

Пензенский государственный университет
Кафедра «Вычислительная техника»

В.Н. Дубинин, В.В. Вяткин

**Разработка распределенных систем автоматики на
основе функциональных блоков IEC 61499 в
системе NxtStudio**

*Методические указания
по выполнению лабораторных работ*

Пенза 2020

Оглавление

Лабораторная работа № 1. Разработка функциональных блоков и приложений

Лабораторная работа № 2. Разработка САТ-блоков

Лабораторная работа № 3. Изучение коммуникационных блоков

Лабораторная работа № 4. Создание интерфейса «человек-машина» для системы охлаждения дата-центра

Приложение 1. Листинг итогового САТ-блока для работы № 3

Список литературы

Лабораторная работа № 1

Разработка функциональных блоков и приложений

Цель работы:

- а) Получить опыт работы с интегрированной средой разработки (ИСР) NxtStudio: как создавать, компилировать, тестировать и модифицировать типы базисных функциональных блоков (ФБ).
- б) Научиться использовать типы базисных ФБ в типах составных ФБ.
- в) Выполнить распределенное удаленное приложение (FLASHER_TESTR).

Порядок выполнения работы

1. Изучить ИСР NxtStudio.
2. Разработать и протестировать базисный ФБ для вычисления функции $X^2 - Y^2$.
3. Модифицировать базисный ФБ из п.2, добавив в него функцию вычисления суммы $X + Y + Z$.
4. Разработать составной ФБ для вычисления функции $X^2 - Y^2$.
5. Создание, развертывание и выполнение распределенного приложения FLASHER.

Для выполнения работы необходимо использовать материал из раздела «Основные сведения», находящийся непосредственно ниже.

Основные сведения

Установка программного обеспечения NxtControl

Инсталляция программного обеспечения фирмы *NxtControl* на платформе *Windows* хорошо поддерживается соответствующим инсталляционным пакетом. Перед установкой необходимо иметь свежую версию пакета *Microsoft.NET*. Необходимые инструкции отображаются в мастере установки.

Функционирование ИСР *NxtStudio* интуитивно понятно для пользователей, знакомых с любой современной средой программирования. Особенно данная ИСР структурно похожа на среду программирования *Visual Studio*.

В данном лабораторном практикуме используется система *NxtStudio* версии 2.0, однако все готовые рекомендуемые *NxtStudio*-проекты, которые ориентированы на данную версию, можно адаптировать для и более поздних версий путем указания соответствующих библиотек или, в более сложных случаях, путем перекомпиляции проектов.

Создание базисного ФБ в NxtStudio

В *NxtStudio* нельзя непосредственно создать тип ФБ. Это можно сделать только в рамках некоторого «решения» (*solution*), которое является структурным аналогом «проекта» (*project*) в других средах программирования.

Создадим новое решение с названием *Tutorial* (выбор меню *File->New->Solution...*), как показано на рис. 1а, затем установим курсор на элемент *Basic* в дереве эксплорера (проводника) решений в левой части экрана, нажмем правую кнопку мыши и выберем *New Item* в выпадающем меню (рис. 1б). В появившемся диалоговом окне зададим имя *X2Y2* для типа ФБ.

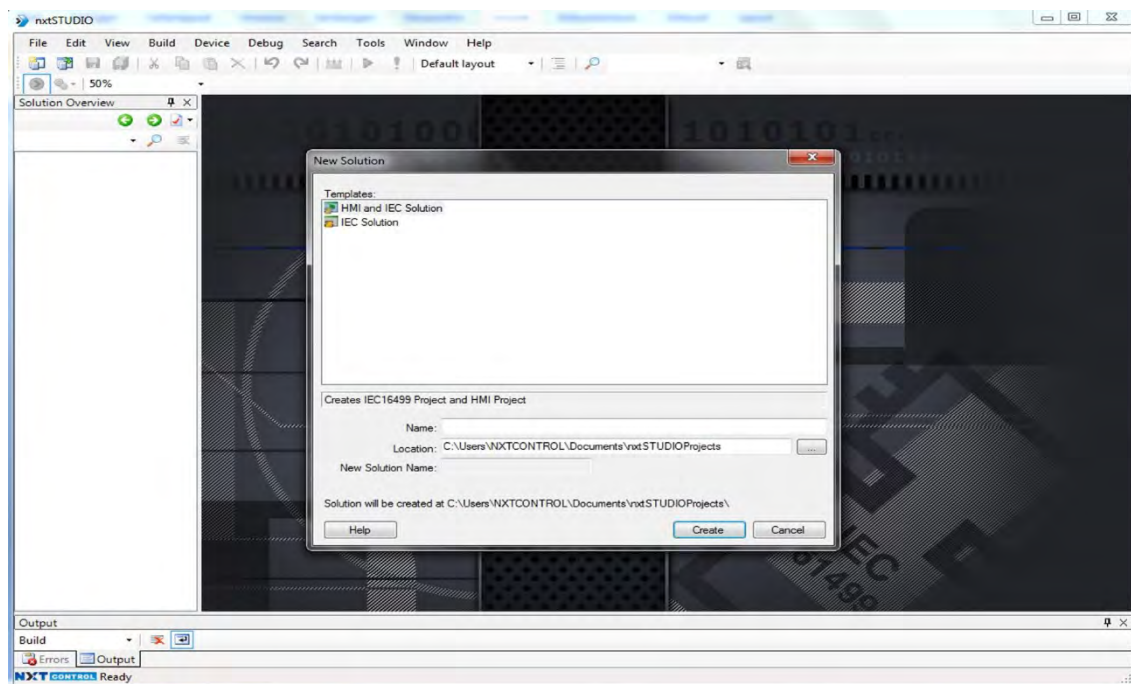


Рис. 1а. Создание решения

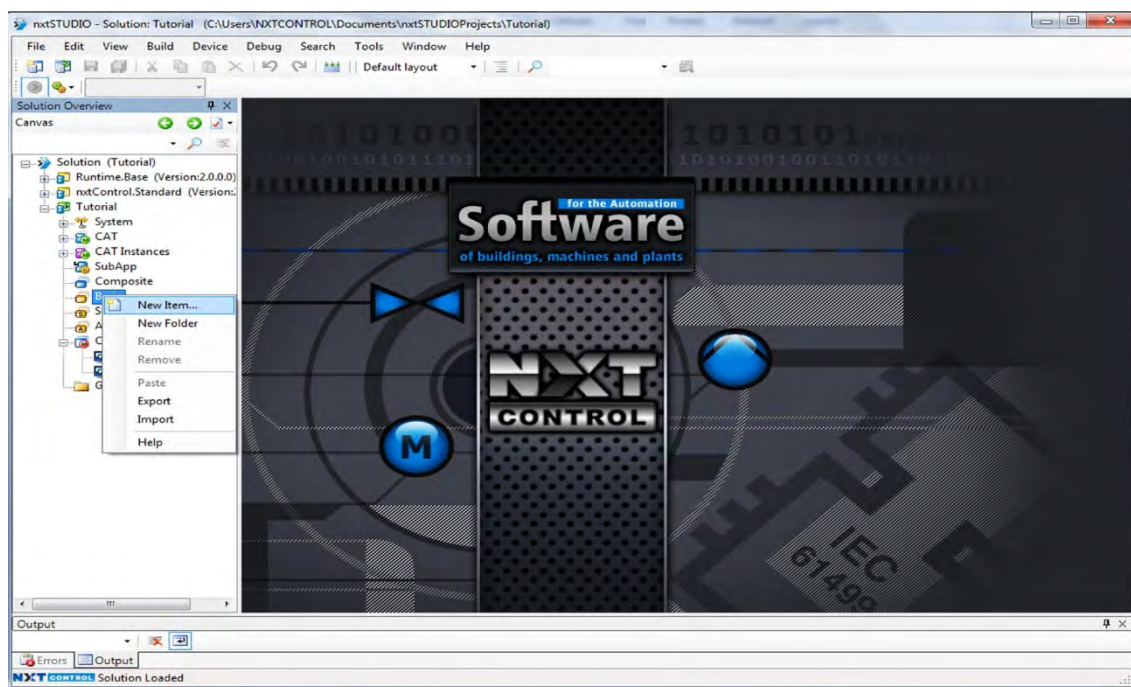


Рис. 1б. Создание нового типа базисного ФБ.

После этого на экране можно увидеть тип ФБ, созданный со стандартным интерфейсом, как показано на рис. 2а. Перетащим границу *Панели интерфейсов* вправо и начнем редактировать элементы интерфейса в таблице, как показано на рис. 2б. При этом удаляем событийные входы *INIT*

и *INITO* и переименовываем информационные вход и выход *QI* и *QO* в *X* и *OUT*, соответственно, изменив попутно их тип на *REAL*. Также добавляем еще один вход *Y* типа *REAL*.

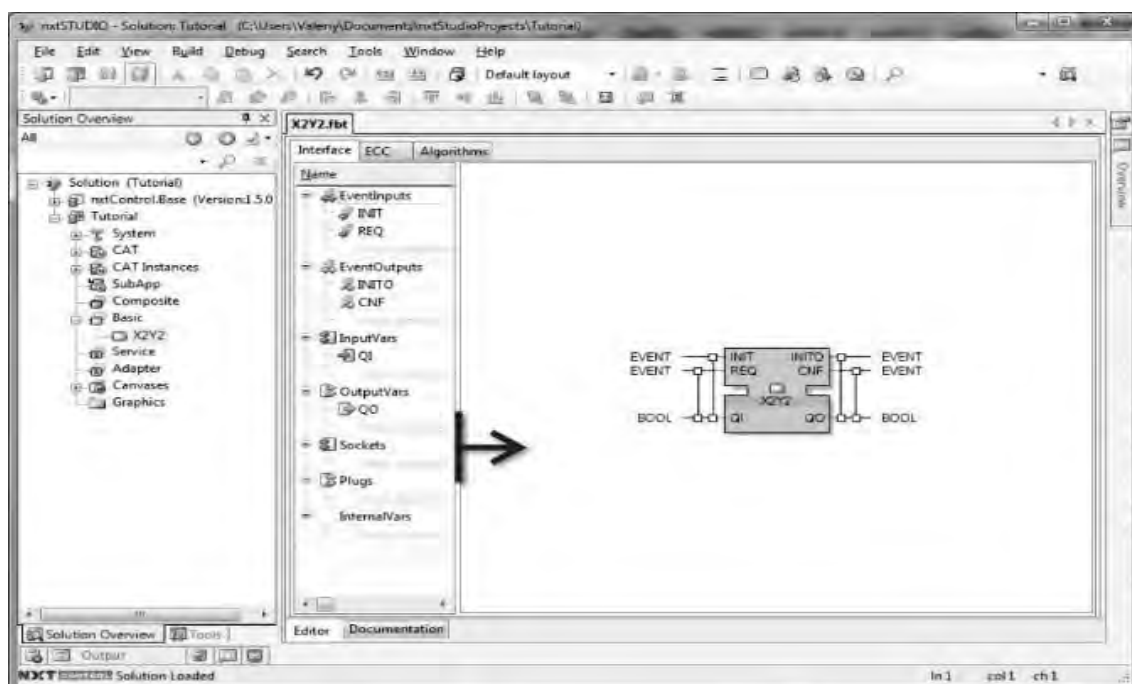


Рис. 2а. Открытие вновь созданного типа ФБ.

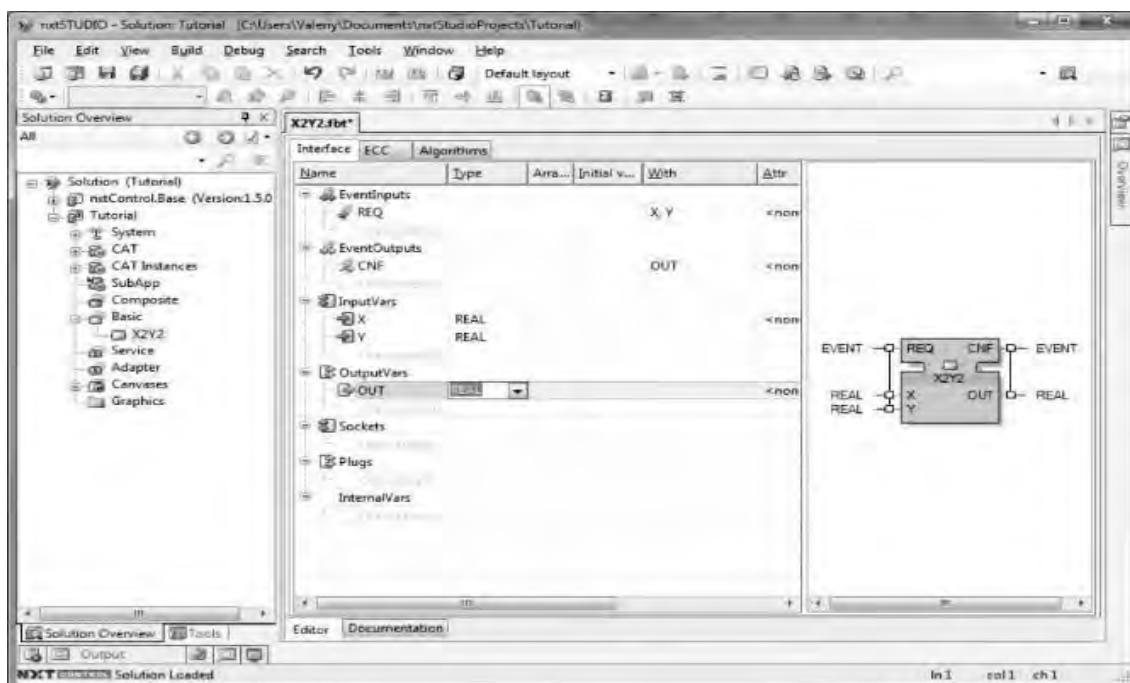


Рис. 2б. Модификация интерфейса созданного типа ФБ.

Чтобы создать связь между событийным входом/выходом и информационным входом/выходом, используем поле *With* в таблице интерфейса.

Теперь переходим на вкладку *ECC* (*Execution Control Chart* - диаграмма управления выполнением) и редактируем *ECC* как показано на рис. 3а. Нужно будет удалить состояние *INIT*, поскольку в этом простом примере нет необходимости в инициализации. Для этого кликаем правой кнопкой мыши по состоянию и выбираем пункт меню *Delete*. Также можно выбрать состояние левым щелчком мыши и нажать клавишу *Del*. После этого удалим алгоритм *INIT* (щелчок правой кнопкой мыши на вкладке *Algorithm* и выбор пункта меню *Delete algorithm*) и отредактируем алгоритм *REQ*, как показано на рис. 3б.

