Tester VFS169

Table of contents

System pomocy dla programu TESTER_169 jest integralną częścią programu.

Zastosowanie programu

Program znajduje zastosowanie do wizualizacji parametrów technicznych na teście oraz automatycznej archiwizacji danych.
Program służyć również możne do zarządzania przepływem informacji wykorzystując Microsoft SQL Server® 2008 Express.

Oprogramowanie nie może być wykorzystywane do innych celów oraz zgodnie z licencją.

Wymagania systemowe

Elementy konieczne do uruchomienia programu

- Procesor 1 GHz lub szybszy
- 1 GB pamięci RAM
- 16 GB wolnego miejsca na dysku twardym
- Urządzenie graficzne DirectX 9 ze sterownikiem WDDM 1.0 lub nowszym

Ustawienie systemowe:

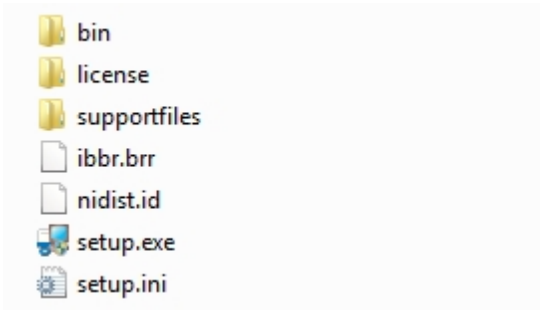
- Microsoft SQL Server® 2008 Express
- Microsoft .NET Framework 4

Instalacja oprogramowania

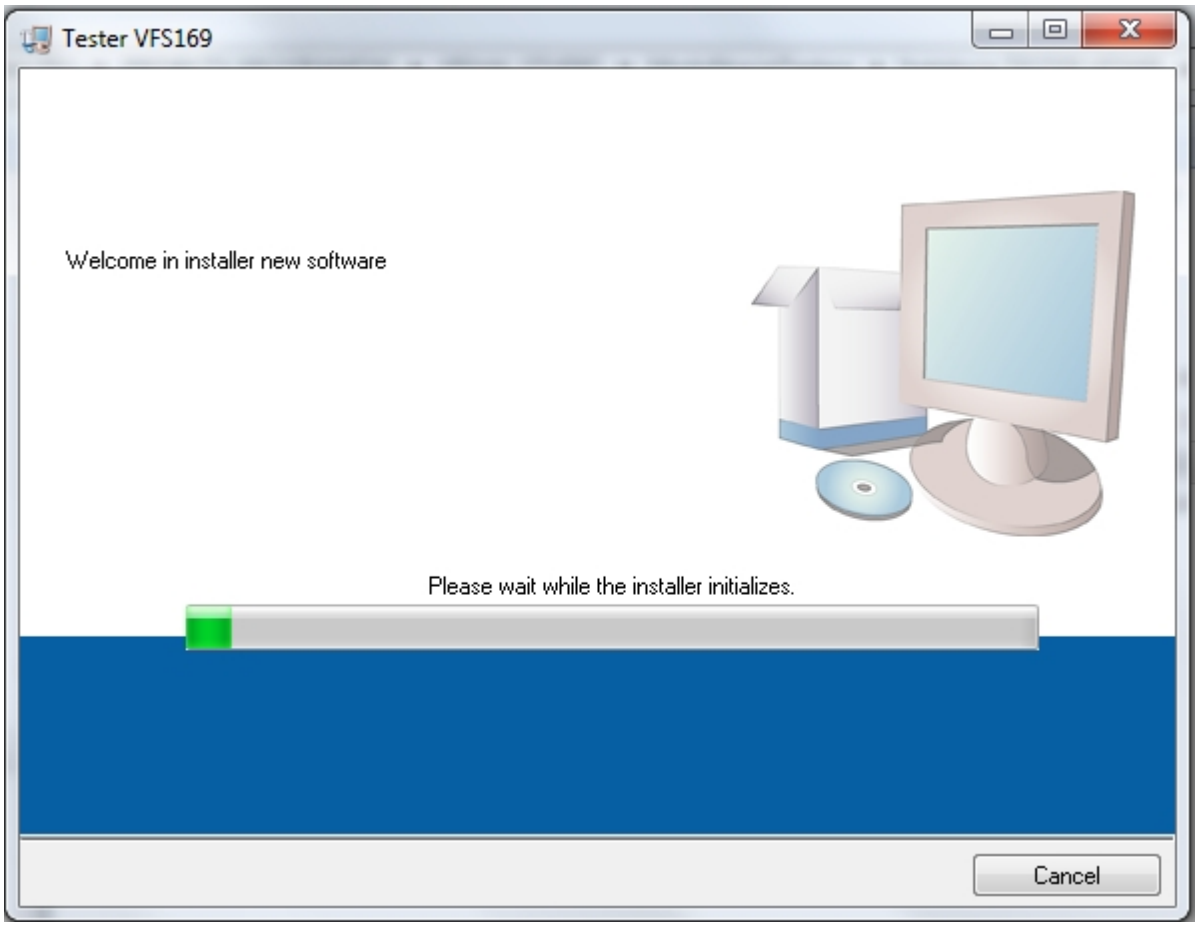
Instalacja oprogramowania powinna następować tylko przy wykorzystaniu oficjalnego instalatora.

Instalalacja oprogramowania

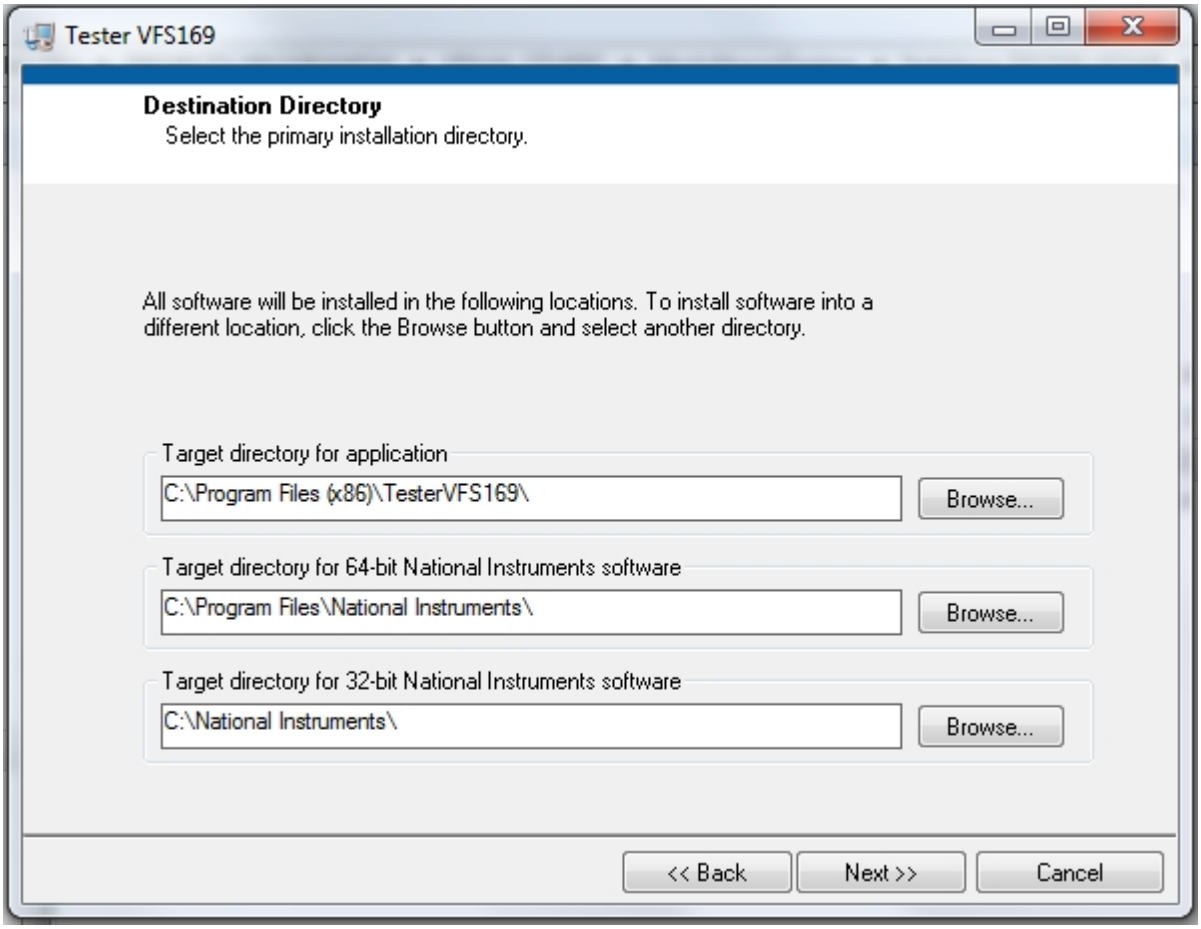
1. W folderze instalacyjnym odszukać *Setup.exe*



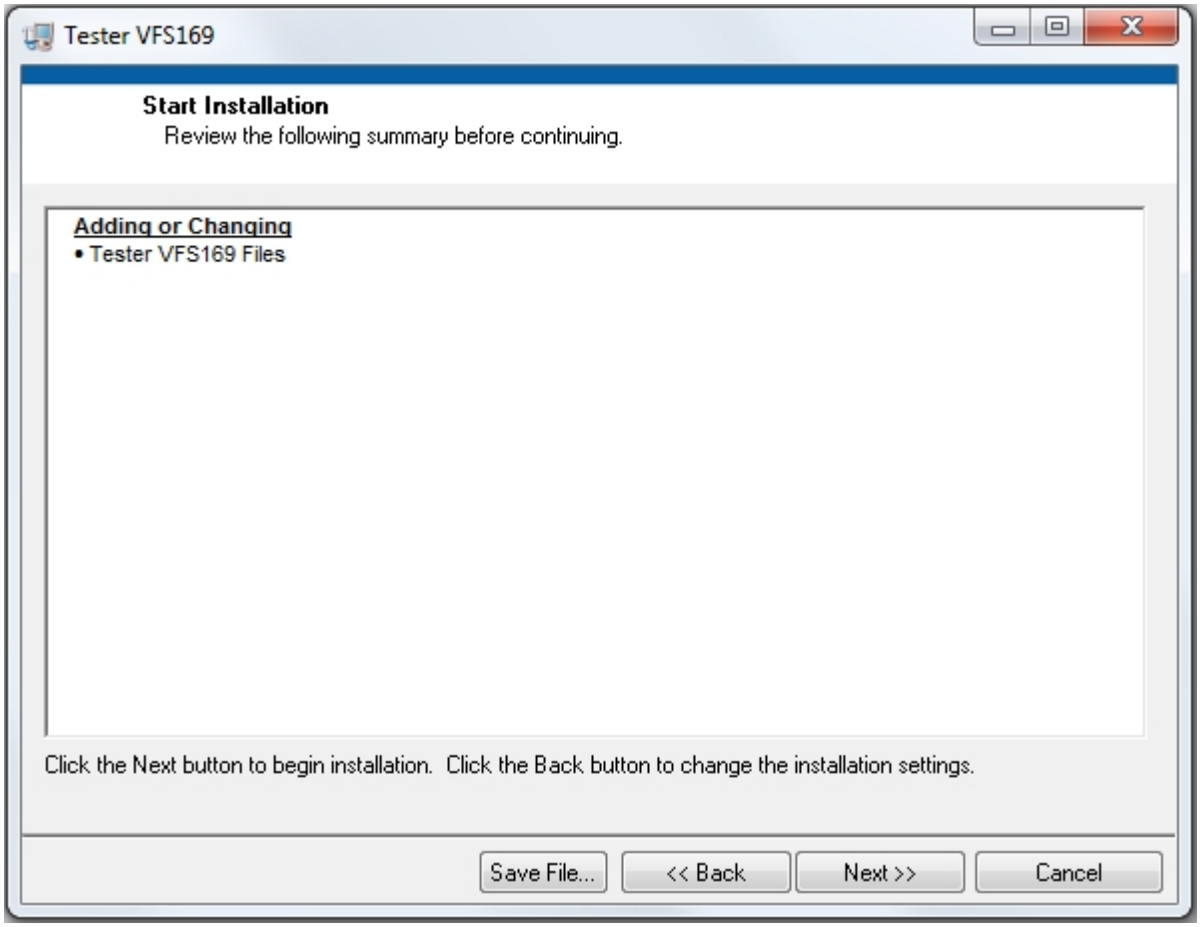
2. Otwieranie instalatora



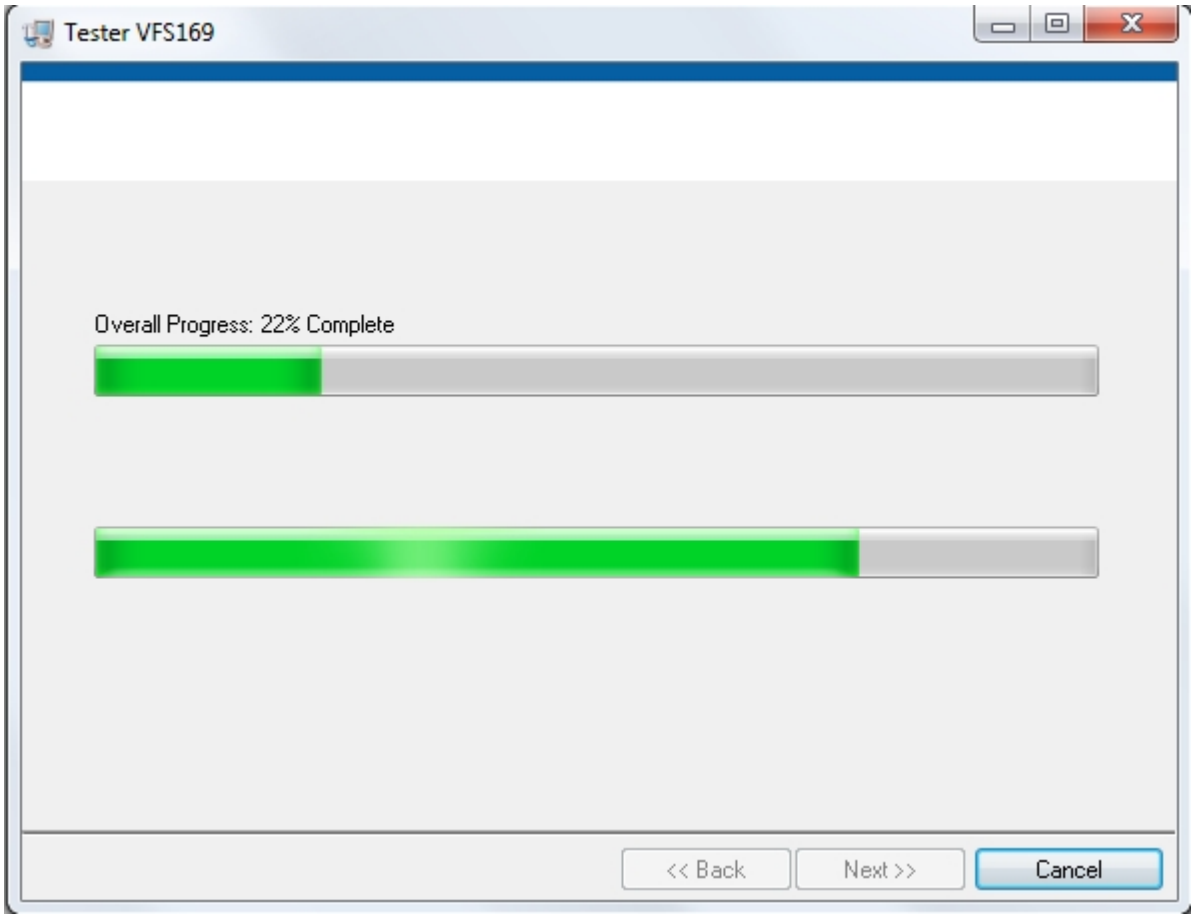
3. Wybór folderów instalacyjnych



4. Podsumowanie instalacji



5. Rozpoczęcie procesu instalacji

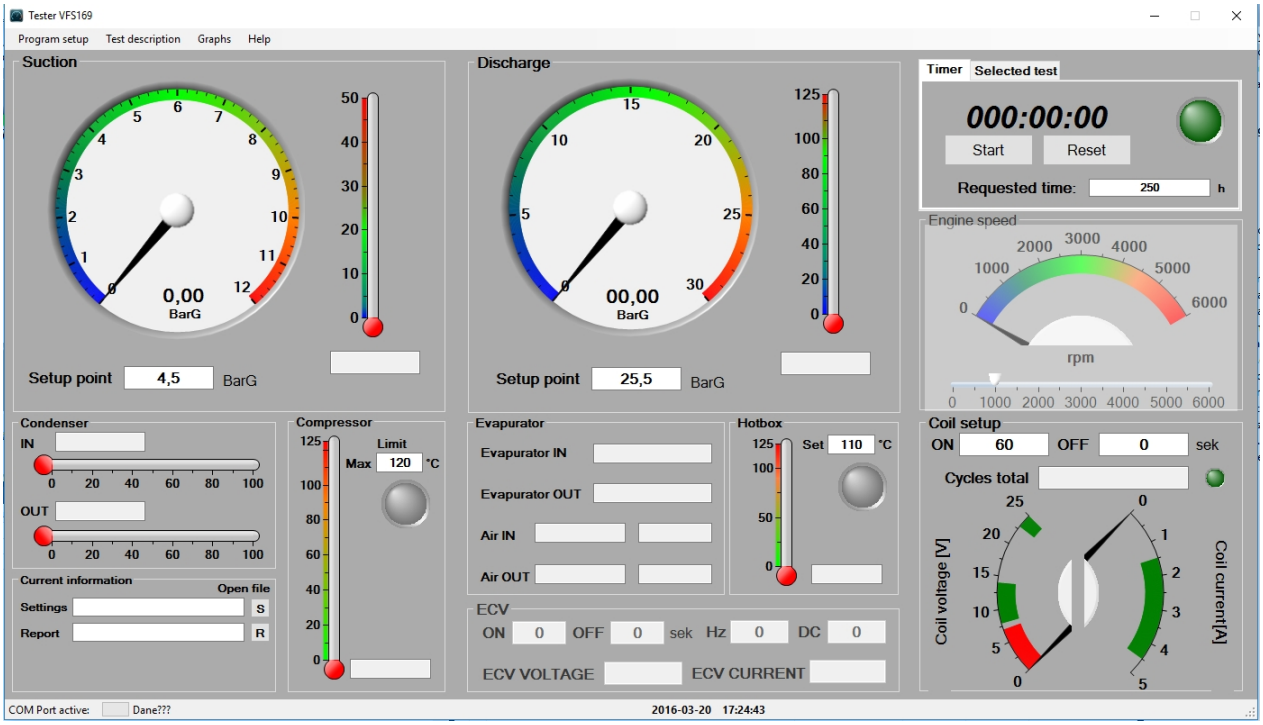


Podstawowa obsługa

1. Ekran główny
2. Ekran wykresów
3. Ekran systemu AC
4. Ekran terminala
5. Zarzadzanie katalogami

Ekran główny

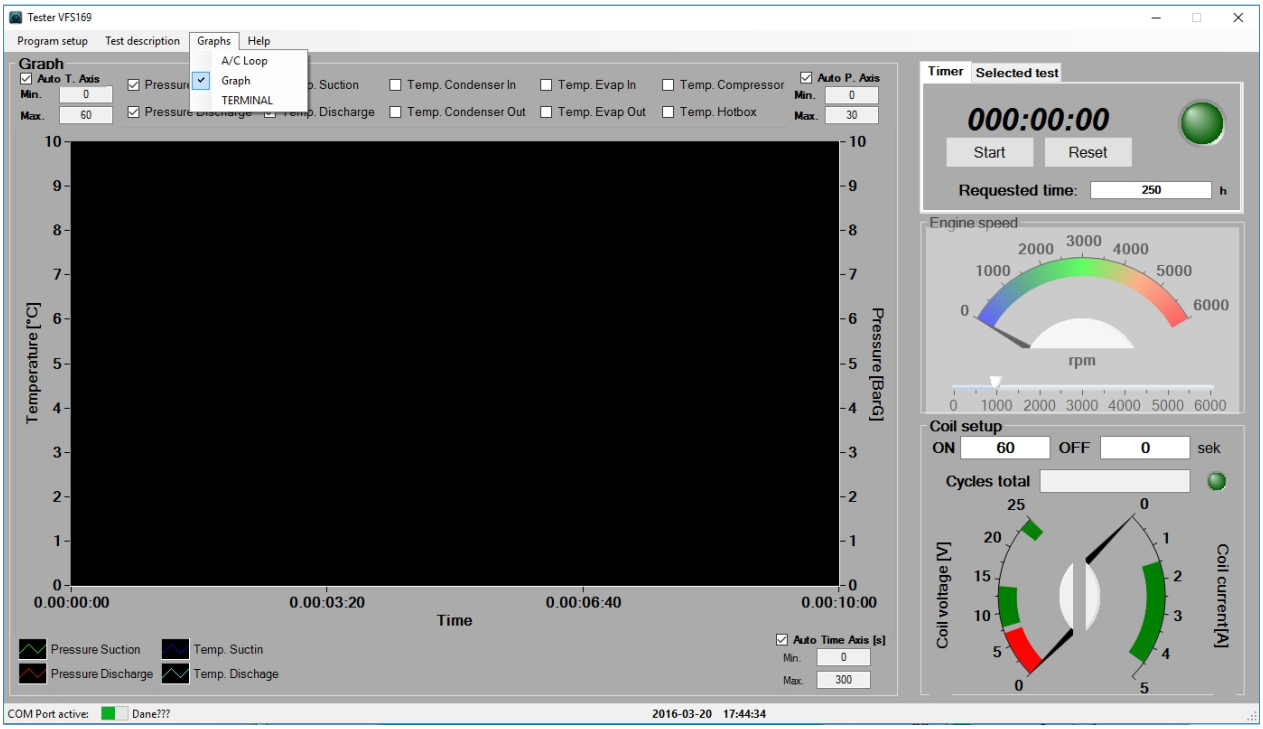
Ekran początkowy



Ekran wykresów

Ekran wykresów realizuje graficzną prezentacje wyników napływających w czasie rzeczywistym.

