**Индивидуальное задание**

1) развить навыки сбора, анализа и обобщения информации в сети Интернет (с учетом основных требований информационной безопасности) и в научной и технической литературе по теме выпускной квалификационной работы;

2) разработать техническое задание, сформулировать назначения, цели и задачи разработки программного обеспечения, охарактеризовать объекты автоматизации и сформулировать требования к программному обеспечению;

3) разработать программного обеспечения с использованием основных методов и инструментов.

**Содержание**

[1 Описание программы 7](#_Toc482984797)

[1.1 Общие сведения 7](#_Toc482984798)

[1.2 Функциональное назначение 14](#_Toc482984799)

[1.3 Описание логической структуры 14](#_Toc482984800)

[1.4 Используемые технические средства 19](#_Toc482984801)

[1.5 Входные данные 19](#_Toc482984802)

[1.6 Выходные данные 20](#_Toc482984803)

[1.7 Архитектура программного обеспечения 20](#_Toc482984804)

[2 Текст программы 22](#_Toc482984805)

[2.1 Главный модуль программы 22](#_Toc482984806)

[2.2 Модуль Constants 26](#_Toc482984807)

[2.3 Модуль Signal 27](#_Toc482984808)

[2.4 Компонент Module 34](#_Toc482984809)

[2.5 Модуль PLC 39](#_Toc482984810)

[2.6 Модуль Generator 40](#_Toc482984811)

[2.7 Модуль ConnectProSim 43](#_Toc482984812)

[Заключение 44](#_Toc482984813)

[Список использованных источников 45](#_Toc482984814)

**Введение**

В середине 90-х гг. прошлого века микропроцессорная техника достигла состояния надежности, обеспечивающего возможность управления технологическим процессом с помощью контроллеров. Препятствиями на пути более активного применения микропроцессорных средств стали: человеческий фактор (привычка пользоваться индивидуальными приборами и ключами), нехватка финансирования для ввода новых энергоблоков в период с 1995 г., а также неприспособленность технологического оборудования к автоматизации.

Построение автоматизированных систем управления технологическим процессом (АСУ ТП) на базе микропроцессорной техники позволило значительно повысить качество систем управления, открыло новые перспективы автоматизации. Под АСУ ТП обычно понимается целостное решение, обеспечивающее автоматизацию основных операций технологического процесса на производстве в целом или каком-то его участке, выпускающем относительно завершённое изделие.

Понятие «автоматизированный», в отличие от понятия «автоматический», подчёркивает необходимость участия человека в отдельных операциях, как в целях сохранения контроля над процессом, так и в связи со сложностью или нецелесообразностью автоматизации отдельных операций.

Целью данной работы является реализация визуализации системы управления охлаждением производственного объекта.

Система охлаждения представляет собой аппараты воздушного охлаждения («сухие градирни») – 4 шт., каждый аппарат имеет 3 секции, в которых установлены по 4 вентиляторных агрегата (всего 48 шт.). Управление происходит посекционно. Система в автоматическом режиме контролирует температуру теплоносителя (водный раствор), который поступает для охлаждения, включая или выключая секции по заданному алгоритму работы. Подача теплоносителя осуществляется с помощью насосов, которые удерживают определенное давление.

Входными данными являются показания с датчиков – входные сигналы: температуры, давления, текущие состояния секций и насосов. На основе них алгоритм программы предпринимает действие для управления объектом автоматизации. Действия являются выходными данными - выходными сигналами: включение или выключение секции, насоса.

Реализация алгоритма автоматизации системы управления охлаждения происходит с помощью программируемого логического контроллера (ПЛК) Siemens SIMATIC S7-300, который включает в себя сигнальные и функциональные модули для работы с входными и выходными сигналами.

Система визуализации — это верхний уровень АСУ ТП, благодаря которому, пользователь работает с визуальным представлением системы. На этом уровне идет контроль хода производства: обеспечивается связь с нижними уровнями, откуда осуществляется сбор данных, визуализация и диспетчеризация (мониторинг) хода технологического процесса. Это уровень HMI, SCADA. На этом уровне задействован человек, т.е. оператор (диспетчер). Он осуществляет локальный контроль технологического оборудования через так называемый человеко-машинный интерфейс (HMI - Human Machine Interface). К нему относятся: мониторы, графические панели, которые устанавливаются локально на пультах управления и шкафах автоматики.

Визуализация системы охлаждения происходит с помощью среды разработки Siemens WinCC Flexible и панели визуализации Simatic HMI Station TP 277. С помощью данной панели, оператор, находящийся на месте работы, может наблюдать за происходящими процессами на объекте автоматизации и контролировать их.

Помимо этого, в ходе работы возникла потребность в тестировании программного обеспечения для логического контроллера, применяемого для управления системы охлаждения. Поэтому было принято решения для создания программного обеспечения для симуляции оборудования – сигнальных и функциональных модулей. Для реализации данного программного обеспечения использовалась среда программирования Microsoft Visual C# 2008 и язык программирования C#, а также программа симуляции входных и выходных сигналов Simatic S7-PLCSIM.

Задачи работы:

1. изучить предметную область;
2. изучить основные принципы устройства оборудования визуализации и автоматизации;
3. изучить инструмент разработки для визуализации (Siemens WinCC Flexible) и симуляции (язык программирования C# и Microsoft Visual C# 2008);
4. разработать программу визуализации;
5. разработать программу симуляции;

Задачей программного обеспечения является уменьшение количества ошибок при работе на производственном объекте, повышение качества труда, уменьшение затрат на обслуживание.

# Описание программы

## Общие сведения

Наименование программного обеспечения: «Программная визуализация системы управления охлаждением производственного объекта».

Исходным языком программирования является C#. Среда разработки – Siemens WinCC Flexible и Microsoft Visual C# 2008.

Для визуализации автоматизации промышленных объектов используются панели оператора.

Панели операторов SIMATIC могут использоваться для решения задач оперативного управления и мониторинга на уровне производственных машин и установок во всех областях промышленного производства, а также в системах автоматизации зданий.

Для реализации программного обеспечения используется панель оператора SIMATIC HMI Station TP 277 (рисунок 1.1).



Рисунок 1.1 – Station TP 277

Simatic WinCC Flexible — программное обеспечение для создания человеко-машинного интерфейса (рисунок 1.2), составная часть семейства систем автоматизации Simatic, производимых компанией Siemens AG.

Предназначено для решения комплекса задач человеко-машинного интерфейса: от разработки проекта отдельно взятой панели оператора до разработки мощных систем человеко-машинного интерфейса с архитектурой клиент/сервер. Оно объединяет в себе простоту работы с пакетом Simatic ProTool и широкие функциональные возможности Simatic WinCC.

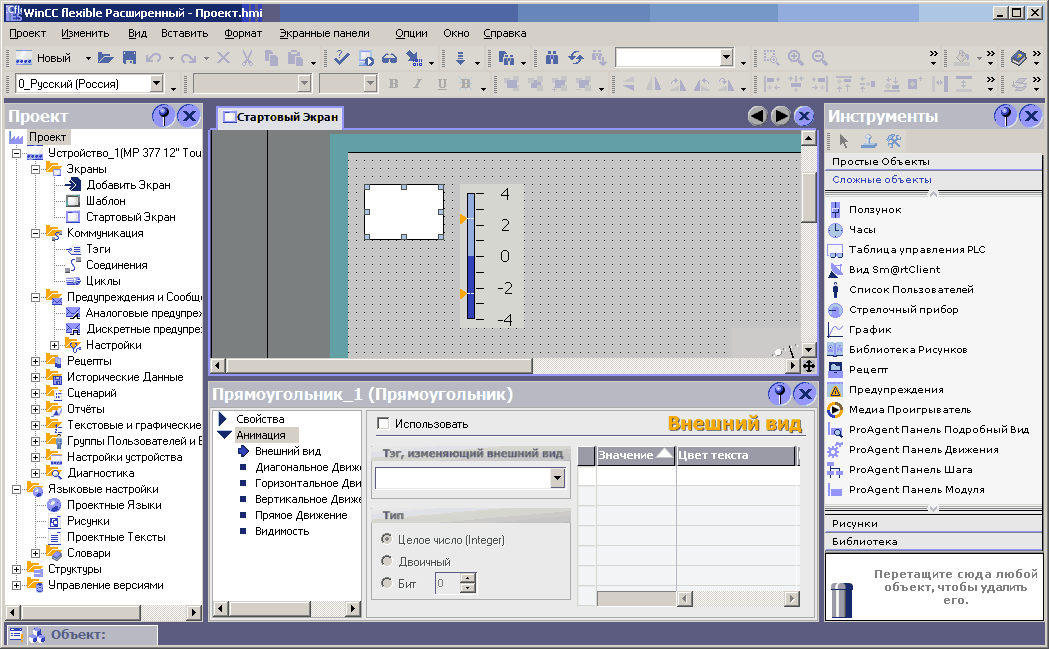


Рисунок 1.2 – WinCC Flexible

Проект в данной среде представляет организационную структуру, состоящую из разделов: экраны, коммуникация, сообщения, рецепты, исторические данные, сценарий, отчеты, текстовые и графические списки, группы пользователей, настройки устройства, языковые настройки, структуры и управления версия. Все это позволяет создать разностороннюю программу визуализации, применяя те или иные средства.

Экраны позволяют задавать содержимое, которое будет выводить на панель операторов (рисунок 1.3 и рисунок 1.4).