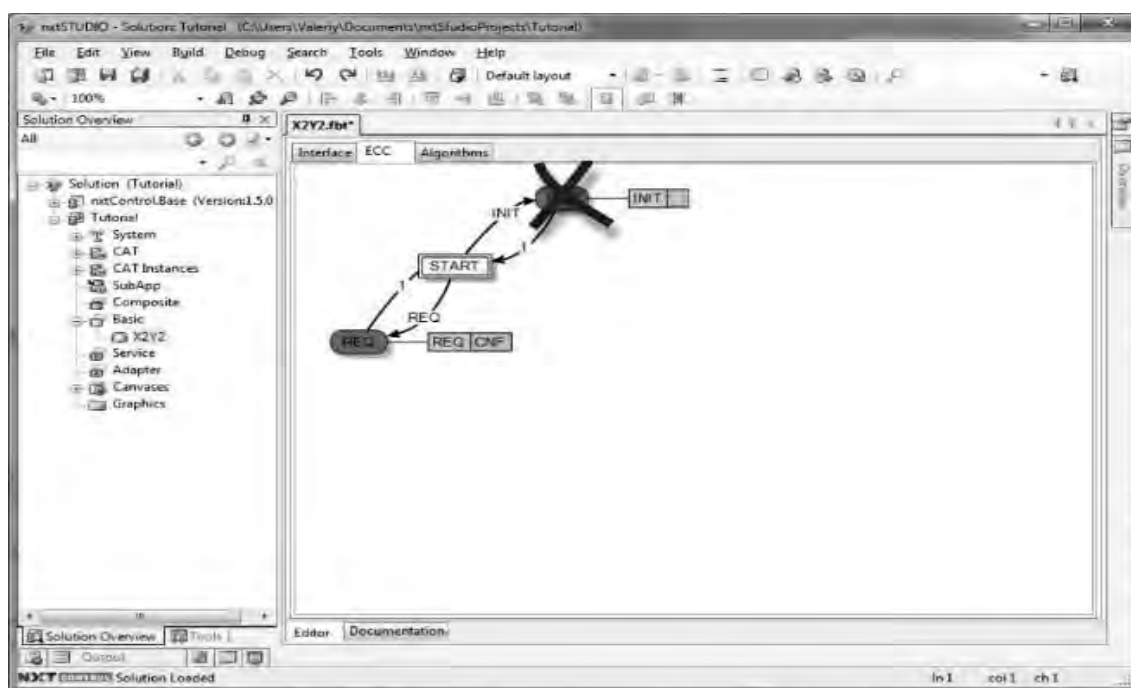


Рис. 3а. Удаление состояния INIT в диаграмме ECC.

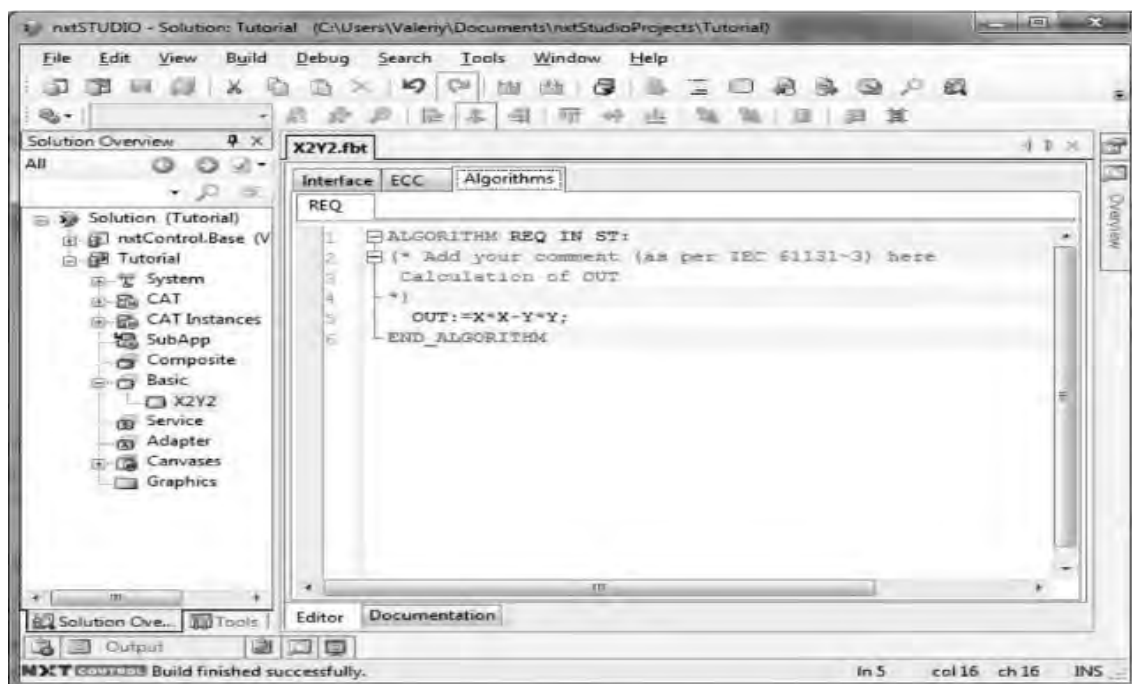


Рис. 3б. Модификация алгоритма REQ.

Теперь ФБ готов и может быть протестирован. Переходим в меню отладки *Debug* и выбираем пункт *Run*. При этом появится отладочное окно (рис. 4).

Для тестирования ФБ необходимо задать определенные значения на информационных входах и инициировать событие *REQ*, кликнув на соответствующей кнопке рядом с ним. Введем параметры *X* и *Y*, нажмем кнопку *REQ* и посмотрим результат: выходное событие *CNF* (в виде вспышки «лампы») и вычисленное значение в поле *OUT*. С левой стороны можно наблюдать текущее состояние *ECC*. Чтобы остановить отладку и вернуться в режим редактирования, нажмите кнопку *Stop*, как показано на рис.4.

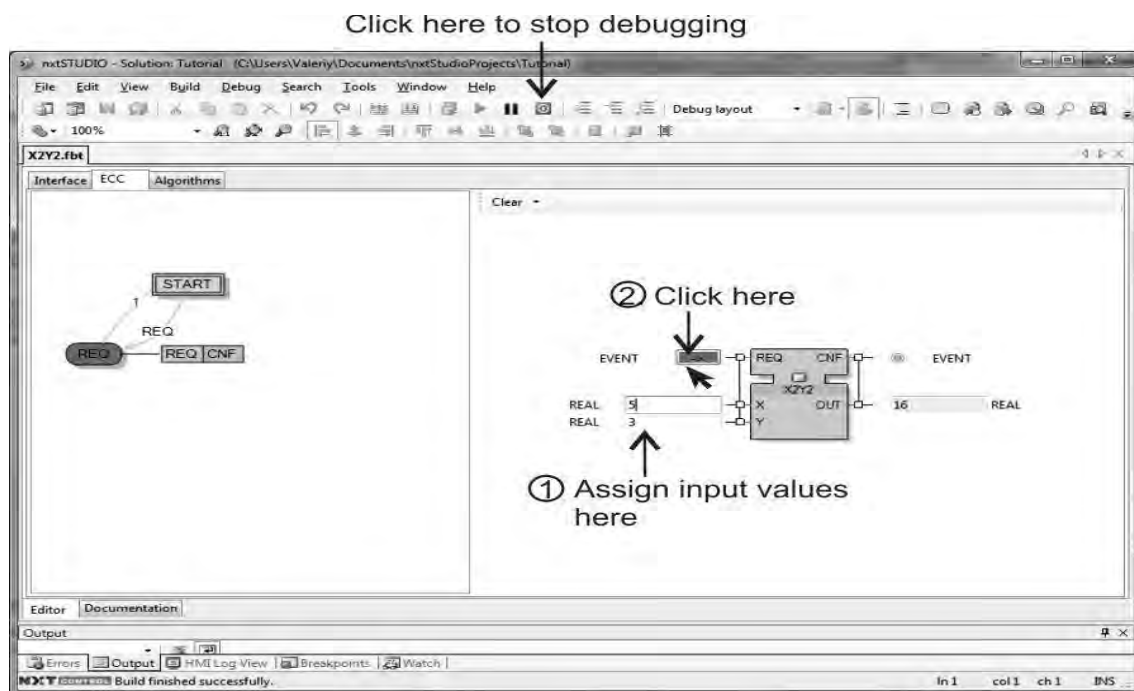


Рис. 4. Окно отладки для функционального блока X2Y2.

Поэкспериментируем с добавлением нескольких входов и выходов в ФБ и с модификацией *ECC*. Например, сделаем следующие изменения: добавим информационный вход *Z* (типа *REAL*), событийный вход *XYZ*, событийный выход *CNF2* и информационный выход *OUT2* (типа *REAL*). Когда блок активируется событием *XYZ*, он должен вычислить $OUT2 = X + Y + Z$ и вывести сигнал *CNF2*.

Чтобы создать новый тип ФБ на основе старого, необходимо скопировать этот ФБ в *Дерево решений*, а затем вставить его под другим именем, например, *X2Y2_XYZ*. Чтобы скопировать его, перейдите в *Дерево решений*, щелкните правой кнопкой мыши на старом ФБ и выберите *Copy* из выпадающего меню (рис. 5а). Затем переместите курсор в корень списка базисных ФБ (по имени *Basic*), щелкните правой кнопкой мыши и выберите команду *Paste*. В появившемся диалоговом окне введите новое имя ФБ. Результат будет выглядеть как на рис. 5б.

Замечание: имя в форме ФБ справа может быть обновлено только после сохранения решения (нажатие комбинации клавиш *Ctrl-S* или кнопка

сохранения на панели инструментов) и двойном щелчке мыши на новом ФБ в дереве решений.

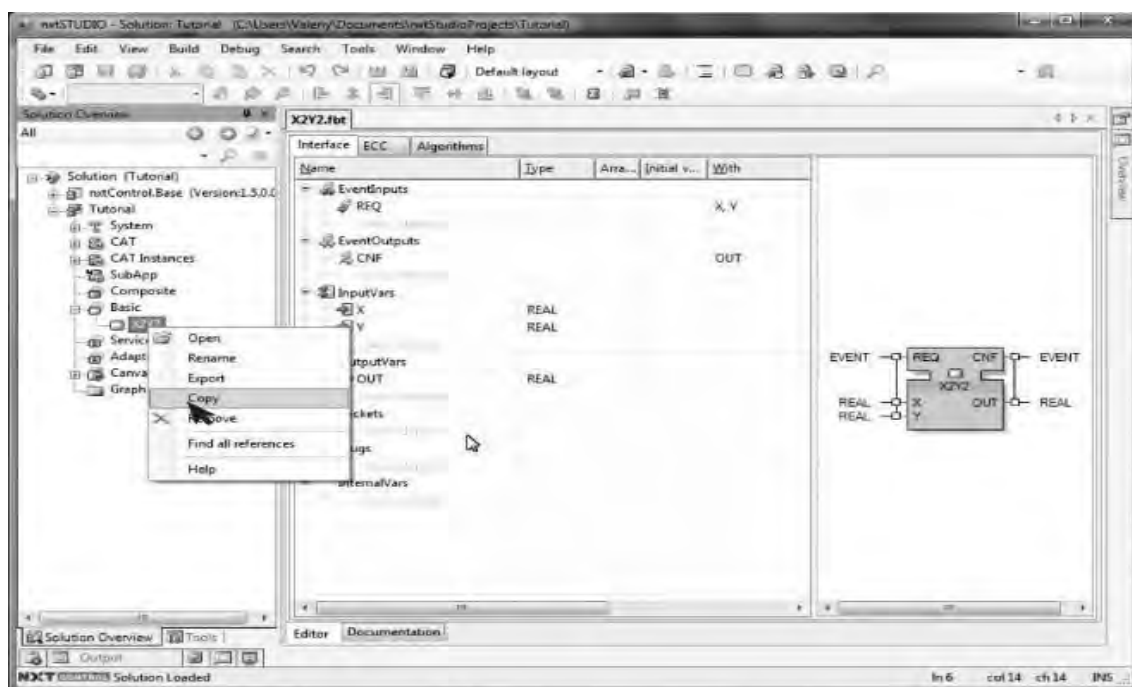


Рис. 5а. Создание нового ФБ на основе старого (копирование)

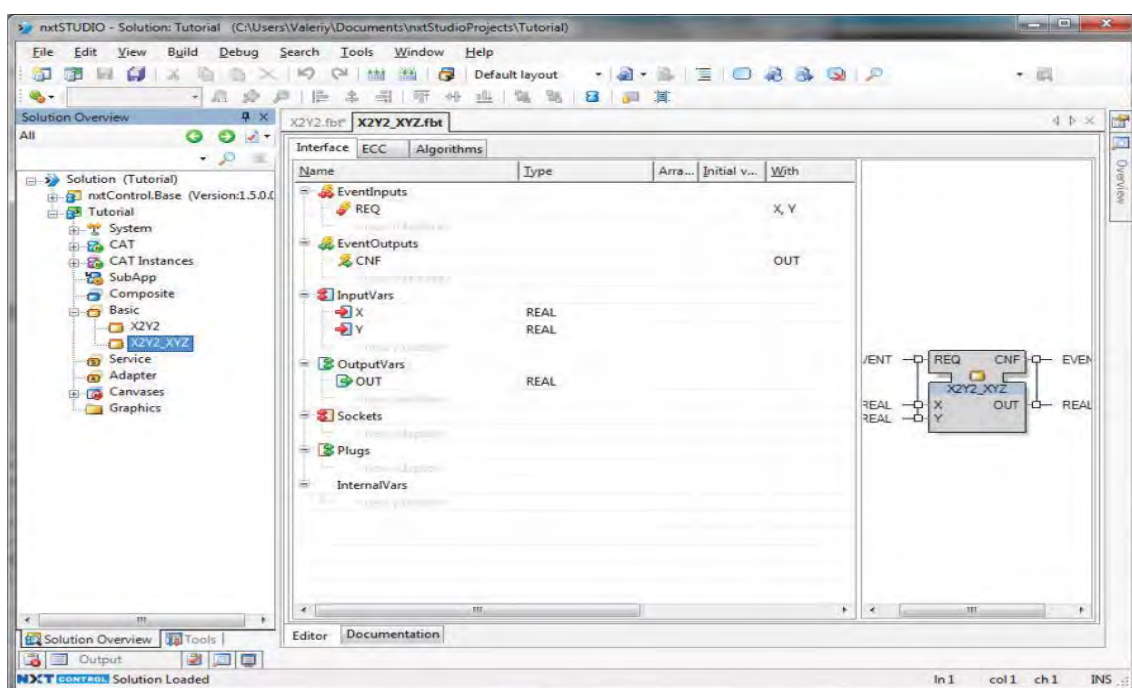


Рис. 5б. Создание нового ФБ на основе старого (добавление и назначение нового имени)

Интерфейс полученного блока показан на рис. 6а. На следующем шаге добавим новый алгоритм, назовем его *XYZ_COMP*. На вкладке *Algorithms* щелкнем правой кнопкой мыши на существующем алгоритме *REQ* и выберем *Add algorithm* (рис. 6б). После этого добавим строку: $OUT2:=X+Y+Z$; в новый алгоритм.