Aktywacja okna wykresów następuje w menu Graphs -> Graph



Ekran ten może być modyfikowany poprzez:

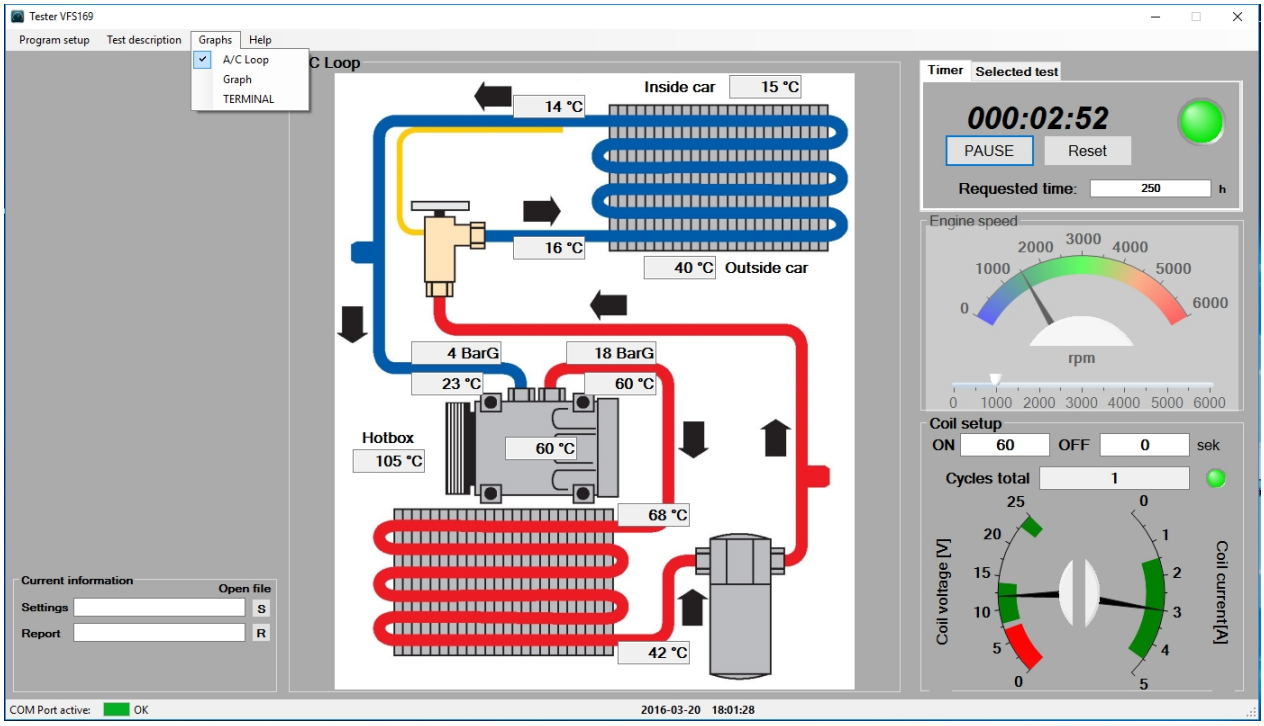
- Wyświetlanie tylko interesujących parametrów
- Modyfikacja osi temperatury
- Modyfikacja osi ciśnienia

- Modyfikacja osi czasu

Ekran systemu AC

Ekran systemu AC służy w celach dydaktycznej reprezentacji wyników w systemie klimatyzacji.

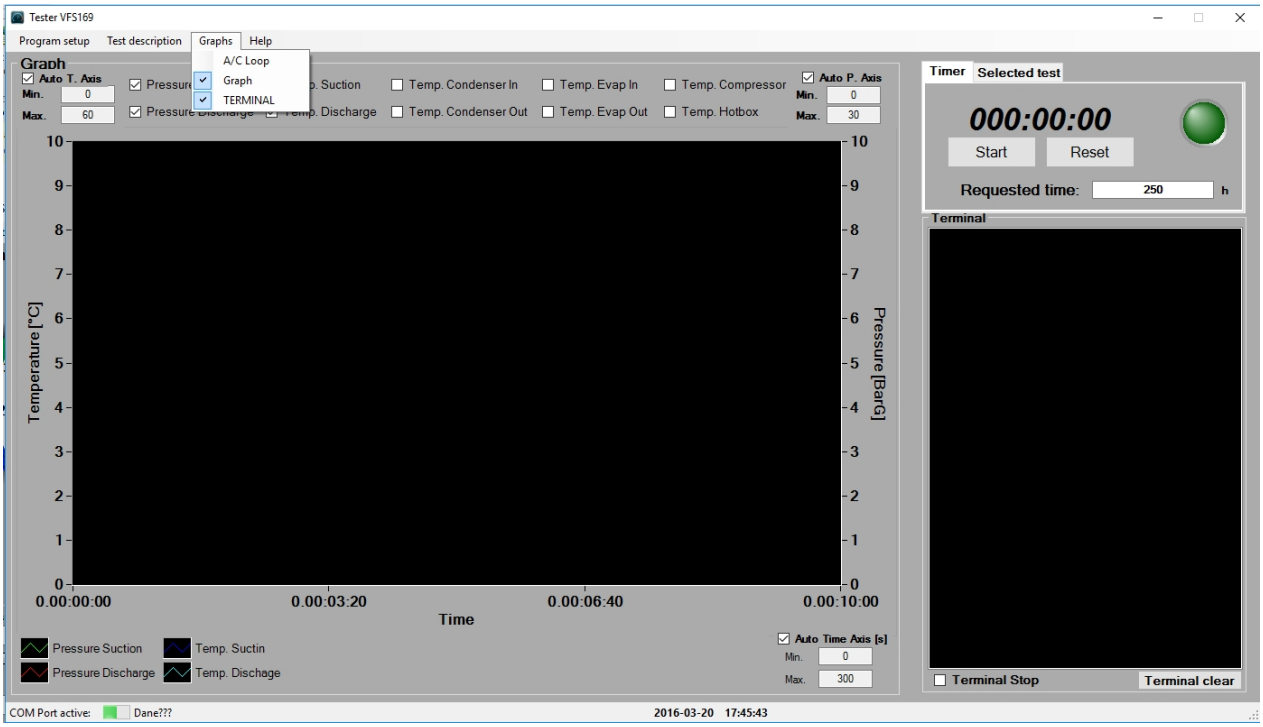
Aktywacja okna wykresów następuje w menu Graphs -> A/C Loop



Ekran terminala

Terminal reprezentuje przesyłane dane przez Serial Port dla temperatur.
Okno terminala może byś uruchomione niezależnie od wyboru reprezentacji graficzne (Ekran główny, Ekran Wykresów, Ekran Systemu)

Aktywacja okna wykresów następuje w menu Graphs -> Terminal



Opcje terminala:

- ***Terminal Stop*** - Wstrzymanie wyświetlania informacji
- ***Terminal clear*** - Wyczyszczenie przestrzeni danych

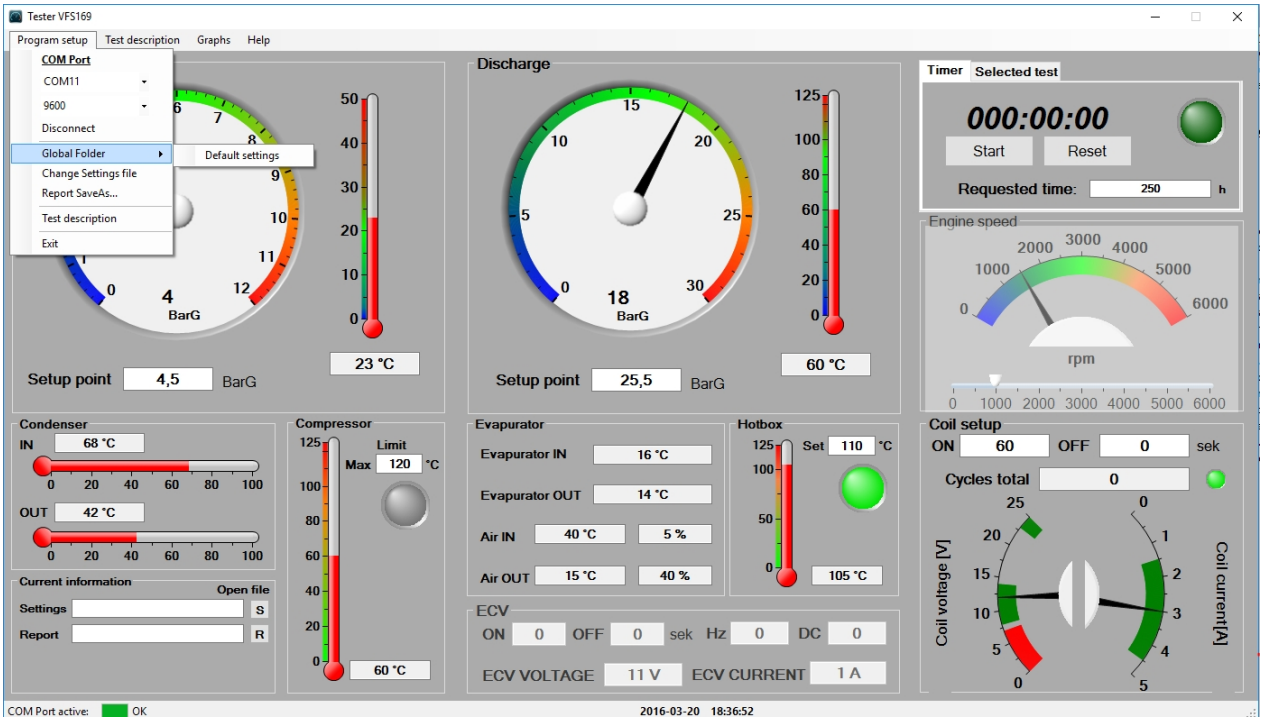
Zarzadzanie katalogami

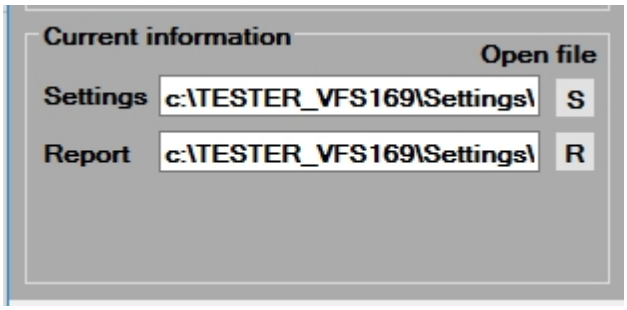
Zarządzanie folderami odbywa się w menu Program Setup

Ustawienie folderów domyślnych dla:

- c:\TESTER_VFS169\Settings\Settings.xml
- c:\TESTER_VFS169\Settings\Report.txt

można ustawić poprzez Program Setup -> Global Folder -> Default settings





Ustawione foldery można dowolnie zmienić korzystając z opcji:

- Program Setup -> Change Settings File - Zmiana miejsca położenia dla Settings.xml
- Program Setup -> report Save As - Zmiana miejsca położenia dla pliku raportu

Przyciski "S" oraz "R" otwierają zdefiniowane pliki "Settings.xml" oraz plik raportu.

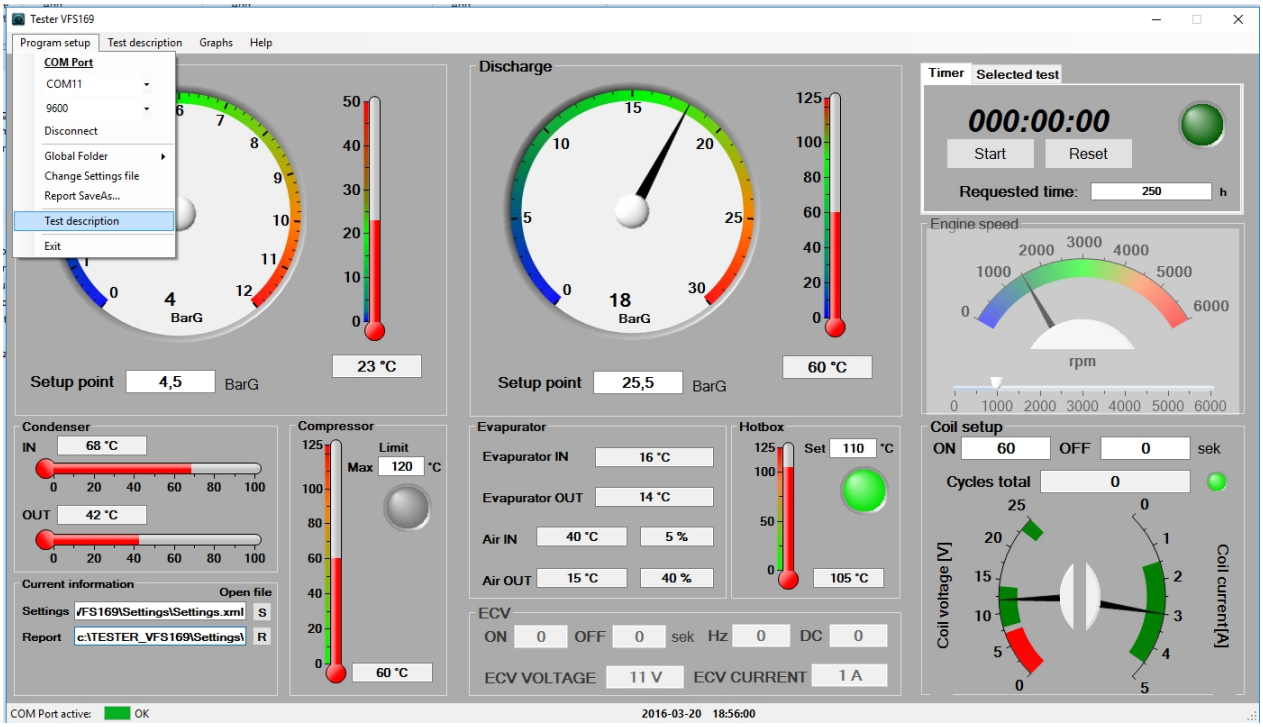
Podwójne klikniecie na ścieżkę dostępu kopiuje lokalizacji.

Obsługa zaawansowana

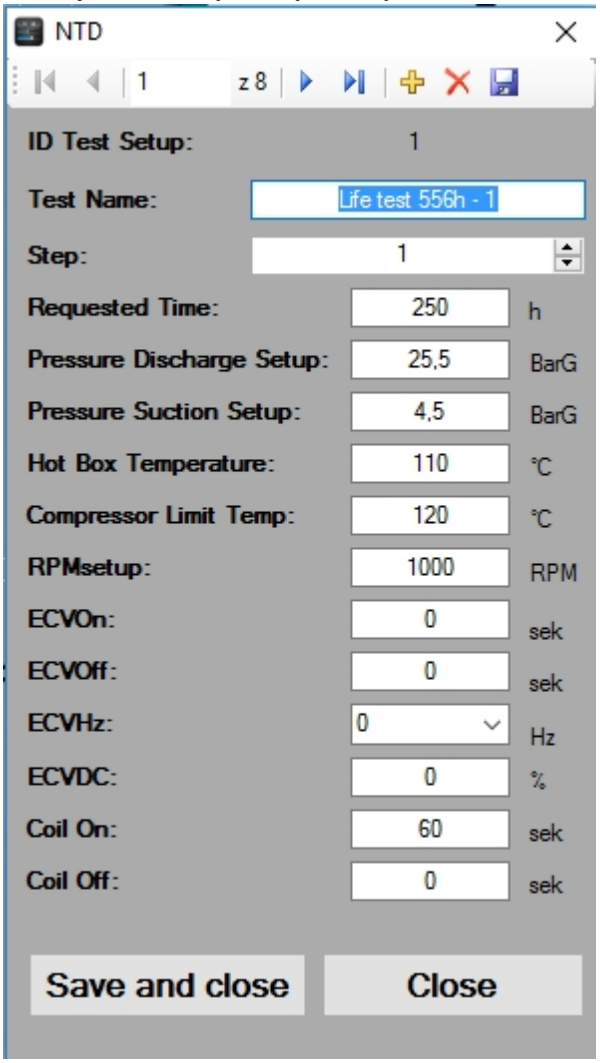
1. Definiowanie testu
2. Definiowanie rozpoczęcia testu
3. Dodawanie komentarz podczas testu
4. Zakończenie test
5. Kontynuacja testu po przerwaniu działania programu

Definiowanie testu

Definicja nowego testu do bazy danych możliwa jest po wybraniu opcji menu Program Setup -> Test Description



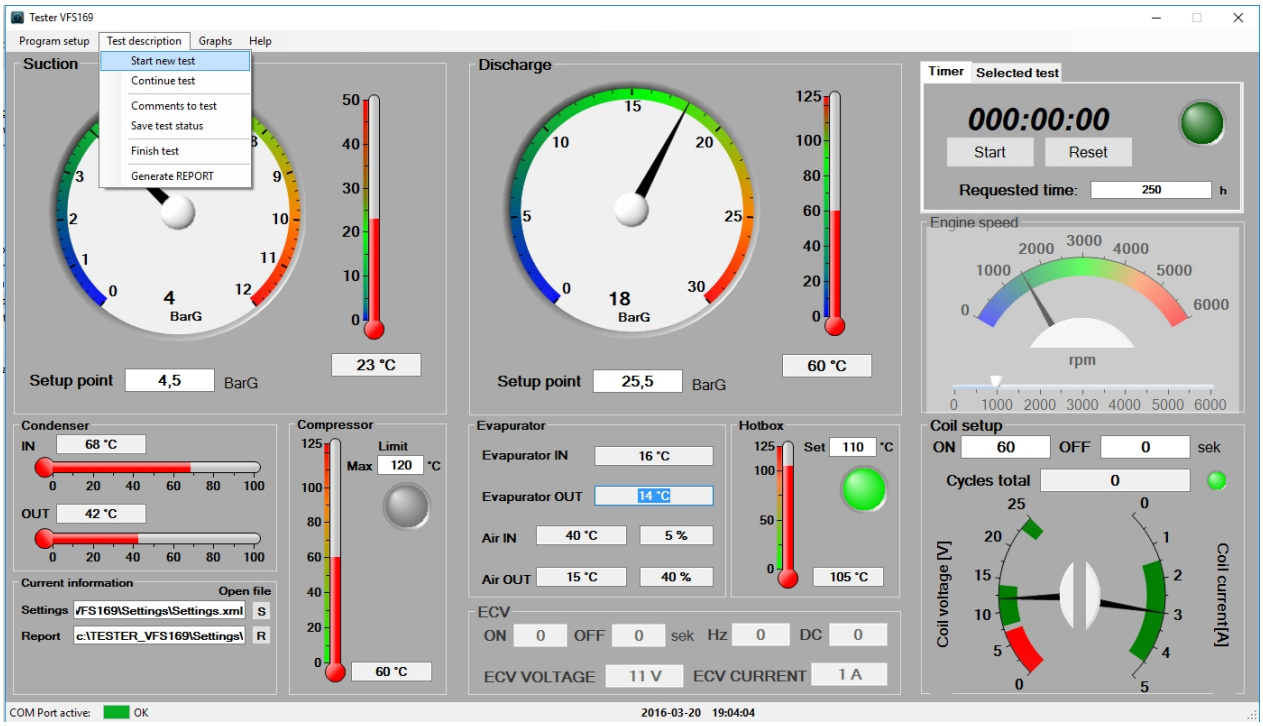
Definicja testu odbywa się w nowym oknie



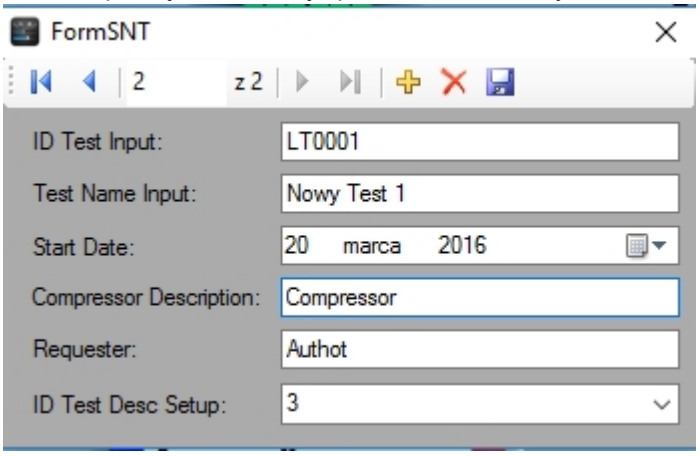
Dane dodawane są do bazy danych.
Każda zmiana musi być zapisana ręcznie.