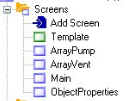


Рисунок 1.3 – Набор экранов

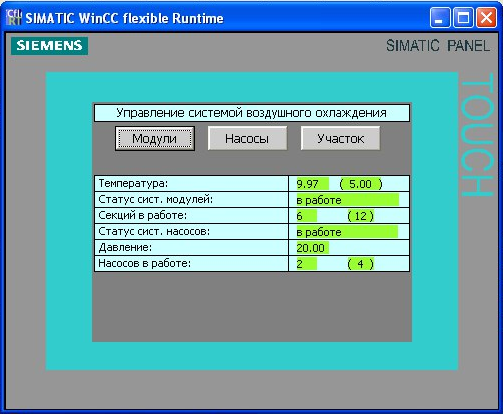


Рисунок 1.4 – Пример экрана

Коммуникация определяет связь с программой автоматизации: подключение и выбранные данные (рисунок 1.5 и рисунок 1.6).

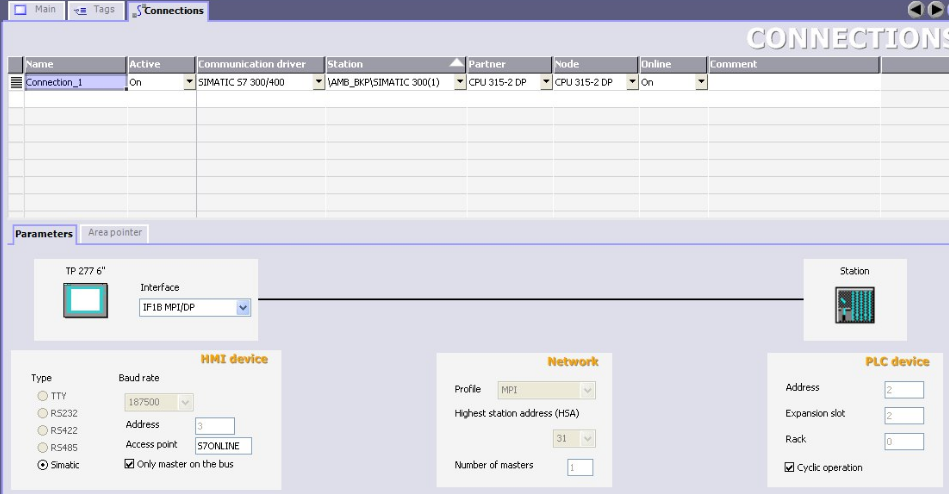


Рисунок 1.5 – Установка соединения с ПЛК

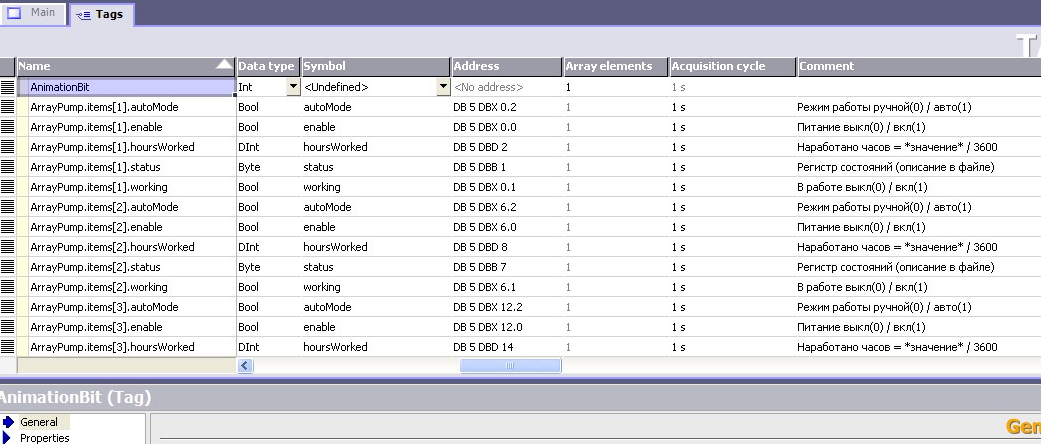


Рисунок 1.6 – Выборка данных из ПЛК

Текстовые и графические списки используются непосредственно в представлении информации на панелях экрана (рисунок 1.7 и рисунок 1.8).

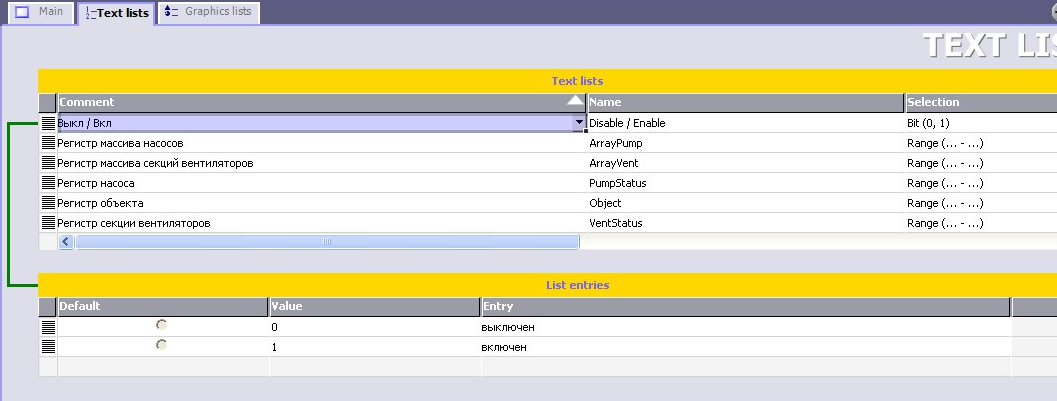


Рисунок 1.7 – Текстовые списки

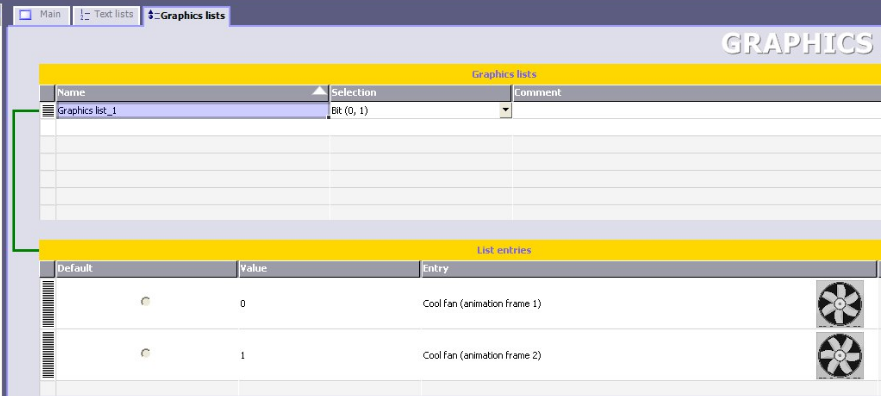


Рисунок 1.8 – Графические списки

В ходе работы возникла потребность в тестировании программного обеспечения для логического контроллера, применяемого для управления системы охлаждения. Поэтому было принято решения для создания программного обеспечения для симуляции оборудования – сигнальных и функциональных модулей для работы с входными и выходными сигналами. Для реализации данного программного обеспечения использовалась среда программирования Microsoft Visual C# 2008 и язык программирования C#, а также программа симуляции входных сигналов Simatic S7-PLCSIM (рисунок 1.9).

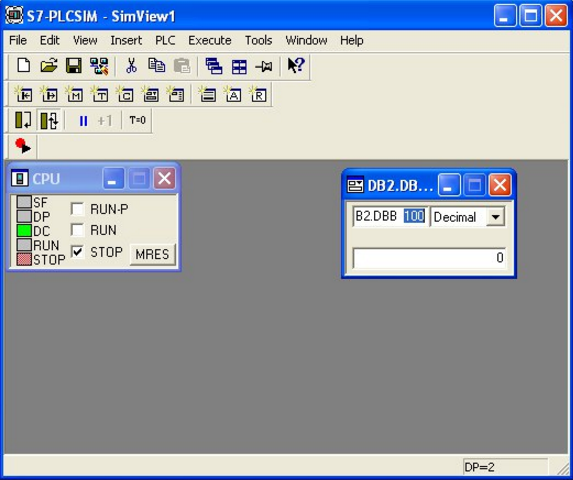


Рисунок 1.9 – Утилита S7-PLCSIM

Утилита Simatic S7-PLCSIM имеет ограниченный набор инструментов для симуляции оборудования. Например, невозможно задать какую-либо функцию, по которой должно вычисляться нужное значение в данный момент времени. Однако, для таких случает Simatic S7-PLCSIM предусматривает специальный COM интерфейс S7ProSim, с помощью которого можно расширить возможности данной утилиты.

COM (англ. Component Object Model — объектная модель компонентов; произносится как [ком]) — это технологический стандарт от компании Microsoft, предназначенный для создания программного обеспечения на основе взаимодействующих компонентов, каждый из которых может использоваться во многих программах одновременно. Стандарт воплощает в себе идеи полиморфизма и инкапсуляции объектно-ориентированного программирования.

Component Object Model поддерживает несколько языков программирования: Visual Basic 6.0, Visual Basic .Net, Visual C++ 6.0 и C#.

Данная программа (рисунок 1.10) была выполнена с помощью языка программирования C# и объектно-ориентированного подхода.



Рисунок 1.10 – Симулятор оборудования

В программе заданы входные и выходные сигналы оборудования, которые необходимо симулировать, а также функции, по которым задаются необходимые значения. Например, значение текущей температуры генерируется в программе с помощью следующей функции (рисунок 1.11).

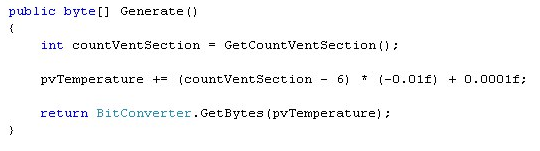


Рисунок 1.11 – Расчет температуры

Функционал симулятора включает в себя включение и выключение (рисунок 2.10).

