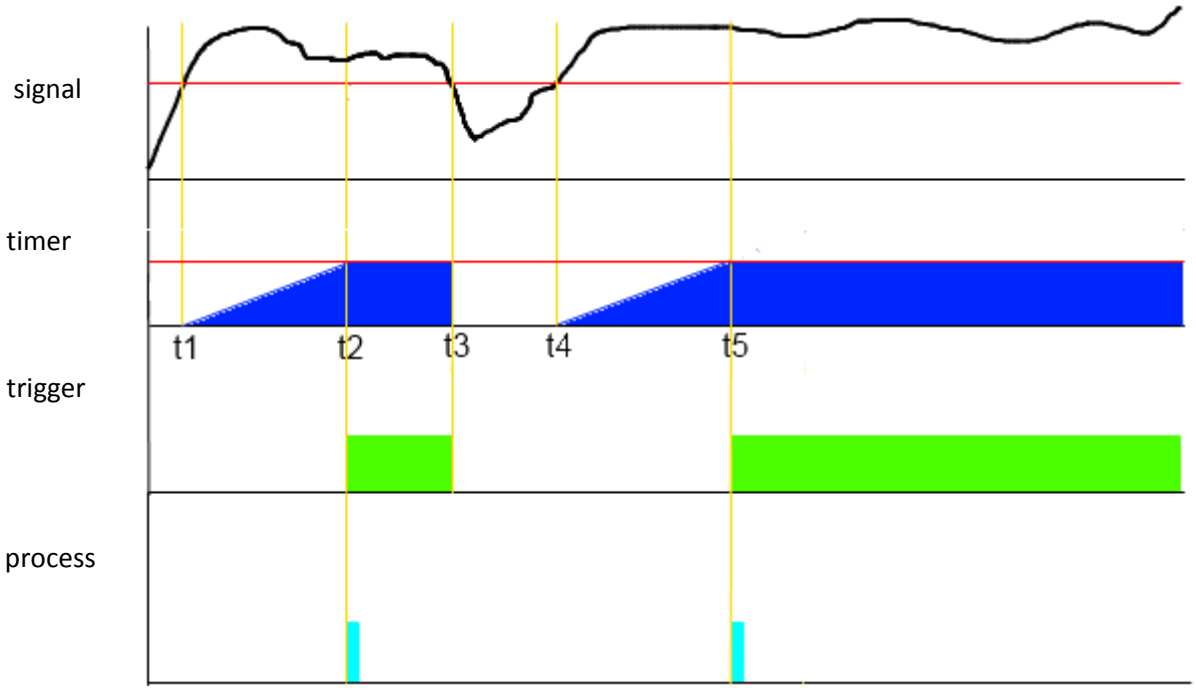


Fig.5.6. Example number 2.

At time t1, the signal value exceeded the threshold, the countdown timer started.

At time t2, the timer counted the specified pause, and the trigger triggered - the user process started.

At time t3, the signal value fell below the threshold, the timer and the trigger reset.

At time t4, the signal value again exceeded the threshold, the countdown timer started.

At time t5, the timer counted the pause again, and the trigger again triggered - the user process started.

Example 3. An integer signal, the condition "value <threshold, delay> 0".

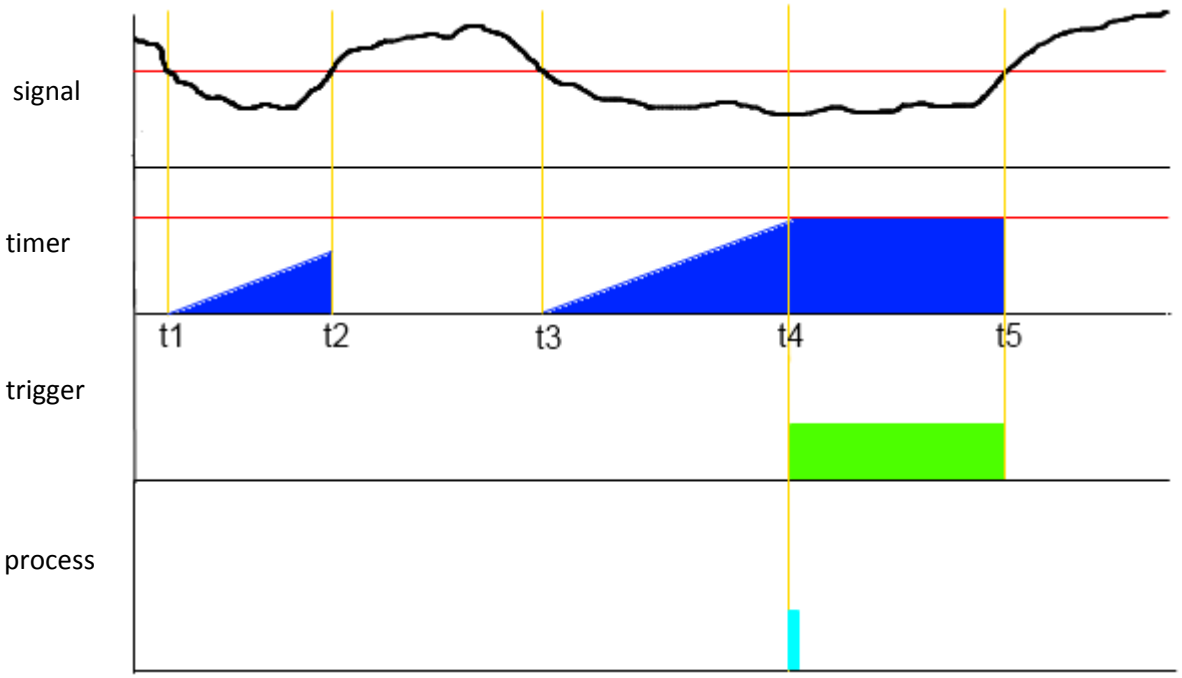


Fig.5.7. Example №3.

At time t1, the signal value became below the threshold, the timer countdown started.

At time t2, the value of the signal became higher than the threshold, the timer did not have time to count the specified time interval and was reset to zero.

At time t3, the value of the signal is again below the threshold, the countdown time has started again.

At time t4, the timer counted the specified pause, and the trigger triggered - the user process started.

At time t5, the signal value is above the threshold, the timer and the trigger are reset.

Example 4. A signal of type bool, the condition "FP value (positive edge), delay> 0".