Definiowanie rozpoczęcia testu

Każdy test powinno się rozpoczynać od wprowadzenia danych rozpoczęcia testu.
Jest to możliwe w menu Test description -> Start new test



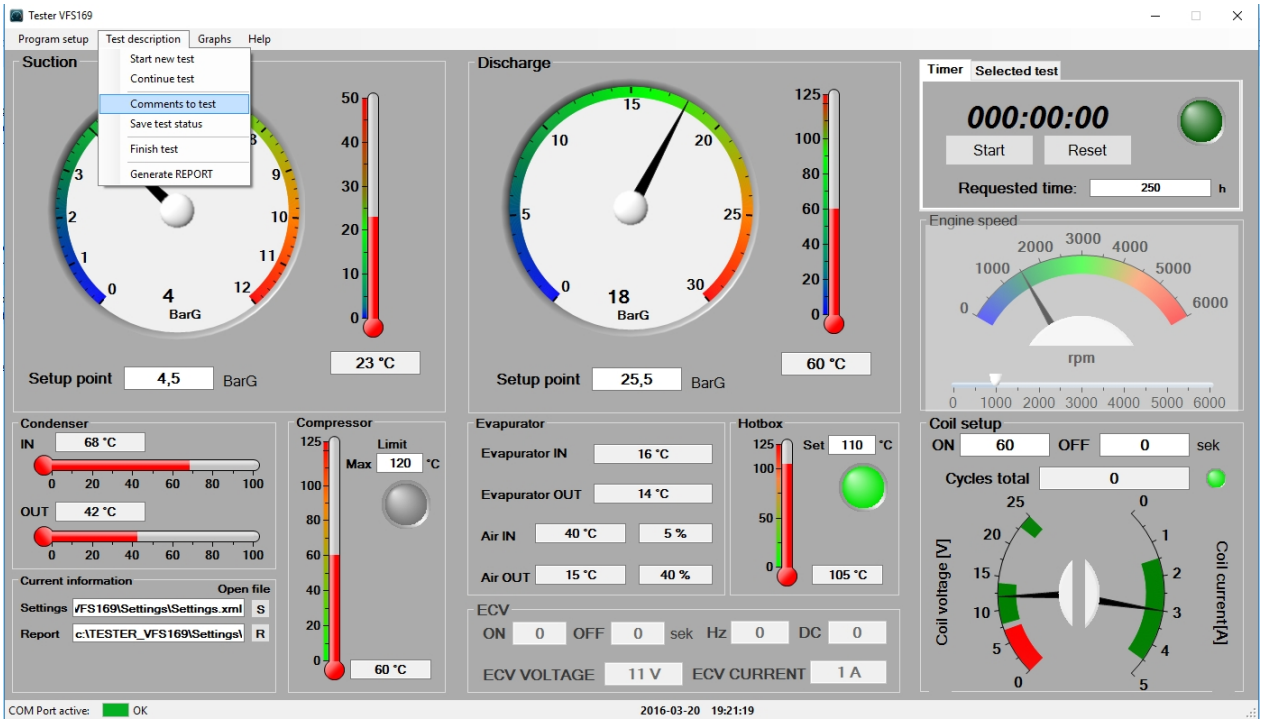
Dane rozpoczęcia testu są wprowadzane w nowym oknie:



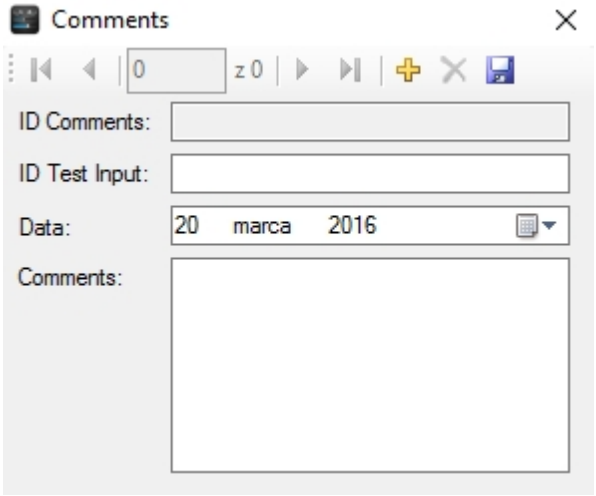
Dane dodawane są do bazy danych.
Każda zmiana musi być zapisana ręcznie.

Dodawanie komentarz podczas testu

Istnieje możliwość dodania komentarza do aktualnego testu.
Jest to możliwe w menu Test description -> Comments to test



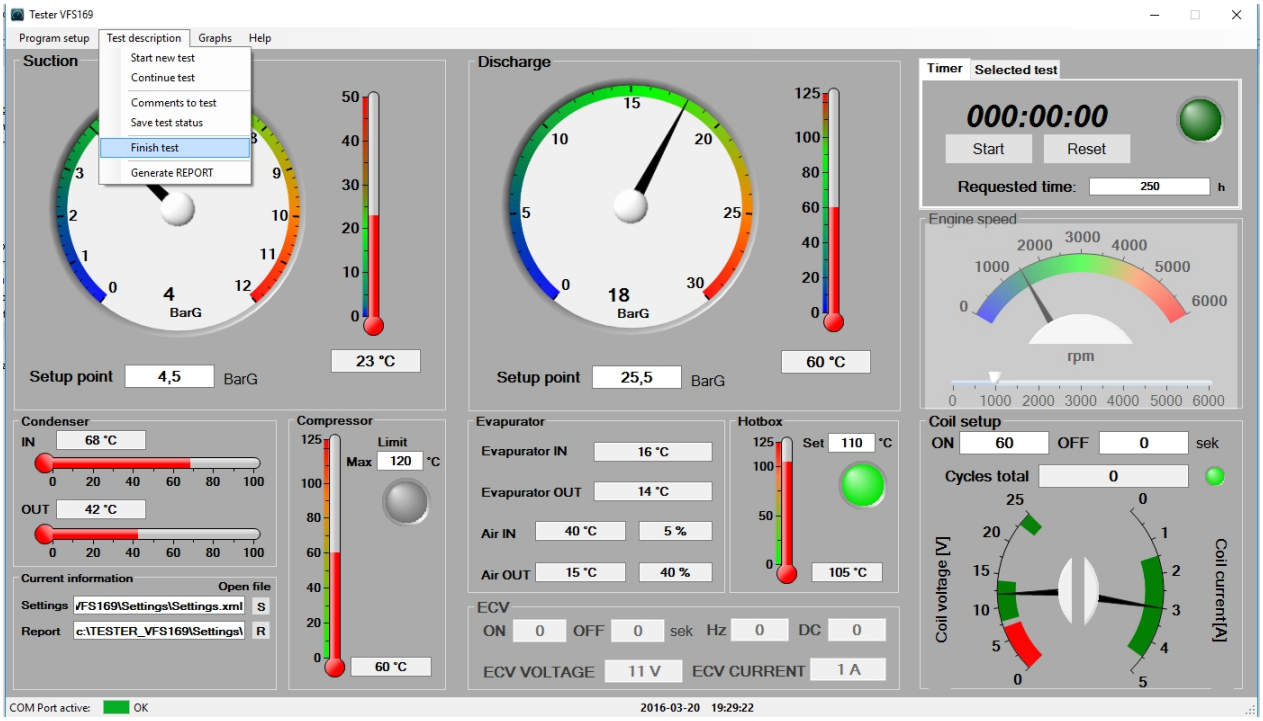
Dane komentarza do testu są wprowadzane w nowym oknie:



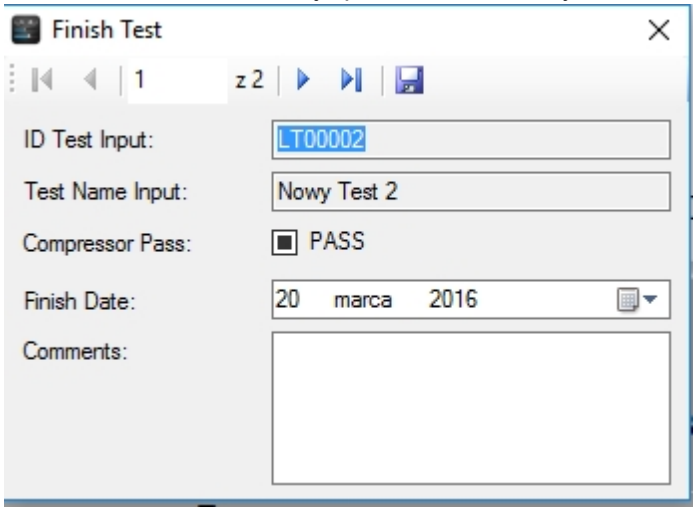
Dane dodawane są do bazy danych.
Każda zmiana musi być zapisana ręcznie.

Zakończenie test

Każdy test powinien zakończyć się posumowaniem dodawanym do bazy danych.
Jest to możliwe w menu Test description -> Finish test.



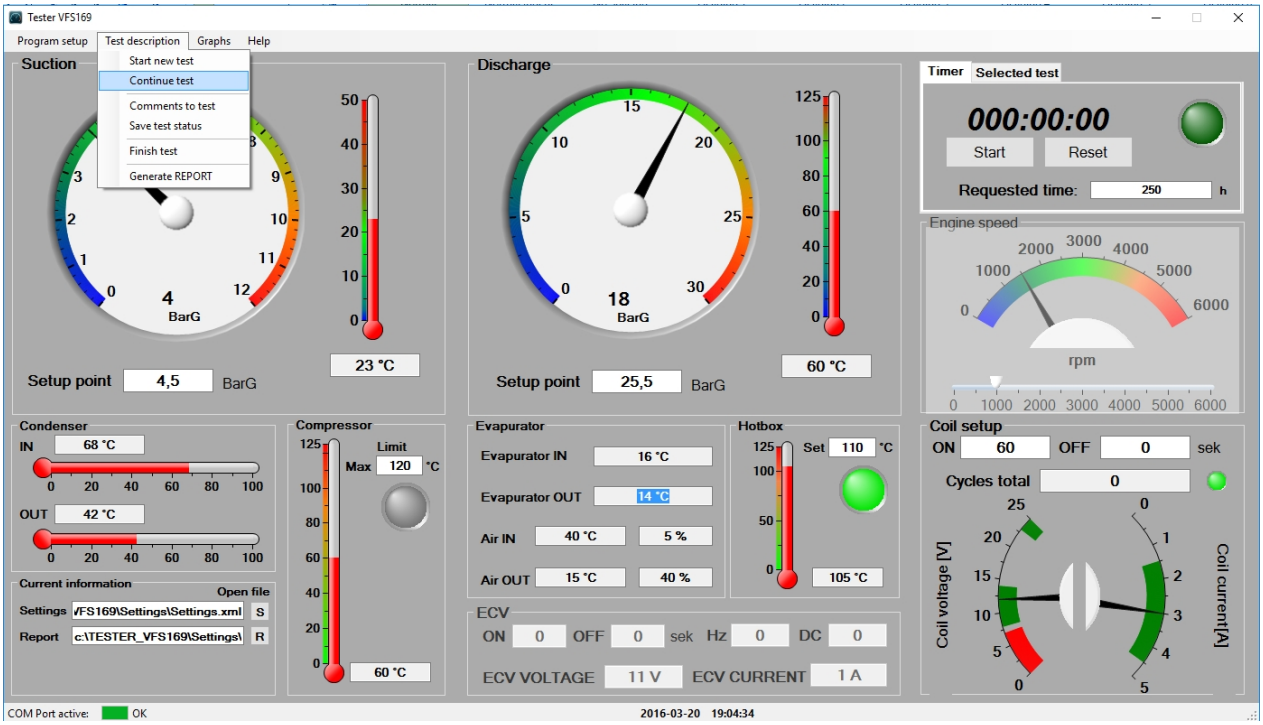
Dane zakończenia testu są wprowadzane w nowym oknie:



Każda zmiana musi być zapisana ręcznie.

Kontynuacja testu po przerwaniu działania programu

Opcja z menu Test description -> Continue test umożliwia wczytanie parametrów testu po świadomym przerwaniu działania pracy programu.
Dane do przywrócenia znajdują się w "Settings.xml".



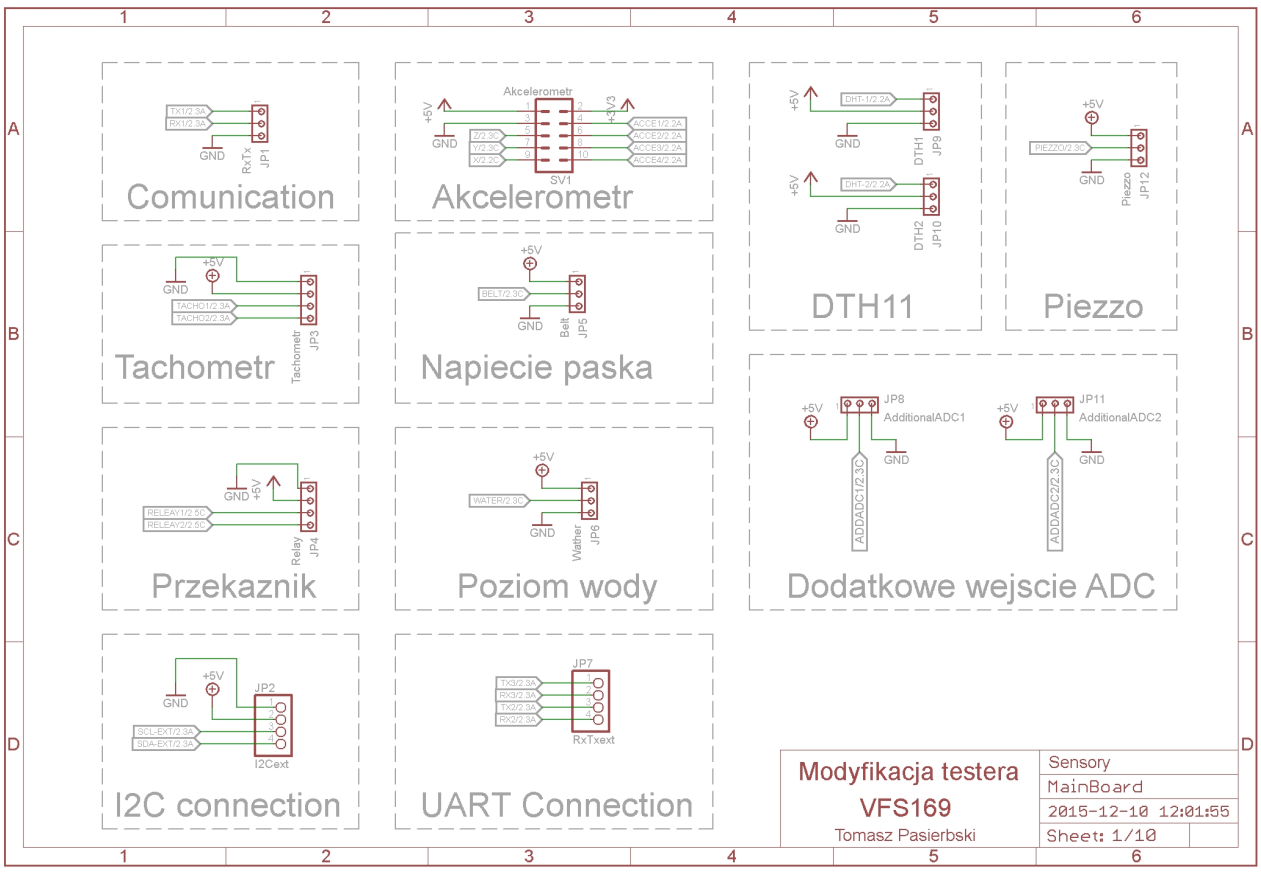
Dane testu można zapisać w dowolnym momencie korzystając z opcji Test description -> Save test status. Status również zapisywany jest automatycznie przy zamykaniu programu.