## Функциональное назначение

Программное обеспечение предназначено для визуализации управления системой охлаждения производственного объекта, а также для симуляции оборудования.

## Описание логической структуры

Данная работы включает в себя две подпрограммы: визуализация и симуляция (рисунок 1.12).

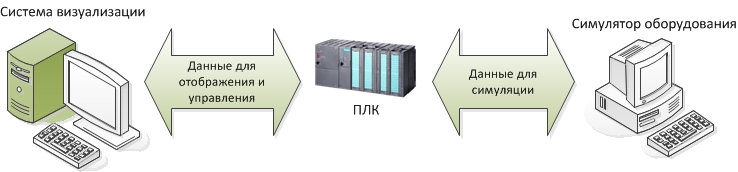


Рисунок 1.12 – Структура проекта

Программная реализация визуализации – это проект в среде WinCC Flexbile, который состоит из набора экранов (рисунок 1.3), отображающие различную информацию о объекте автоматизации:

* Main – главное окно панели. На ней отображается общая информация о объекте и происходящих процессах (рисунок 1.13);

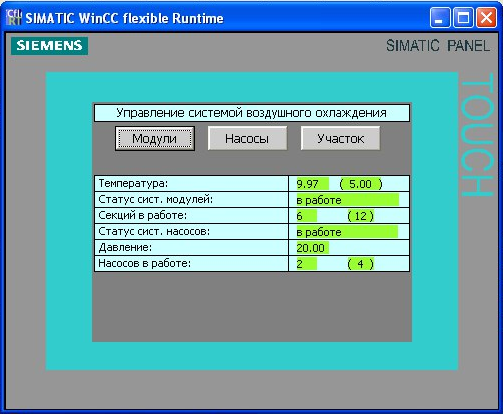


Рисунок 1.13 – Главное окно

* ArrayVent – выводит информацию о секция вентиляторов (рисунок 2.14);

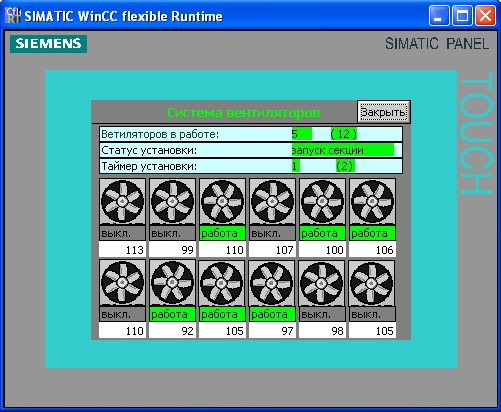


Рисунок 1.14 – Окно ArrayVent

* ArrayPump – выводит информацию о насосах (рисунок 1.15);

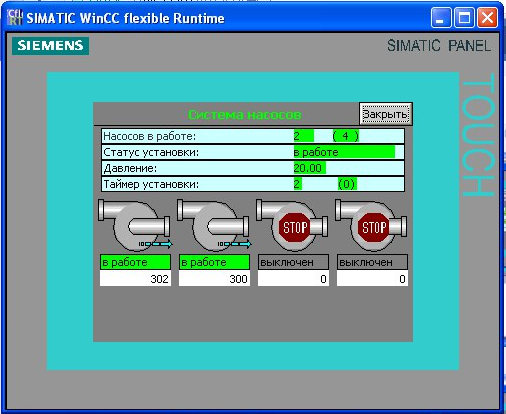


Рисунок 1.15 – Окно ArrayPump

* ObjectProperties – выводит текущую температуре на объекте (рисунок 2.16);

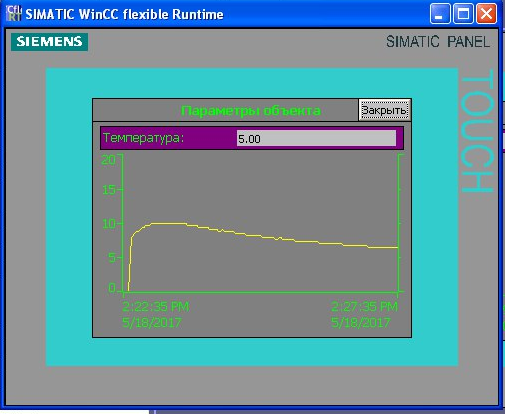


Рисунок 1.16 – Окно ObjectProperties

В основе каждого экрана лежат простые элементы и графические объекты (рисунок 1.17), которые предоставлены средой WinCC Flexible, а также данные из набора тегов, которые необходимо отобразить (рисунок 1.18).

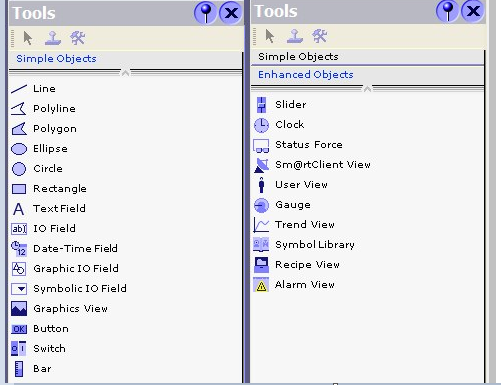


Рисунок 1.17 – Инструменты разработки

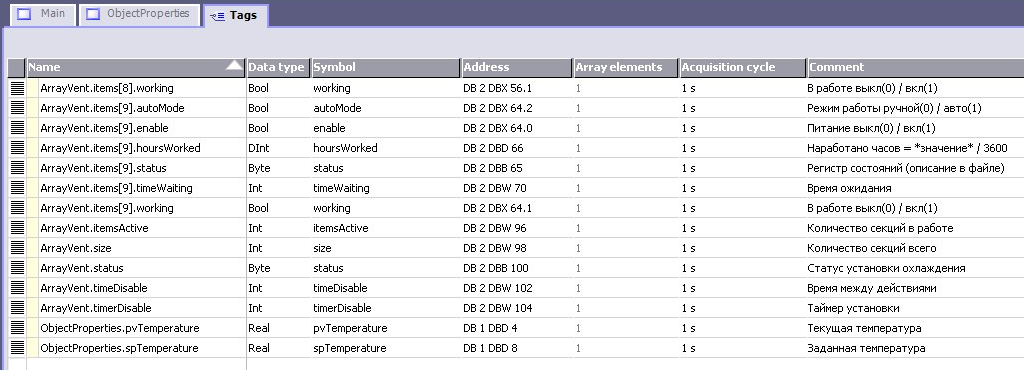


Рисунок 1.18 – Набор тегов программы

Программная реализация симулятора – это программа, разработанная с помощью среды Microsoft Visual C# 2008 и языка программирования C# (рисунок 1.19).

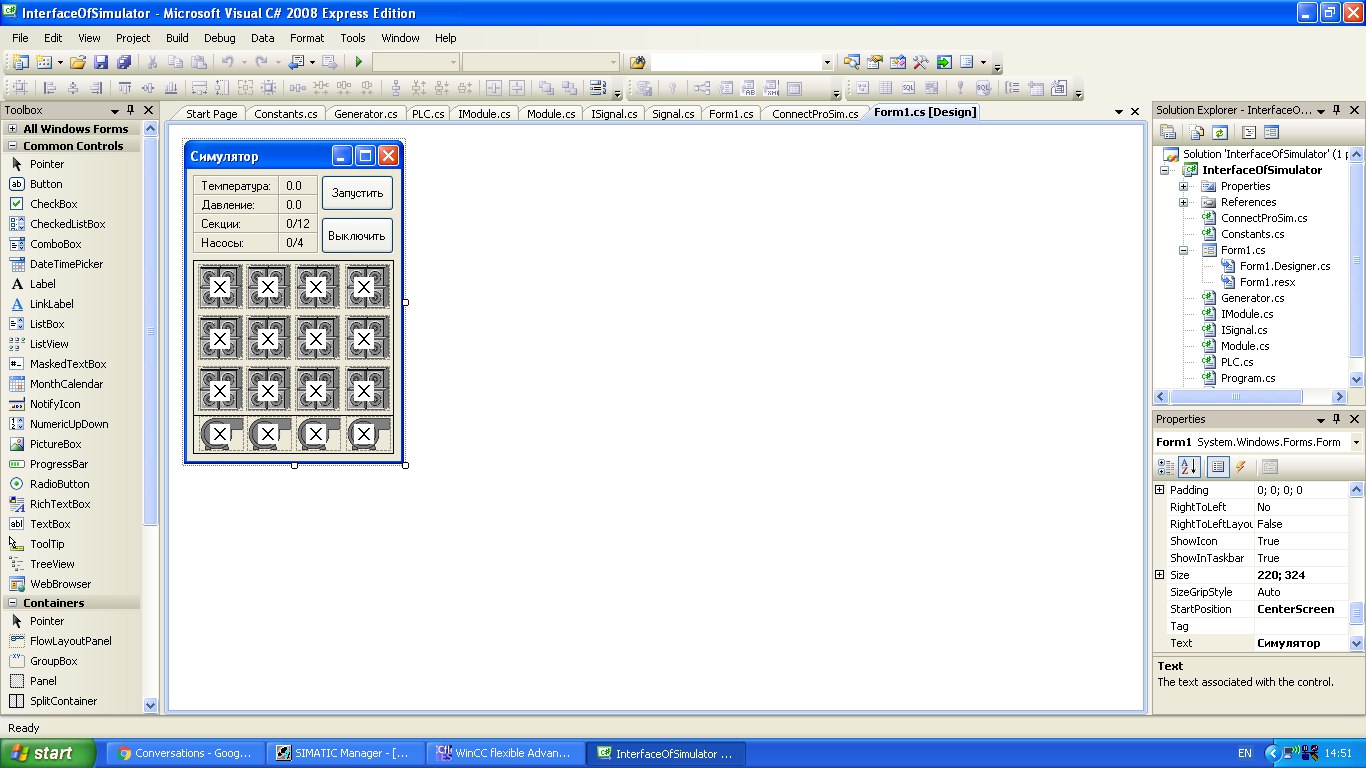


Рисунок 1.19 – Проект Microsoft Visual C# 2008

Данная программа, по заданным в ней алгоритмам, генерирует необходимые входные сигналы, которые использует программируемый логический контроллер для анализа текущей работы объекта автоматизации. Таким образом происходит симуляция оборудования.