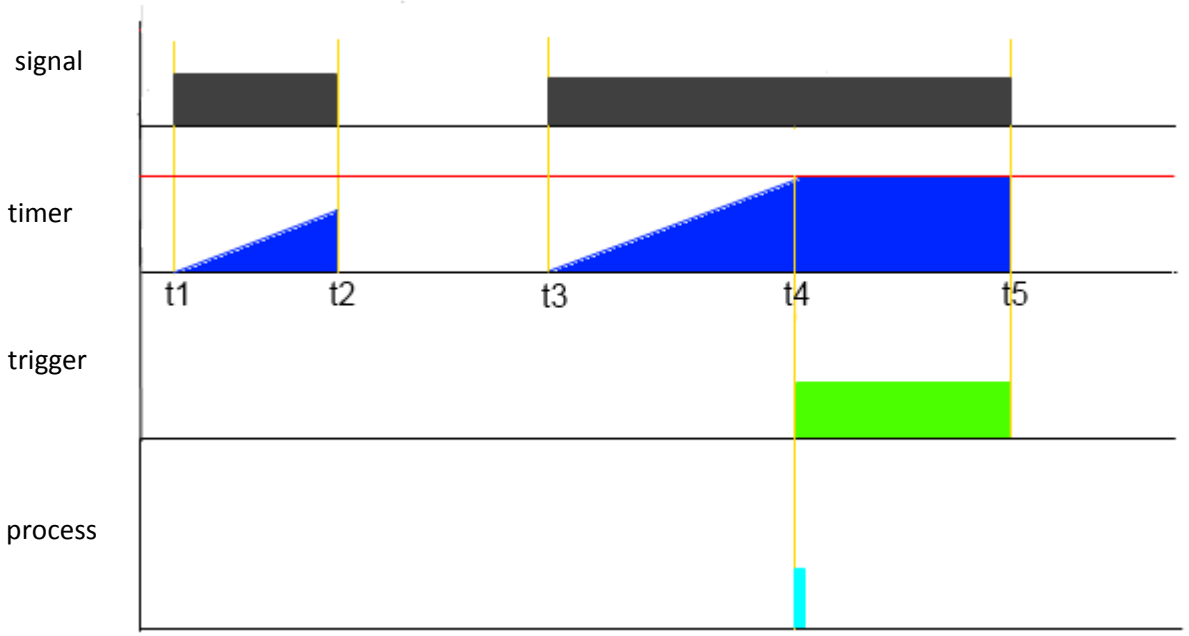


Fig.5.8. Example №4.

At the moment t1, the value of the signal became TRUE, the countdown of the timer started.

At time t2, the signal value became FALSE, the timer did not have time to count the set time interval and was reset to zero.

At time t3, the value of the signal is again TRUE, the timer time has started again.

At time t4, the timer counted the specified pause, and the trigger triggered - the user process started.

At the time t5, the signal value became FALSE, the timer and the trigger reset.

Example 5. A signal of type bool, the condition "FN value (negative edge), delay == 0".

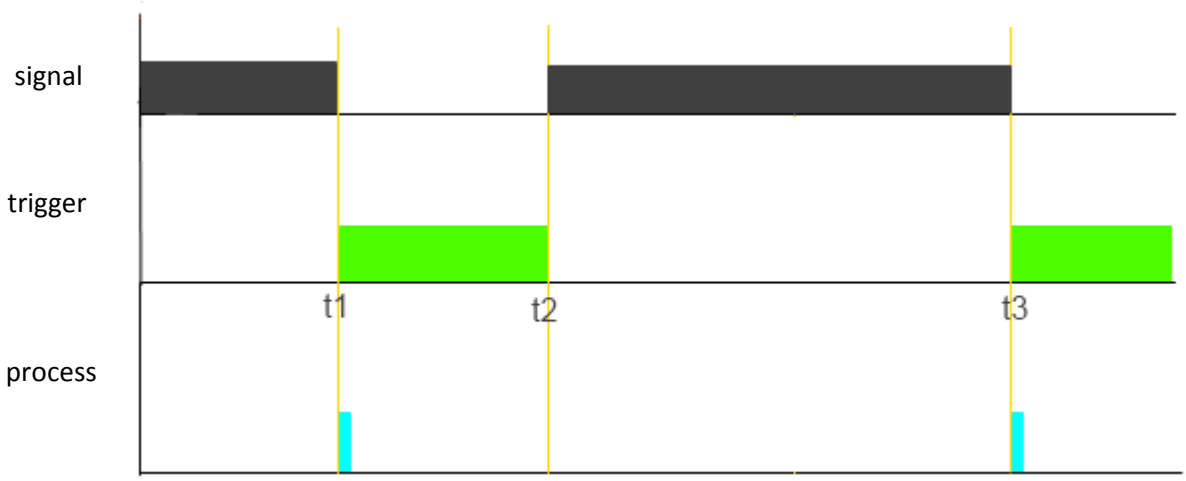


Fig.5.9. Example №5.

At time t1, the signal value became FALSE, the trigger triggered - the user process started.

At time t2, the value of the signal became TRUE, the flip-flop was reset.

At time t3, the signal value again became FALSE, the trigger triggered - the user process started.

6. Report on active events

On the control panel, click the button with the table.

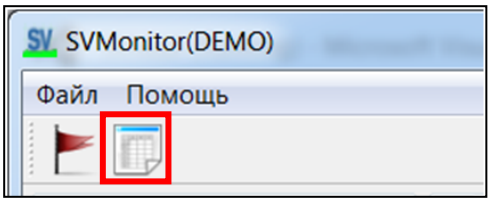


Fig.6.1. Open the report window.

The report window opens.

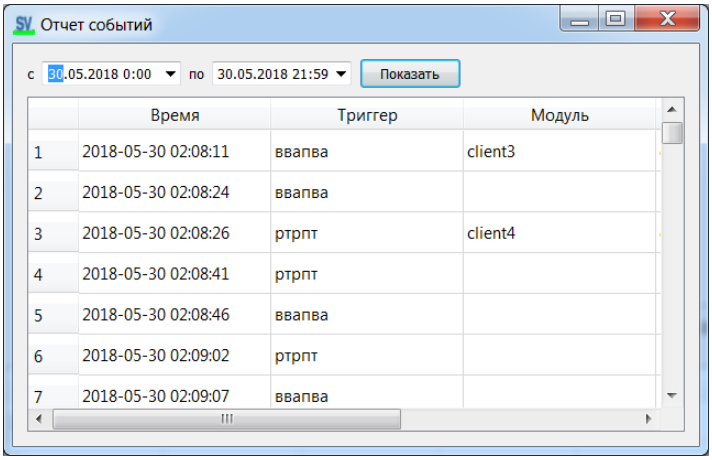


Fig.6.2. Report window.

The report contains the trigger time of the trigger, the triggering condition.

7. Viewing the data archive

To view the recorded data files, you must run the SVViewer.exe application.

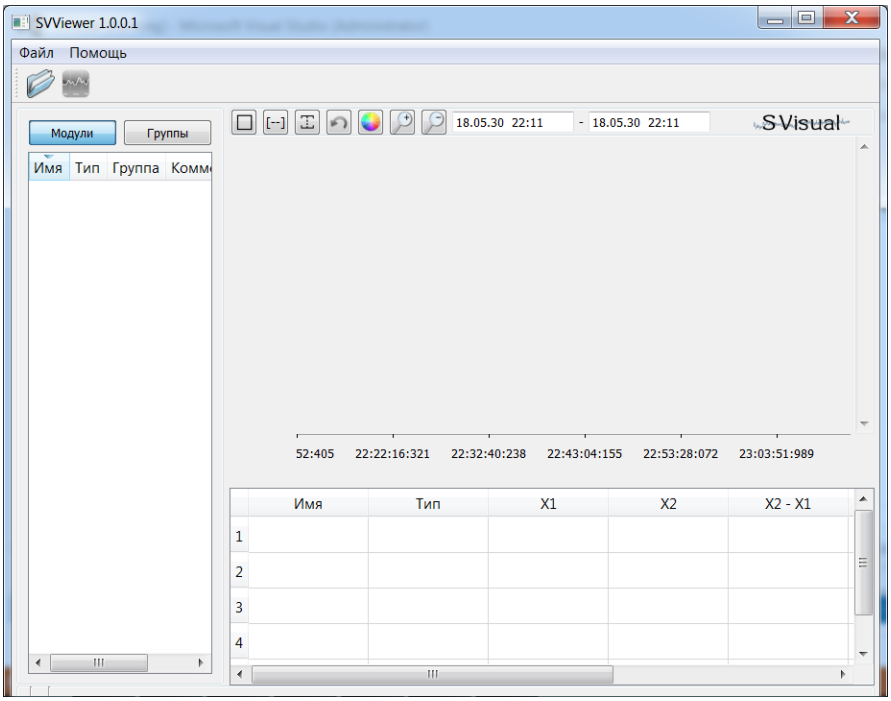


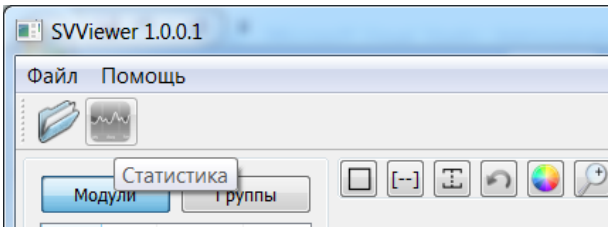
Fig.7.1. Archive viewer window.