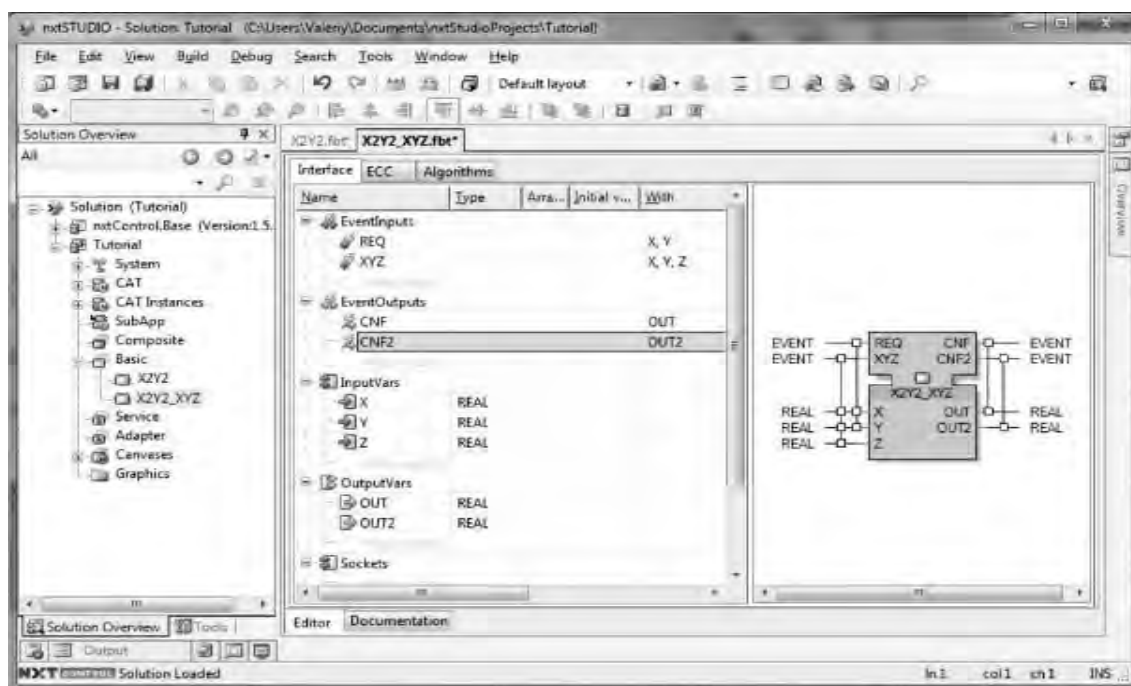


Рис. 6а. Интерфейс нового ФБ

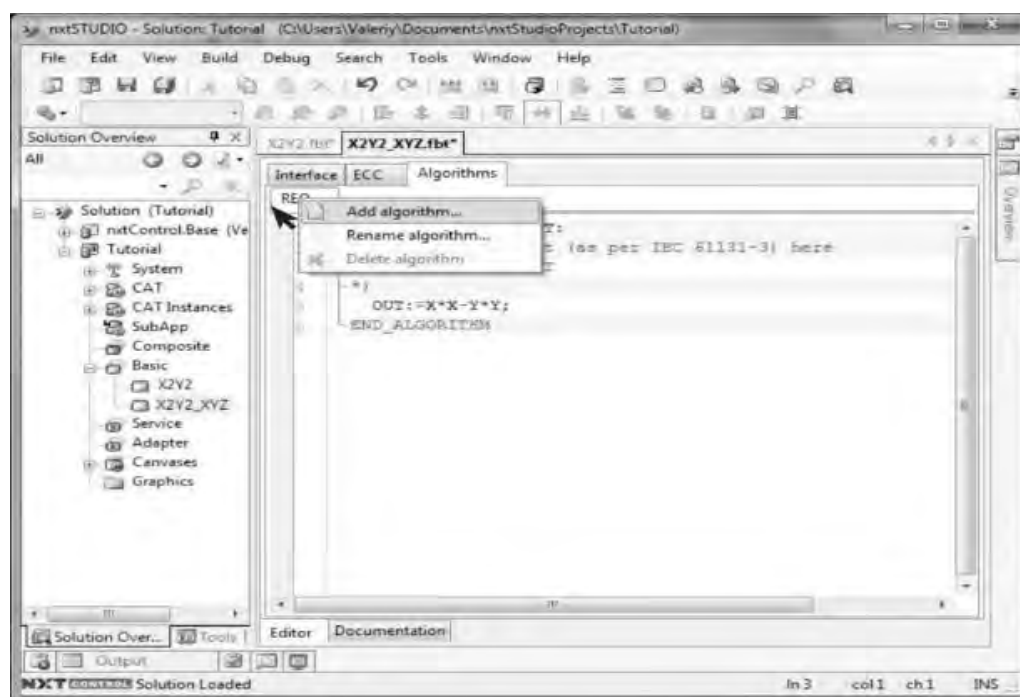


Рис. 6б. Создание нового алгоритма

Следующим шагом является редактирование диаграммы *ECC* путем добавления нового состояния, в котором будет вызываться только что созданный алгоритм. Назовем состояние *XYZ_S*.

Перейдя на панель *ECC*, вызовите всплывающее меню, щелкнув правой кнопкой мыши, затем выберите *New State*, как показано на рис. 7а. Редактирование диаграммы *ECC* интуитивно понятно. Чтобы изменить название состояния, просто нажмите на него. Чтобы нарисовать линию перехода между двумя состояниями, наведите курсор на состояние-источник, найдите положение, где курсор в виде стрелки изменяет свою форму на изображение руки, и, нажав и удерживая левую кнопку мыши, перетащите его в конечное состояние (рис.7б). Далее нажмите на полученную связь, чтобы изменить значение с *XYZ_S* (по умолчанию условие перехода будет равно имени события-приемника) на *XYZ* (это новое событие, которое мы создали на предыдущем шаге, редактируя интерфейс).

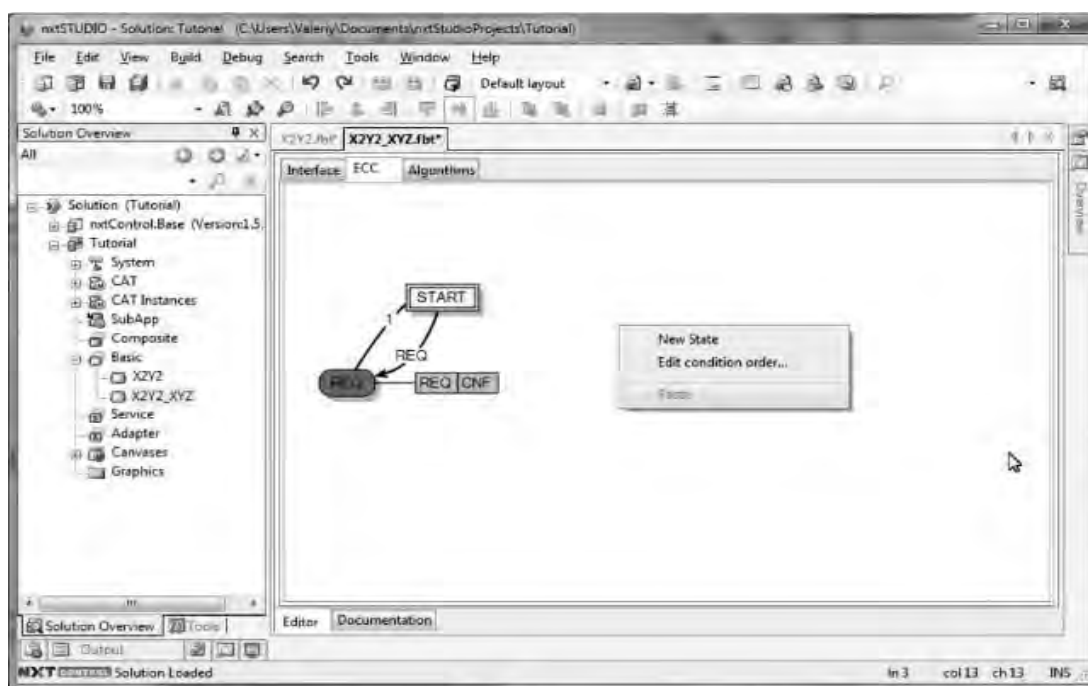


Рис. 7а. Создание нового состояния в ECC

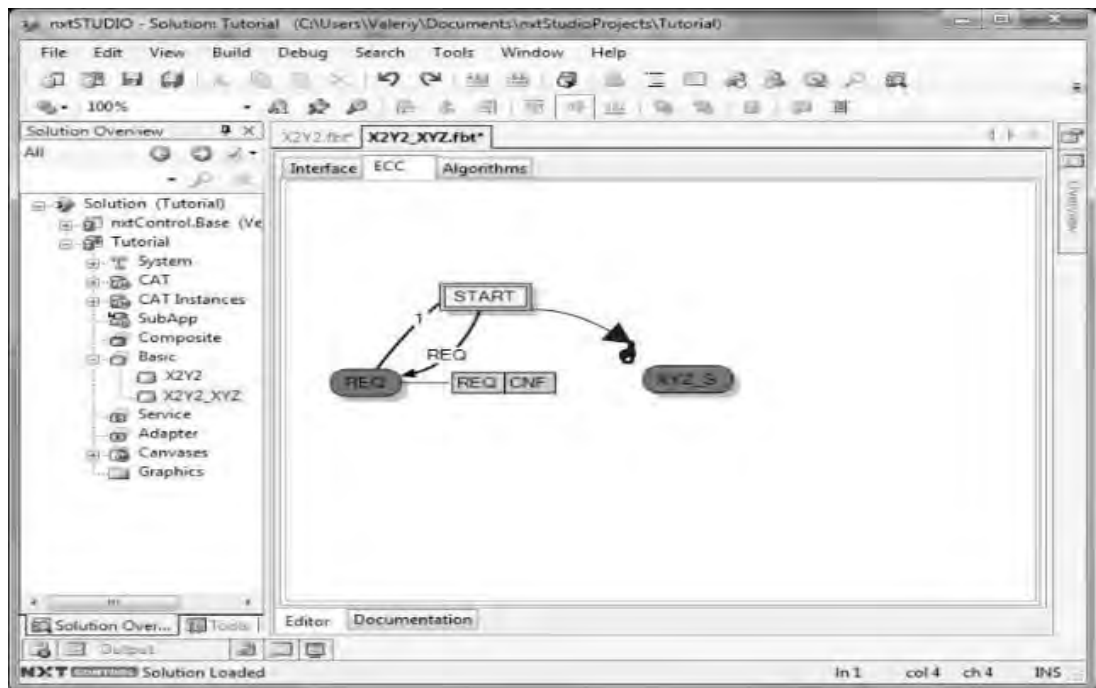


Рис. 7а. Добавление связей

После добавления нового состояния, мы должны создать новую акцию (иначе, *ЕС*-акцию) для этого состояния для того, чтобы прикрепить к нему алгоритм и определить, какое выходное событие будет выдано. Все эти действия можно легко выполнить, нажимая правой кнопкой мыши на состоянии или части акции. Результат модификации диаграммы *ЕСС* показан на рис.8.

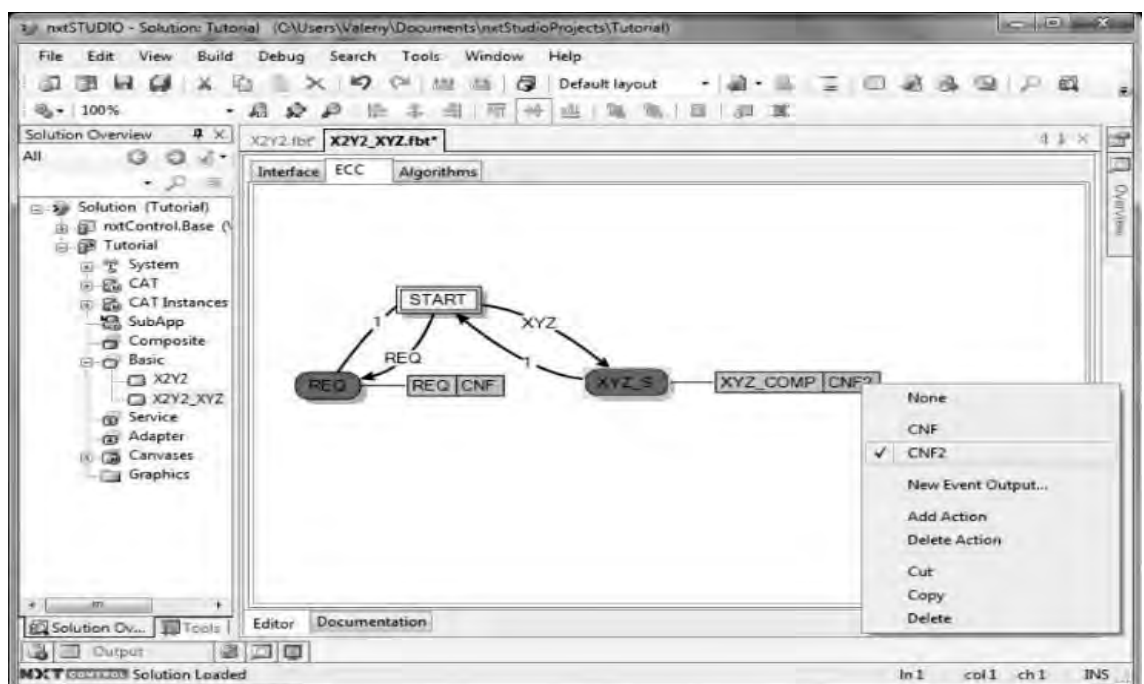


Рис. 8. Модификация диаграммы ЕСС

Теперь можно протестировать этот новый ФБ, как и раньше, используя опцию *Run* из меню отладки *Debug*.

Создание составного ФБ в NxtStudio

Создадим новый тип составного ФБ с именем *X2Y2_CMP*, как показано на рис. 9. Для этого слева, в дереве эксплорера решений выберем элемент *Composite*, правый клик мыши на нем и далее выбор *New Item*. Затем нужно выполнить те же действия, что и для базисного ФБ, а именно: дать ему имя, определить интерфейс и сохранить.