Структура программы приведена на рисунке 1.20:

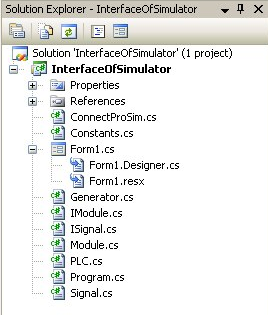


Рисунок 1.20 – Структура программы

* Form – описывает главное окно программы;
* Constants – класс, описывающий набор констант и наименований данной программы;
* Signal – файл, описывающий классы цифровых и аналоговых сигналов, а также коллекцию, энумератор и компаратор для работы с ними: Signal, SignalDigital, SignalAnalog, SignalDigitalInput, SignalAnalogInput, SignalDigitalOutput, SignalCollection, SignalEnumerator и SignalComparator;
* Module – файл, описывающий классы различных модулей: Module, ModuleInput, ModuleOutput;
* PLC – класс, симулирующий работу оборудования; Реализует цикл работы оборудования и паттерн синглтон;
* Generator – класс, описывающий различные функций, которые генерируют нужные значения сигналов. Реализует паттерн стратегия;
* ConnectProSim – класс, реализующий соединение с утилитой S7 ProSim. Реализует паттерн синглтон.

## Используемые технические средства

Техническое обеспечение необходимое для работы программы:

* оперативная память 512 Мб и выше;
* процессор Intel Pentium IV 800 МГц и выше;
* существование логических и/или физических дисков со свободным дисковым пространством 2 Мб и выше;
* клавиатура;
* манипулятор «мышь».

## Входные данные

Входными данными для программного обеспечения «Программная визуализация системы управления охлаждением производственного объекта» являются:

* Управляющие выходные сигналы программы ПЛК Siemens S7-300

## Выходные данные

Выходными данными программного обеспечения «Программная визуализация системы управления охлаждением производственного объекта» являются:

* Сгенерированные программой симулятора показания датчиков и оборудования

## Архитектура программного обеспечения

Схема организации логики программного обеспечения представлена на рисунке 1.21.

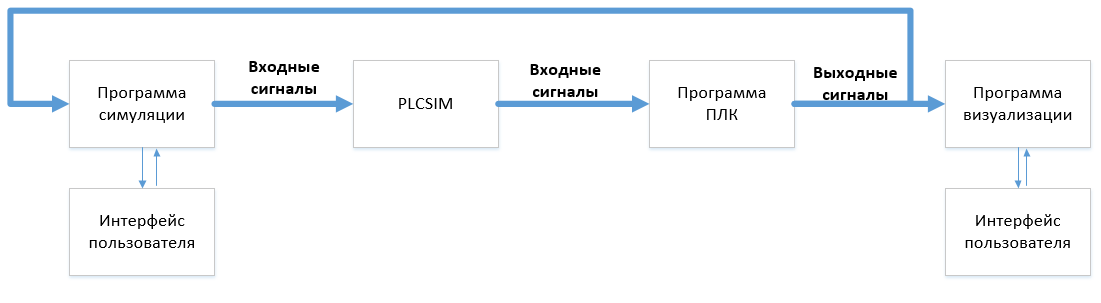
 Рисунок 1.21 – Схема организации логики приложения

Схема организации логики программы симуляции представлена на рисунке 1.22.

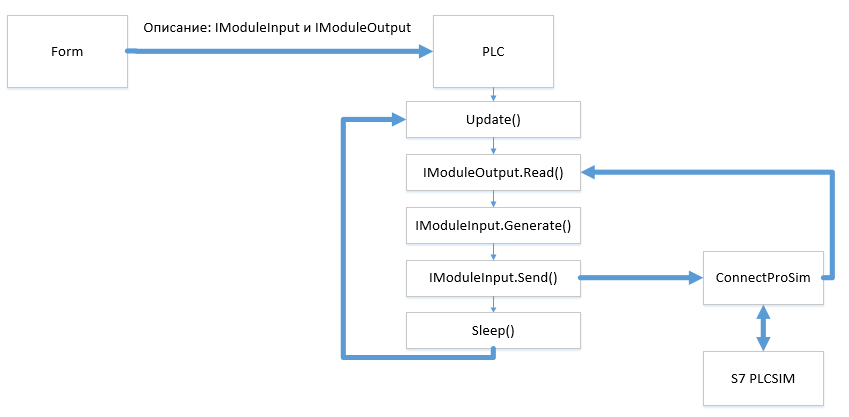


Рисунок 1.22 – Схема организации логики программы симуляции

Схема организации логики программы визуализации представлена на рисунке 1.23.