All operations in the preview window are similar to those described above for SVMonitor.

To add data files, click the "open" button in the control panel and select the archive files.

7.1. View statistics for archived data

To display the histogram of the signal change, press the "statistics" button on the control panel.



A window for building a histogram will open. Drag the selected signal to the window.

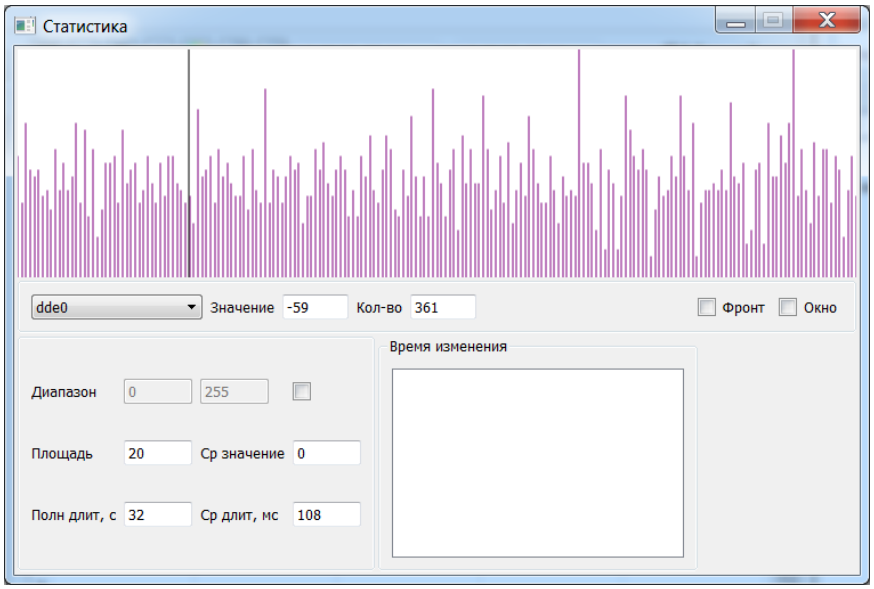


Fig. 7.2. Statistics window.

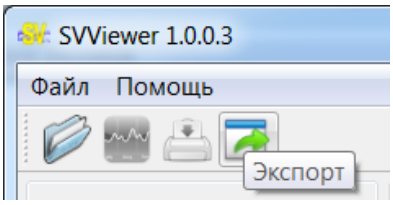
The window displays the histogram of the signal values, the histogram area to the marker on the left, the average value, and the duration of the current signal value.

The RBM on the graph selects a specific signal value.

7.2. Exporting data to a file

Data export is possible in a text file, json file and in a xlsx file.

To open the export window, click the Export button on the control panel.



A window will open for exporting data.

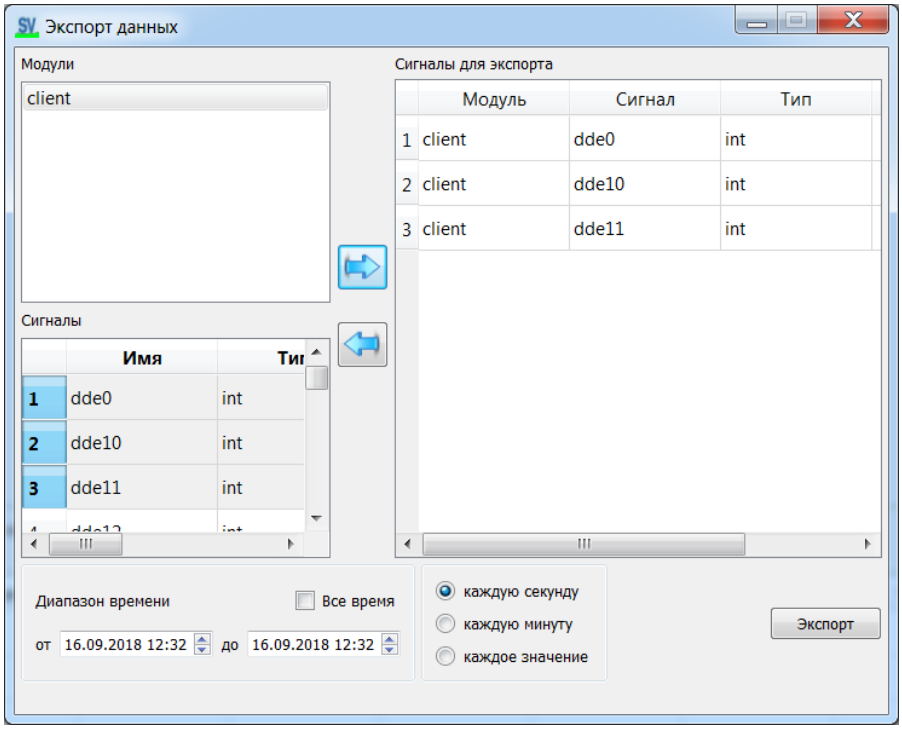


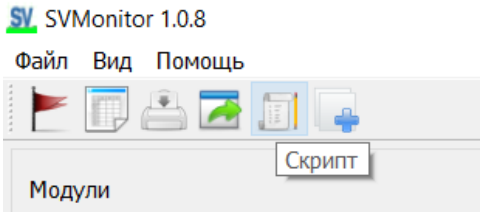
Fig.7.3. Export window.

Select the signals to export, select the time and click the "Export" button.