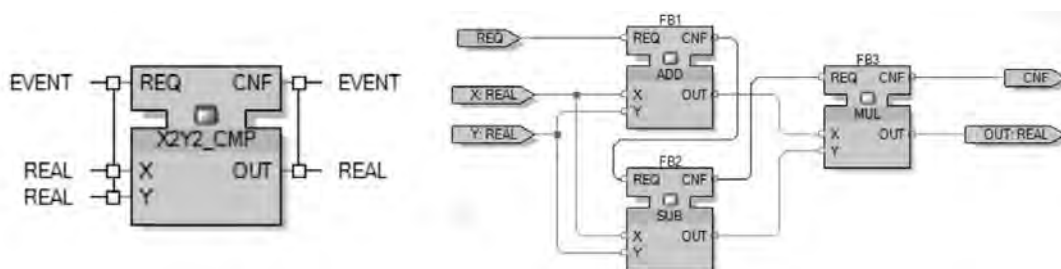


Рис. 9. Интерфейс и содержимое составного ФБ для вычисления функции $X^2 - Y^2$

Чтобы создать тело ФБ, переходим из вкладки *Interface* на вкладку *Composite*, как показано на рис. 10. Для добавления новых экземпляров ФБ, используем всплывающее меню, вызывающееся нажатием правой кнопки мыши.

К сожалению, в текущей версии *NxtStudio* удобная для пользователя среда тестирования базисных ФБ недоступна в отношении составных ФБ. Поэтому составные ФБ могут быть протестированы только в составе приложений, которые будут обсуждаться в следующих разделах.

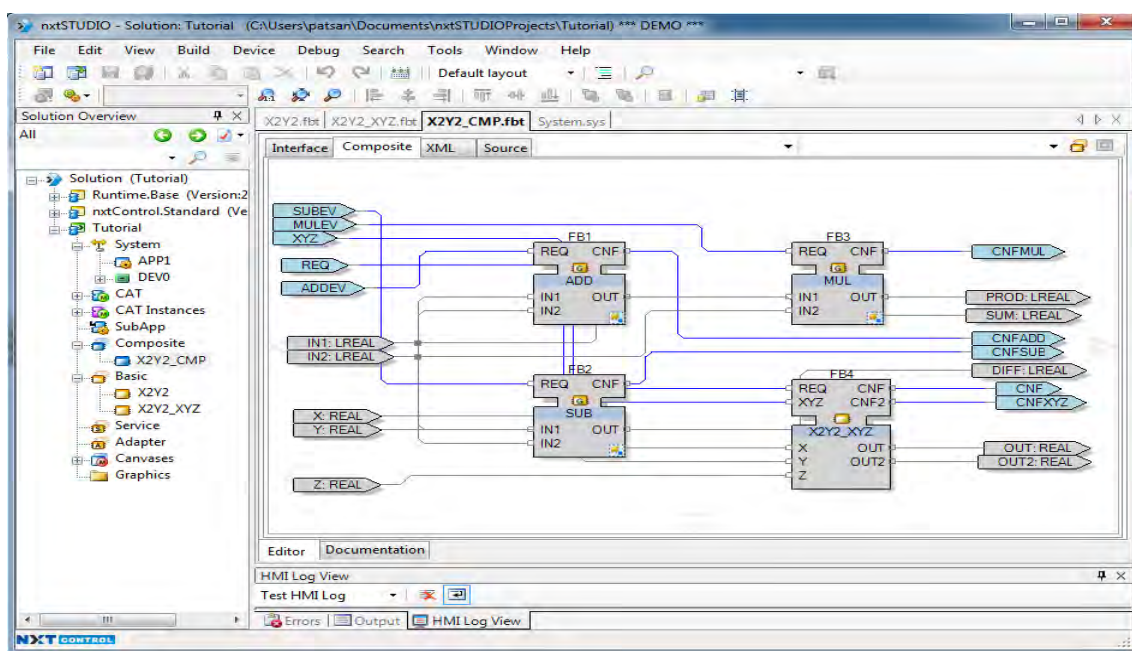


Рис. 10. Добавление компонентных ФБ в сеть составного ФБ

Далее создадим новый составной блок, как показано ниже. Добавим в него следующие компонентные ФБ: ADD, SUB, MUL, а также созданный нами ранее блок X2Y2_XYZ. Сначала создаем интерфейс, как показано на рис. 11, а затем сеть ФБ, как показано на рис. 12. Следует обратить внимание на то, что ФБ типов ADD, SUB и MUL могут быть найдены в библиотеках системы NxtStudio, как показано на рис. 13.

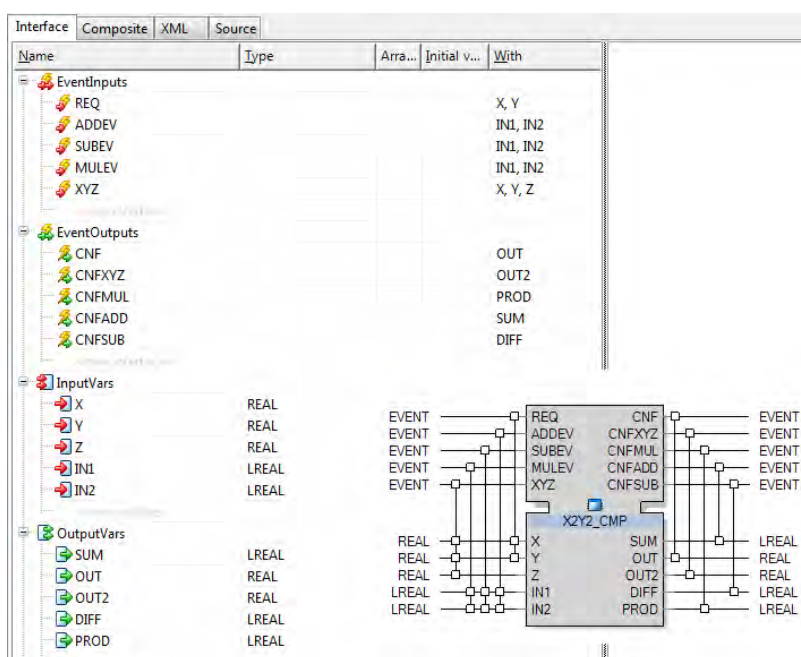


Рис. 11. Интерфейс составного ФБ

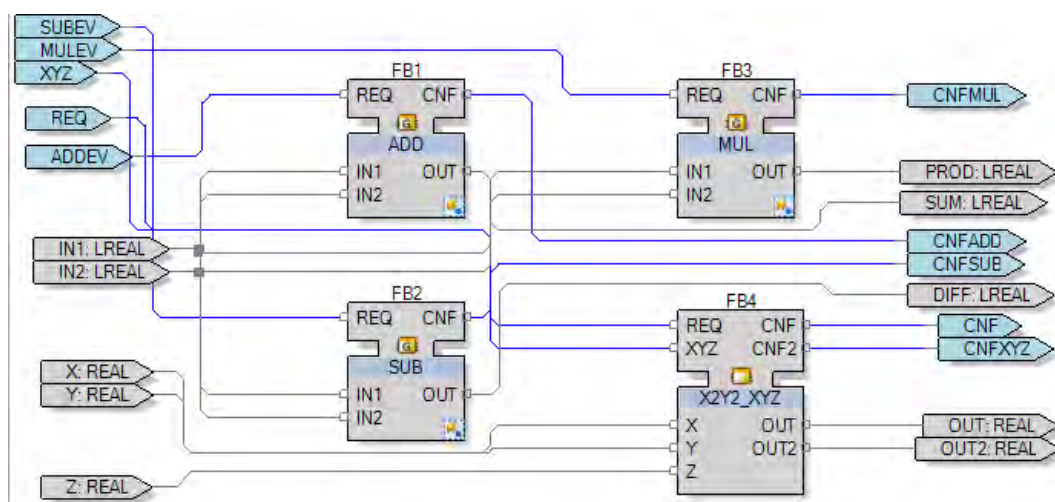


Рис. 12. Сеть составного ФБ

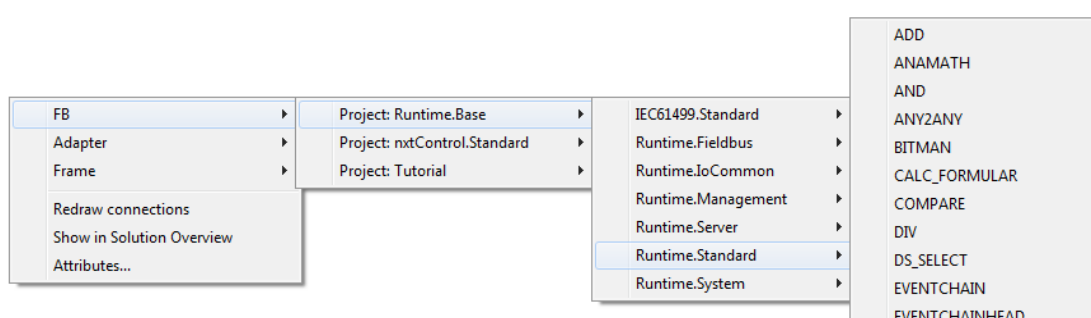


Рис 13. Готовые ФБ в библиотеках системы NxtStudio

Как уже упоминалось ранее, нельзя напрямую протестировать составной блок в отладчике, поэтому необходимо создать функционально-блочное приложение (ФБ-приложение, или просто приложение). Чтобы сделать это, необходимо дважды щелкнуть на элементе *APP1* в разделе *System* дерева эксплорера решений (с левой стороны), как показано на рис. 14. Затем можно просто перетащить составной блок *XYZ_COMP* из эксплорера решений на холст в середину. В дальнейшем, в рамках данного приложения можно создать простой человеко-машинный интерфейс (ЧМИ), содержащий кнопки и текстовые поля для имитации ввода событий и сопутствующих данных, а также «светодиодов» (LEDs) и текстовые поля для визуализации выходных событий и связанных с ними данных.

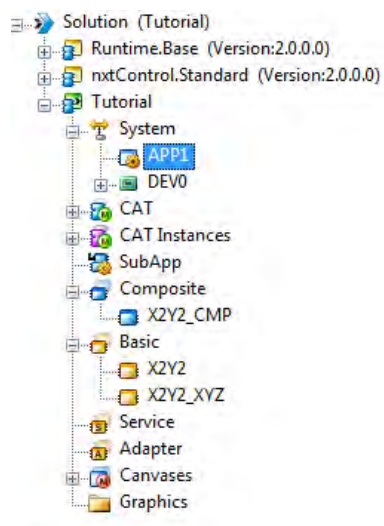


Рис. 14. Создание приложения.

Создание распределенного приложения FLASHER

Далее мы будем использовать существующую систему ФБ под названием *FLASHER*. Идея распределенной реализации *FLASHER* показана на рис. 15. Здесь, микропроцессор устройства ввода параметров считывает все входные события и значения параметров, а затем отправляет их на приемные устройства в сети. Другое устройство (устройство принятия решений) получает эти значения и генерирует значения *on/off* сигналов для каждого из светодиодов, которые также публикуются в сети. И, наконец, устройство вывода изображения, которое в данном случае представляет собой блок из четырех светодиодов со встроенным микропроцессором, принимает значения и отображает их.

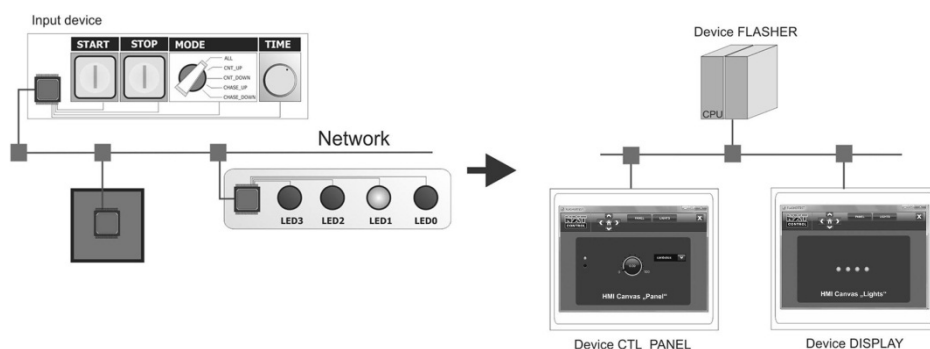


Рис. 15. Система FLASHER, реализованная на трех устройствах.

Рассмотрим детали создания конфигурации распределенной системы. В качестве отправной точки создадим копию централизованной системы FLASHER из материала курса и переименуем ее в FLASHERR (для этого необходимо просто вырезать и вставить папку FLASHER). Откроем решение FLASHERR и изменим архитектуру аппаратных средств, как показано на рис. 16.

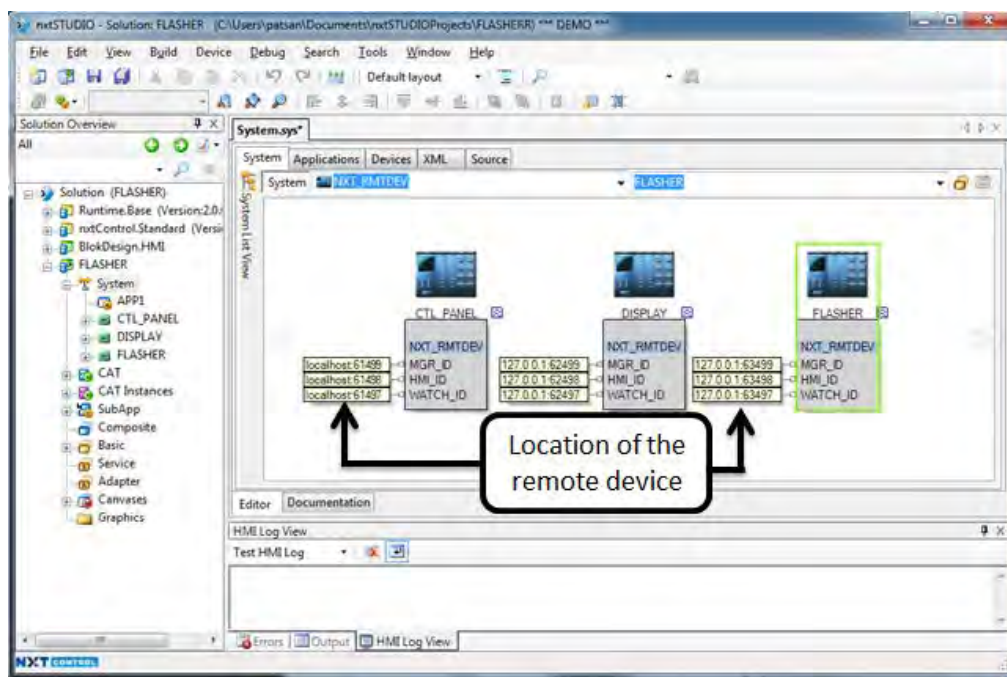


Рис. 16. Устройства системы FLASHERR_TEST3D: два локальных устройства (CTL_PANEL и DISPLAY) и удаленное устройство FLASHER

Для отображения функциональных блоков приложения на устройства, щелкаем на значок приложения (APP1) в дереве и следуем инструкциям из рис. 17: щелкнуть правой кнопкой мыши по ФБ, выбрать Mapping в выпадающем меню, а затем выбрать ресурс устройства для размещения этого ФБ.