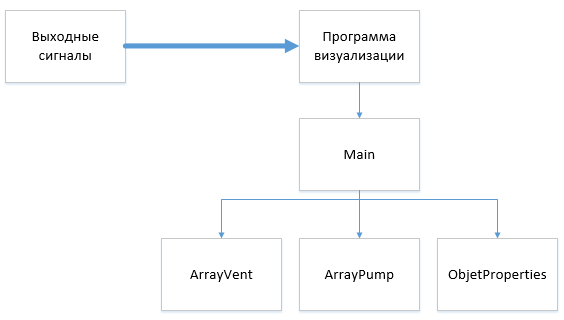


Рисунок 1.23 – Схема организации логики программы визуализации

# Текст программы

## Главный модуль программы

Form.cs – описывает главное окно программы

|  |
| --- |
| 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.Threading;  using System.Reflection;  using S7PROSIMLib;  namespace InterfaceOfSimulator  {  public partial class Form1 : Form  {  delegate void SetTextCallback(string text);  public Form1()  {  InitializeComponent();  }    private void UpdateUI()  {  while (mIsWorkUI)  {  this.labelTemperature.BeginInvoke((MethodInvoker)(() => this.labelTemperature.Text = Generator.GetTemperature() + ""));  this.labelPressure.BeginInvoke((MethodInvoker)(() => this.labelPressure.Text = Generator.GetPressure() + ""));  this.labelCountVentSection.BeginInvoke((MethodInvoker)(() => this.labelCountVentSection.Text = Generator.GetVentSection() + "/" + Constants.COUNT\_VENT\_SECTION));  this.labelCountPump.BeginInvoke((MethodInvoker)(() => this.labelCountPump.Text = Generator.GetPump() + "/" + Constants.COUNT\_PUMP));  int index = 1;  for (int curByte = 24; curByte < 26; curByte++)  {  for (int curBit = 0; curBit < Constants.BYTE\_SIZE; curBit++)  {  ISignal signalDO = PLC.Instance.GetSignalOutput(curByte, curBit);  bool isWork = BitConverter.ToBoolean(signalDO.Data, 0);  Label currentLabel = new Label();  currentLabel = (Label)(this.Controls.Find("labelSignalDO" + index, true).FirstOrDefault());  if (!isWork)  {  currentLabel.BeginInvoke((MethodInvoker)(() => currentLabel.Text = "X"));  currentLabel.BackColor = System.Drawing.Color.Red;  }  else  {  currentLabel.BeginInvoke((MethodInvoker)(() => currentLabel.Text = "O"));  currentLabel.BackColor = System.Drawing.Color.GreenYellow;  }  index++;  }  }  Thread.Sleep(200);  }  }  private void button1\_Click(object sender, EventArgs e)  {  ConnectProSim.Instance.Connect();  ConnectProSim.Instance.Connection.SetScanMode(ScanModeConstants.ContinuousScan);  //выходные статусы секций вентиляторов  SignalDigitalOutput signalDO\_24\_0 = new SignalDigitalOutput(24, 0, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.Q\_VS\_WORKING);  SignalDigitalOutput signalDO\_24\_1 = new SignalDigitalOutput(24, 1, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.Q\_VS\_WORKING);  SignalDigitalOutput signalDO\_24\_2 = new SignalDigitalOutput(24, 2, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.Q\_VS\_WORKING);  SignalDigitalOutput signalDO\_24\_3 = new SignalDigitalOutput(24, 3, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.Q\_VS\_WORKING);  SignalDigitalOutput signalDO\_24\_4 = new SignalDigitalOutput(24, 4, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.Q\_VS\_WORKING);  SignalDigitalOutput signalDO\_24\_5 = new SignalDigitalOutput(24, 5, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.Q\_VS\_WORKING);  SignalDigitalOutput signalDO\_24\_6 = new SignalDigitalOutput(24, 6, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.Q\_VS\_WORKING);  SignalDigitalOutput signalDO\_24\_7 = new SignalDigitalOutput(24, 7, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.Q\_VS\_WORKING);  SignalDigitalOutput signalDO\_25\_0 = new SignalDigitalOutput(25, 0, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.Q\_VS\_WORKING);  SignalDigitalOutput signalDO\_25\_1 = new SignalDigitalOutput(25, 1, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.Q\_VS\_WORKING);  SignalDigitalOutput signalDO\_25\_2 = new SignalDigitalOutput(25, 2, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.Q\_VS\_WORKING);  SignalDigitalOutput signalDO\_25\_3 = new SignalDigitalOutput(25, 3, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.Q\_VS\_WORKING);  //выходные статусы насосов  SignalDigitalOutput signalDO\_25\_4 = new SignalDigitalOutput(25, 4, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.Q\_PUMP\_WORKING);  SignalDigitalOutput signalDO\_25\_5 = new SignalDigitalOutput(25, 5, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.Q\_PUMP\_WORKING);  SignalDigitalOutput signalDO\_25\_6 = new SignalDigitalOutput(25, 6, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.Q\_PUMP\_WORKING);  SignalDigitalOutput signalDO\_25\_7 = new SignalDigitalOutput(25, 7, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.Q\_PUMP\_WORKING);  ModuleOutput moduleOutput = new ModuleOutput();  moduleOutput.Add(signalDO\_24\_0);  moduleOutput.Add(signalDO\_24\_1);  moduleOutput.Add(signalDO\_24\_2);  moduleOutput.Add(signalDO\_24\_3);  moduleOutput.Add(signalDO\_24\_4);  moduleOutput.Add(signalDO\_24\_5);  moduleOutput.Add(signalDO\_24\_6);  moduleOutput.Add(signalDO\_24\_7);  moduleOutput.Add(signalDO\_25\_0);  moduleOutput.Add(signalDO\_25\_1);  moduleOutput.Add(signalDO\_25\_2);  moduleOutput.Add(signalDO\_25\_3);  moduleOutput.Add(signalDO\_25\_4);  moduleOutput.Add(signalDO\_25\_5);  moduleOutput.Add(signalDO\_25\_6);  moduleOutput.Add(signalDO\_25\_7);  //секция №1  SignalDigitalInput signalDI\_16\_0 = new SignalDigitalInput(16, 0, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_VS\_ENABLE, new Generator.ConstantBit(true));  SignalDigitalInput signalDI\_16\_1 = new SignalDigitalInput(16, 1, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_VS\_WORKING, new Generator.OutputValue(signalDO\_24\_0));  SignalDigitalInput signalDI\_16\_2 = new SignalDigitalInput(16, 2, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_VS\_AUTOMODE, new Generator.ConstantBit(true));  //секция №2  SignalDigitalInput signalDI\_16\_3 = new SignalDigitalInput(16, 3, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_VS\_ENABLE, new Generator.ConstantBit(true));  SignalDigitalInput signalDI\_16\_4 = new SignalDigitalInput(16, 4, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_VS\_WORKING, new Generator.OutputValue(signalDO\_24\_1));  SignalDigitalInput signalDI\_16\_5 = new SignalDigitalInput(16, 5, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_VS\_AUTOMODE, new Generator.ConstantBit(true));  //секция №3  SignalDigitalInput signalDI\_16\_6 = new SignalDigitalInput(16, 6, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_VS\_ENABLE, new Generator.ConstantBit(true));  SignalDigitalInput signalDI\_16\_7 = new SignalDigitalInput(16, 7, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_VS\_WORKING, new Generator.OutputValue(signalDO\_24\_2));  SignalDigitalInput signalDI\_17\_0 = new SignalDigitalInput(17, 0, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_VS\_AUTOMODE, new Generator.ConstantBit(true));  //секция №4  SignalDigitalInput signalDI\_17\_1 = new SignalDigitalInput(17, 1, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_VS\_ENABLE, new Generator.ConstantBit(true));  SignalDigitalInput signalDI\_17\_2 = new SignalDigitalInput(17, 2, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_VS\_WORKING, new Generator.OutputValue(signalDO\_24\_3));  SignalDigitalInput signalDI\_17\_3 = new SignalDigitalInput(17, 3, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_VS\_AUTOMODE, new Generator.ConstantBit(true));  //секция №5  SignalDigitalInput signalDI\_17\_4 = new SignalDigitalInput(17, 4, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_VS\_ENABLE, new Generator.ConstantBit(true));  SignalDigitalInput signalDI\_17\_5 = new SignalDigitalInput(17, 5, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_VS\_WORKING, new Generator.OutputValue(signalDO\_24\_4));  SignalDigitalInput signalDI\_17\_6 = new SignalDigitalInput(17, 6, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_VS\_AUTOMODE, new Generator.ConstantBit(true));  //секция №6  SignalDigitalInput signalDI\_17\_7 = new SignalDigitalInput(17, 7, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_VS\_ENABLE, new Generator.ConstantBit(true));  SignalDigitalInput signalDI\_18\_0 = new SignalDigitalInput(18, 0, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_VS\_WORKING, new Generator.OutputValue(signalDO\_24\_5));  SignalDigitalInput signalDI\_18\_1 = new SignalDigitalInput(18, 1, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_VS\_AUTOMODE, new Generator.ConstantBit(true));  //секция №7  SignalDigitalInput signalDI\_18\_2 = new SignalDigitalInput(18, 2, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_VS\_ENABLE, new Generator.ConstantBit(true));  SignalDigitalInput signalDI\_18\_3 = new SignalDigitalInput(18, 3, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_VS\_WORKING, new Generator.OutputValue(signalDO\_24\_6));  SignalDigitalInput signalDI\_18\_4 = new SignalDigitalInput(18, 4, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_VS\_AUTOMODE, new Generator.ConstantBit(true));  //секция №8  SignalDigitalInput signalDI\_18\_5 = new SignalDigitalInput(18, 5, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_VS\_ENABLE, new Generator.ConstantBit(true));  SignalDigitalInput signalDI\_18\_6 = new SignalDigitalInput(18, 6, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_VS\_WORKING, new Generator.OutputValue(signalDO\_24\_7));  SignalDigitalInput signalDI\_18\_7 = new SignalDigitalInput(18, 7, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_VS\_AUTOMODE, new Generator.ConstantBit(true));  //секция №9  SignalDigitalInput signalDI\_19\_0 = new SignalDigitalInput(19, 0, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_VS\_ENABLE, new Generator.ConstantBit(true));  SignalDigitalInput signalDI\_19\_1 = new SignalDigitalInput(19, 1, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_VS\_WORKING, new Generator.OutputValue(signalDO\_25\_0));  SignalDigitalInput signalDI\_19\_2 = new SignalDigitalInput(19, 2, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_VS\_AUTOMODE, new Generator.ConstantBit(true));  //секция №10  SignalDigitalInput signalDI\_19\_3 = new SignalDigitalInput(19, 3, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_VS\_ENABLE, new Generator.ConstantBit(true));  SignalDigitalInput signalDI\_19\_4 = new SignalDigitalInput(19, 4, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_VS\_WORKING, new Generator.OutputValue(signalDO\_25\_1));  SignalDigitalInput signalDI\_19\_5 = new SignalDigitalInput(19, 5, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_VS\_AUTOMODE, new Generator.ConstantBit(true));  //секция №11  SignalDigitalInput signalDI\_19\_6 = new SignalDigitalInput(19, 6, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_VS\_ENABLE, new Generator.ConstantBit(true));  SignalDigitalInput signalDI\_19\_7 = new SignalDigitalInput(19, 7, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_VS\_WORKING, new Generator.OutputValue(signalDO\_25\_2));  SignalDigitalInput signalDI\_20\_0 = new SignalDigitalInput(20, 0, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_VS\_AUTOMODE, new Generator.ConstantBit(true));  //секция №12  SignalDigitalInput signalDI\_20\_1 = new SignalDigitalInput(20, 1, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_VS\_ENABLE, new Generator.ConstantBit(true));  SignalDigitalInput signalDI\_20\_2 = new SignalDigitalInput(20, 2, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_VS\_WORKING, new Generator.OutputValue(signalDO\_25\_3));  SignalDigitalInput signalDI\_20\_3 = new SignalDigitalInput(20, 3, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_VS\_AUTOMODE, new Generator.ConstantBit(true));  //насос №1  SignalDigitalInput signalDI\_20\_4 = new SignalDigitalInput(20, 4, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_PUMP\_ENABLE, new Generator.ConstantBit(true));  SignalDigitalInput signalDI\_20\_5 = new SignalDigitalInput(20, 5, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_PUMP\_WORKING, new Generator.OutputValue(signalDO\_25\_4));  SignalDigitalInput signalDI\_20\_6 = new SignalDigitalInput(20, 6, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_PUMP\_AUTOMODE, new Generator.ConstantBit(true));  //насос №2  SignalDigitalInput signalDI\_20\_7 = new SignalDigitalInput(20, 7, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_PUMP\_ENABLE, new Generator.ConstantBit(true));  SignalDigitalInput signalDI\_21\_0 = new SignalDigitalInput(21, 0, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_PUMP\_WORKING, new Generator.OutputValue(signalDO\_25\_5));  SignalDigitalInput signalDI\_21\_1 = new SignalDigitalInput(21, 1, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_PUMP\_AUTOMODE, new Generator.ConstantBit(true));  //насос №3  SignalDigitalInput signalDI\_21\_2 = new SignalDigitalInput(21, 2, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_PUMP\_ENABLE, new Generator.ConstantBit(true));  SignalDigitalInput signalDI\_21\_3 = new SignalDigitalInput(21, 3, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_PUMP\_WORKING, new Generator.OutputValue(signalDO\_25\_6));  SignalDigitalInput signalDI\_21\_4 = new SignalDigitalInput(21, 4, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_PUMP\_AUTOMODE, new Generator.ConstantBit(true));  //насос №4  SignalDigitalInput signalDI\_21\_5 = new SignalDigitalInput(21, 5, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_PUMP\_ENABLE, new Generator.ConstantBit(true));  SignalDigitalInput signalDI\_21\_6 = new SignalDigitalInput(21, 6, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_PUMP\_WORKING, new Generator.OutputValue(signalDO\_25\_7));  SignalDigitalInput signalDI\_21\_7 = new SignalDigitalInput(21, 7, PointDataTypeConstants.S7\_Bit, Constants.TitleSignal.I\_PUMP\_AUTOMODE, new Generator.ConstantBit(true));  ModuleInput moduleInput = new ModuleInput();  moduleInput.Add(signalDI\_16\_0);  moduleInput.Add(signalDI\_16\_1);  moduleInput.Add(signalDI\_16\_2);  moduleInput.Add(signalDI\_16\_3);  moduleInput.Add(signalDI\_16\_4);  moduleInput.Add(signalDI\_16\_5);  moduleInput.Add(signalDI\_16\_6);  moduleInput.Add(signalDI\_16\_7);  moduleInput.Add(signalDI\_17\_0);  moduleInput.Add(signalDI\_17\_1);  moduleInput.Add(signalDI\_17\_2);  moduleInput.Add(signalDI\_17\_3);  moduleInput.Add(signalDI\_17\_4);  moduleInput.Add(signalDI\_17\_5);  moduleInput.Add(signalDI\_17\_6);  moduleInput.Add(signalDI\_17\_7);  moduleInput.Add(signalDI\_18\_0);  moduleInput.Add(signalDI\_18\_1);  moduleInput.Add(signalDI\_18\_2);  moduleInput.Add(signalDI\_18\_3);  moduleInput.Add(signalDI\_18\_4);  moduleInput.Add(signalDI\_18\_5);  moduleInput.Add(signalDI\_18\_6);  moduleInput.Add(signalDI\_18\_7);  moduleInput.Add(signalDI\_19\_0);  moduleInput.Add(signalDI\_19\_1);  moduleInput.Add(signalDI\_19\_2);  moduleInput.Add(signalDI\_19\_3);  moduleInput.Add(signalDI\_19\_4);  moduleInput.Add(signalDI\_19\_5);  moduleInput.Add(signalDI\_19\_6);  moduleInput.Add(signalDI\_19\_7);  moduleInput.Add(signalDI\_20\_0);  moduleInput.Add(signalDI\_20\_1);  moduleInput.Add(signalDI\_20\_2);  moduleInput.Add(signalDI\_20\_3);  moduleInput.Add(signalDI\_20\_4);  moduleInput.Add(signalDI\_20\_5);  moduleInput.Add(signalDI\_20\_6);  moduleInput.Add(signalDI\_20\_7);  moduleInput.Add(signalDI\_21\_0);  moduleInput.Add(signalDI\_21\_1);  moduleInput.Add(signalDI\_21\_2);  moduleInput.Add(signalDI\_21\_3);  moduleInput.Add(signalDI\_21\_4);  moduleInput.Add(signalDI\_21\_5);  moduleInput.Add(signalDI\_21\_6);  moduleInput.Add(signalDI\_21\_7);  SignalAnalogInput signalAI\_72 = new SignalAnalogInput(72, PointDataTypeConstants.S7\_DoubleWord, Constants.TitleSignal.ID\_PVTEMPERATURE, new Generator.CurrentTemperature(moduleOutput));  moduleInput.Add(signalAI\_72);  SignalAnalogInput signalAI\_78 = new SignalAnalogInput(78, PointDataTypeConstants.S7\_DoubleWord, Constants.TitleSignal.ID\_PRESSURE, new Generator.CurrentPressure(moduleOutput));  moduleInput.Add(signalAI\_78);  PLC.Instance.Load();  PLC.Instance.AddModuleOutput(moduleOutput);  PLC.Instance.AddModuleInput(moduleInput);  PLC.Instance.Start();  mIsWorkUI = true;  mThreadUI = new Thread(UpdateUI);  mThreadUI.Start();  }  private void button2\_Click(object sender, EventArgs e)  {  PLC.Instance.Stop();  ConnectProSim.Instance.Disconnect();  mIsWorkUI = false;  mThreadUI.Join();  }  private void Form1\_FormClosing(object sender, FormClosingEventArgs e)  {  ConnectProSim.Instance.Disconnect();  }  private Thread mThreadUI;  private volatile bool mIsWorkUI;  }  } |