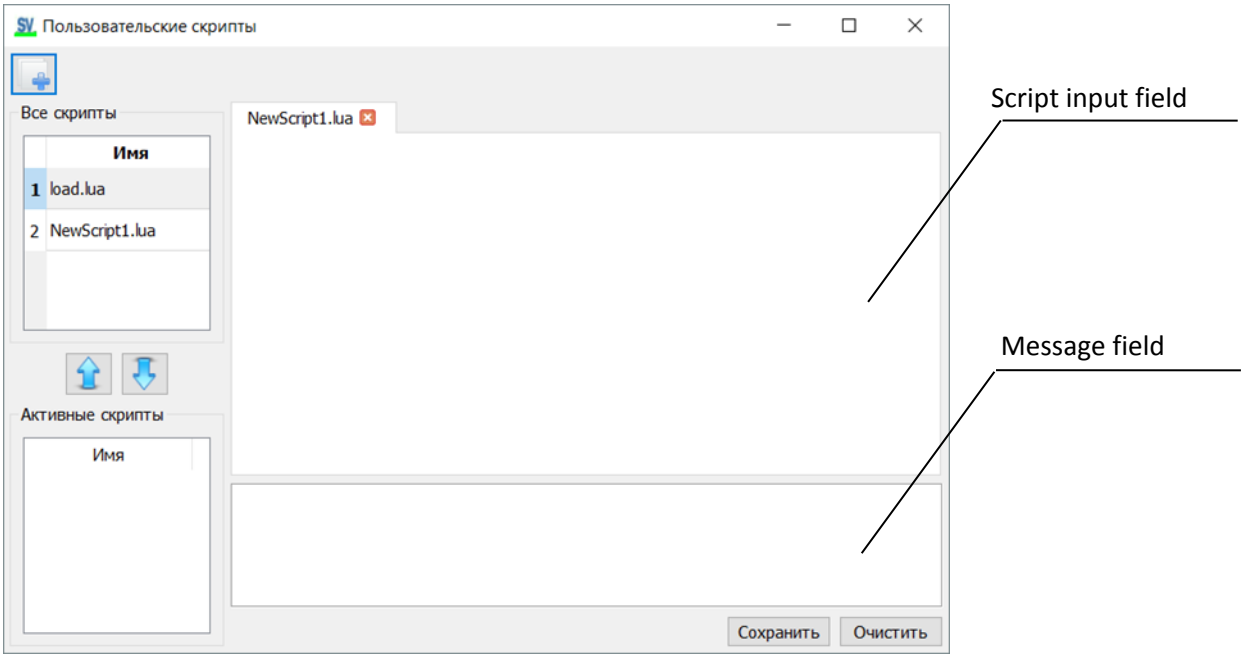
Note: for SVMonitor export only signals currently viewed can be exported.

8. User scripts - creating virtual signals

To open the script window, click the “Script” button on the control panel.



A window for creating scripts will open.



All written scripts are input to the Lua language interpreter.

To create a new script file, click on the “Create Script” button above.

In the “All scripts” list, double-click on the script name to open the script for editing.

The script is executed cyclically for each value of the signal.

To start the script, select the script in the "All scripts" list and click the "Add to active" arrow.

In addition to user messages (the printMess function), error messages in the script appear in the message output field.

The script called “load.lua” runs only once when adding it to active scripts, this allows you to create and initialize global variables in it that can be used in other scripts.

To delete a script, you need to click RMB on the selected one in the “All scripts” list.

In addition to standard Lua functions, the following SV functions are available:

- message output
```
void printMess(const std::string& mess);
```
- receiving signal time, in ms
```
uint64_t getTimeValue(const std::string& module, const std::string& signal);
```
- getting the signal type values Bool
```
bool getBoolValue(const std::string& module, const std::string& signal);
```
- getting the signal type values Int
```
int getIntValue(const std::string& module, const std::string& signal);
```
- getting the signal type values Float
```
float getFloatValue(const std::string& module, const std::string& signal);
```
- setting the signal type Bool
```
void setBoolValue(const std::string& signal, bool value,uint64_t time);
```
- setting the signal typeInt
```
void setIntValue(const std::string& signal, int value,uint64_t time);
```
- setting the signal typeFloat
```
void setFloatValue(const std::string& signal, float value,uint64_t time);
```

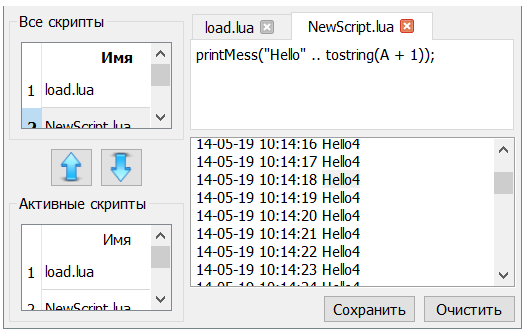
Example 1. Displaying a text message.

1. Open the load.lua script in the input field and type:
   A = 3; -- create and initialize a variable

2. Open the NewScript.lua script in the input field and type:
   printMess("Hello" .. tostring(A + 1));

3. Add both scripts to active.

4. In the output field you should see:
   Hello4



Example 2. Signal Value Conversion.

1. For example, let a sine wave be given:

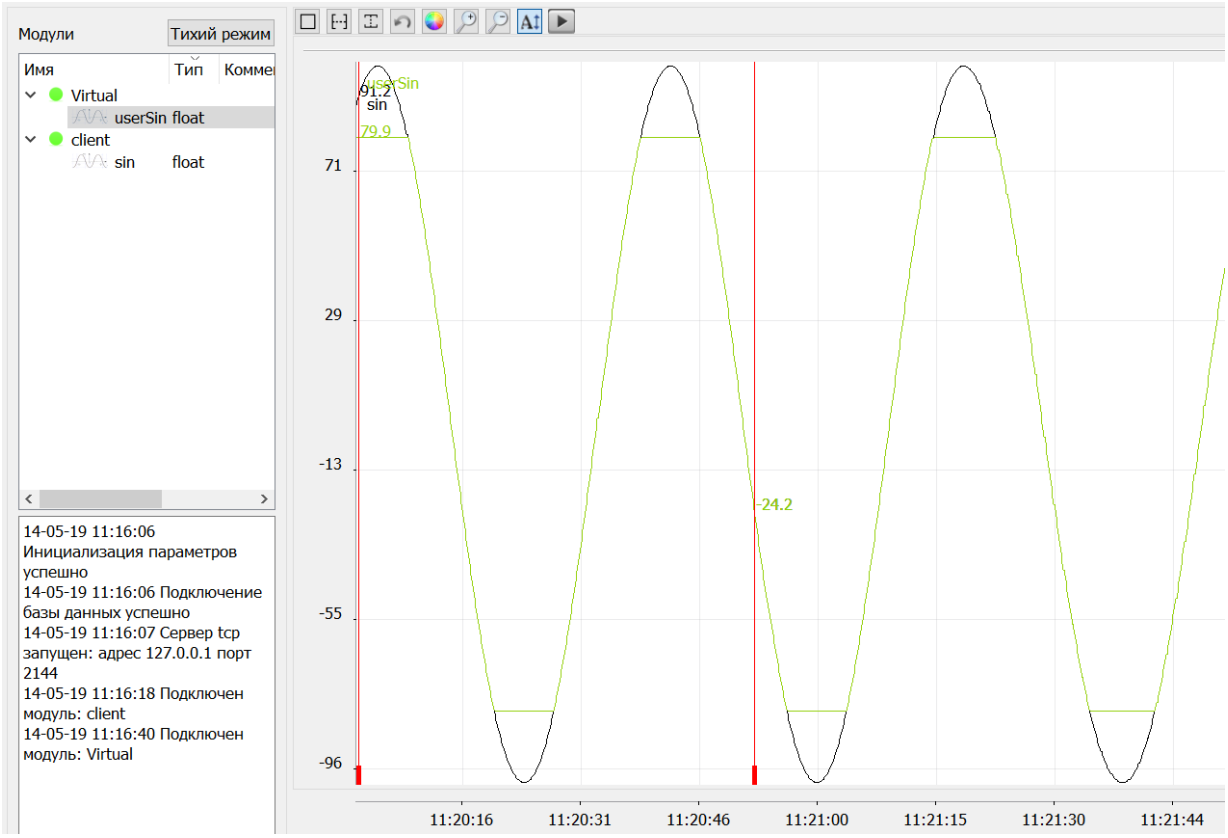
```
int cp = 0;
while(true){

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

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

     Sleep(100);
}
```

2. Open the NewScript.lua script in the input field and type:

```
-- get signal time
tm = getTimeValue("client", "sin");

-- get signal value
val = getFloatValue("client", "sin");

-- cut off the top of the sinusoid
usVal = (val > 80) and 80 or val;

-- cut off the bottom of the sinusoid
usVal = (usVal < -80) and -80 or usVal;

-- create a new virtual signal "userSin"
setFloatValue("userSin", usVal, tm);
```

3. Make the script active.

4. The “Virtual” module will be connected, on the chart you should see the following:



Example 3. Highlight signal values.