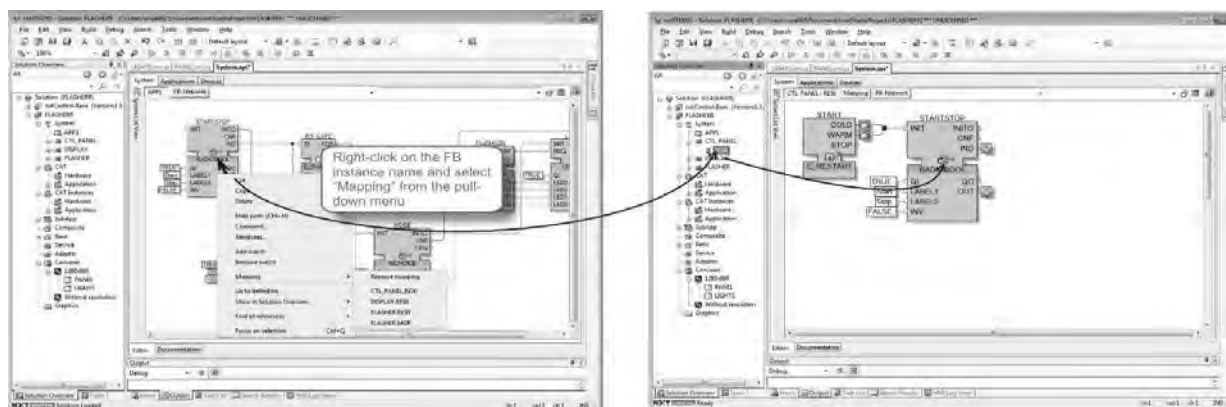


Рис. 17. Отображение функционального блока **STARTSTOP** на устройство CTL_PANEL

Следует отметить, что *NxtStudio* будет автоматически создавать соединение между двумя ФБ, если они отображаются на один и тот же ресурс, и будет вставлять коммуникационные блоки для ФБ, размещенных на различных устройствах. Это проиллюстрировано на рис.18.

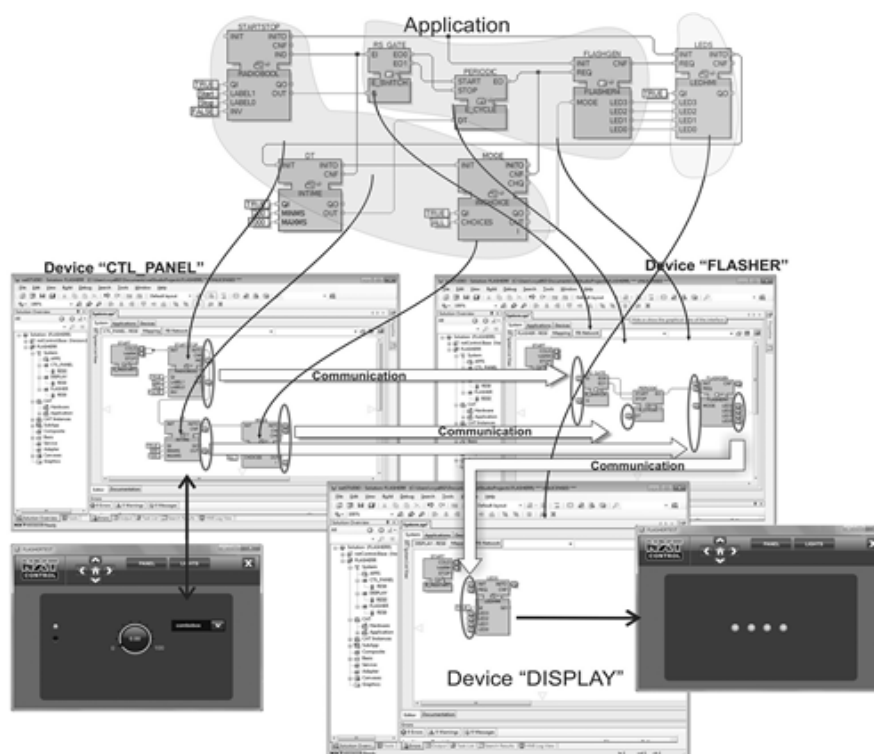


Рис. 18. Автоматическое добавление коммуникационных ФБ в устройства FLASHER и DISPLAY

Полная структура распределенной системы FLASHERR показана на рис. 19. Здесь устройство FLASHER (справа) не производит никаких видимых результатов (нет связей). На самом деле он генерирует выходные значения и посылает их на устройство DISPLAY, учитывая входные параметры, полученные от устройства CTL_PANEL.

Существует два коммуникационных потока: между устройствами CTL_PANEL и FLASHER, а также между устройствами FLASHER и DISPLAY.

Конфигурация системы FLASHERR может быть выполнена в сети из нескольких (1 или 2) подключенных к сети устройств.

В конфигурации системы на рис. 16, значение идентификатора (ID) менеджера устройства FLASHER установлено в *localhost:63499*. Эта установка используется, когда удаленное устройство выполняется на локальном хосте *localhost* (то есть, на том же компьютере, на котором выполняется *NxtStudio*). В противном случае, необходимо исправить эту настройку, заменив значение *localhost* на символическое имя или IP-адрес устройства, на котором будет запущена удаленная копия устройства. Таким образом, несколько удаленных устройств могут быть выполнены одновременно на локальном или удаленном компьютере.

Рис. 19 схематично показывает выполнение проекта *FLASHERR* на двух компьютерах, работающих в сети. Следует отметить, что можно купить дополнительную лицензию на выполнение копии устройства *NXT_RMTDEV* на другом компьютере. Вместо *NXT_RMTDEV* может использоваться более простой тип устройства *RMT_DEV*. Реализация этого типа устройства недоступна в *NxtStudio*, однако доступна в ИСР *FBDK*, имеющей среду выполнения *FBRT*, и в ИСР *4DIAC*, основанной на среде выполнения *FORTE*. Однако автоматическое отображение ФБ на устройство *RMT_DEV* возможно только в ограниченной области, так как эти ИСР реализуют только устройства Класса 1 (*Class 1*). При этом программный код для сетевого взаимодействия должен быть включен вручную и тип устройства

(*FLASHER.fbt*) должен быть указан на устройстве (с помощью *FBDK* или *4DIAC*).

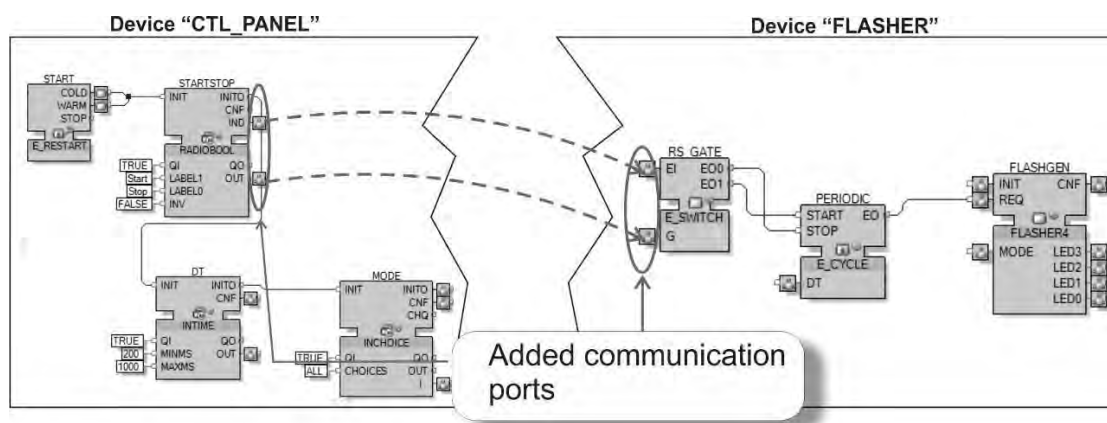


Рис. 19. Распределение приложения FLASHER на два устройства.

3.3. Развертывание и выполнение

Чтобы распределить (развернуть) решение на том же самом компьютере, надо следовать нижеприведенным шагам.

1) На панели *Devices* (Панель устройств) переключите *Active Network Profile* на *Local Test* (рис. 20).

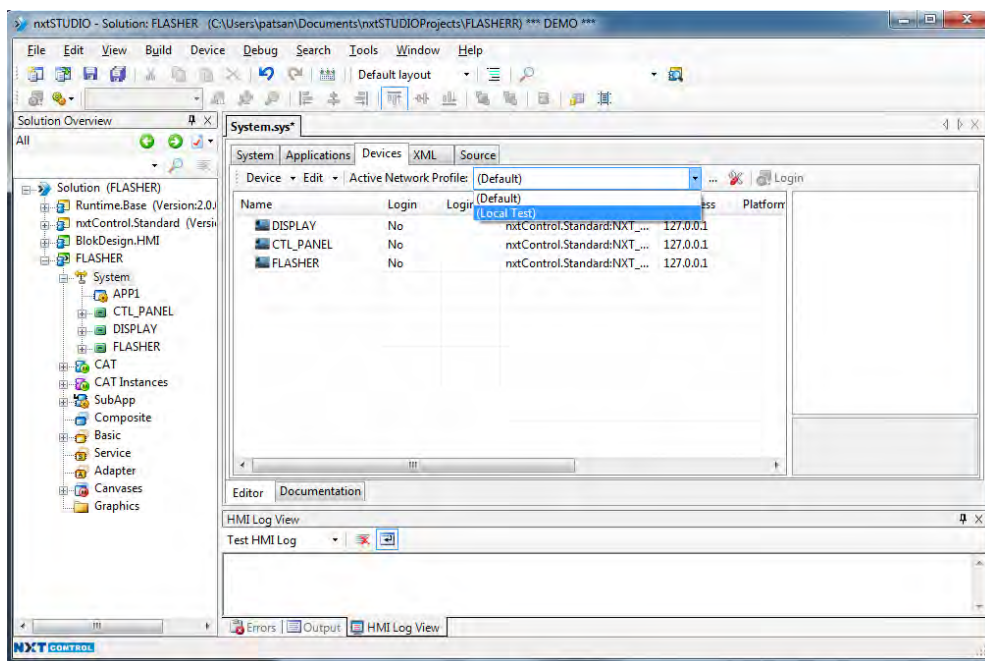


Рис. 20. Переключение на профиль Local Test

2) Установить состояние *Login* в используемых устройствах: *DISPLAY*, *CTL_PANEL* и *FLASHER*. Для этого перейдите на *Панель устройств* и используйте выпадающее меню, как показано на рис. 21. Это требуется для того, чтобы иметь возможность удаленной отладки и просмотра значений переменных процесса.

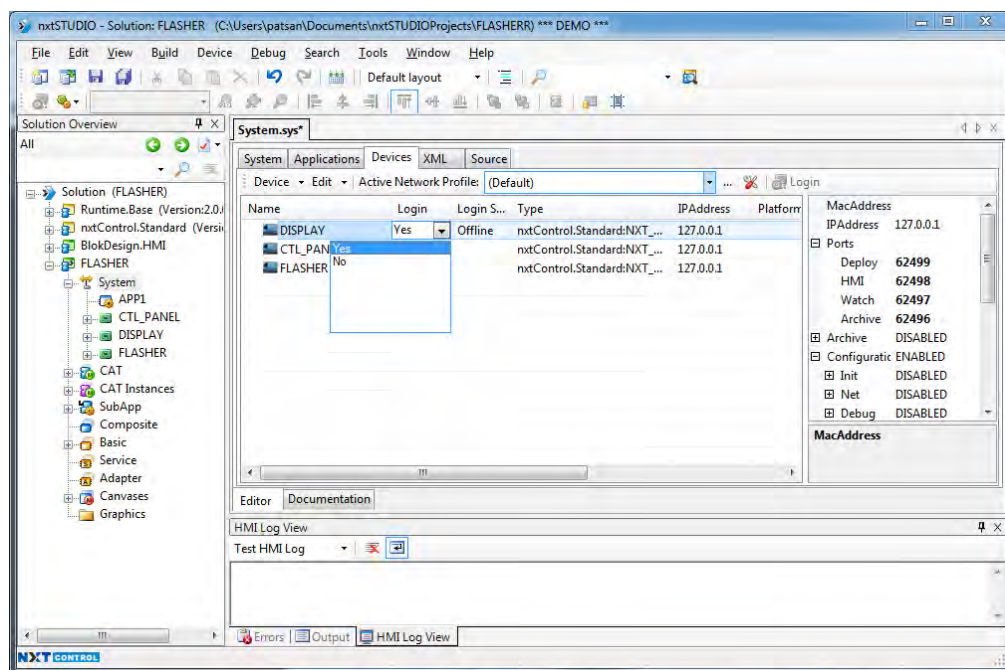


Рис. 21. Разрешение входа на устройство (Login -> Yes).

3) Убедитесь, что все устройства имеют разные номера портов (рис. 22). Обычно номера портов для локальных тестов устанавливаются системой автоматически, например, группа портов 61499, 61498, 61497, 61496 используется для устройства *CTL_PANEL*, группы портов 62499, 62498, 62497, 62496 используется для устройства *DISPLAY* и т.д.