## Модуль Constants

Constants – класс, описывающий набор констант и наименований данной программы;

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace InterfaceOfSimulator  {  public static class Constants  {  public static class TitleSignal  {  public static string I\_VS\_ENABLE = "I\_VS\_ENABLE";  public static string I\_VS\_WORKING = "I\_VS\_WORKING";  public static string I\_VS\_AUTOMODE = "I\_VS\_AUTOMODE";  public static string Q\_VS\_WORKING = "Q\_VS\_WORKING";  public static string I\_PUMP\_ENABLE = "I\_PUMP\_ENABLE";  public static string I\_PUMP\_WORKING = "I\_PUMP\_WORKING";  public static string I\_PUMP\_AUTOMODE = "I\_PUMP\_AUTOMODE";  public static string Q\_PUMP\_WORKING = "Q\_PUMP\_WORKING";  public static string ID\_PVTEMPERATURE = "ID\_PVTEMPERATURE";  public static string ID\_PRESSURE = "ID\_PRESSURE";  }  public static int COUNT\_VENT\_SECTION = 12;  public static int COUNT\_PUMP = 4;  public static int BYTE\_SIZE = 8;  }  } |

## Модуль Signal

Signal – файл, описывающий классы цифровых и аналоговых сигналов, а также коллекцию, энумератор и компаратор для работы с ними: Signal, SignalDigital, SignalAnalog, SignalDigitalInput, SignalAnalogInput, SignalDigitalOutput, SignalCollection, SignalEnumerator и SignalComparator;