Dokumentacja techniczna

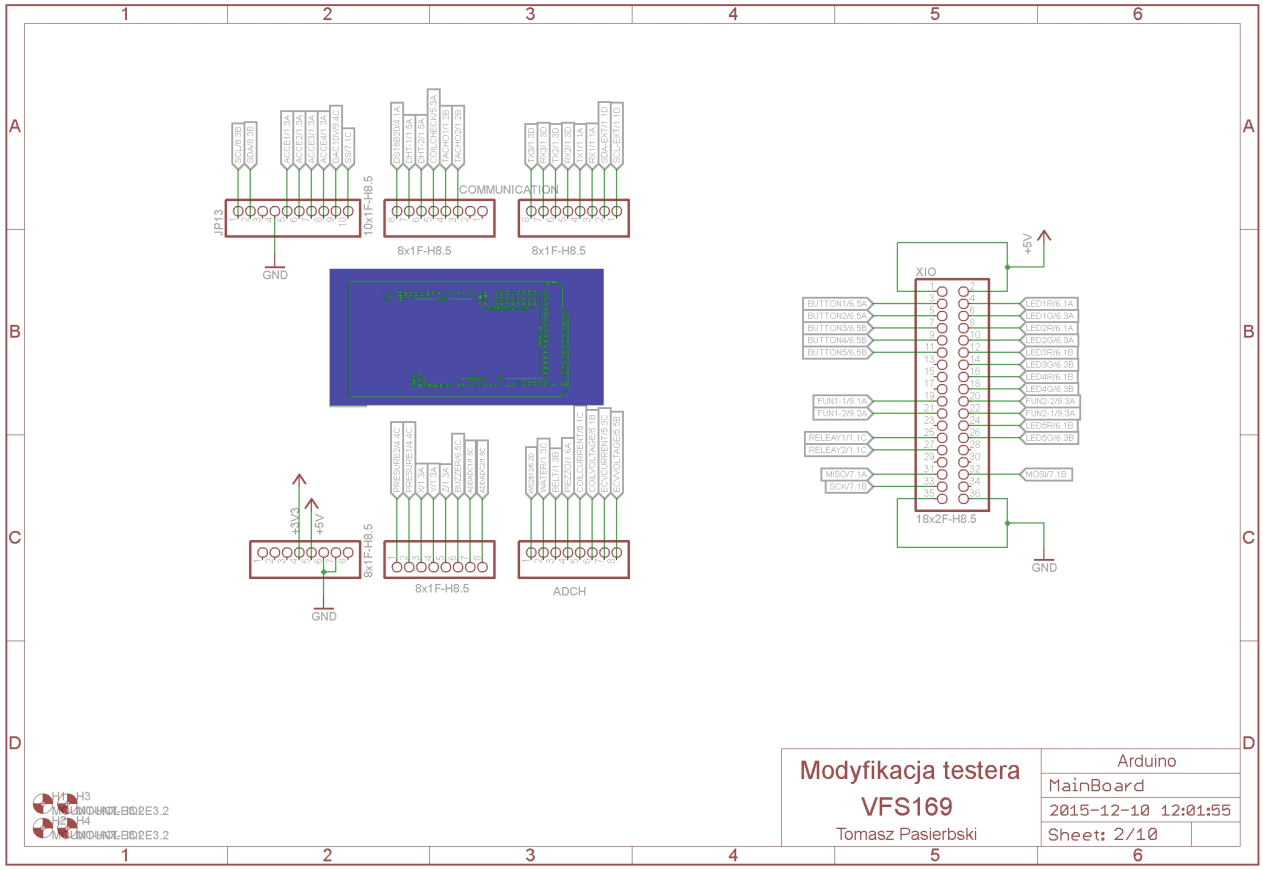
1. Schemat
2. PCB

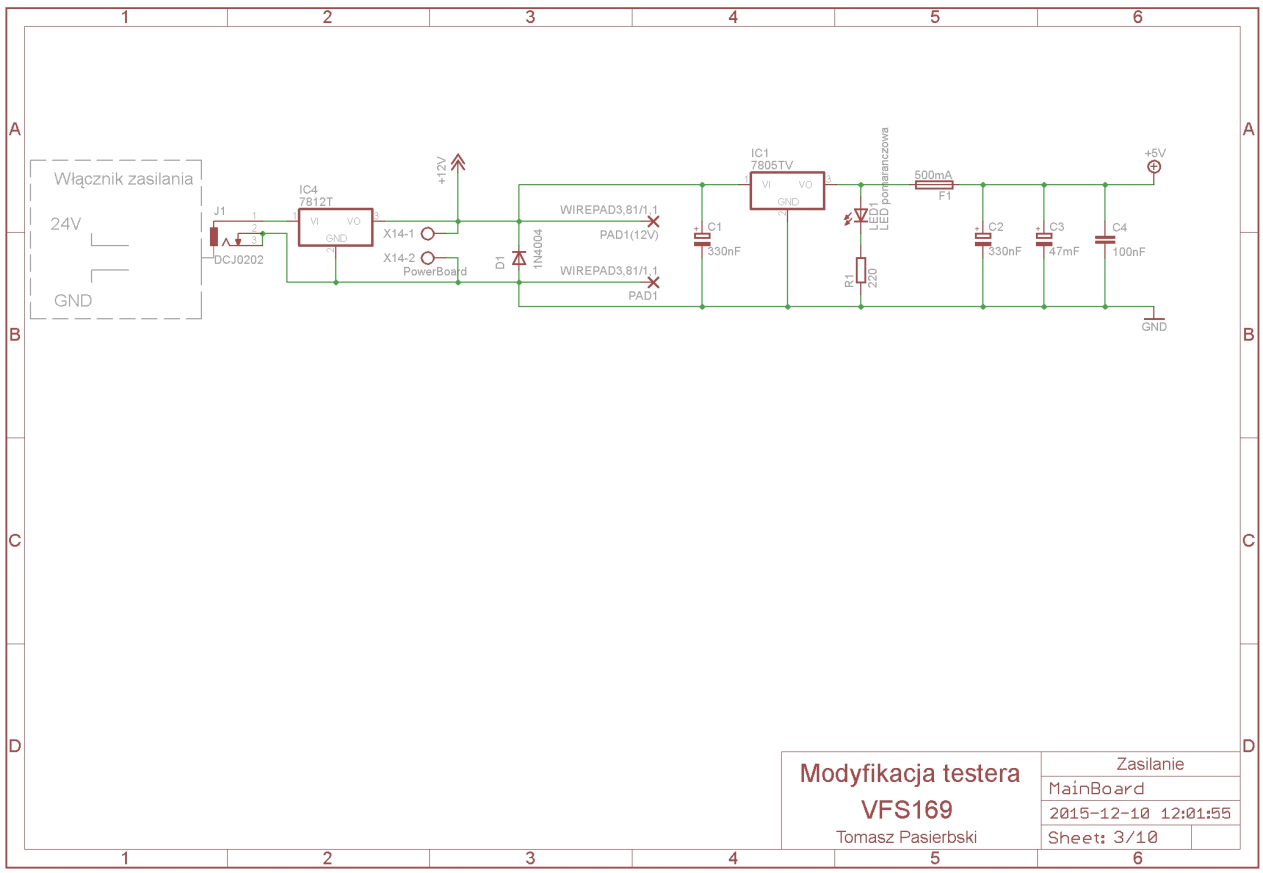
Arkusz	Opis
1/10	Sensory
2/10	Arduino
3/10	Zasilanie
4/10	Czujniki pomiarowe (Temperatura, Cisnienie)
5/10	Pomiary elektryczne
6/10	Interface uzytkownika
7/10	Karta SD
8/10	RTC
9/10	Sterowanie zewnętrzne (wentylatory, falownik)
10/10	Spis tresci

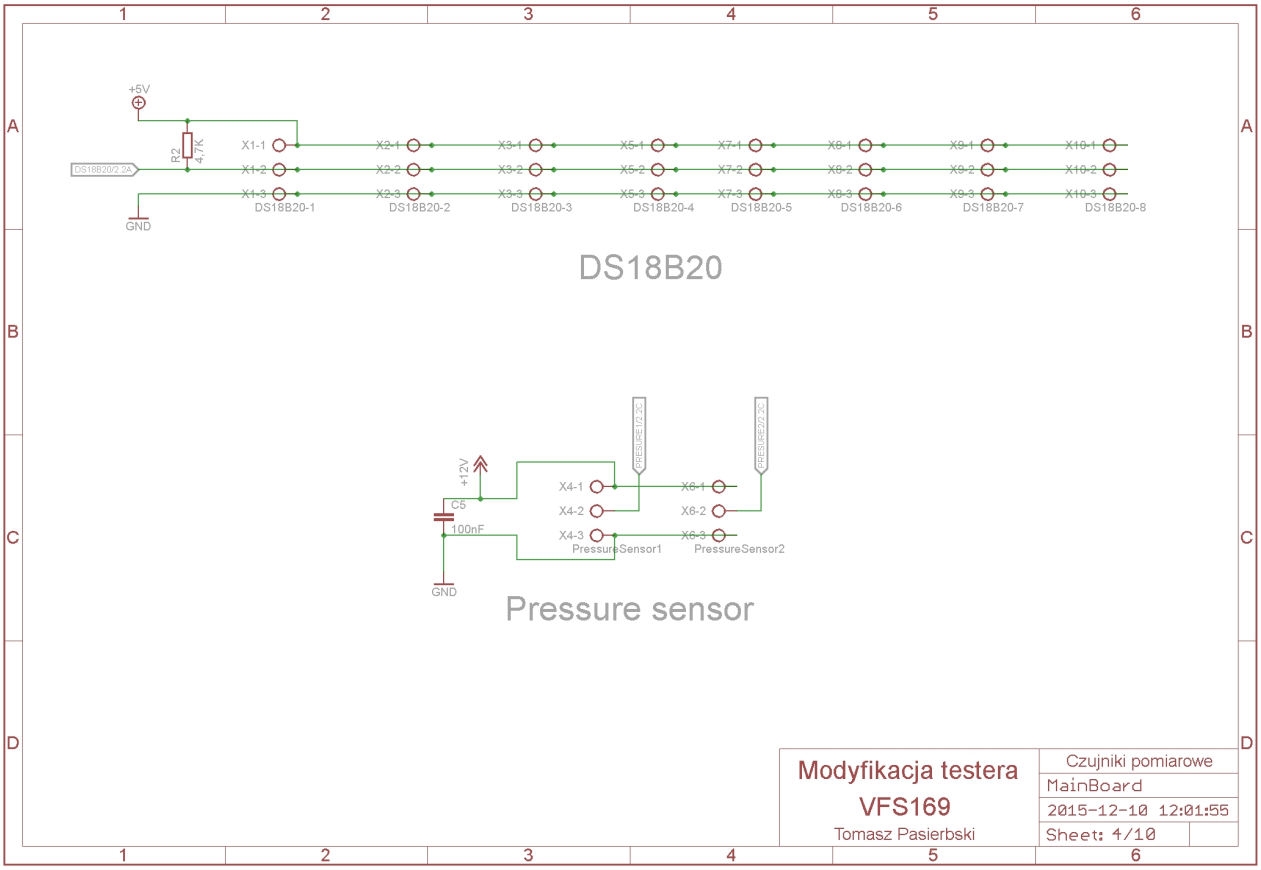
Modyfikacja testera VFS169 Tomasz Pasierbski	MainBoard 2015-12-10 12:01:55 Sheet: 10/10

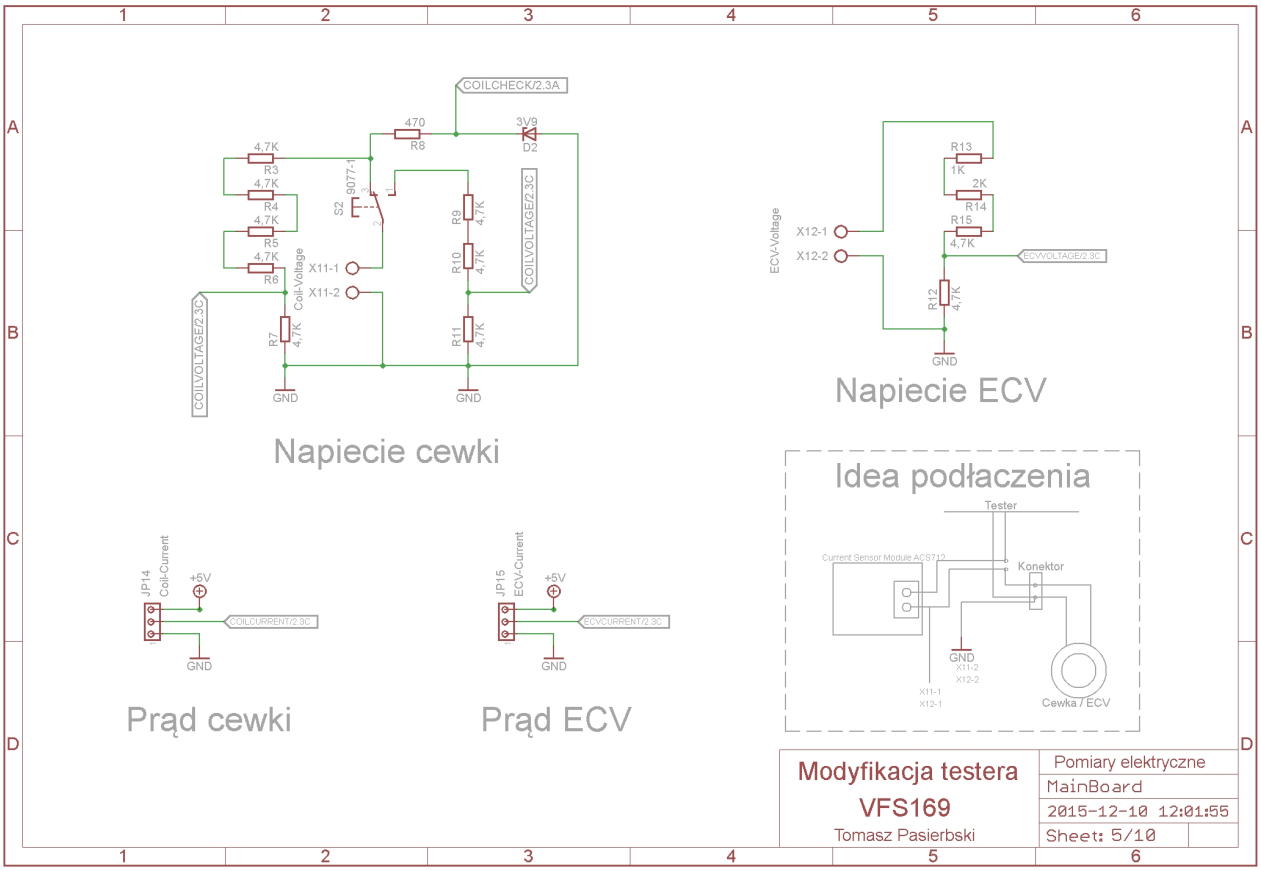


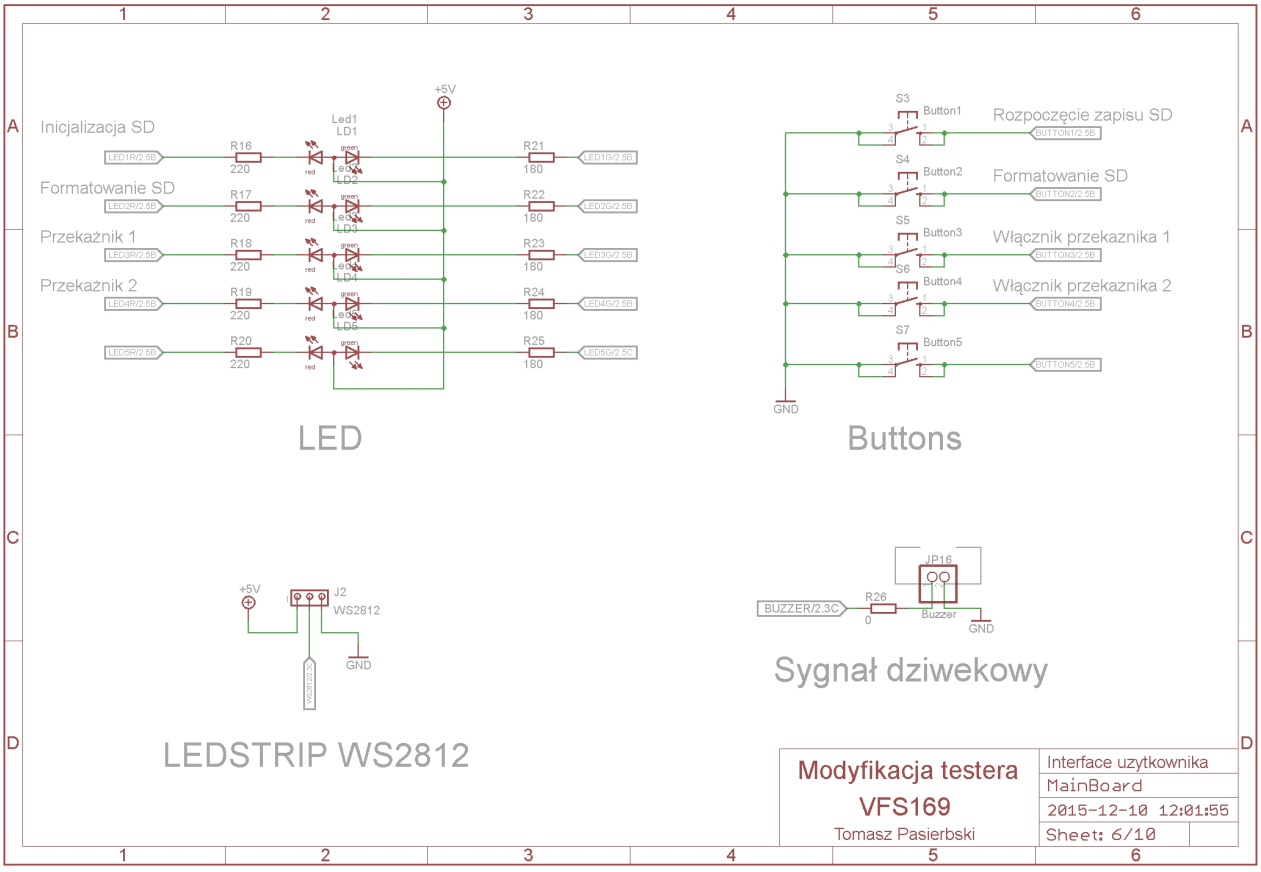
Modyfikacja testera VFS169 Tomasz Pasierbski	Sensory MainBoard 2015-12-10 12:01:55 Sheet: 1/10

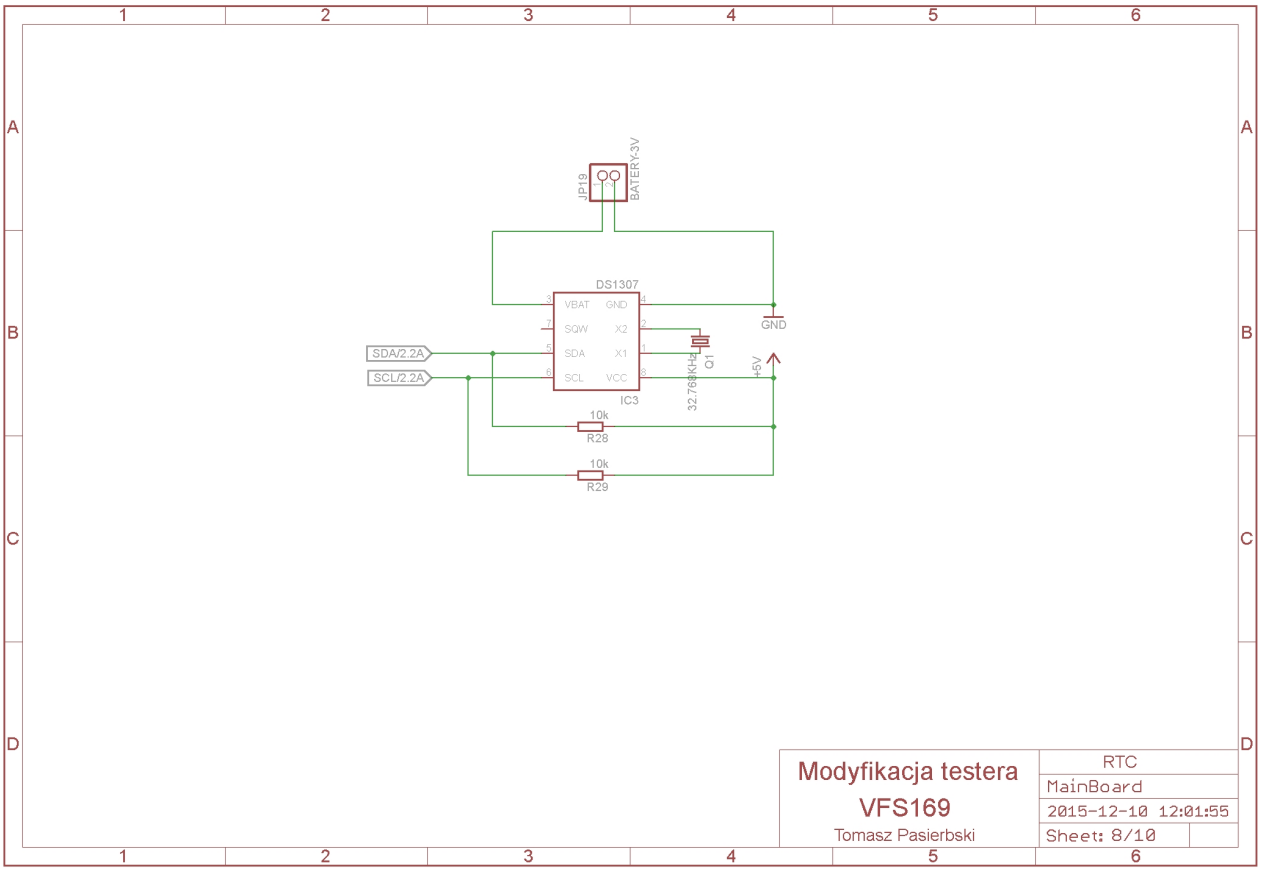

JP13
10x1F-H8.5
COMMUNICATION
8x1F-H8.5
8x1F-H8.5
GND
XIO
+5V
18x2F-H8.5
GND
+3V3
+5V
8x1F-H8.5
GND
8x1F-H8.5
ADCH
Modyfikacja testera
VFS169
Tomasz Pasierbski
Arduino
MainBoard
2015-12-10 12:01:55
Sheet: 2/10