1. For example, let a sine wave be given:

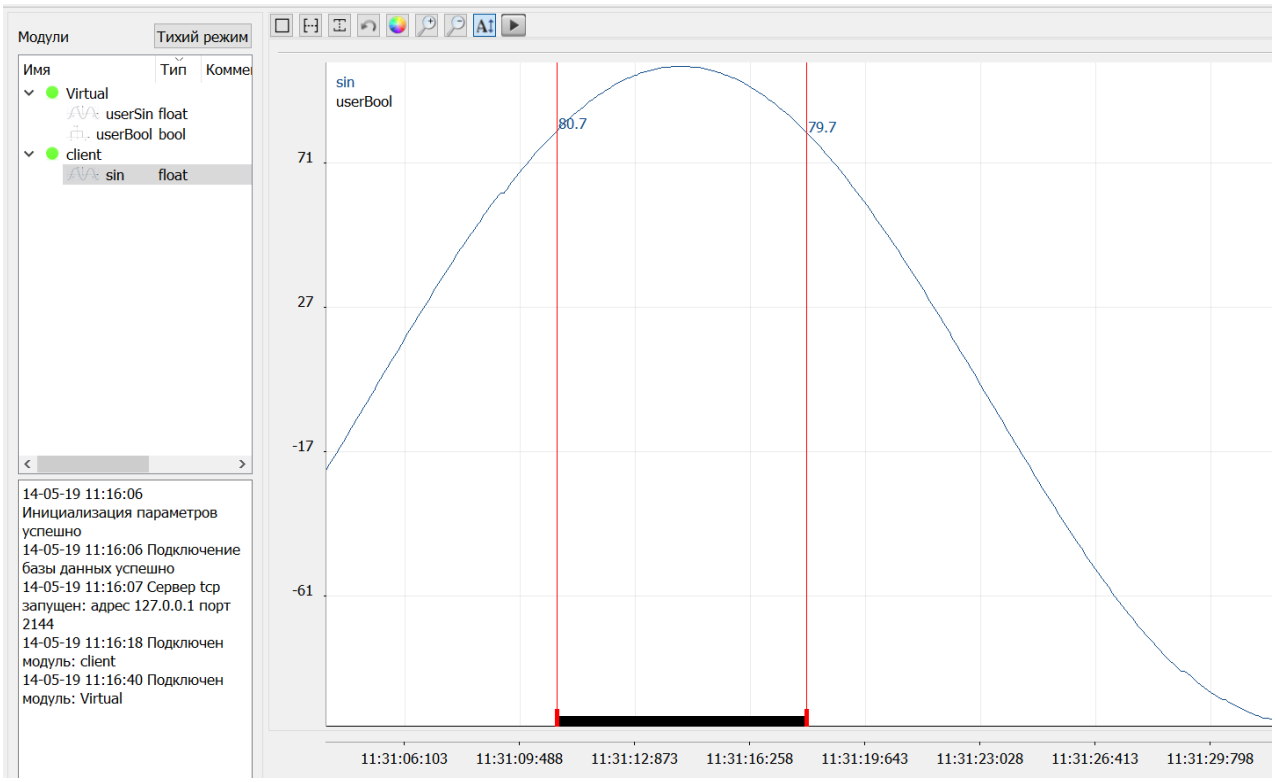
```
int cp = 0;
while(true){

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

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

     Sleep(100);
}
```

2. Open the NewScript.lua script in the input field and type:

```
-- get signal time
tm = getTimeValue("client", "sin");

-- get signal value
val = getFloatValue("client", "sin");

-- get a boolean signal
usVal = (val> 80);

-- create a new virtual signal "userBool"
setBoolValue("userBool", usVal, tm);
```

3. Make the script active.

4. The “Virtual” module will be connected, on the chart you should see the following:



9. Browser view

The browser can only view currently active signals.

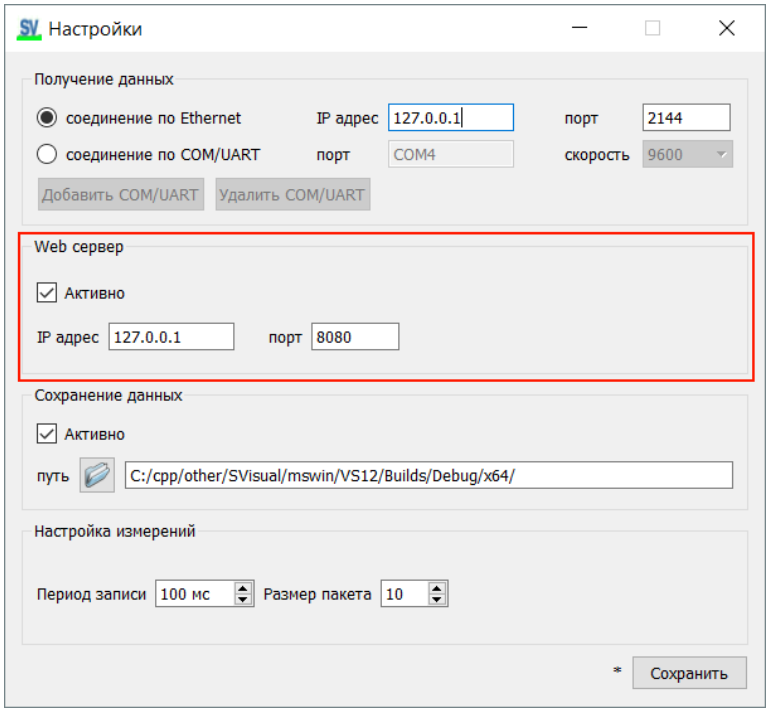


Fig. 9.1 Settings window.

In the settings window, specify the IP address and port for the web server, reboot SVMonitor.