|  |
| --- |
| using System;  using System.Collections;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using S7PROSIMLib;  namespace InterfaceOfSimulator  {  public class Signal : ISignal  {  public Signal(int byteIndex, PointDataTypeConstants dataType, string title)  {  ByteIndex = byteIndex;  DataType = dataType;  Data = BitConverter.GetBytes(0);  Title = title;  }  public int ByteIndex  {  get { return mByteIndex; }  private set { mByteIndex = value; }  }  public PointDataTypeConstants DataType  {  get { return mDataType; }  private set { mDataType = value; }  }  public virtual byte[] Data  {  get { return mData; }  set { mData = value; }  }  public virtual Object GetDataObject()  {  Object tempObject = null;  if(DataType == PointDataTypeConstants.S7\_Bit)  {  tempObject = BitConverter.ToBoolean(Data, 0);  }  else if(DataType == PointDataTypeConstants.S7\_Byte)  {  tempObject = BitConverter.ToInt16(Data, 0);  }  else if (DataType == PointDataTypeConstants.S7\_Word)  {  tempObject = BitConverter.ToSingle(Data, 0);  }  else if (DataType == PointDataTypeConstants.S7\_DoubleWord)  {  tempObject = BitConverter.ToSingle(Data, 0);  }  return tempObject;  }  public virtual string Title  {  get { return mTitle; }  private set { mTitle = value; }  }  private int mByteIndex;  private PointDataTypeConstants mDataType;  private byte[] mData;  private string mTitle;  }  public class SignalDigital : Signal, ISignalDigital  {  public SignalDigital(int byteIndex, int bitIndex, PointDataTypeConstants dataType, string title)  : base(byteIndex, dataType, title)  {  BitIndex = bitIndex;  }  public int BitIndex  {  get { return mBitIndex; }  private set { mBitIndex = value; }  }  private int mBitIndex;  }  public class SignalAnalog : Signal, ISignalAnalog  {  public SignalAnalog(int byteIndex, PointDataTypeConstants dataType, string title)  : base(byteIndex, dataType, title)  {  }  }  public sealed class SignalDigitalInput : SignalDigital, ISignalInput  {  public SignalDigitalInput(int byteIndex, int bitIndex, PointDataTypeConstants dataType, string title, IGenerationMethod method)  : base(byteIndex, bitIndex, dataType, title)  {  mGenerationMethod = method;  }  public void Generate()  {  Data = mGenerationMethod.Generate();  }  public void Send()  {  Object dataReformed = GetDataObject();  ConnectProSim.Instance.Connection.WriteInputPoint(ByteIndex, BitIndex, ref dataReformed);  if (System.Runtime.InteropServices.Marshal.IsComObject(dataReformed))  System.Runtime.InteropServices.Marshal.ReleaseComObject(dataReformed);  dataReformed = null;  }  private IGenerationMethod mGenerationMethod;  }  public sealed class SignalAnalogInput : SignalAnalog, ISignalInput  {  public SignalAnalogInput(int byteIndex, PointDataTypeConstants dataType, string title, IGenerationMethod method)  : base(byteIndex, dataType, title)  {  mGenerationMethod = method;  }  public void Generate()  {  Data = mGenerationMethod.Generate();  }  public void Send()  {  byte[] bufferArray = Data;  Array.Reverse(bufferArray);  Object dataReformed = new Object();  dataReformed = bufferArray;  ConnectProSim.Instance.Connection.WriteInputImage(ByteIndex, ref dataReformed);  }  private IGenerationMethod mGenerationMethod;  }  public sealed class SignalDigitalOutput : SignalDigital, ISignalOutput  {  public SignalDigitalOutput(int byteIndex, int bitIndex, PointDataTypeConstants dataType, string title)  : base(byteIndex, bitIndex, dataType, title)  {  }  public void Read()  {  Object tempData = null;  ConnectProSim.Instance.Connection.ReadOutputPoint(ByteIndex, BitIndex, DataType, ref tempData);  bool value = false;  if (tempData is bool)  {  value = (bool)tempData;  }  if(System.Runtime.InteropServices.Marshal.IsComObject(tempData))  System.Runtime.InteropServices.Marshal.ReleaseComObject(tempData);  tempData = null;  Data = BitConverter.GetBytes(value);  }  }  public sealed class SignalCollection : ISignalCollection  {  public SignalCollection()  {  mSignalCollection = new List<ISignal>();  }  public ISignal this[int index]  {  get { return (ISignal)mSignalCollection[index]; }  set { mSignalCollection[index] = value; }  }  public IEnumerator<ISignal> GetEnumerator()  {  return new SignalEnumerator(this);  }  IEnumerator IEnumerable.GetEnumerator()  {  return new SignalEnumerator(this);  }  public bool Contains(ISignal signal)  {  bool isContains = false;  if (signal is ISignalDigital)  {  foreach (ISignal item in mSignalCollection)  {  if (item is ISignalDigital && new SignalComparator().Equals((ISignalDigital) item, (ISignalDigital) signal))  {  isContains = true;  break;  }  }  }  else if (signal is ISignalAnalog)  {  foreach (ISignal item in mSignalCollection)  {  if(item is ISignalAnalog && new SignalComparator().Equals((ISignalAnalog) item, (ISignalAnalog) signal))  {  isContains = true;  break;  }  }  }    return isContains;  }  public bool Contains(int byteIndex, int bitIndex)  {  bool isContains = false;  foreach (ISignal item in mSignalCollection)  {  if (item is ISignalDigital && new SignalComparator().Equals((ISignalDigital)item, byteIndex, bitIndex))  {  isContains = true;  break;  }  }  return isContains;  }  public bool Contains(int byteIndex)  {  bool isContains = false;  foreach (ISignal item in mSignalCollection)  {  if (item is ISignalAnalog && new SignalComparator().Equals((ISignalAnalog)item, byteIndex))  {  isContains = true;  break;  }  }  return isContains;  }  public void Add(ISignal item)  {  if (!Contains(item))  {  mSignalCollection.Add(item);  }  }  public void Clear()  {  mSignalCollection.Clear();  }  public void CopyTo(ISignal[] array, int arrayIndex)  {  if (array == null)  throw new ArgumentNullException("The array can't be null!");  if (arrayIndex < 0)  throw new ArgumentOutOfRangeException("The starting array index can't be negative!");  if (Count > array.Length - arrayIndex + 1)  throw new ArgumentException("The destination array has fewer elements than the collection!");  for (int i = 0; i < mSignalCollection.Count; i++)  {  array[i + arrayIndex] = mSignalCollection[i];  }  }  public int Count  {  get { return mSignalCollection.Count; }  }  public bool IsReadOnly  {  get { return false; }  }  public bool Remove(ISignal item)  {  bool isRemove = false;  for (int i = 0; i < mSignalCollection.Count; i++)  {  ISignal currentItem = mSignalCollection[i];  if (new SignalComparator().Equals(currentItem, item))  {  mSignalCollection.RemoveAt(i);  isRemove = true;  break;  }  }  return isRemove;  }  private List<ISignal> mSignalCollection;  }  public sealed class SignalEnumerator : IEnumerator<ISignal>  {  public SignalEnumerator(SignalCollection collection)  {  mSignalCollection = collection;  mCurrentIndex = -1;  mCurrentSignal = default(ISignal);  }  public bool MoveNext()  {  if (++mCurrentIndex >= mSignalCollection.Count())  {  return false;  }  else  {  mCurrentSignal = mSignalCollection[mCurrentIndex];  }  return true;  }  public void Reset()  {  mCurrentIndex = -1;  }  void IDisposable.Dispose()  {  }  public ISignal Current  {  get { return mCurrentSignal; }  }  object IEnumerator.Current  {  get { return Current; }  }  private SignalCollection mSignalCollection;  private int mCurrentIndex;  private ISignal mCurrentSignal;  }  public sealed class SignalComparator : EqualityComparer<ISignal>  {  public override bool Equals(ISignal x, ISignal y)  {  bool isEquals = false;  if (x is ISignalDigital && y is ISignalDigital)  {  isEquals = Equals((ISignalDigital)x, (ISignalDigital)y);  }  else if (x is ISignalAnalog && y is ISignalAnalog)  {  isEquals = Equals((ISignalAnalog)x, (ISignalAnalog)y);  }  return isEquals;  }  public bool Equals(ISignalDigital x, ISignalDigital y)  {  bool isEquals = false;  if (x.ByteIndex == y.ByteIndex && x.BitIndex == y.BitIndex)  {  isEquals = true;  }  return isEquals;  }  public bool Equals(ISignalAnalog x, ISignalAnalog y)  {  bool isEquals = false;  if (x.ByteIndex == y.ByteIndex)  {  isEquals = true;  }  return isEquals;  }  public bool Equals(ISignalDigital signalDigital, int byteIndex, int bitIndex)  {  bool isEquals = false;  if (signalDigital.ByteIndex == byteIndex && signalDigital.BitIndex == bitIndex)  {  isEquals = true;  }  return isEquals;  }  public bool Equals(ISignalAnalog signalAnalog, int byteIndex)  {  bool isEquals = false;  if (signalAnalog.ByteIndex == byteIndex)  {  isEquals = true;  }  return isEquals;  }  public override int GetHashCode(ISignal item)  {  int hashCode = item.ByteIndex ^ item.Data.GetHashCode();  return hashCode.GetHashCode();  }  }  } |

Определение интерфейсов

|  |
| --- |
| using System;  using System.Collections;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using S7PROSIMLib;  namespace InterfaceOfSimulator  {  public interface ISignal  {  int ByteIndex { get; }  PointDataTypeConstants DataType { get; }  byte[] Data { get; }  Object GetDataObject();  string Title { get; }  }  public interface ISignalDigital : ISignal  {  int BitIndex { get; }  }  public interface ISignalAnalog : ISignal  {  }  public interface ISignalInput : ISignal  {  void Generate();  void Send();  }  public interface ISignalOutput : ISignal  {  void Read();  }  public interface ISignalCollection : ICollection<ISignal>  {  bool Contains(int byteIndex, int bitIndex);  bool Contains(int byteIndex);  }  } |

## Компонент Module

Module – файл, описывающий классы различных модулей: Module, ModuleInput, ModuleOutput;

|  |
| --- |
| using System;  using System.Collections;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace InterfaceOfSimulator  {  public class Module : IModule  {  public Module()  {  mSignalCollection = new SignalCollection();  }  public virtual void Add(ISignal signal)  {  mSignalCollection.Add(signal);  }  public virtual ISignal Get(int byteIndex, int bitIndex)  {  ISignal buffer = null;  foreach (ISignal item in mSignalCollection)  {  if (item is ISignalDigital && item.ByteIndex == byteIndex)  {  if (((ISignalDigital)item).BitIndex == bitIndex)  {  buffer = item;  }  }  }  return buffer;  }  public virtual ISignal Get(int byteIndex)  {  ISignal buffer = null;  foreach (ISignal item in mSignalCollection)  {  if (item is ISignalAnalog && item.ByteIndex == byteIndex)  {  buffer = item;  }  }  return buffer;  }  public virtual bool Contains(int byteIndex, int bitIndex)  {  bool isContains = false;  if(mSignalCollection.Contains(byteIndex, byteIndex))  {  isContains = true;  }  return isContains;  }  public virtual bool Contains(int byteIndex)  {  bool isContains = false;  if (mSignalCollection.Contains(byteIndex))  {  isContains = true;  }  return isContains;  }  public virtual ISignalCollection GetSignalCollection()  {  return mSignalCollection;  }    private ISignalCollection mSignalCollection;  }  public sealed class ModuleInput: Module, IModuleInput  {  public ModuleInput() : base()  {    }  public override void Add(ISignal item)  {  if (item is ISignalInput)  {  base.Add(item);  }  }  public void Add(ISignalInput item)  {  base.Add(item);  }  public void Generate()  {  foreach (ISignalInput item in base.GetSignalCollection())  {  item.Generate();  }  }  public void Send()  {  foreach (ISignalInput item in base.GetSignalCollection())  {  item.Send();  }  }  }  public sealed class ModuleOutput : Module, IModuleOutput  {  public ModuleOutput() : base()  {  }  public override void Add(ISignal item)  {  if (item is ISignalOutput)  {  base.Add(item);  }  }  public void Add(ISignalOutput item)  {  base.Add(item);  }  public void Read()  {  foreach (ISignalOutput item in base.GetSignalCollection())  {  item.Read();  }  }  }  public sealed class ModuleCollection : IModuleCollection  {  public ModuleCollection()  {  mModuleCollection = new List<IModule>();  }  public IModule this[int index]  {  get { return (IModule) mModuleCollection[index]; }  set { mModuleCollection[index] = value; }  }  public IEnumerator<IModule> GetEnumerator()  {  return new ModuleEnumerator(this);  }  IEnumerator IEnumerable.GetEnumerator()  {  return new ModuleEnumerator(this);  }  public bool Contains(IModule signal)  {  bool isContains = false;  //-- описать алгоритм поиска одинаковых модулей    return isContains;  }  public void Add(IModule item)  {  if (!Contains(item))  {  mModuleCollection.Add(item);  }  }  public void Clear()  {  mModuleCollection.Clear();  }  public void CopyTo(IModule[] array, int arrayIndex)  {  if (array == null)  throw new ArgumentNullException("The array can't be null!");  if (arrayIndex < 0)  throw new ArgumentOutOfRangeException("The starting array index can't be negative!");  if (Count > array.Length - arrayIndex + 1)  throw new ArgumentException("The destination array has fewer elements than the collection!");  for (int i = 0; i < mModuleCollection.Count; i++)  {  array[i + arrayIndex] = mModuleCollection[i];  }  }  public int Count  {  get { return mModuleCollection.Count; }  }  public bool IsReadOnly  {  get { return false; }  }  public bool Remove(IModule item)  {  bool isRemove = false;  for (int i = 0; i < mModuleCollection.Count; i++)  {  IModule currentItem = mModuleCollection[i];  if (new ModuleComparator().Equals(currentItem, item))  {  mModuleCollection.RemoveAt(i);  isRemove = true;  break;  }  }  return isRemove;  }  private List<IModule> mModuleCollection;  }  public sealed class ModuleEnumerator : IEnumerator<IModule>  {  public ModuleEnumerator(ModuleCollection collection)  {  mModuleCollection = collection;  mCurrentIndex = -1;  mCurrentModule = default(IModule);  }  public bool MoveNext()  {  if (++mCurrentIndex >= mModuleCollection.Count())  {  return false;  }  else  {  mCurrentModule = mModuleCollection[mCurrentIndex];  }  return true;  }  public void Reset()  {  mCurrentIndex = -1;  }  void IDisposable.Dispose()  {  }  public IModule Current  {  get { return mCurrentModule; }  }  object IEnumerator.Current  {  get { return Current; }  }  private ModuleCollection mModuleCollection;  private int mCurrentIndex;  private IModule mCurrentModule;  }  public sealed class ModuleComparator : EqualityComparer<IModule>  {  public override bool Equals(IModule x, IModule y)  {  bool isEquals = false;  //--- сравнение на идентичность  return isEquals;  }  public override int GetHashCode(IModule item)  {  int hashCode = item.GetHashCode();  return hashCode.GetHashCode();  }  }  } |