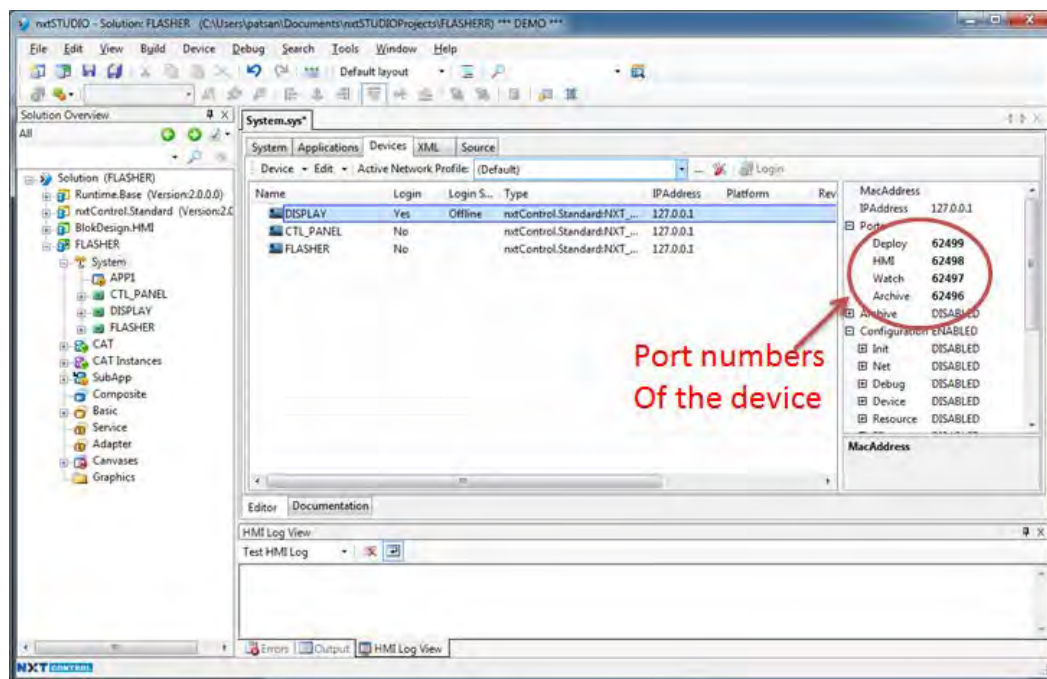


Рис. 22. Установка номеров портов устройств.

4) Выберите все устройства, представляющие интерес, например, *DISPLAY*, *CTL_PANEL* и *FLASHER*, и запустите программные контроллеры *SoftPLC*, выбрав соответствующий пункт в меню, вызываемом нажатием правой кнопки мыши (рис. 23). Это будет проявляться в трех окнах командной строки *Windows* с сообщениями среды выполнения *NxtForte* (рис. 24).

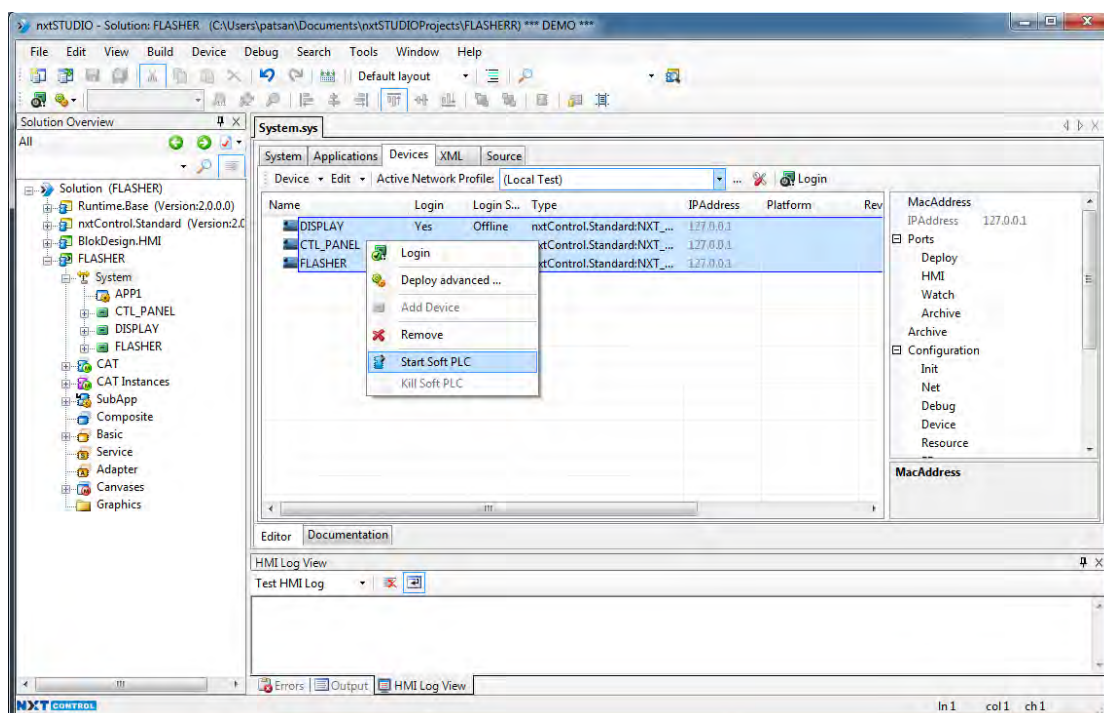


Рис. 23. Запуск программных контроллеров Soft-PLC

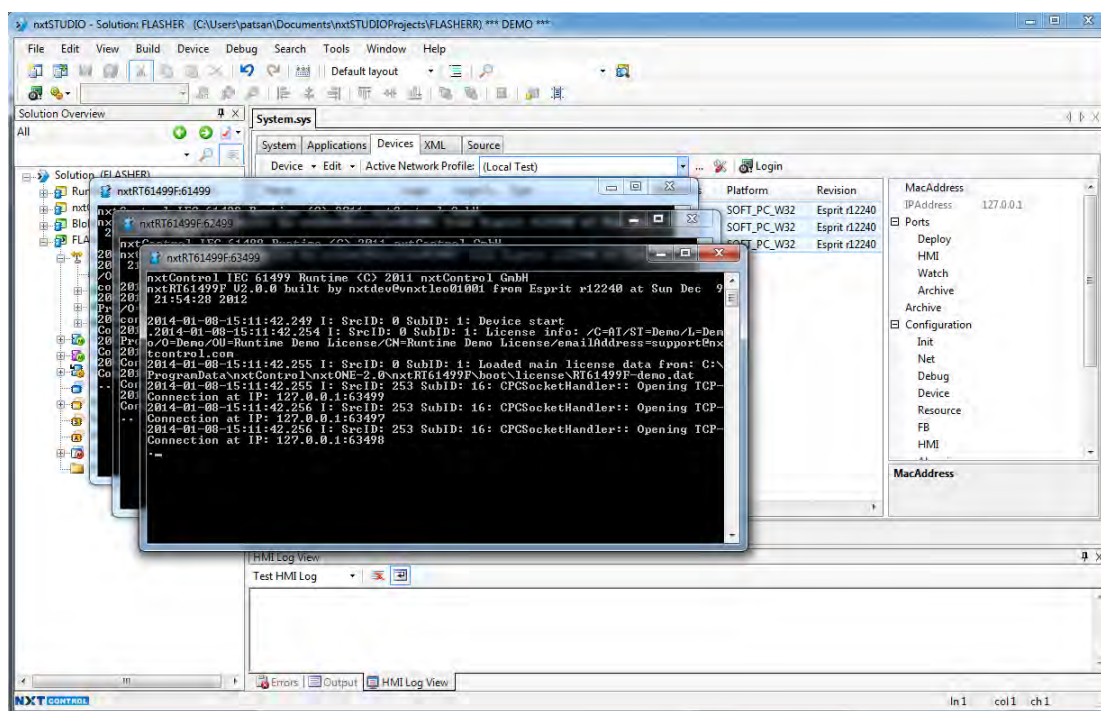


Рис. 24. Программный контроллер Soft-PLC в работе.

5) Разместите ФБ- приложения на запущенных устройствах. Для этого перейдите в диалог *Advanced Deployment* (рис. 25). Щелкните правой кнопкой мыши на каждом устройстве и выберите пункт меню *Delete persistent data*. Когда постоянные данные будут удалены, нажмите кнопку *Deploy*. Загрузка может оказаться неудачной, если постоянные данные не были удалены; иногда удаление должно быть выполнено во второй раз. Важно также удалить из устройства ранее загруженные ФБ-приложения, прежде чем загружать новое. Поэтому желательно выбрать опцию *Clear before deploy* во вкладке *Advanced*. Если эта опция установлена, система *NxtStudio* перед развертыванием будет всегда очищать ранее загруженные приложения.

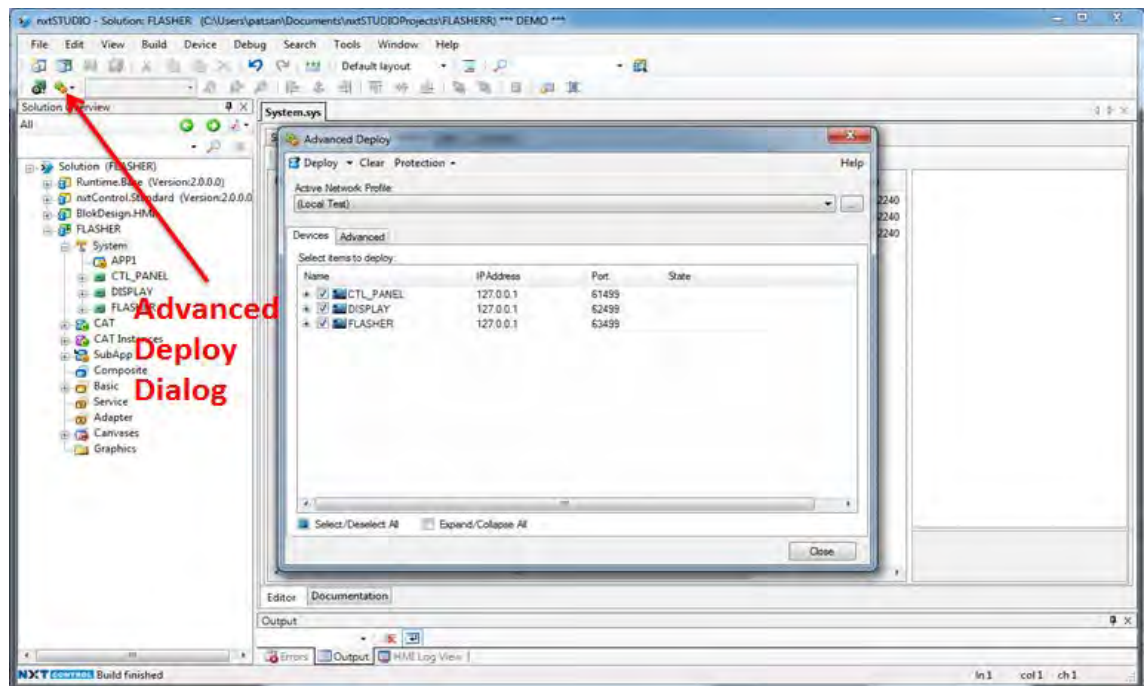


Рис. 25. Размещение систем ФБ на устройствах.

6) Войдите на запущенные устройства , нажав кнопку Login (рис. 26).

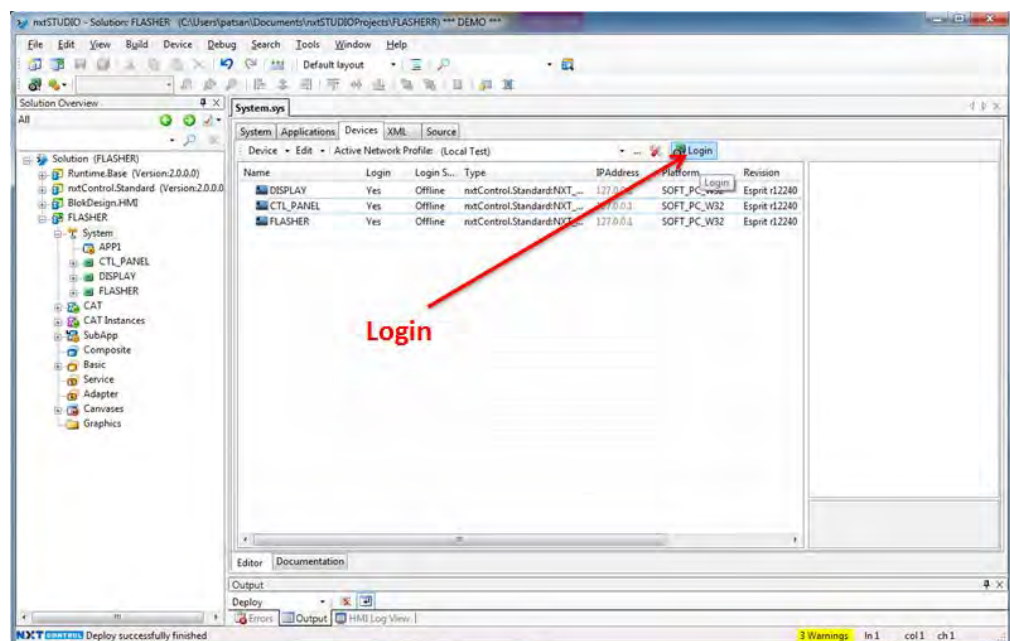


Рис. 26. Вход на стартовавшие устройства.

7) Запустите панель человеко-машинного интерфейса (HMI) (рис.27).

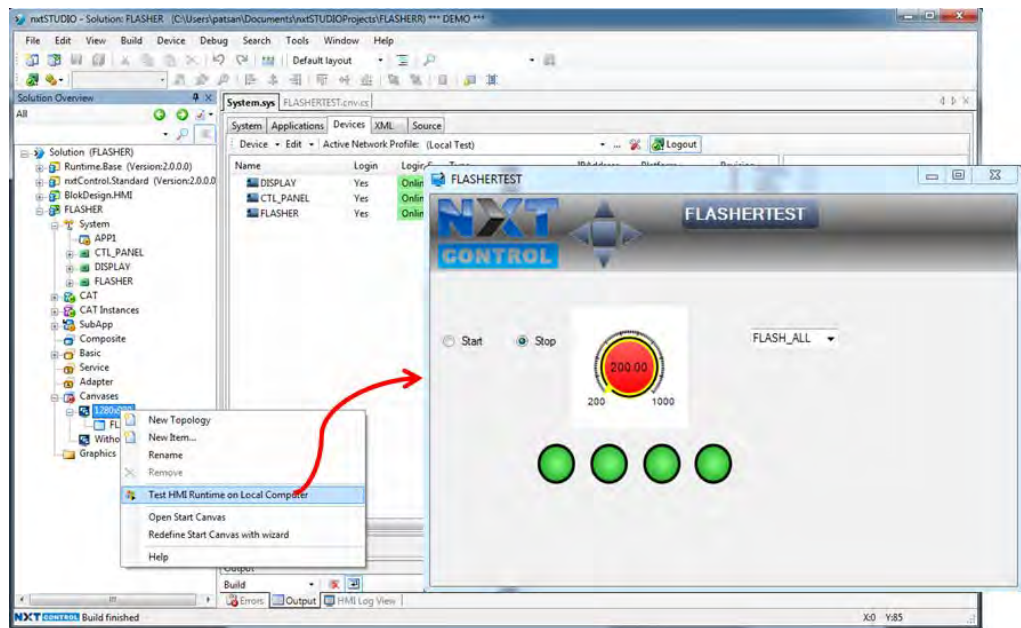


Рис. 27. Запуск панели человеко-машинного интерфейса (HMI).