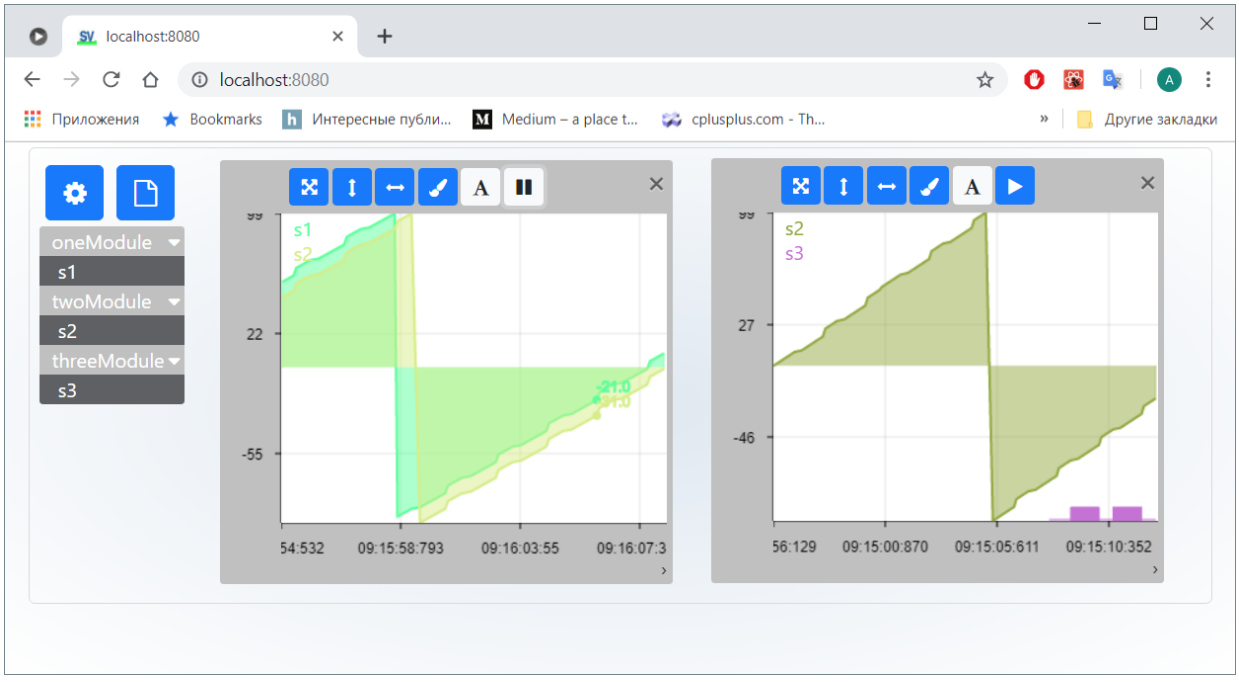


Рис. 9.2 Browser window.

Open a browser, in the address bar write the address that you specified in the settings window.

All viewing operations on the browser page are similar to those described above for SVMonitor.

10. Zabbix agent support

In the settings window, specify the IP address and port for the Zabbix-agent, reboot SVMonitor.

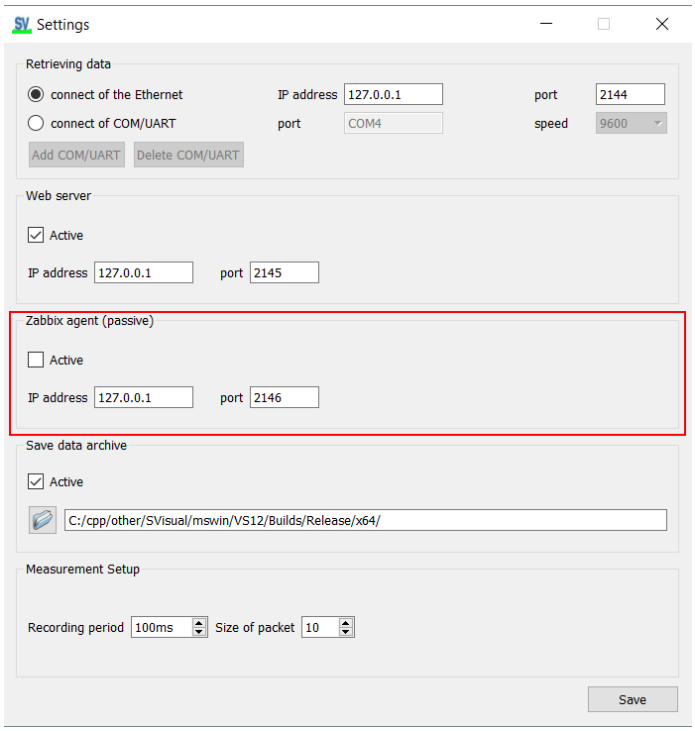


Fig. 10.1 Settings window

To transmit the selected signal to the Zabbix server:

- Zabbix agent should send to the address the name of the signal specified in the settings in the format: "SignalNameModuleName" (together);

- in response, it will receive the last signal value at the moment.

To do this, add a user parameter entry in the Zabbix agent ini-file.