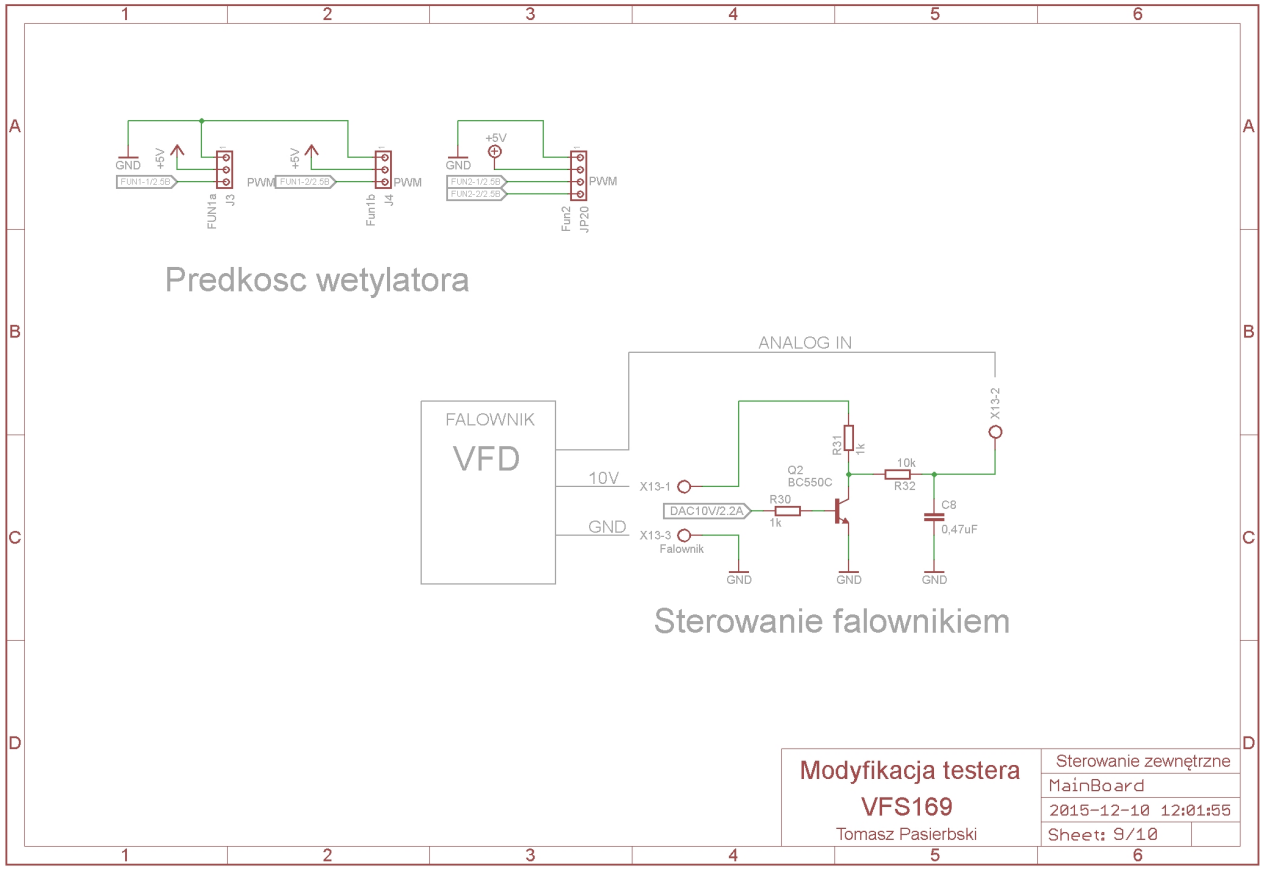

Włącznik zasilania
24V
GND
J1
DCJ0202
IC4
7812T
+12V
X14-1
X14-2
PowerBoard
D1
1N4004
WIREPAD3,81/1.1
PAD1(12V)
WIREPAD3,81/1.1
PAD1
C1
330nF
IC1
7805TV
LED1
LED pomarańczowa
R1
220
500mA
F1
C2
330nF
C3
47mF
C4
100nF
+5V
GND
Modyfikacja testera
VFS169
Tomasz Pasierbski
Zasilanie
MainBoard
2015-12-10 12:01:55
Sheet: 3/10


+5V
R2
4.7K
DS18B20/2.2A
GND
X1-1
X1-2
X1-3
DS18B20-1
X2-1
X2-2
X2-3
DS18B20-2
X3-1
X3-2
X3-3
DS18B20-3
X5-1
X5-2
X5-3
DS18B20-4
X7-1
X7-2
X7-3
DS18B20-5
X8-1
X8-2
X8-3
DS18B20-6
X9-1
X9-2
X9-3
DS18B20-7
X10-1
X10-2
X10-3
DS18B20-8
DS18B20
+12V
C5
100nF
GND
X4-1
X4-2
X4-3
PressureSensor1
X6-1
X6-2
X6-3
PressureSensor2
Pressure sensor
Modyfikacja testera
VFS169
Tomasz Pasierbski
Czujniki pomiarowe
MainBoard
2015-12-10 12:01:55
Sheet: 4/10

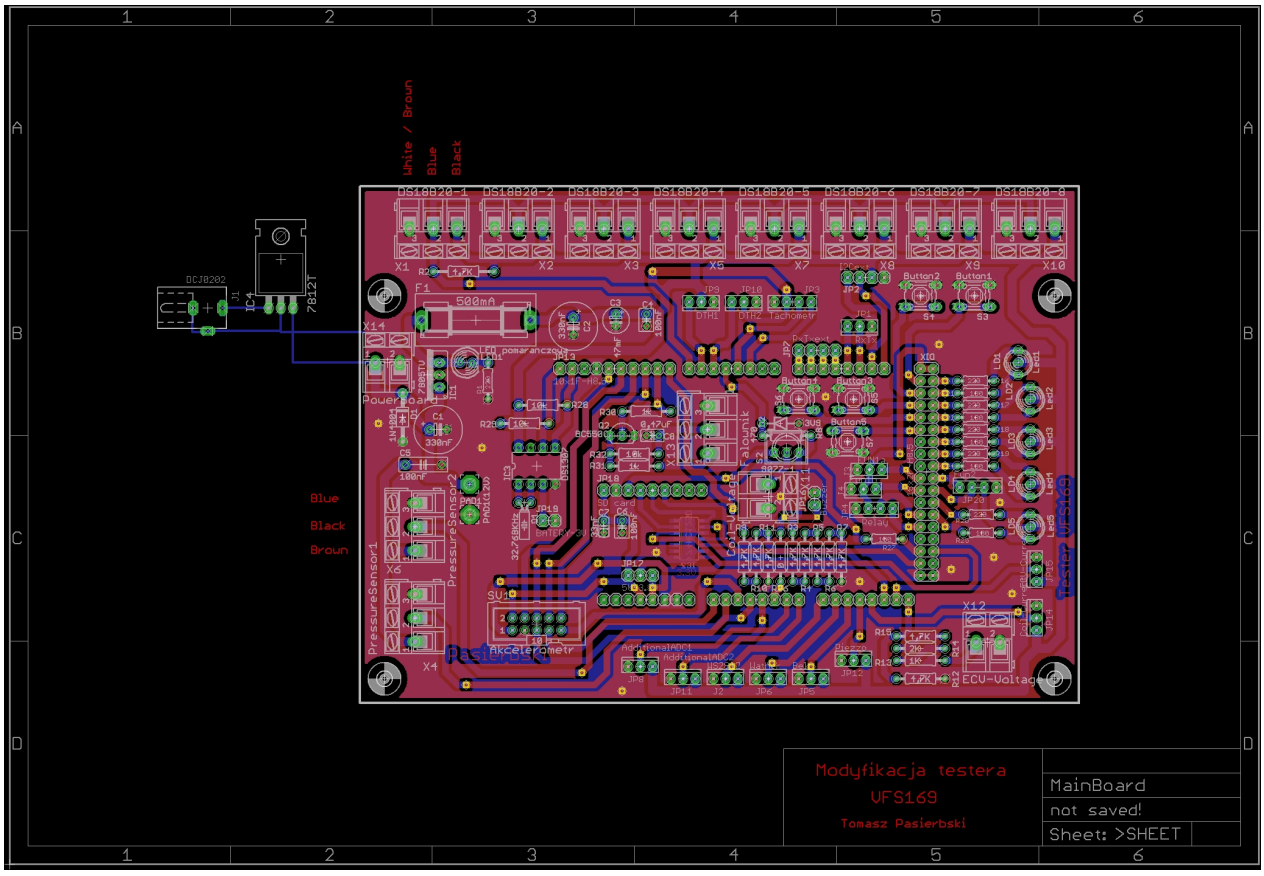

COILCHECK/2.3A
470
R8
3V9
D2
4.7K
R3
4.7K
R4
4.7K
R5
4.7K
R6
S2
9077-1
X11-1
X11-2
Coil-Voltage
R7
4.7K
R9
4.7K
R10
4.7K
R11
4.7K
COILVOLTAGE/2.3C
GND
Napiecie cewki
ECV-Voltage
X12-1
X12-2
R13
1K
R14
2K
R15
4.7K
R12
4.7K
ECVVOLTAGE/2.3C
Napiecie ECV
JP14
Coil-Current
+5V
COILCURRENT/2.3C
Prąd cewki
JP15
ECV-Current
ECVCURRENT/2.3C
Prąd ECV
Idea podłaczenia
Tester
Current Sensor Module ACS712
Konektor
Cewka / ECV
Modyfikacja testera
VFS169
Tomasz Pasierbski
Pomiary elektryczne
MainBoard
2015-12-10 12:01:55
Sheet: 5/10

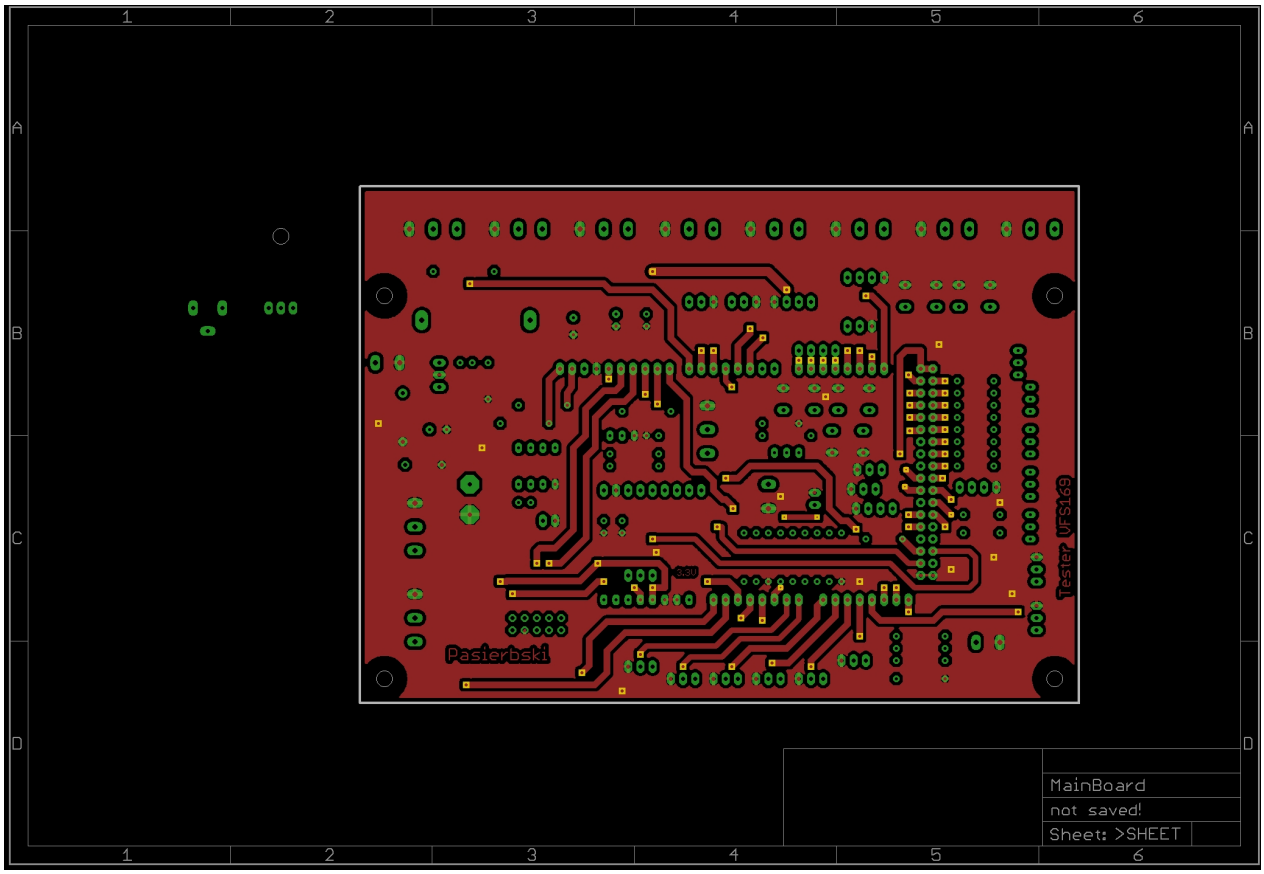

Inicjalizacja SD
Formatowanie SD
Przekaźnik 1
Przekaźnik 2
LED
Button1
Rozpoczęcie zapisu SD
Button2
Formatowanie SD
Button3
Włącznik przekaznika 1
Button4
Włącznik przekaznika 2
Button5
Buttons
LEDSTRIP WS2812
Sygnał dziwekowy
Modyfikacja testera
VFS169
Tomasz Pasierbski
Interface uzytkownika
MainBoard
2015-12-10 12:01:55
Sheet: 6/10

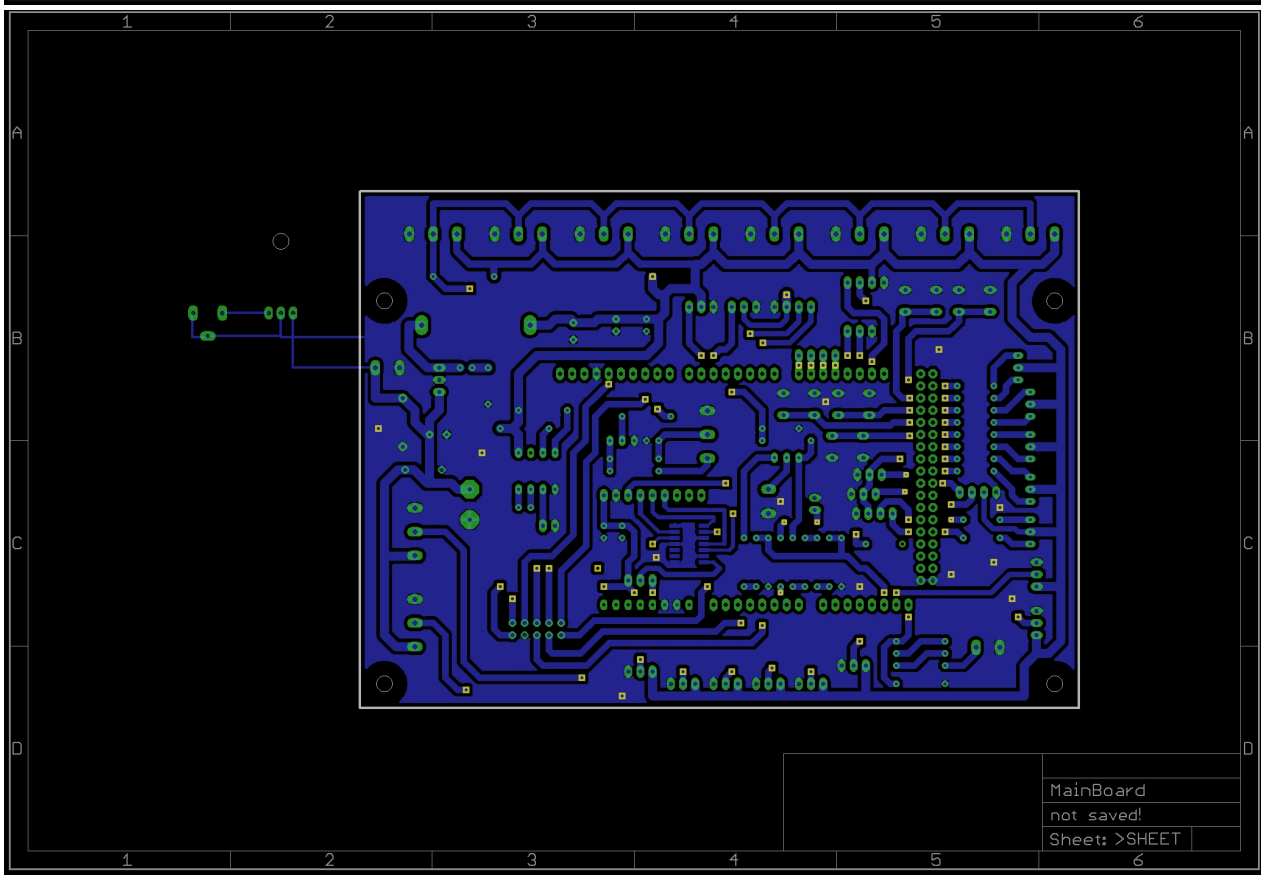

DS1307
IC3
BATERY-3V
SDA/2.2A
SCL/2.2A
32.768KHz
Q1
10k
R28
10k
R29
Modyfikacja testera
VFS169
Tomasz Pasierbski
RTC
MainBoard
2015-12-10 12:01:55
Sheet: 8/10

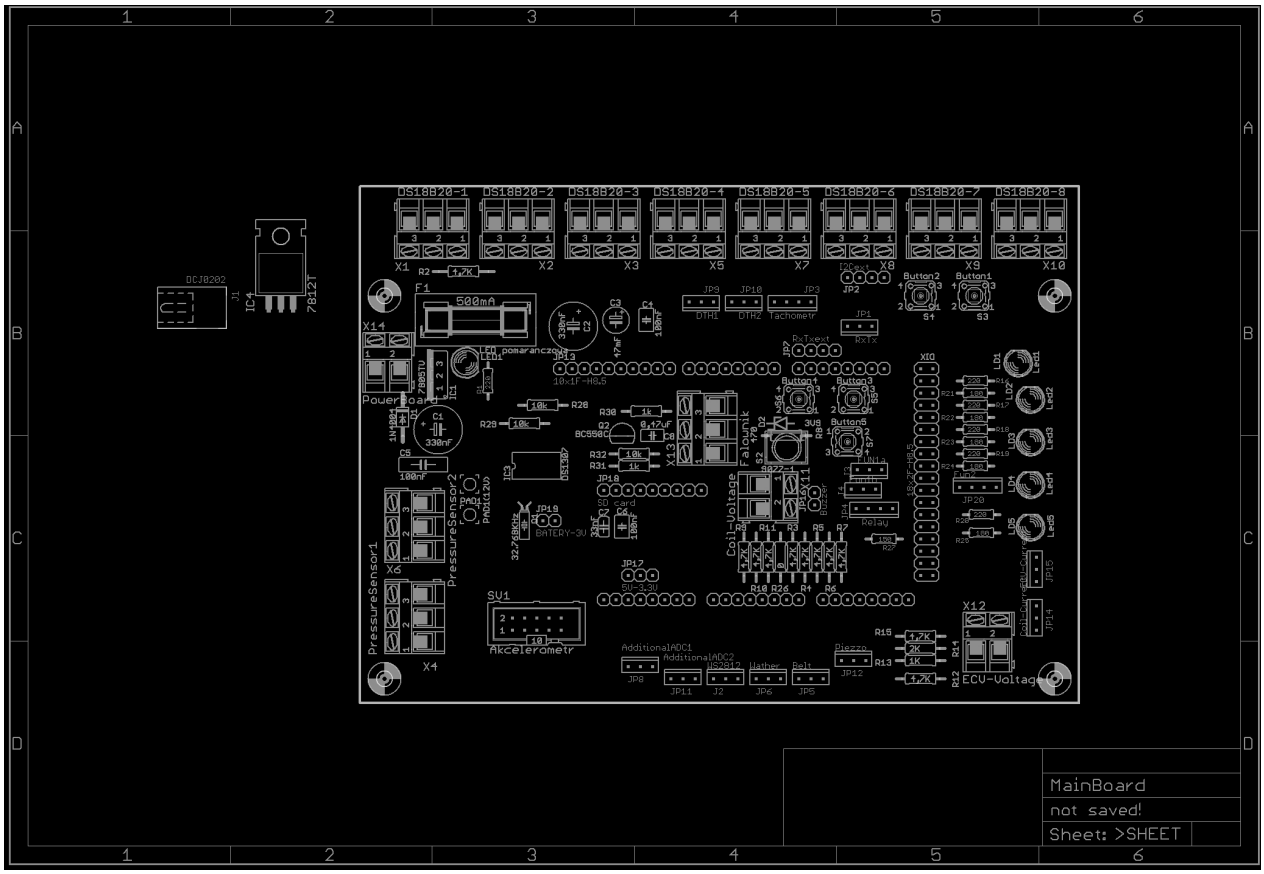


PCB




Pasierbski
Tester VFS169
MainBoard
not saved!
Sheet: >SHEET


MainBoard
not saved!
Sheet: >SHEET



Kod programu

1. PC
2. Elektronika

PC

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO.Ports;                  //uzycie derektywy Ports
using System.Xml;                       //uzycie dyrektyw XML
using System.IO;                        //uzycie dyrektyw IO

namespace Tester_VFS169
{
    public partial class FormMain : Form
    {



        public static double[] zmienne;
        /// <value>
```

```
        /// Zmienne do obsługi portu COM
        /// </value>
        int InttializeLenght = 0;
        int baudRate;