8) Наблюдайте за переменными, кликая на них правой кнопкой мыши (рис. 28). Попробуйте разные опции в панели HMI и посмотрите, как изменяются переменные в устройствах (рис. 29).

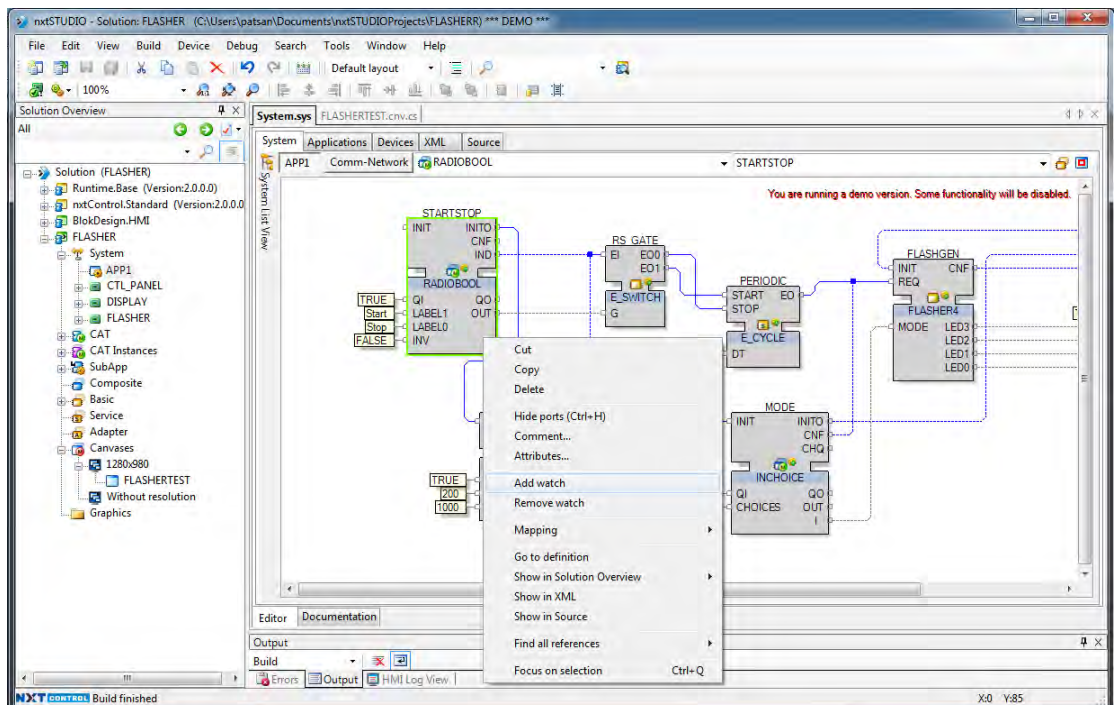


Рис. 28. Установка наблюдаемых переменных.

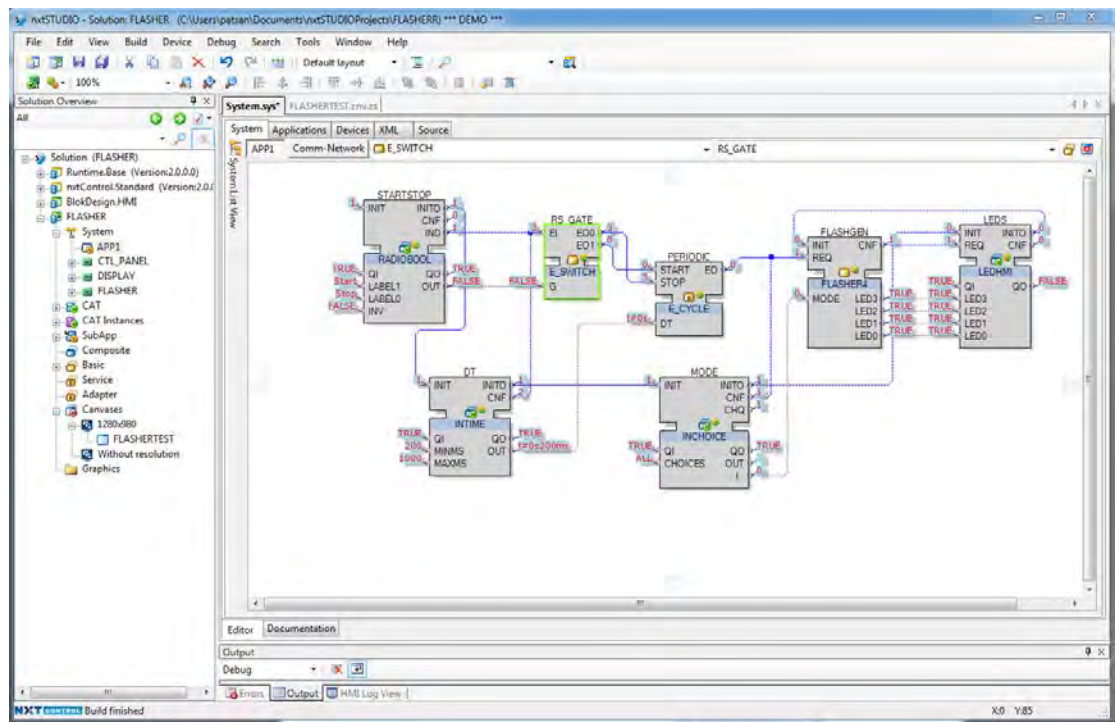


Рис. 29. Наблюдение за значениями переменных.

Лабораторная работа № 2

Разработка САТ-блоков

Цель работы – получить навыки разработки: а) САТ-блоков в среде NxtStudio; б) имитационной модели в виде функционально-блочного приложения, включающего САТ-блоки, с продвинутым человеко-машинным интерфейсом (HMI).

1. Разработка интерфейса «человек-машина» и связанной с ним сети функциональных блоков

В данном разделе работы речь пойдет о создании человеко-машинного интерфейса (HMI) в виде САТ-блока для объекта автоматизации «Вертикальный цилиндр».

- 1) Откройте проект, названный PnP2.
- 2) Правой кнопкой мыши щелкните на САТ-> Application и выберите New Item, затем задайте имя VCylinder для создаваемого САТ-приложения (рис. 1).

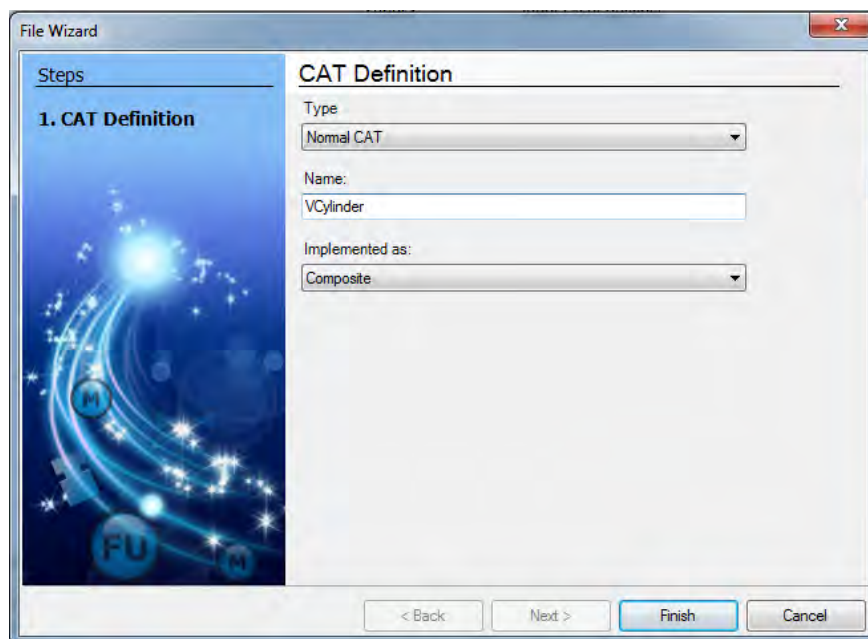


Рис. 1. Добавление нового САТ-приложения

3) Правой кнопкой мыши щелкните на только что созданном элементе CAT, выбирая VCylinder-> IThis:VCylinder_HMI, далее выберите Add HMI Symbol, назовите его HMI и удалите другой объект с именем sDefault (рис. 2).

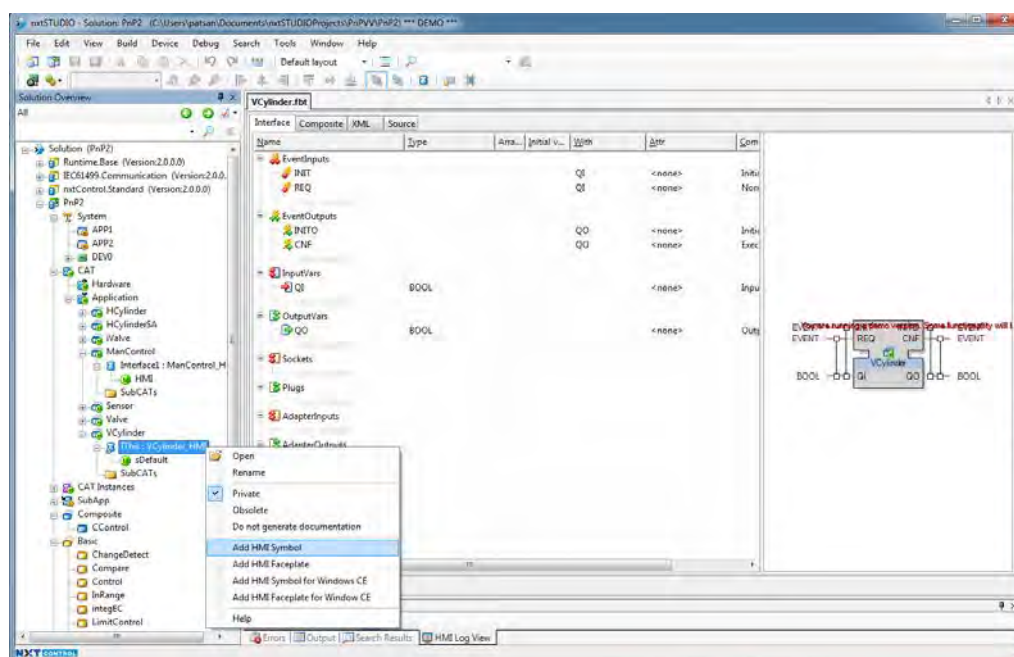


Рис. 2. Добавление символа HMI

4) Измените его интерфейс, как показано ниже на рис. 3.

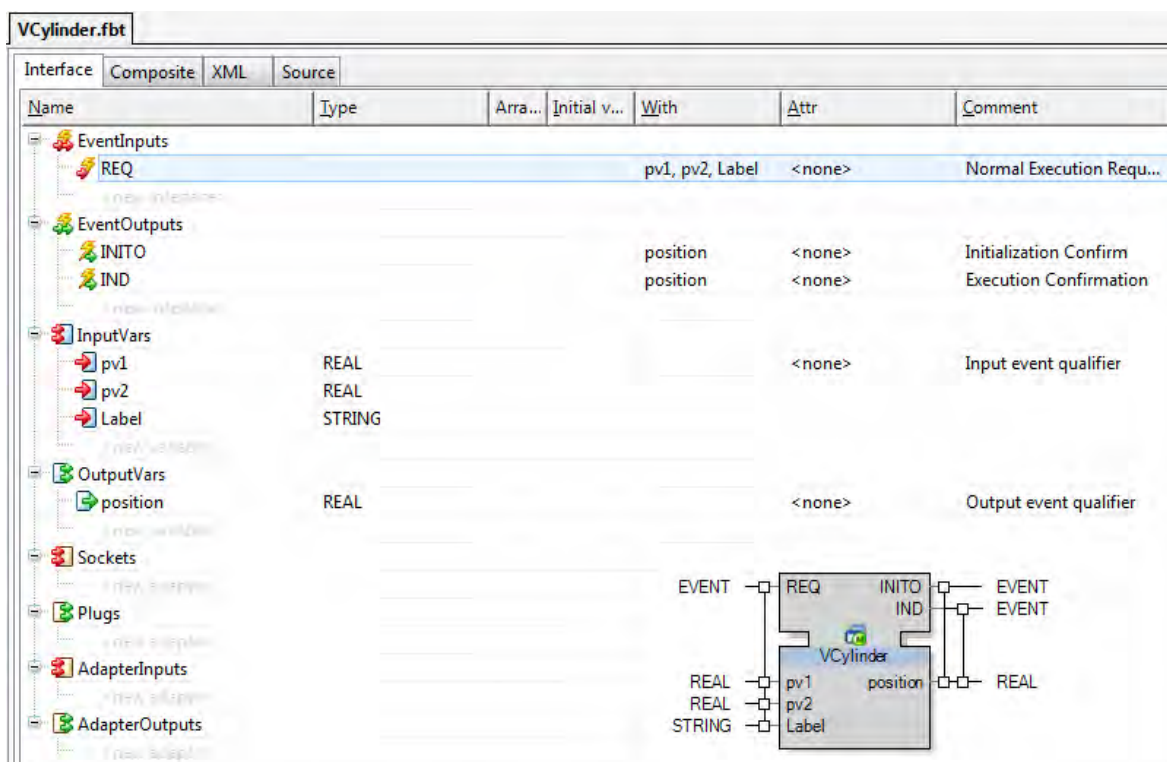


Рис. 3. Интерфейс VCylinder

5) Теперь откройте VCylinder_HMI и измените его интерфейс так, как показано на рисунке ниже.