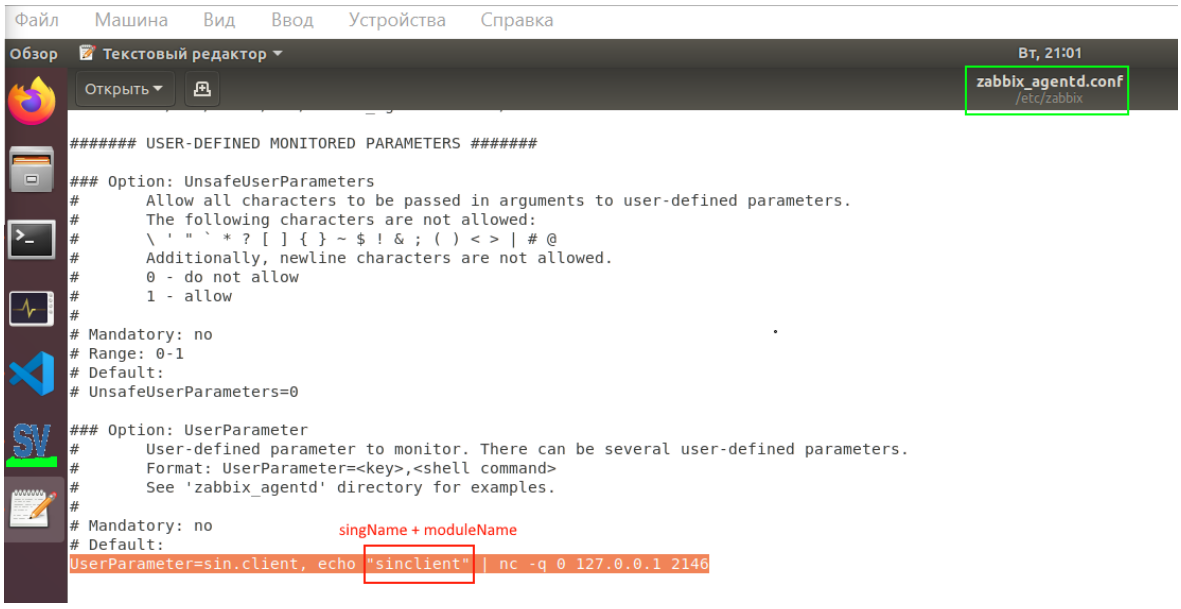


Fig. 10.2 etc/zabbix/zabbix_agentd.conf

11. Description of internal software architecture

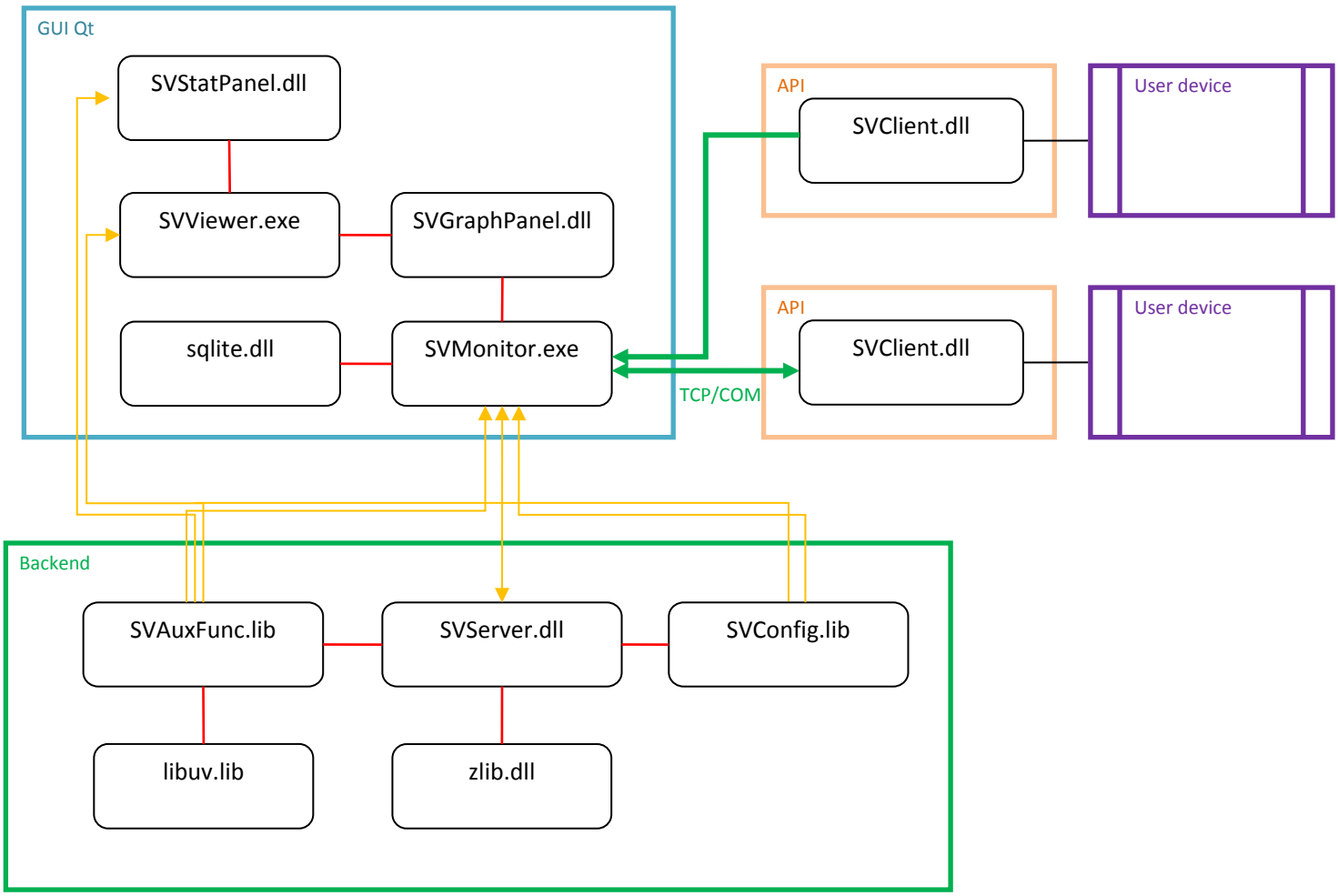


Fig.11.1. Software architecture.

- SVClient.dll – a cross-platform library for transferring data from the client, written in C ++, only STL. Native sockets from SVAuxFunc.lib. Has a C interface.

- SVMonitor.exe – the main graphic part of the software is written in C ++, Qt + STL. To get data from the client, it uses QSerial for COM and libuv (in SVAuxFunc.lib) for TCP. The received data is transferred to the server library SVServer.dll. To display the graphs, use the SVGraphPanel.dll library. The sqlite.dll database is used to store signal names, triggers, event statistics.

- SVServer.dll – server library, written in C ++, only STL. It buffers the transmitted data to display the graph (for SVMonitor), writes the data archive to disk, tracks the cycle of triggering. The archive is compressed with zlib.dll. It does not deal directly with the data. Can be used separately from SVMonitor in the console application, as a service for writing an archive.

-SVConfig.lib – stores the configuration of the system: restrictions on the amount of data received (number of signals, number of modules, triggers), global types of structures.

-SVViewer.exe – viewing the archive of signals, C ++ Qt. Uses the SVGraphPanel.dll library.

Free Style Description.

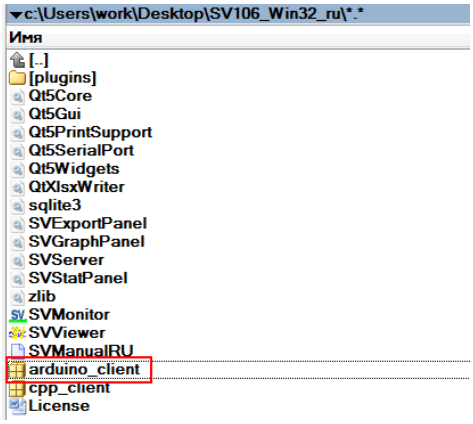
SVMonitor receives data via TCP or COM, transfers data to SVServer for processing.

All signals are added dynamically by SVServer, all data collections are in SVServer.

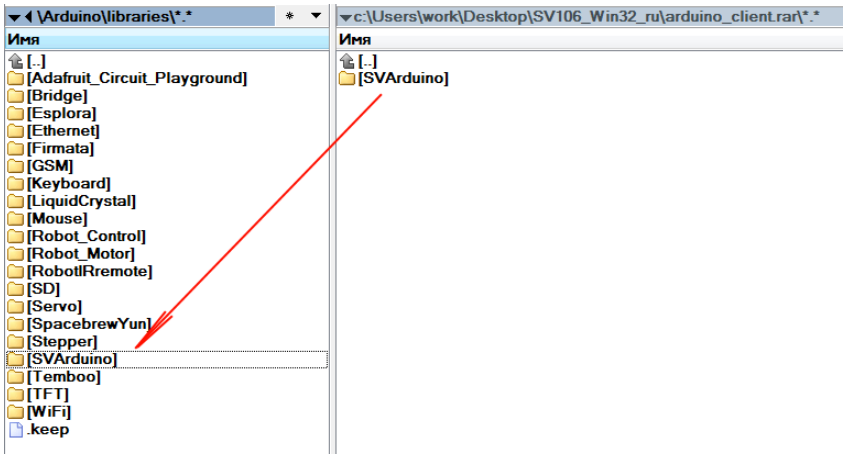
The principle of operation of SVServer: a circular buffer, the client writes the signals to the current buffer position, at the same time the buffer reader "runs" the buffer reader in a separate thread. The reader stores for each signal the archive, the archive size is 10 minutes. After 10 minutes, the archive is flushed to disk, that is, the data is compressed and written to the file by sending it for 10 minutes. Another thread is responsible for handling custom triggers. Triggers are added / removed from SVMonitor.

Appendix 1. Customer for MK Arduino

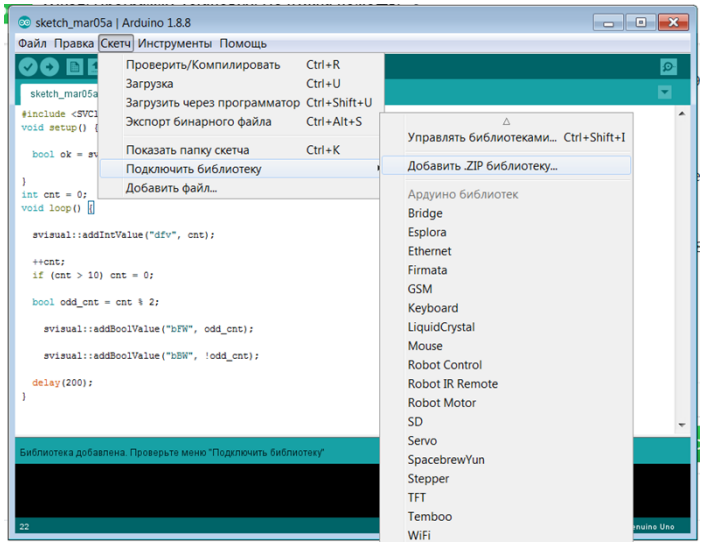
The client is in the folder with the software installed:



Unzip where you have the libraries for the Arduino:



Connect the library to the Arduino IDE:



You do not have to care about the data transfer, the data transfer occurs automatically by timer interrupts, timer 2 is used.