Определение интерфейсов

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace InterfaceOfSimulator  {  public interface IModule  {  void Add(ISignal signal);  ISignal Get(int byteIndex, int bitIndex);  ISignal Get(int byteIndex);  bool Contains(int byteIndex, int bitIndex);  bool Contains(int byteIndex);  ISignalCollection GetSignalCollection();  }  public interface IModuleInput : IModule  {  void Add(ISignalInput item);  void Send();  void Generate();  }  public interface IModuleOutput : IModule  {  void Add(ISignalOutput item);  void Read();  }  public interface IModuleCollection : ICollection<IModule>  {  }  } |

## Модуль PLC

PLC – класс, симулирующий работу оборудования; Реализует цикл ра-боты оборудования и паттерн синглтон;

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading;  namespace InterfaceOfSimulator  {  public sealed class PLC  {  public static PLC Instance  {  get  {  if (mController == null)  {  mController = new PLC();  }  return mController;  }  }  public void Dispose()  {  }  public PLC Controller  {  private get { return mController; }  set { mController = value; }  }  public void Load()  {  isWork = false;  mThead = new Thread(Update);  mModuleInput = new ModuleInput();  mModuleOutput = new ModuleOutput();  }  public void Start()  {  isWork = true;  mThead.Start();  }  private void Update()  {  while (isWork)  {  mModuleOutput.Read();  mModuleInput.Generate();  mModuleInput.Send();  Thread.Sleep(100);  }  }  public void Stop()  {  isWork = false;  mThead.Join();  }  public void AddModuleInput(IModuleInput moduleInput)  {  mModuleInput = moduleInput;  }  public void AddModuleOutput(IModuleOutput moduleOutput)  {  mModuleOutput = moduleOutput;  }  public ISignal GetSignalInput(int byteIndex, int bitIndex)  {  return mModuleInput.Get(byteIndex, bitIndex);  }  public ISignal GetSignalInput(int byteIndex)  {  return mModuleInput.Get(byteIndex);  }  public ISignal GetSignalOutput(int byteIndex, int bitIndex)  {  return mModuleOutput.Get(byteIndex, bitIndex);  }  public ISignal GetSignalOutput(int byteIndex)  {  return mModuleOutput.Get(byteIndex);  }  private volatile bool isWork;  private Thread mThead;  private IModuleInput mModuleInput;  private IModuleOutput mModuleOutput;  private static PLC mController;  }  } |

## Модуль Generator

Generator – класс, описывающий различные функций, которые генерируют нужные значения сигналов. Реализует паттерн стратегия;

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Data;  namespace InterfaceOfSimulator  {  public interface IGenerationMethod  {  byte[] Generate();  }  public sealed class Generator  {  public sealed class ConstantBit : IGenerationMethod  {  public ConstantBit(bool bit)  {  mBit = bit;  }  public byte[] Generate()  {  return BitConverter.GetBytes(mBit);  }  private bool mBit;  }  public sealed class RandomBit : IGenerationMethod  {  public byte[] Generate()  {  bool result = rnd.Next(2) > 0 ? true : false;  return BitConverter.GetBytes(result);  }  }  public sealed class OutputValue : IGenerationMethod  {  public OutputValue(ISignalOutput signalOutput)  {  mSignalOutput = signalOutput;  }  public byte[] Generate()  {  return mSignalOutput.Data;  }  private ISignalOutput mSignalOutput;  }  public sealed class CurrentTemperature : IGenerationMethod  {  public CurrentTemperature(IModuleOutput moduleOutput)  {  mModuleOutput = moduleOutput;  }  public byte[] Generate()  {  int countVentSection = GetCountVentSection();  pvTemperature += (countVentSection - 6) \* (-0.01f) + 0.0001f;  return BitConverter.GetBytes(pvTemperature);  }  private int GetCountVentSection()  {  int countVentSection = 0;  foreach (ISignalOutput item in mModuleOutput.GetSignalCollection())  {  if (item.Title == Constants.TitleSignal.Q\_VS\_WORKING)  {  bool isWorking = false;  isWorking = BitConverter.ToBoolean(item.Data, 0);  if (isWorking)  {  countVentSection++;  }  }  }  ventSection = countVentSection;  return countVentSection;  }  private IModuleOutput mModuleOutput;  }  public sealed class CurrentPressure : IGenerationMethod  {  public CurrentPressure(IModuleOutput moduleOutput)  {  mModuleOutput = moduleOutput;  }  public byte[] Generate()  {  int countPump = GetCountPump();  pressure = countPump \* 10.0f;  return BitConverter.GetBytes(pressure);  }  private int GetCountPump()  {  int countPump = 0;  foreach (ISignalOutput item in mModuleOutput.GetSignalCollection())  {  if (item.Title == Constants.TitleSignal.Q\_PUMP\_WORKING)  {  bool isWorking = false;  isWorking = BitConverter.ToBoolean(item.Data, 0);  if (isWorking)  {  countPump++;  }  }  }  pump = countPump;  return countPump;  }  private IModuleOutput mModuleOutput;  }  public static float GetTemperature()  {  return pvTemperature;  }  public static float GetPressure()  {  return pressure;  }  public static int GetVentSection()  {  return ventSection;  }  public static int GetPump()  {  return pump;  }  private static Random rnd = new Random(DateTime.Now.Millisecond);  private static float pvTemperature = 0.0f;  private static float pressure = 0.0f;  private static int ventSection = 0;  private static int pump = 0;  }  } |

## Модуль ConnectProSim

ConnectProSim – класс, реализующий соединение с утилитой S7 ProSim. Реализует паттерн синглтон.

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using S7PROSIMLib;  namespace InterfaceOfSimulator  {  public sealed class ConnectProSim  {  protected ConnectProSim()  {    }  public static ConnectProSim Instance  {  get  {  if (mInstance == null)  {  mInstance = new ConnectProSim();  }  return mInstance;  }  }  public void Connect()  {  mSimulator = new S7ProSim();  mSimulator.Connect();  }  public void Disconnect()  {  if (mSimulator != null)  {  mSimulator.Disconnect();  Dispose();  }  }  public void Dispose()  {  System.Runtime.InteropServices.Marshal.ReleaseComObject(mSimulator);  mSimulator = null;  }  public S7ProSim Connection  {  get  {  return mSimulator;  }  }  private static ConnectProSim mInstance;  private S7ProSim mSimulator;  }  } |

# Заключение

В рамках прохождения преддипломной практики мною были получены умения и навыки сбора, критического анализа и обобщения информации в сети Интернет (с учетом основных требований информационной безопасности) и в научной и технической литературе по теме выпускной квалификационной работы при самостоятельном проектировании, конструировании и тестировании программного обеспечения. Были получены знания о средствах объектно-ориентированного языка программирования C#, позволяющего реализовать взаимодействие с Component Object Model интерфейса S7ProSim утилиты Simatic S7-PLCSIM для тестирования программного обеспечения ПЛК. Кроме этого, была изучена среда WinCC Flexible, позволяющая реализовать взаимодействие с ПЛК Siemens Simatic S7-300 для визуализации и управления объектом автоматизации. Также были получены навыки проектирования, разработки и тестирования программного обеспечения; проведения и анализ результатов компьютерных экспериментов по теме выпускной квалификационной работы.

Список использованных источников

1. Кривченко, И.В. AVR - микроконтроллеры: очередной этап на пути развития / И. В. Кривченко // Компоненты и технологии. – 2002. – № 3. – С. 5-14.
2. Википедия – свободная энциклопедия [Электронный ресурс] / Wikimedia Foundation – режим доступа: http://www.wikipedia.org/wiki/Simatic\_Step\_7, свободный. – Загл. с экрана. Яз. рус., англ.
3. Siemens – Global WebSite [Электронный ресурс] / Siemens – режим доступа: http://www.siemens.com, свободный. – Загл. с экрана. Англ.
4. Бергер, Г. Автоматизация посредством STEP 7 с использованием STL и SCL и програмируемых контроллеров SIMATIC S7-300/400 / Г. Бергер. – Нюрнберг : Siemens AG, – 2001. – 776 с.