        /// <value>
        /// Tablica zmiennych przychodzacych z SerialPort
        /// </value>
        string[] ZmienneCOM = new string[1];

        /// <value>
        /// Zmienne dla stopera
        /// </value>
        private int t = 0;          //  sekundy
        private int s1 = 0;         //  minuty
        private int s2 = 0;         //  godiny
        public static bool dzialanie = false;

        /// <value>
        /// Obliczenie ilości cykli cewki
        /// </value>
        int iloscCykli = 0;
        int iloscCykliLast = 0;

        /// <value>
        /// Deklaracja folderów i plików
        /// </value>
        public static string GlobalFolder;
        public static string TestSettings;// = "Settings.xml";
        public static string PlikRaportu;// = "Report.txt";

        private static bool TerminalStop = false;

        public FormMain()
        {
            InitializeComponent();
        }

        /// <summary>
        /// Zamykanie aplikacji z wykorzystaniem potwierdzenia.
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void Form1_FormClosing(object sender, FormClosingEventArgs e)
        {

#if DEBUG

#else
                    if (MessageBox.Show("Continue this application", "Close",
MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.Yes)
                    {
                        e.Cancel = true;
                    }
#endif


            //zamkniecie portu COM
            COM.Close();

        }

        private void exitToolStripMenuItem1_Click(object sender, EventArgs e)
        {
```

```
            Application.Exit();
        }


        /// <summary>
        /// Obsługa portu COM
        /// Automatyczna detekcja podłaczonych portów COM
        /// Podłaczenie do wybranych ustawien
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void timerCOM_Tick(object sender, EventArgs e)
        {
            string[] ports = SerialPort.GetPortNames();

            try
            {
                if (InttializeLenght != ports.Length)
                {
                    InttializeLenght = ports.Length;

                    StripCOM.Items.Clear();
                    for (int j = 0; j < InttializeLenght; j++)
                    {
                        StripCOM.Items.Add(ports[j]);

                    }
                    StripCOM.Text = ports[0];
                    StripBaud.SelectedIndex = 0;
                    StripProgressCOM.Value = 50;
                    StripConnect.Enabled = true;
                }
            }

            catch
            {
                MessageBox.Show("COM Ports not connected!!!\n" + "Check your device",
"COM Port", MessageBoxButtons.OK, MessageBoxIcon.Error);
                StripCOM.Text = "";
                StripBaud.Text = "";
                StripProgressCOM.Value = 0;

            }
        }


        /// <summary>
        /// Połaczenie sie do ustawionego SERILA PORT
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void StripConnect_Click(object sender, EventArgs e)
        {

            try
            {
                if (StripConnect.Text == "Connect!!!")
                {
                    baudRate = Convert.ToInt32(StripBaud.Text);
                    COM.PortName = StripCOM.Text;
                    COM.BaudRate = baudRate;

                    COM.Open();
                    timerCOM.Stop();
```

```
                StripProgressCOM.Value = 100;
                StripConnect.Text = "Disconnect";
            }

            else
            {
                COM.Close();
                StripProgressCOM.Value = 50;
                StripConnect.Text = "Connect!!!";
                StripStatusDane.Text = "Dane???";
                timerCOM.Start();
            }
        }



        catch
        {
            MessageBox.Show("COM Ports not connected!!!\n" + "Check your device",
"COM Port", MessageBoxButtons.OK, MessageBoxIcon.Error);
            StripCOM.Text = "";
            StripBaud.Text = "";
            StripProgressCOM.Value = 0;
        }

    }



    /// <summary>
    /// Metoda wpisywania wartosci do TEXT BOX
    /// </summary>
    /// <example>
    /// SetTextBox(textBox1, zmienne[3].ToString());
    /// textBox1 ===== nazwa kontrolki
    /// zmienne[3].ToString() ===== wartosc przekazywana
    /// </example>
    /// <param name="tb"></param>
    /// <param name="value"></param>
    public void SetTextBox(TextBox tb, string value)
    {
        if (InvokeRequired)
        {
            this.Invoke(new Action<TextBox, string>(SetTextBox), new object[] { tb,
value });
            return;
        }

        tb.Text = value;
    }


    /// <summary>
    /// Metoda wpisywania wartosci do Label
    /// </summary>
    /// <example>
    /// SetTextLabel(label1, zmienne[3].ToString());
    /// textBox1 ===== nazwa kontrolki
    /// zmienne[3].ToString() ===== wartosc przekazywana
    /// </example>
    /// <param name="lab"></param>
    /// <param name="value"></param>
    public void SetTextLabel(Label lab, string value)
```

```
        {
            if (InvokeRequired)
            {
                this.Invoke(new Action<Label, string>(SetTextLabel), new object[] { lab,
value });
                return;
            }

            lab.Text = value;

        }

        /// <summary>
        /// Metoda wpisywania wartosci do TERMINALA
        /// </summary>
        /// <param name="tb"></param>
        /// <param name="value"></param>
        public void SetTextBoxLines(TextBox tbl, string value)
        {
            if (InvokeRequired)
            {
                this.Invoke(new Action<TextBox, string>(SetTextBoxLines), new object[]
{ tbl, value });
                return;
            }

            tbl.AppendText(value+Environment.NewLine);
        }


        /// <summary>
        /// Wyswietlanie danych przychodzacych z SerialPort na kontrolki
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void COM_DataReceived(object sender, SerialDataReceivedEventArgs e)
        {
            if (COM.IsOpen)
            {
                try
                {
                    ///<remarks>
                    ///Rozbicie tekstu przychodzącegoz SerialPort na tablice zmiennych
                    ///</remarks>
                    ZmienneCOM = COM.ReadLine().Replace('.',
',').Split(';');          //19 wartosci



                    ///<remarks>
                    ///Wpisywanie zmiennych do Terminala
                    ///</remarks>
                    if (tERMINALToolStripMenuItem.Checked == true && TerminalStop ==
false)
                    {
                    SetTextBoxLines(Terminal, ZmienneCOM[0] + " ; " + ZmienneCOM[1] + " ;
" + ZmienneCOM[2] + " ; " + ZmienneCOM[3]
                    + " ; " + ZmienneCOM[4] + " ; " + ZmienneCOM[5] + " ; " +
ZmienneCOM[6] + " ; " + ZmienneCOM[7] + " ; " + ZmienneCOM[8]);
                    }



                    //double[]
```

```
                    zmienne = Array.ConvertAll(ZmienneCOM, Double.Parse);


                    ///<remarks>
                    ///Zapisywanie zanych do pliku
                    ///</remarks>
                    WriteDataLog(PlikRaportu, DateTime.Now.ToShortDateString() + ";" +
DateTime.Now.ToLongTimeString()
                                + ";" + String.Format("{0:D3}:{1:D2}:{2:d2}", s2, s1, t)
                                + ";" + zmienne[0].ToString() + ";" + zmienne[1].ToString()
+ ";" + zmienne[2].ToString()
                                + ";" + zmienne[3].ToString() + ";" + zmienne[5].ToString()
+ ";" + zmienne[4].ToString() + ";" + zmienne[6].ToString()
                                + ";" + zmienne[7].ToString() + ";" + zmienne[8].ToString()
+ ";" + zmienne[9].ToString() + ";" + zmienne[10].ToString()
                                + ";" + zmienne[11].ToString() + ";" + zmienne[12].ToString()
+ ";" + zmienne[13].ToString() + ";" + zmienne[14].ToString()
                                + ";" + zmienne[15].ToString() + ";" + zmienne[16].ToString()
+ ";" + zmienne[17].ToString() + ";" + zmienne[18].ToString()
                                + ";" + zmienne[19].ToString());



                    StripStatusDane.Text = "OK";

                    /*
                    0       //Serial.print  (Tsuc);
                    1       //Serial.print  (Tdis);
                    2       //Serial.print  (TcondenserIn);
                    3       //Serial.print  (TcondenserOut);
                    4       //Serial.print  (TevapuratorIn);
                    5       //Serial.print  (TevapuratorOut);
                    6       //Serial.print  (Thotbox);
                    7       //Serial.print  (Tcompressor);
                    8       //Serial.print  (TairIn);
                    9       //Serial.print  (HairIn);
                    10      //Serial.print  (TairOut);
                    11      //Serial.print  (HairOut);
                    12      //Serial.print  (Psuc);
                    13      //Serial.print  (Pdis);
                    14      //Serial.print  (COILvoltage);
                    15      //Serial.print  (readCurrent(COILcurrent),3);
                    16      //Serial.print  (ECVvoltage);
                    17      //Serial.print  (readCurrent(ECVcurrent),3);
                    18      //Serial.print  (Cewka);
                    19      //Serial.print  (rpm);
                    */



                    ///<remarks>
                    ///Wysyłanie danych do GRAPH CHART
                    ///</remarks>
                    WFPressureSuction.PlotYAppend(zmienne[12]);
                    WFPressureDischarge.PlotYAppend(zmienne[13]);

                    WFTempCompressor.PlotYAppend(zmienne[7]);
                    WFTempCondenserIn.PlotYAppend(zmienne[2]);
                    WFTempCondenserOut.PlotYAppend(zmienne[3]);
                    WFTempDischarge.PlotYAppend(zmienne[1]);
                    WFTempEvapIn.PlotYAppend(zmienne[5]);
                    WFTempEvapOut.PlotYAppend(zmienne[4]);
                    WFTempHotbox.PlotYAppend(zmienne[6]);
```

```
WFTempSuction.PlotYAppend(zmienne[0]);



///<remarks>
///Obsługa kontrolek z podziałem na funkcje
///</remarks>
///


//Discharge
gaugeDischarge.Value = zmienne[13];
SetTextLabel(gaugeDischargeLAB, zmienne[13].ToString());
SetTextBox(TBACDischargePress, zmienne[13].ToString() + " BarG");

thermometerDischarge.Value = zmienne[1];
SetTextBox(thermometerDischargeTB, zmienne[1].ToString() + " °C");
SetTextBox(TBACDischargeTemp, zmienne[1].ToString() + " °C");


//Suction
gaugeSuction.Value = zmienne[12];
SetTextLabel(gaugeSuctionLAB, zmienne[12].ToString());
SetTextBox(TBACSuctionPress, zmienne[12].ToString() + " BarG");

thermometerSuction.Value = zmienne[0];
SetTextBox(thermometerSuctionTB, zmienne[0].ToString() + " °C");
SetTextBox(TBACSuctionTemp, zmienne[0].ToString() + " °C");

//Condenser
thermometerCondIn.Value = zmienne[2];
SetTextBox(thermometerCondInTB, zmienne[2].ToString() + " °C");
SetTextBox(TBACCondenserINTemp, zmienne[2].ToString() + " °C");
thermometerCondOut.Value = zmienne[3];
SetTextBox(thermometerCondOutTB, zmienne[3].ToString() + " °C");
SetTextBox(TBACCondenserOUTTemp, zmienne[3].ToString() + " °C");

//Compressor
thermometerCompressor.Value = zmienne[7];
SetTextBox(thermometerCompressorTB, zmienne[7].ToString() + " °C");
SetTextBox(TBACCompressorTemp, zmienne[7].ToString() + " °C");

if (zmienne[7] >= Convert.ToDouble(textBoxCompressorLimit.Text))
{
    ledCompressor.Value = true;
}
else
{
    ledCompressor.Value = false;
}


//Evapurator
SetTextBox(thermometerEvapInTB, zmienne[5].ToString() + " °C");
SetTextBox(TBACEvapINTemp, zmienne[5].ToString() + " °C");
SetTextBox(thermometerEvapOutTB, zmienne[4].ToString() + " °C");
SetTextBox(TBACEvapOUTTemp, zmienne[4].ToString() + " °C");

//Air in Evapurator
SetTextBox(thermometerAirInTB, zmienne[8].ToString() + " °C");
SetTextBox(TBACAirINTemp, zmienne[8].ToString() + " °C");
SetTextBox(thermometerAirOutTB, zmienne[10].ToString() + " °C");
SetTextBox(TBACAirOutTemp, zmienne[10].ToString() + " °C");
```

```
                    SetTextBox(HumidityAirInTB, zmienne[9].ToString() + " %");
                    SetTextBox(HumidityAirOutTB, zmienne[11].ToString() + " %");


                    //Hotbox
                    thermometerHotbox.Value = zmienne[6];
                    SetTextBox(thermometerHotboxTB, zmienne[6].ToString() + " °C");
                    SetTextBox(TBACHotboxTemp, zmienne[6].ToString() + " °C");


                    if (zmienne[6] >= (Convert.ToDouble(textBoxHBmax.Text)-5) &
(zmienne[6] <= Convert.ToDouble(textBoxHBmax.Text)+5))
                    {
                        ledHotbox.Value = true;
                    }
                    else
                    {
                        ledHotbox.Value = false;
                    }


                    //Coil

                    meterCoilVoltage.Value = zmienne[14];
                    meterCoilCurrent.Value = zmienne[15];

                    if (zmienne[18] == 1)
                    {
                        ledCoil.Value = true;
                    }
                    else
                    {
                        ledCoil.Value = false;
                    }

                    if (zmienne[18] == 1 & iloscCykliLast == 0 & dzialanie == true)
                    {
                        iloscCykli++;
                        iloscCykliLast = 1;
                    }
                    if (zmienne[18] == 0 & iloscCykliLast == 1)
                    {
                        iloscCykliLast = 0;
                    }

                    SetTextBox(textBoxCoilCycles, iloscCykli.ToString());


                    //ECV
                    SetTextBox(textBoxECVvoltage, zmienne[16].ToString() + " V");
                    SetTextBox(textBoxECVcurrent, zmienne[17].ToString() + " A");

                    //RPM
                    meterRPM.Value = zmienne[19];


                }
                catch
                {
                    StripStatusDane.Text = "Brak danych";
                }

            }
```

```
        }



        private void Form1_Load(object sender, EventArgs e)
        {
            // TODO: Ten wiersz kodu wczytuje dane do tabeli
'databaseDataSet.ShotTestInfo' . Możesz go przenieść lub usunąć.
            this.shotTestInfoTableAdapter.Fill(this.databaseDataSet.ShotTestInfo);
            // TODO: Ten wiersz kodu wczytuje dane do tabeli
'databaseDataSet.TestDescriptionInput' . Możesz go przenieść lub usunąć.
            this.testDescriptionInputTableAdapter.Fill(this.databaseDataSet.TestDescripti
onInput);
            // TODO: Ten wiersz kodu wczytuje dane do tabeli
'databaseDataSet.TestDescriptionSetup' . Możesz go przenieść lub usunąć.
            this.testDescriptionSetupTableAdapter.Fill(this.databaseDataSet.TestDescripti
onSetup);
            // TODO: Ten wiersz kodu wczytuje dane do tabeli
'databaseDataSet.TestParameters' . Możesz go przenieść lub usunąć.
            this.testParametersTableAdapter.Fill(this.databaseDataSet.TestParameters);


#if DEBUG
            timerStoper.Interval = 10;        //Ustwienie timera dla stopera na 1 sekude

            buttonDisconnect.Visible = true;
#else
            timerStoper.Interval = 1000;        //Ustwienie timera dla stopera na 1
sekude
            SplashScreen SS = new SplashScreen();
            SS.ShowDialog();                    //Wyswietlenie splash scrren
#endif


        }




        /// <summary>
        /// Stoper działania testera
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void timerStoper_Tick(object sender, EventArgs e)
        {
            t++;

            // Obliczenia dla minut
            if (t >= 60)
            {
                t = 0;
                s1++;
            }


            // Obliczenia dla godzin
            if (s1 >= 60)
            {
                s1 = 0;
                s2++;
```

```
            }
            //Wyswietlanie czasu działania testu
            labelStoper.Text = String.Format("{0:D3}:{1:D2}:{2:d2}", s2, s1, t);
        }



        /// <summary>
        /// Obsługa przycisku "START" / "PAUSE"
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void buttonStart_Click(object sender, EventArgs e)
        {
            if (dzialanie == false)
            {
                timerStoper.Enabled = true;
                buttonStart.Text = "PAUSE";
                dzialanie = true;
                ledStoper.Value = true;

            }
            else
            {
                timerStoper.Enabled = false;
                buttonStart.Text = "START";
                dzialanie = false;
                ledStoper.Value = false;


            }
        }


        /// <summary>
        /// Obsługa przycisku "RESET"
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void buttonReset_Click(object sender, EventArgs e)
        {

#if DEBUG
            {
                timerStoper.Enabled = false;
                s2 = 0; s1 = 0; t = 0;
                labelStoper.Text = "000:00:00";
                buttonStart.Text = "START";
                dzialanie = false;
                ledStoper.Value = false;
                iloscCykli = 0;
                iloscCykliLast = 0;
            }

#else
                    if (MessageBox.Show("Continue this test", "Reset",
MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.No)
                    {
                        timerStoper.Enabled = false;
                        s2 = 0; s1 = 0; t = 0;
                        labelStoper.Text = "000:00:00";
                        buttonStart.Text = "START";
                        dzialanie = false;
```

```
                    ledStoper.Value = false;
                    iloscCykli = 0;
                    iloscCykliLast = 0;
                }
#endif

        }


        /// <summary>
        /// Wyswietlanie okna dialogowego "About"
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void aboutToolStripMenuItem_Click(object sender, EventArgs e)
        {
            AboutBox AB = new AboutBox();
            AB.ShowDialog();
        }


        /// <summary>
        /// Wyswietlanie okna dialogowego "AC loop parameters"
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void aCLoopToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (aCLoopToolStripMenuItem.Checked == true)
            {
                BoxDischrge.Visible = true;
                BoxSuction.Visible = true;
                BoxCompressor.Visible = true;
                BoxEvapurator.Visible = true;
                BoxCondenser.Visible = true;
                BoxECV.Visible = true;
                BoxHotbox.Visible = true;
                BoxChart.Visible = false;
                BoxInformation.Visible = true;
                BoxAC.Visible = false;


                aCLoopToolStripMenuItem.Checked = false;
                chartToolStripMenuItem.Checked = false;
            }

            else
            {
                BoxDischrge.Visible = false;
                BoxSuction.Visible = false;
                BoxCompressor.Visible = false;
                BoxEvapurator.Visible = false;
                BoxCondenser.Visible = false;
                BoxECV.Visible = false;
                BoxHotbox.Visible = false;
                BoxChart.Visible = false;
                BoxInformation.Visible = true;
                BoxAC.Visible = true;

                aCLoopToolStripMenuItem.Checked = true;
                chartToolStripMenuItem.Checked = false;
            }

        }
```

```
/// <summary>
/// Wyswietlanie Splash Screen
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void splashScreenToolStripMenuItem_Click(object sender, EventArgs e)
{
    SplashScreen SS = new SplashScreen();
    SS.ShowDialog();                    //Wyswietlenie splash scrren
}