It is only necessary to update the variables for writing.

When connecting to COM, you should not use the output of messages on the COM port (Serial.Print(), Serial.Write()).

Interface

```
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);


};
```

connectOfEhernet - function for Ethernet connection. It takes the IP address of the Arduino module, the address of the SVMonitor server. Returns TRUE if connection is established.

connectOfWiFi - function for connection over WiFi. It takes the name of the network and the password, the address of the SVMonitor server. Returns TRUE if connection is established.

connectOfCOM - function for connection to COM port. It takes the name of the module and the transmission rate. Returns TRUE if connection is established.

addBoolValue(const char* name, bool value, bool onlyPosFront = false) – function is used to write a variable of type bool.

name - the name of the variable, should not contain the combinations "= end=" and "=begin=" and exceed the length of 24 characters;

value - the value of the variable;

onlyPosFront - if only the positive edge of the signal is to be recorded.

addIntValue(const char* name, int value) – function is used to write a variable type int.

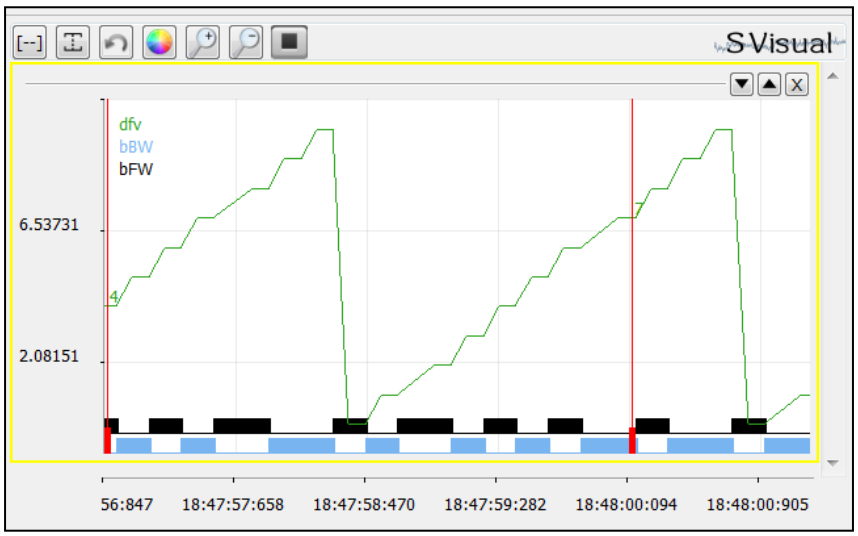
name - the name of the variable, should not contain the combinations "= end=" and "=begin=" and exceed the length of 24 characters;

value is the value of the variable;

Example sketch

```
#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);
}
```

As a result, you will see:



Appendix 2. Client for C / C ++.

Written in C ++. Can be used in multi-threaded applications.

```
#pragma once

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

// соед-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);
```

Example

```
#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;
}
```

Appendix 3. Client for NET (C #).

Is a wrapper of the C ++ client library.

```
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);
        }
    }
}
```

Example of use

```
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);

         }
```

Appendix 4. Client for Python.

Is a wrapper of the C ++ client library.

```
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
```

Example of use

```
from SVClient import SVClient

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

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

Appendix 5. Rules of writing a client for any MK and devices

For each variable you write, you must create an array (let it) for 10 values. And update the variable for the current internal loop - interrupt (let) 100 ms (see Annex 7).

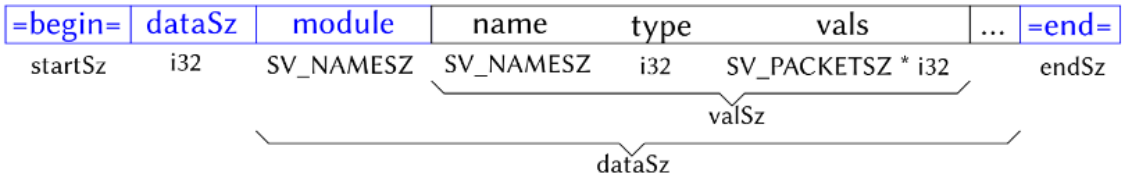
```
bool sv_client::addValue(const char* name, valueType type, value val, bool onlyPosFront){

		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;
	}
```

Every 100 ms (see Appendix 7) for interrupts, you should check whether the logged variable was updated in the current cycle and update it with the previous value, if it was not updated:

```
	// 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) {

               isConnect_ = (isConnect_) ? sendData() : false;

               curCycCnt_ = 0;
        }
        else curCycCnt_++;
```

Every second (see Appendix 7) on a socket or COM, you need to send the following data structure:



Package start marker

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

Total data size

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

Module name

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

Data recorded over the past second

```
dataRef * data;
for (int i = 0; i < sz; ++i){

       data = (dataRef *)values_.at(i);

       send(data, vlSz);
}
```

The end-of-packet marker

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

More details see the sketch for the Arduino MC.

Appendix 6. API for obtaining current values

To the SVMonitor address on the socket, you can send the following commands in JSON format:

- Revert all signals

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

Answer

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

- Revert all triggers

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

Answer

```
{
"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
}
```

- Return the current signal value

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

Signal – наименование сигнала
Module – наименование модуля

Answer

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

If the signal is not found, the answer is:

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

Appendix 7. Setting the Frequency of Data Acquisition

The frequency of recording and retrieving data is specified by parameters in SVMonitor.ini, the file is located in the installation folder of the software.

```
cycleRecMs = 100 мс – signal recording cycle - maximum recording frequency
packetSz = 10 – data packet size
```

The default setting corresponds to a recording frequency of 10 Hz, sending a packet every second, a packet size of 10 values.

For example, you want to send data every minute, at a frequency of 1 Hz (value per second), then you install:

```
cycleRecMs = 1000 мс
packetSz = 60
```

cycleRecMs should not be set less than 10 ms.

If you changed these parameters in SVMonitor.ini, then the changes must be made in the:

- initialize the record viewer SViewer.ini;

- for the client library sv_client.dll before calling the method:

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

- for the Arduino client, fix it in a file sv_structurs.h:

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

Appendix 8. Python script for sending email

Can be used to handle event triggers.

```
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')
```

Ini file

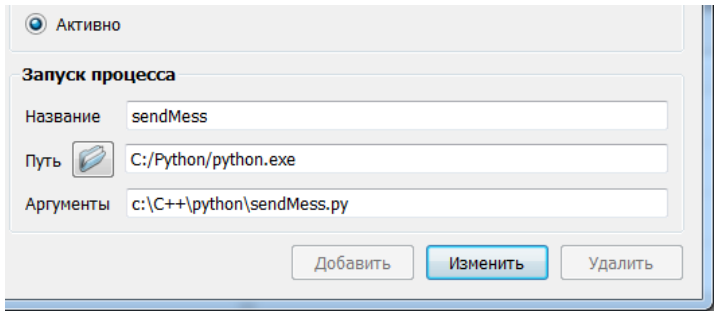
[Param]

recvAddr = recv@mail.ru

recvPassw = 12345

sendAddr = user1@mail.ru user2@mail.ru

Example of use



License