/// <summary>
/// Wyswietlanie okna dialogowego "Chart"
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void pressuresToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (chartToolStripMenuItem.Checked == true)
    {
        BoxDischrge.Visible = true;
        BoxSuction.Visible = true;
        BoxCompressor.Visible = true;
        BoxEvapurator.Visible = true;
        BoxCondenser.Visible = true;
        BoxECV.Visible = true;
        BoxHotbox.Visible = true;
        BoxInformation.Visible = true;
        BoxChart.Visible = false;

        BoxAC.Visible = false;


        aCLoopToolStripMenuItem.Checked = false;
        chartToolStripMenuItem.Checked = false;
    }

    else
    {
        BoxDischrge.Visible = false;
        BoxSuction.Visible = false;
        BoxCompressor.Visible = false;
        BoxEvapurator.Visible = false;
        BoxCondenser.Visible = false;
        BoxECV.Visible = false;
        BoxHotbox.Visible = false;
        BoxInformation.Visible = false;
        BoxChart.Visible = true;

        BoxAC.Visible = false;

        aCLoopToolStripMenuItem.Checked = false;
        chartToolStripMenuItem.Checked = true;
    }



    WFPressureSuction.Visible = true;
    WFPressureDischarge.Visible = true;
    WFTempCompressor.Visible = false;
    WFTempCondenserIn.Visible = false;
```

```
            WFTempCondenserOut.Visible = false;
            WFTempDischarge.Visible = true;
            WFTempEvapIn.Visible = false;
            WFTempEvapOut.Visible = false;
            WFTempHotbox.Visible = false;
            WFTempSuction.Visible = true;
            PlotBoxTempSuction.Checked = true;
            PlotBoxTempDischarge.Checked = true;
        }

        /// <summary>
        /// Zapisywanych danych do XML przy zamknieciu programu
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void FormMain_FormClosed(object sender, FormClosedEventArgs e)
        {

#if DEBUG

#else


            XmlTextWriter xml = null;

            try
            {
                //tworzenie pliku
                xml = new XmlTextWriter(TestSettings, System.Text.Encoding.UTF8);
                xml.Formatting = Formatting.Indented;
                xml.WriteStartDocument(true);

                xml.WriteComment("Last test parameters before application close");

                //<Settings global>>
                xml.WriteStartElement("Settings");

                //<test time>
                xml.WriteStartElement("time");
                xml.WriteElementString("Hour", s2.ToString());
                xml.WriteElementString("Minute", s1.ToString());
                xml.WriteElementString("Sekunds", t.ToString());
                xml.WriteEndElement();

                xml.WriteStartElement("CoilCyclesTotal");
                xml.WriteElementString("Cykles", iloscCykli.ToString());
                xml.WriteEndElement();

                xml.WriteStartElement("DeafaltSettingsLocation");
                xml.WriteElementString("GlobalLocation", GlobalFolder.ToString());
                xml.WriteElementString("SettingsLocation", TestSettings.ToString());
                xml.WriteElementString("ReportLocation", PlikRaportu.ToString());
                xml.WriteEndElement();

                //xml.WriteStartElement("TestParameters");
                //xml.WriteElementString("SelectedTest", TestSelected.SelectedText);
                //xml.WriteEndElement();

                xml.WriteEndElement();

                COM.Close();
            }

            catch (Exception exc)
```

```
            {
                MessageBox.Show("Error writing settings for files (" + exc.Message +
")");

            }

            finally
            {
                xml.Close();
            }
#endif
        }



        /// <summary>
        /// Oczytywanie danych do XML dla opcji Test Description -> Continu test
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void continueTestToolStripMenuItem_Click(object sender, EventArgs e)
        {

            if (openFileSetting.ShowDialog() == DialogResult.OK)
            {
                TestSettings = openFileSetting.FileName;
            }

            XmlTextReader xml = null;

            try
            {
                xml = new XmlTextReader(TestSettings);

                xml.ReadStartElement("Settings");

                xml.Read();

                //<time>
                xml.ReadStartElement("time");
                s2 = int.Parse(xml.ReadElementString("Hour"));
                s1 = int.Parse(xml.ReadElementString("Minute"));
                t = int.Parse(xml.ReadElementString("Sekunds"));
                xml.ReadEndElement();

                xml.ReadStartElement("CoilCyclesTotal");
                iloscCykli = int.Parse(xml.ReadElementString("Cykles"));
                xml.ReadEndElement();


                xml.ReadStartElement("DeafaltSettingsLocation");
                GlobalFolder = xml.ReadElementString("GlobalLocation");
                TestSettings = xml.ReadElementString("SettingsLocation");
                PlikRaportu = xml.ReadElementString("ReportLocation");

                xml.ReadEndElement();


                xml.ReadEndElement();

                //Wyswietlanie czasu działania testu
                labelStoper.Text = String.Format("{0:D3}:{1:D2}:{2:d2}", s2, s1, t);
                textBoxCoilCycles.Text = iloscCykli.ToString();
```

```
                //Wyswetlanie Informcji o folderach
                string RepS = TestSettings.Replace("/", @"\");
                string RepR = PlikRaportu.Replace("/", @"\");
                textBoxSettingsInfo.Text = RepS;
                textBoxReportInfo.Text = RepR;
            }

            catch (Exception exc)
            {
                MessageBox.Show("Błąd w odczycie pliku (" + exc.Message + ")");
            }
            finally
            {
                xml.Close();
            }

        }



        /// <summary>
        /// Szybkie za zamykanie połaczenia - TYLKO DEBUG
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void button2_Click(object sender, EventArgs e)
        {
            COM.Close();
            StripProgressCOM.Value = 50;
            StripConnect.Text = "Connect!!!";
            StripStatusDane.Text = "Dane???";
            timerCOM.Start();
        }


        /// <summary>
        /// Inicjalizacja pliku raportu
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void setupToolStripMenuItem_Click(object sender, EventArgs e)
        {

            if (SaveFileReport.ShowDialog() == DialogResult.OK)
            {
                PlikRaportu = SaveFileReport.FileName;
                textBoxReportInfo.Text = PlikRaportu;


                if (!File.Exists(@PlikRaportu))
                {
                    using (FileStream fileStream = new FileStream(PlikRaportu,
FileMode.Append, FileAccess.Write, FileShare.None))
                    {
                        using (StreamWriter streamWriter = new StreamWriter(fileStream))
                        {
                            streamWriter.WriteLine("DURABILITY TESTER VFS169");
                            streamWriter.WriteLine("Rozpoczęcie Testu: " +
DateTime.Now.ToLongDateString() + " " + DateTime.Now.ToLongTimeString());
                            streamWriter.WriteLine("DataNow;TomeNow;TimeOfTest;Tsuc;TdisT
condenserIn;TcondenserOut;TevapuratorIn;TevapuratorOut;Thotbox;Tcompressor;TairIn;HairIn;
TairOut;HairOut;Psuc;Pdis;COILvoltage;COILcurrent;ECVvoltage;ECVcurrent;Cewka;rpm");
                            streamWriter.Close();
                        }
```

```
            }
        }
    }
}

/// <summary>
/// Otwieranie ustawienia formulaża z komentazem
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void commentsToTestToolStripMenuItem_Click(object sender, EventArgs e)
{
    FormComments FC = new FormComments();
    FC.Show();
}


/// <summary>
/// Otwieranie formulaża kończącergo test
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void finishTestToolStripMenuItem_Click(object sender, EventArgs e)
{
    FormFinishTest FFinish = new FormFinishTest();
    FFinish.Show();
}

/// <summary>
/// Ustawienie Folderu i pliku Settings
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void backlogFolderToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (saveFileSetting.ShowDialog() == DialogResult.OK)
    {
        TestSettings = saveFileSetting.FileName;
        textBoxSettingsInfo.Text = TestSettings;
    }
}

/// <summary>
/// Ustawienie globalnego folderu dla aplikacji
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void globalFolderToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (folderBrowserGlobal.ShowDialog() == DialogResult.OK)
    {
        GlobalFolder = folderBrowserGlobal.SelectedPath;
        SaveFileReport.InitialDirectory = GlobalFolder;
        saveFileSetting.InitialDirectory = GlobalFolder;
        openFileSetting.InitialDirectory = GlobalFolder;
    }
}


/// <summary>
/// Metoda zapisywania danych do pliku .txt
/// </summary>
/// <param name="logFileName"></param>
/// <param name="data"></param>
```

```
        public static void WriteDataLog(string logFileName, string data)
        {
            if (File.Exists(logFileName) & dzialanie == true)
            {
                using (FileStream fileStream = new FileStream(logFileName,
FileMode.Append, FileAccess.Write, FileShare.None))
                {
                    using (StreamWriter streamWriter = new StreamWriter(fileStream))
                    {
                        streamWriter.WriteLine(data);
                        streamWriter.Close();
                    }
                }
            }
        }


        /// <summary>
        /// Ustawienie standardowych scieżek do raportu i testu
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void defaultSettingsToolStripMenuItem_Click(object sender, EventArgs e)
        {
            string TestSettingsDir = "c:/TESTER_VFS169/Settings/";
            string PlikRaportyDir = "c:/TESTER_VFS169/Reports/";

            TestSettings = "c:/TESTER_VFS169/Settings/Settings.xml";
            PlikRaportu = "c:/TESTER_VFS169/Settings/Report.txt";
            GlobalFolder = "c:/TESTER_VFS169/Settings/";


            DirectoryInfo dirSetting = null;
            if (!Directory.Exists(@TestSettings))
            {
                dirSetting = Directory.CreateDirectory(@TestSettingsDir);
            }


            DirectoryInfo dirReport = null;
            if (!Directory.Exists(@TestSettings))
            {
                dirReport = Directory.CreateDirectory(@PlikRaportyDir);
            }


            string RepS = TestSettings.Replace("/", @"\");
            string RepR = PlikRaportu.Replace("/", @"\");
            textBoxSettingsInfo.Text = RepS;
            textBoxReportInfo.Text = RepR;
        }


        /// <summary>
        /// Kopiowanie do schowka scieżki do pliku SETTINGS
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ButCopySettings_Click(object sender, EventArgs e)
        {
            try
            {
                System.Diagnostics.Process.Start(@textBoxSettingsInfo.Text);
                //System.Windows.Forms.Clipboard.SetText(textBoxSettingsInfo.Text);
```

```
            }
            catch
            {
                MessageBox.Show("Lack of correct file", "Warning");
            }

        }


        /// <summary>
        /// Kopiowanie do schowka scieżki do pliku TEST
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void butCopyReport_Click(object sender, EventArgs e)
        {
            try
            {
                System.Diagnostics.Process.Start(@textBoxReportInfo.Text);
                //System.Windows.Forms.Clipboard.SetText(textBoxReportInfo.Text);
            }
            catch
            {
                MessageBox.Show("Lack of correct file", "Warning");
            }


        }

        /// <summary>
        /// Otwieranie formy definicji nowego testu
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void newTestDescriptionToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            FormNTD FNTD = new FormNTD();
            FNTD.ShowDialog();
        }


        /// <summary>
        /// Wyswietlanie aktualnej daty i godziny w pasku dolnym
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void timerZegar_Tick(object sender, EventArgs e)
        {
            StripStatusTime.Text = DateTime.Now.ToShortDateString() + "    " +
DateTime.Now.ToLongTimeString();
        }

        /// <summary>
        /// Otwieranie formy Rozpoczecia nowego testu
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void startNewTestToolStripMenuItem_Click(object sender, EventArgs e)
        {
            FormSNT FSNT = new FormSNT();
            FSNT.Show();
        }
```

```
/// <summary>
/// Wyswietlanie wykresu oraz skali Pressure Suction
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void checkBox1_CheckedChanged(object sender, EventArgs e)
{
    if (PlotBoxPressureSuction.Checked)
    {
        WFPressureSuction.Visible = true;
        legendItem1.Visible = true;
    }
    else
    {
        WFPressureSuction.Visible = false;
        legendItem1.Visible = false;
    }

    if (!PlotBoxPressureDischarge.Checked & !PlotBoxPressureSuction.Checked)
    {
        AxisPressure.Visible = false;
    }
    else
    {
        AxisPressure.Visible = true;
    }
}

/// <summary>
/// Wyswietlanie wykresu oraz skali PressureDischatge
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void PlotPressureDischarge_CheckedChanged(object sender, EventArgs e)
{
    if (PlotBoxPressureDischarge.Checked)
    {
        WFPressureDischarge.Visible = true;
        legendItem2.Visible = true;
    }
    else
    {
        WFPressureDischarge.Visible = false;
        legendItem2.Visible = false;
    }

    if (!PlotBoxPressureDischarge.Checked & !PlotBoxPressureSuction.Checked)
    {
        AxisPressure.Visible = false;
    }
    else
    {
        AxisPressure.Visible = true;
    }
}

/// <summary>
/// Wyswietlanie wykresu oraz skali TempSuction
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void PlotBoxTempSuction_CheckedChanged(object sender, EventArgs e)
```

```
        {
            if (PlotBoxTempSuction.Checked)
            {
                WFTempSuction.Visible = true;
                legendItem3.Visible = true;
            }
            else
            {
                WFTempSuction.Visible = false;
                legendItem3.Visible = false;
            }

            if (!PlotBoxTempCompressor.Checked & !PlotBoxTempCondenserIn.Checked &
!PlotBoxTempCondenserOut.Checked & !PlotBoxTempDischarge.Checked &
!PlotBoxTempEvapIn.Checked & !PlotBoxTempEvapOut.Checked & !PlotBoxTempHotbox.Checked &
!PlotBoxTempSuction.Checked)
            {
                AxisTemperature.Visible = false;
            }
            else
            {
                AxisTemperature.Visible = true;
            }
        }


        /// <summary>
        /// Wyswietlanie wykresu oraz skali TempDischarge
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void PlotBoxTempDischarge_CheckedChanged(object sender, EventArgs e)
        {
            if (PlotBoxTempDischarge.Checked)
            {
                WFTempDischarge.Visible = true;
                legendItem4.Visible = true;
            }
            else
            {
                WFTempDischarge.Visible = false;
                legendItem4.Visible = false;
            }
            if (!PlotBoxTempCompressor.Checked & !PlotBoxTempCondenserIn.Checked &
!PlotBoxTempCondenserOut.Checked & !PlotBoxTempDischarge.Checked &
!PlotBoxTempEvapIn.Checked & !PlotBoxTempEvapOut.Checked & !PlotBoxTempHotbox.Checked &
!PlotBoxTempSuction.Checked)
            {
                AxisTemperature.Visible = false;
            }
            else
            {
                AxisTemperature.Visible = true;
            }
        }


        /// <summary>
        /// Wyswietlanie wykresu oraz skali CONDENSERIn
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void PlotBoxTempCondenserIn_CheckedChanged(object sender, EventArgs e)
        {
```

```
            if (PlotBoxTempCondenserIn.Checked)
            {
                WFTempCondenserIn.Visible = true;
                legendItem5.Visible = true;
            }
            else
            {
                WFTempCondenserIn.Visible = false;
                legendItem5.Visible = false;
            }
            if (!PlotBoxTempCompressor.Checked & !PlotBoxTempCondenserIn.Checked &
!PlotBoxTempCondenserOut.Checked & !PlotBoxTempDischarge.Checked &
!PlotBoxTempEvapIn.Checked & !PlotBoxTempEvapOut.Checked & !PlotBoxTempHotbox.Checked &
!PlotBoxTempSuction.Checked)
            {
                AxisTemperature.Visible = false;
            }
            else
            {
                AxisTemperature.Visible = true;
            }
        }

        /// <summary>
        /// Wyswietlanie wykresu oraz skali CondenserOut
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void PlotBoxTempCondenserOut_CheckedChanged(object sender, EventArgs e)
        {
            if (PlotBoxTempCondenserOut.Checked)
            {
                WFTempCondenserOut.Visible = true;
                legendItem6.Visible = true;
            }
            else
            {
                WFTempCondenserOut.Visible = false;
                legendItem6.Visible = false;
            }
            if (!PlotBoxTempCompressor.Checked & !PlotBoxTempCondenserIn.Checked &
!PlotBoxTempCondenserOut.Checked & !PlotBoxTempDischarge.Checked &
!PlotBoxTempEvapIn.Checked & !PlotBoxTempEvapOut.Checked & !PlotBoxTempHotbox.Checked &
!PlotBoxTempSuction.Checked)
            {
                AxisTemperature.Visible = false;
            }
            else
            {
                AxisTemperature.Visible = true;
            }
        }


        /// <summary>
        /// Wyswietlanie wykresu oraz skali EVAPIN
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void PlotBoxTempEvapIn_CheckedChanged(object sender, EventArgs e)
        {
            if (PlotBoxTempEvapIn.Checked)
            {
                WFTempEvapIn.Visible = true;
```

```
                legendItem7.Visible = true;
            }
            else
            {
                WFTempEvapIn.Visible = false;
                legendItem7.Visible = false;
            }
            if (!PlotBoxTempCompressor.Checked & !PlotBoxTempCondenserIn.Checked &
!PlotBoxTempCondenserOut.Checked & !PlotBoxTempDischarge.Checked &
!PlotBoxTempEvapIn.Checked & !PlotBoxTempEvapOut.Checked & !PlotBoxTempHotbox.Checked &
!PlotBoxTempSuction.Checked)
            {
                AxisTemperature.Visible = false;
            }
            else
            {
                AxisTemperature.Visible = true;
            }
        }

        /// <summary>
        /// Wyswietlanie wykresu oraz skali EVAPOUT
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void PlotBoxTempEvapOut_CheckedChanged(object sender, EventArgs e)
        {
            if (PlotBoxTempEvapOut.Checked)
            {
                WFTempEvapOut.Visible = true;
                legendItem8.Visible = true;
            }
            else
            {
                WFTempEvapOut.Visible = false;
                legendItem8.Visible = false;
            }
            if (!PlotBoxTempCompressor.Checked & !PlotBoxTempCondenserIn.Checked &
!PlotBoxTempCondenserOut.Checked & !PlotBoxTempDischarge.Checked &
!PlotBoxTempEvapIn.Checked & !PlotBoxTempEvapOut.Checked & !PlotBoxTempHotbox.Checked &
!PlotBoxTempSuction.Checked)
            {
                AxisTemperature.Visible = false;
            }
            else
            {
                AxisTemperature.Visible = true;
            }
        }

        /// <summary>
        /// Wyswietlanie wykresu oraz skali COMPRESSOR
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void PlotBoxTempCompressor_CheckedChanged(object sender, EventArgs e)
        {
            if (PlotBoxTempCompressor.Checked)
            {
                WFTempCompressor.Visible = true;
                legendItem9.Visible = true;
            }
            else
            {
```

```
                WFTempCompressor.Visible = false;
                legendItem9.Visible = false;
            }
            if (!PlotBoxTempCompressor.Checked & !PlotBoxTempCondenserIn.Checked &
!PlotBoxTempCondenserOut.Checked & !PlotBoxTempDischarge.Checked &
!PlotBoxTempEvapIn.Checked & !PlotBoxTempEvapOut.Checked & !PlotBoxTempHotbox.Checked &
!PlotBoxTempSuction.Checked)
            {
                AxisTemperature.Visible = false;
            }
            else
            {
                AxisTemperature.Visible = true;
            }
        }

        /// <summary>
        /// Wyswietlanie wykresu oraz skali HOTBOX
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void PlotBoxTempHotbox_CheckedChanged(object sender, EventArgs e)
        {
            if (PlotBoxTempHotbox.Checked)
            {
                WFTempHotbox.Visible = true;
                legendItem10.Visible = true;
            }
            else
            {
                WFTempHotbox.Visible = false;
                legendItem10.Visible = false;
            }
            if (!PlotBoxTempCompressor.Checked & !PlotBoxTempCondenserIn.Checked &
!PlotBoxTempCondenserOut.Checked & !PlotBoxTempDischarge.Checked &
!PlotBoxTempEvapIn.Checked & !PlotBoxTempEvapOut.Checked & !PlotBoxTempHotbox.Checked &
!PlotBoxTempSuction.Checked)
            {
                AxisTemperature.Visible = false;
            }
            else
            {
                AxisTemperature.Visible = true;
            }
        }



        /// <summary>
        /// Zmiana stanu pomiedzy auto a Manual Value dla osi TEMPERATURE
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void TaxisBox_CheckedChanged(object sender, EventArgs e)
        {
            if (TaxisBox.Checked)
            {
                TaxisMIN.ReadOnly = true;
                TaxisMAX.ReadOnly = true;
                AxisTemperature.Mode = NationalInstruments.UI.AxisMode.AutoScaleLoose;
            }
            else
            {
                TaxisMIN.ReadOnly = false;
```

```
                TaxisMAX.ReadOnly = false;
                AxisTemperature.Mode = NationalInstruments.UI.AxisMode.Fixed;
                NationalInstruments.UI.Range TAxis = new
NationalInstruments.UI.Range(Convert.ToDouble(TaxisMIN.Text),
Convert.ToDouble(TaxisMAX.Text));
                AxisTemperature.Range = TAxis;
            }
        }


        /// <summary>
        /// Zmiana stanu pomiedzy auto a Manual Value dla osi PRESSURE
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void PaxisBOX_CheckedChanged_1(object sender, EventArgs e)
        {
            if (PaxisBOX.Checked)
            {
                PaxisMIN.ReadOnly = true;
                PaxisMAX.ReadOnly = true;
                AxisPressure.Mode = NationalInstruments.UI.AxisMode.AutoScaleLoose;
            }
            else
            {
                PaxisMIN.ReadOnly = false;
                PaxisMAX.ReadOnly = false;
                AxisPressure.Mode = NationalInstruments.UI.AxisMode.Fixed;
                NationalInstruments.UI.Range PAxis = new
NationalInstruments.UI.Range(Convert.ToDouble(PaxisMIN.Text),
Convert.ToDouble(PaxisMAX.Text));
                AxisPressure.Range = PAxis;
            }
        }


        /// <summary>
        /// Wyswietlanie terminala
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void tERMINALToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (tERMINALToolStripMenuItem.Checked == true)
            {
                BoxCoil.Visible = true;
                BoxRPM.Visible = true;
                BoxTerminal.Visible = false;
                tERMINALToolStripMenuItem.Checked = false;

            }
            else
            {
                BoxCoil.Visible = false;
                BoxRPM.Visible = false;
                BoxTerminal.Visible = true;
                tERMINALToolStripMenuItem.Checked = true;

            }
        }

        /// <summary>
        ///  Wstrzymanie sysyłanie danych do terminala
        /// </summary>
```

```
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void TerminalStopCheck_CheckedChanged(object sender, EventArgs e)
        {
            if (TerminalStopCheck.Checked)
            {
                TerminalStop = true;
            }
            else
            {
                TerminalStop = false;
            }
        }

        private void TerminalClearBut_Click(object sender, EventArgs e)
        {
            Terminal.Text = "";
        }

        private void saveTestStatusToolStripMenuItem_Click(object sender, EventArgs e)
        {

        }


        /// <summary>
        /// Zmiana stanu pomiedzy auto a Manual Value dla osi TIME
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void TimeAxisBOX_CheckedChanged(object sender, EventArgs e)
        {
            if (TimeAxisBOX.Checked)
            {
                TimeAxisMIN.ReadOnly = true;
                TimeAxisMAX.ReadOnly = true;
                TimeAxis.Mode = NationalInstruments.UI.AxisMode.StripChart;
            }
            else
            {
                TimeAxisMIN.ReadOnly = false;
                TimeAxisMAX.ReadOnly = false;
                TimeAxis.Mode = NationalInstruments.UI.AxisMode.Fixed;
                NationalInstruments.UI.Range TAxis = new
NationalInstruments.UI.Range(Convert.ToDouble(TimeAxisMIN.Text),
Convert.ToDouble(TimeAxisMAX.Text));
                TimeAxis.Range = TAxis;
            }
        }


        /// <summary>
        /// Ustawienie wartosci MIN i MAX dla skali Temperature
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void TaxisMIN_TextChanged(object sender, EventArgs e)
        {
            try
            {
                NationalInstruments.UI.Range TAxis = new
NationalInstruments.UI.Range(Convert.ToDouble(TaxisMIN.Text),
Convert.ToDouble(TaxisMAX.Text));
                AxisTemperature.Range = TAxis;
```

```
            }
            catch
            {
                //MessageBox.Show("Wrong ValueType");
            }
        }

        private void TaxisMAX_TextChanged(object sender, EventArgs e)
        {
            try
            {
                NationalInstruments.UI.Range TAxis = new
NationalInstruments.UI.Range(Convert.ToDouble(TaxisMIN.Text),
Convert.ToDouble(TaxisMAX.Text));
                AxisTemperature.Range = TAxis;
            }
            catch
            {
                //MessageBox.Show("Wrong ValueType");
            }
        }


        /// <summary>
        /// Ustawienie wartosci MIN i MAX dla skali Pressure
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void PaxisMIN_TextChanged(object sender, EventArgs e)
        {
            try
            {
                NationalInstruments.UI.Range PAxis = new
NationalInstruments.UI.Range(Convert.ToDouble(PaxisMIN.Text),
Convert.ToDouble(PaxisMAX.Text));
                AxisPressure.Range = PAxis;
            }
            catch
            {
                //MessageBox.Show("Wrong ValueType");
            }
        }

        private void PaxisMAX_TextChanged(object sender, EventArgs e)
        {
            try
            {
                NationalInstruments.UI.Range PAxis = new
NationalInstruments.UI.Range(Convert.ToDouble(PaxisMIN.Text),
Convert.ToDouble(PaxisMAX.Text));
                AxisPressure.Range = PAxis;
            }
            catch
            {
                //MessageBox.Show("Wrong ValueType");
            }
        }


        /// <summary>
        /// Ustawienie wartosci MIN i MAX dla skali TIME
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
```

```
        private void TimeAxisMIN_TextChanged(object sender, EventArgs e)
        {
            try
            {
                NationalInstruments.UI.Range TAxis = new
NationalInstruments.UI.Range(Convert.ToDouble(TimeAxisMIN.Text),
Convert.ToDouble(TimeAxisMAX.Text));
                TimeAxis.Range = TAxis;
            }
            catch
            {
                //MessageBox.Show("Wrong ValueType");
            }
        }

        private void TimeAxisMAX_TextChanged(object sender, EventArgs e)
        {
            try
            {
                NationalInstruments.UI.Range TAxis = new
NationalInstruments.UI.Range(Convert.ToDouble(TimeAxisMIN.Text),
Convert.ToDouble(TimeAxisMAX.Text));
                TimeAxis.Range = TAxis;
            }
            catch
            {
                //MessageBox.Show("Wrong ValueType");
            }
        }


        /// <summary>
        /// Ustawienie automatycznego  Minimum w oknie czasu
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void label36_MouseDoubleClick(object sender, MouseEventArgs e)
        {
            TimeAxisMIN.Text = "0";
        }


        /// <summary>
        /// Kopiowanie linku do pliku raportu
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void textBoxReportInfo_MouseDoubleClick(object sender, MouseEventArgs e)
        {
            try
            {
                System.Windows.Forms.Clipboard.SetText(textBoxReportInfo.Text);
            }
            catch
            {

            }
        }

        /// <summary>
        /// Kopiowanie linku do pliku Settings
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
```

```
        private void textBoxSettingsInfo_MouseDoubleClick(object sender, MouseEventArgs
e)
        {
            try
            {
                System.Windows.Forms.Clipboard.SetText(textBoxSettingsInfo.Text);
            }
            catch
            {

            }
        }

        /// <summary>
        /// Przeładowanie data base dla DataSet
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void button1_Click(object sender, EventArgs e)
        {
            this.testDescriptionInputTableAdapter.Fill(this.databaseDataSet.TestDescripti
onInput);
            this.testDescriptionSetupTableAdapter.Fill(this.databaseDataSet.TestDescripti
onSetup);
            this.testParametersTableAdapter.Fill(this.databaseDataSet.TestParameters);
        }




    }
}
```


Elektronika

```
// Program odczytuje temperaturę kilku czujników

#include <OneWire.h>
#include <DS18B20.h>
#include <DHT.h>

// Numer pinu do którego podłaczasz czujnik
#define ONEWIRE_PIN 7   //Czujnik Temperatury DS18B20
#define DHTPIN1 6       //Czujnik Temperatury DHT21
#define DHTPIN2 5       //Czujnik Temperatury DHT21

//Typ czujnika DHT
#define DHTTYPE DHT21

// Ilość czujników
#define SENSORS_NUM 8
//#define TEMPERATURE_PRECISION 9  //9

// Adresy czujników DS18B20
const byte address[SENSORS_NUM][8] PROGMEM = {
0x28, 0xB4, 0x2D, 0xF0, 0x4, 0x0, 0x0, 0x3E,
0x28, 0x4F, 0xEA, 0xDF, 0x5, 0x0, 0x0, 0x86,
0x28, 0x51, 0xA5, 0xE0, 0x5, 0x0, 0x0, 0x9A,
0x28, 0x4F, 0xAA, 0xE0, 0x5, 0x0, 0x0, 0xBD,
```

```
0x28, 0xAA, 0x66, 0xE0, 0x5, 0x0, 0x0, 0x7C,
0x28, 0xBC, 0x9B, 0xE0, 0x5, 0x0, 0x0, 0xDF,
0x28, 0x10, 0xD, 0xE0, 0x5, 0x0, 0x0, 0xD5,
0x28, 0x15, 0x23, 0xF0, 0x4, 0x0, 0x0, 0xF7
};

//Inicjalizacja bibliotek
OneWire onewire(ONEWIRE_PIN);
DS18B20 sensors(&onewire);

DHT dht1(DHTPIN1, DHTTYPE);
DHT dht2(DHTPIN2, DHTTYPE);

//Definicja zmiennych globalnych
float Tsuc;
float Tdis;
float TcondenserIn;
float TcondenserOut;
float TevapuratorIn;
float TevapuratorOut;
float Thotbox;
float Tcompressor;
float Psuc;
float Pdis;

int P1;
int P2;

float TairIn;
float TairOut;
float HairIn;
float HairOut;


//Definicja przypisanych pinów
const int Button1 = 22;
const int Button2 = 24;
const int Button3 = 30;
const int Przelacznik1 = 26;
const int Przelacznik2 = 28;


long time = 0;
long ButtonDebounce = 500;

//Definicja diod LED
const int LED1g = 25 ;
const int LED1r = 23 ;
const int LED2g = 29 ;
const int LED2r = 27 ;
const int LED3g = 33 ;
const int LED3r = 31 ;
const int LED4g = 37 ;
const int LED4r = 35 ;
const int LED5g = 45 ;
const int LED5r = 43 ;

const int Pressure1 = A0;
const int Pressure2 = A1;
```

```
const int Buzzer = A5;

//Zmienne dla pomiarów elektrycznych
const int COILvPin =      A13;
const int ECVvPin =                A15;
const int COILcurrent =   A12;
const int ECVcurrent =    A14;
const int COILchecked = 4;
int COILcheckedState = 0;
int COILvolt;
int ECVvolt;
float COILvoltage;
float ECVvoltage;

String Cewka;
const unsigned long sampleTime = 100000UL;
// sample over 100ms, it is an exact number of cycles for both 50Hz and 60Hz mains
const unsigned long numSamples = 250UL;
          // choose the number of samples to divide sampleTime exactly, but low enough for the ADC to keep
up
const unsigned long sampleInterval = sampleTime/numSamples;                                    // the sampling
interval, must be longer than then ADC conversion time
//const int adc_zero = 522;
                    // relative digital zero of the arudino input from ACS712 (could make this a variable and
auto-adjust it)
int adc_zero1;
                              // autoadjusted relative digital zero
int adc_zero2;

//definicja tachometru
int ticsPerRev = 2;       // define number of tics per rev of code wheel
float rpm = 0.0;  // I prefer float for rpm
volatile int rpmcount = 0; // volitile because the variable is manipulated during the interrupt
unsigned long timeold = 0; // used to calculate d_t= millis()-timeold;
int d_t;
volatile byte status = LOW;   // set initial state of status LED


void setup() {
  //while(!Serial);
  Serial.begin(9600);

  sensors.begin();
  sensors.request();
  dht1.begin();
  dht2.begin();


//Ustawienia dla przycisku
  pinMode(Button1, INPUT_PULLUP);  //ustawienie rezystowow pull up
  pinMode(Button2, INPUT_PULLUP);  //ustawienie rezystowow pull up
  pinMode(Button3, INPUT_PULLUP);  //ustawienie rezystowow pull up


//Ustawienie dla przełaczkika
  pinMode(Przelacznik1, INPUT_PULLUP);   //digitalWrite(Przelacznik1, HIGH);
  pinMode(Przelacznik2, INPUT_PULLUP);   //digitalWrite(Przelacznik2, HIGH);

//Ustawienia dla diod LED
          pinMode(LED1g, OUTPUT);                  digitalWrite(LED1g, HIGH);
```

```
        pinMode(LED1r, OUTPUT);                 digitalWrite(LED1r, HIGH);
        pinMode(LED2g, OUTPUT);                 digitalWrite(LED2g, HIGH);
        pinMode(LED2r, OUTPUT);                 digitalWrite(LED2r, HIGH);
        pinMode(LED3g, OUTPUT);                 digitalWrite(LED3g, HIGH);
        pinMode(LED3r, OUTPUT);                 digitalWrite(LED3r, HIGH);
        pinMode(LED4g, OUTPUT);                 digitalWrite(LED4g, HIGH);
        pinMode(LED4r, OUTPUT);                 digitalWrite(LED4r, HIGH);
        pinMode(LED5g, OUTPUT);                 digitalWrite(LED5g, HIGH);
        pinMode(LED5r, OUTPUT);                 digitalWrite(LED5r, HIGH);

//Czujnik prądu
        adc_zero1 = determineVQ(COILcurrent);
        adc_zero2 = determineVQ(ECVcurrent);
        pinMode(COILchecked, INPUT);


        //Serial.println("Test VFS169");

}

void loop() {


 //===============Obsługa przycisków ===========
   int Przycisk1  = digitalRead(Button1);                    // read input value
   int Przycisk2  = digitalRead(Button2);
   int Cewka24V = digitalRead(Button3);
   int PrzelacznikA  = digitalRead(Przelacznik1);
   int PrzelacznikB  = digitalRead(Przelacznik2);

 if (Przycisk1 == LOW)
   {
     digitalWrite(LED2r, LOW);
 }
 else    {
     digitalWrite(LED2r, HIGH);
 }
  if (Przycisk2 == LOW)
   {
     digitalWrite(LED1g, LOW);
 }
 else    {
     digitalWrite(LED1g, HIGH);
 }

 if (PrzelacznikA == LOW)
   {
     digitalWrite(LED5r, LOW);
 }
 else    {
     digitalWrite(LED5r, HIGH);
 }
  if (PrzelacznikB == LOW)
   {
     digitalWrite(LED5g, LOW);
 }
 else    {
     digitalWrite(LED5g, HIGH);
 }
```

```
//
================================================================================
====================


//Odczyt temperatur z czujników DHT
  HairIn = dht1.readHumidity();
  TairIn = dht1.readTemperature();
  HairOut = dht2.readHumidity();
  TairOut = dht2.readTemperature();


//Odczyt i obliczenie cisnienia
P1 = analogRead(Pressure1);
        if (P1>100 && P1<925)
                {
                        Pdis = 0.0317 * P1 + (-3.1823);
                }
        else
                {
                        Pdis = NULL;
                }


P2 = analogRead(Pressure2);
        if (P2>100 && P2<925)
                {
                        Psuc = 0.0122 * P2 + (-1.2741);
                }
        else
                {
                        Psuc = NULL;
                }

COILvolt = analogRead(COILvPin);
        if (Cewka24V == HIGH)
                {
                        COILvoltage = COILvolt * (12/1023);                     //dodac przelicznik
                }

        else if (Cewka24V == LOW)
                {
                        COILvoltage = COILvolt * (24/1023);                     //dodac przelicznik
                }


ECVvolt = analogRead(ECVvPin);
ECVvoltage = COILvolt * (12/1023);                      //dodac przelicznik


//Odczyt temperatur z czujników DS18B20
  if (sensors.available())
   {
     Tsuc               = sensors.readTemperature(FA(address[0]));
     Tdis               = sensors.readTemperature(FA(address[1]));
     TcondenserIn       = sensors.readTemperature(FA(address[2]));
     TcondenserOut      = sensors.readTemperature(FA(address[3]));
          TevapuratorIn                 = sensors.readTemperature(FA(address[4]));
          TevapuratorOut = sensors.readTemperature(FA(address[5]));
          Thotbox                               = sensors.readTemperature(FA(address[6]));
```

```
            Tcompressor             = sensors.readTemperature(FA(address[7]));

  sensors.request();
  }

COILcheckedState = digitalRead(COILchecked);
if (COILcheckedState == HIGH)
{
        digitalWrite(LED2g, LOW);
        Cewka = "1";
}
else
{
        digitalWrite(LED2r, LOW);
        Cewka = "0";
}


                Serial.print(Tsuc);                                                     Serial.print(";");
                Serial.print(Tdis);                                                     Serial.print(";");
                Serial.print(TcondenserIn);                                     Serial.print(";");
                Serial.print(TcondenserOut);                            Serial.print(";");
                Serial.print(TevapuratorIn);                            Serial.print(";");
                Serial.print(TevapuratorOut);                           Serial.print(";");
                Serial.print(Thotbox);                                          Serial.print(";");
                Serial.print(Tcompressor);                                      Serial.print(";");
                Serial.print(TairIn);                                           Serial.print(";");
                Serial.print(HairIn);                                           Serial.print(";");
                Serial.print(TairOut);                                          Serial.print(";");
                Serial.print(HairOut);                                          Serial.print(";");
                Serial.print(Psuc);                                                     Serial.print(";");
                Serial.print(Pdis);                                                     Serial.print(";");
                Serial.print(COILvoltage);                              Serial.print(";");
                Serial.print(readCurrent(COILcurrent),3);   Serial.print(";");
                Serial.print(ECVvoltage);                               Serial.print(";");
                Serial.print(readCurrent(ECVcurrent),3);    Serial.print(";");
                Serial.print(Cewka);                                            Serial.print(";");
                Serial.print(rpm);

                Serial.println();

delay(1000);
}



int determineVQ(int PIN) {
        //Serial.print("estimating avg. quiscent voltage:");
        long VQ = 0;
        //read 5000 samples to stabilise value
        for (int i=0; i<5000; i++) {
                VQ += analogRead(PIN);
                delay(1);//depends on sampling (on filter capacitor), can be 1/80000 (80kHz) max.
        }
        VQ /= 5000;
        //Serial.print(map(VQ, 0, 1023, 0, 5000));Serial.println(" mV");
        return int(VQ);
}

float readCurrent(int PIN)
```

```
{
	unsigned long currentAcc = 0;
	unsigned int count = 0;
	unsigned long prevMicros = micros() - sampleInterval ;
	while (count < numSamples)
	{
		if (micros() - prevMicros >= sampleInterval)
		{
			int adc_raw = analogRead(PIN) - adc_zero1;
			currentAcc += (unsigned long)(adc_raw * adc_raw);
			++count;
			prevMicros += sampleInterval;
		}
	}

	//ustawienie poprawnej wartosci
	float rms = sqrt((float)currentAcc/(float)numSamples) * (34.0 / 1024.0);
	return rms;
	//Serial.println(rms);
}

void Sygnal()
{
	// notes in the melody:
	int melody[] = {  262 }; //NOTE_C4, NOTE_G3, NOTE_G3, NOTE_A3, NOTE_G3, 0, NOTE_B3,
NOTE_C4

	// note durations: 4 = quarter note, 8 = eighth note, etc.:
	int noteDurations[] = { 4 }; // 8, 8, 4, 4, 4, 4, 4
	for (int thisNote = 0; thisNote < 1; thisNote++)
	{

		// to calculate the note duration, take one second
		// divided by the note type.
		//e.g. quarter note = 1000 / 4, eighth note = 1000/8, etc.
		int noteDuration = 1000 / noteDurations[thisNote];
		tone(Buzzer, melody[thisNote], noteDuration);

		// to distinguish the notes, set a minimum time between them.
		// the note's duration + 30% seems to work well:
		int pauseBetweenNotes = noteDuration * 1.30;
		delay(pauseBetweenNotes);
		// stop the tone playing:
		noTone(Buzzer);
		delay (1000);
	}
}
```