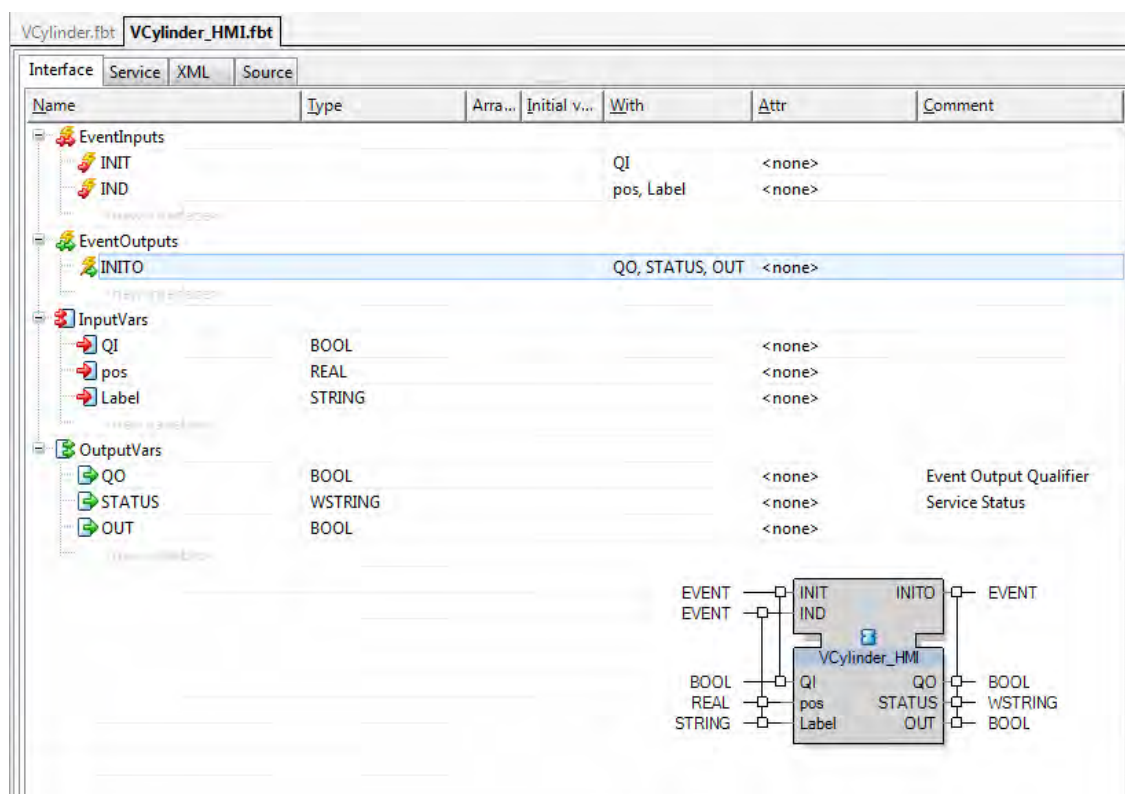


Рис. 4. Интерфейс VCylinder_HMI

6) Вернитесь на вкладку VCylinder, выберите Composite и разработайте составной блок, как показано на рис. 5, причем блоки integEC и ChangeDetect могут быть найдены в пункте Basic для базисных ФБ. VCylinder_HMI – это блок, который вы редактировали на предыдущем шаге. Эти три блока можно перетащить из дерева решений (Solution Overview) в рабочую область редактора (Canvas). Другие два блока: E_RESTART и E_CYCLE можно найти, щелкнув правой кнопкой мыши в рабочей области и выбрав FB-> Project: Runtime Base-> IEC61499.Standard.

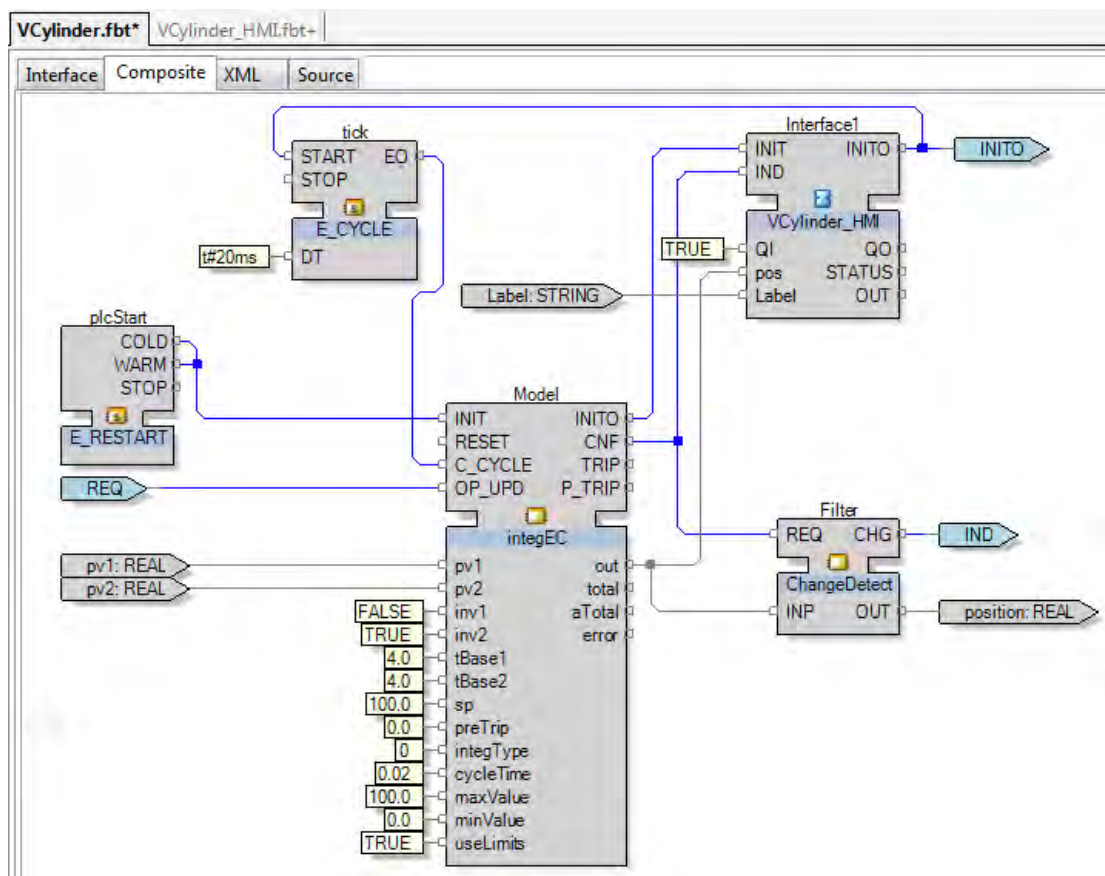


Рис. 5. Проектирование сети ФБ для VCylinder

7) Дважды щелкните на VCylinder-> VCylinder_HMI-> HMI, чтобы открыть рабочую область HMI. Откройте окно свойств, щелкнув на меню View системы и выбрав опцию Properties в нем. Измените размер рабочей области на 161,437 (рис. 6).

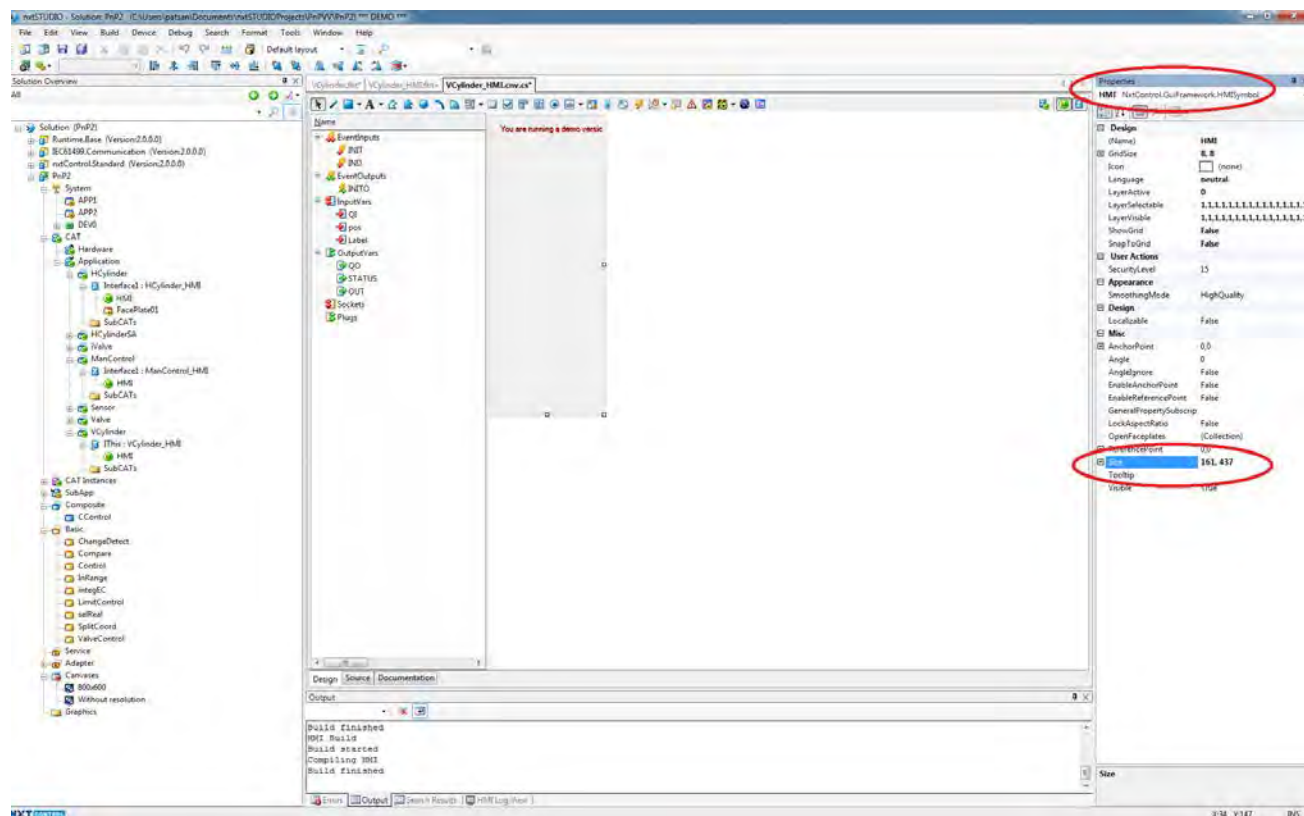


Рис. 6. Изменение размера рабочей области на 161,437

8) Теперь приступим к созданию объекта Цилиндр (Cylinder). Выберите из панели инструментов для конструирования фигур, которая находится между вкладками и рабочей областью, обозначение прямоугольника и нарисуйте произвольный прямоугольник, как показано на рис. 7. Измените свойства 'Pen', 'Location' и 'Size' в соответствии с рис. 7, оставив другие свойства по умолчанию.

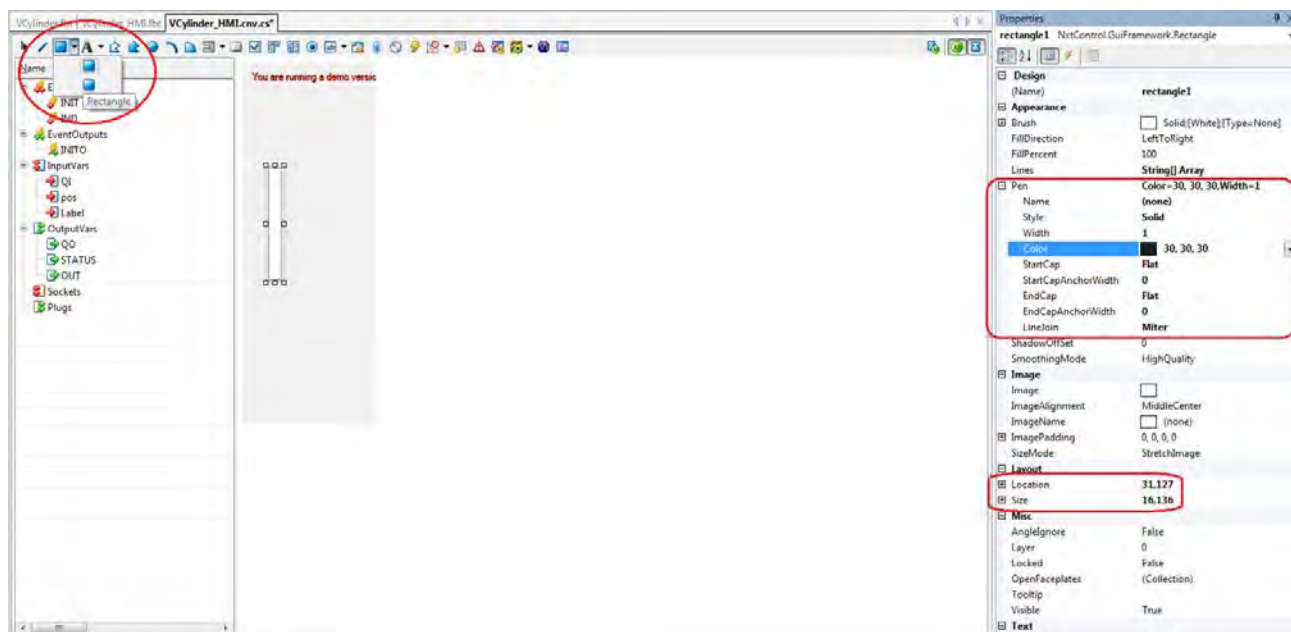


Рис. 7. Создание Цилиндра (шаг 1)

9) Придадим фигуре градиентный цвет. Чтобы сделать это, нужно нажать на кнопку выбора, которая находится за пунктом Brush (Кисть) в окне свойств, и выбрать значения, как показано на рис. 8.

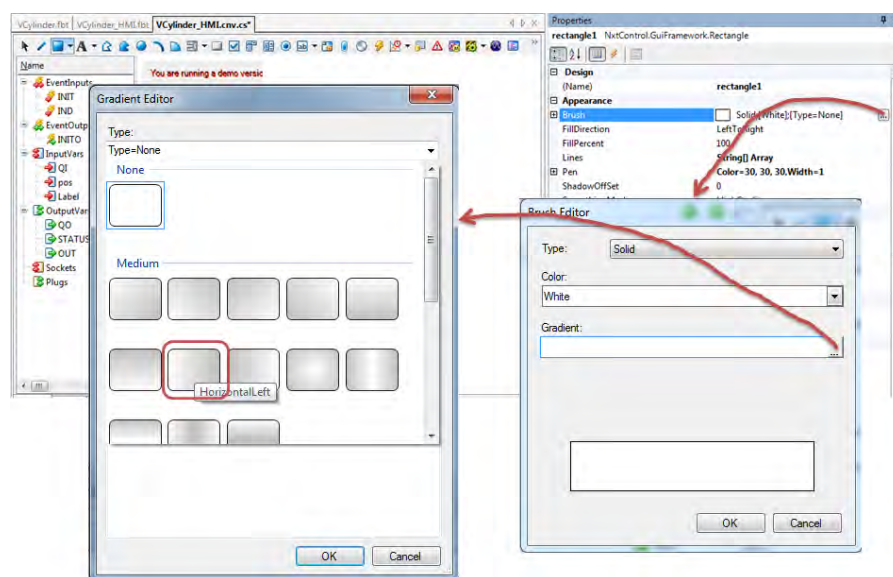


Рис. 8. Создание Цилиндра (шаг 2)

10) Установите флажок Use Color Blend. Используйте три точки установки, установите значения этих точек как показано на рис. 9.

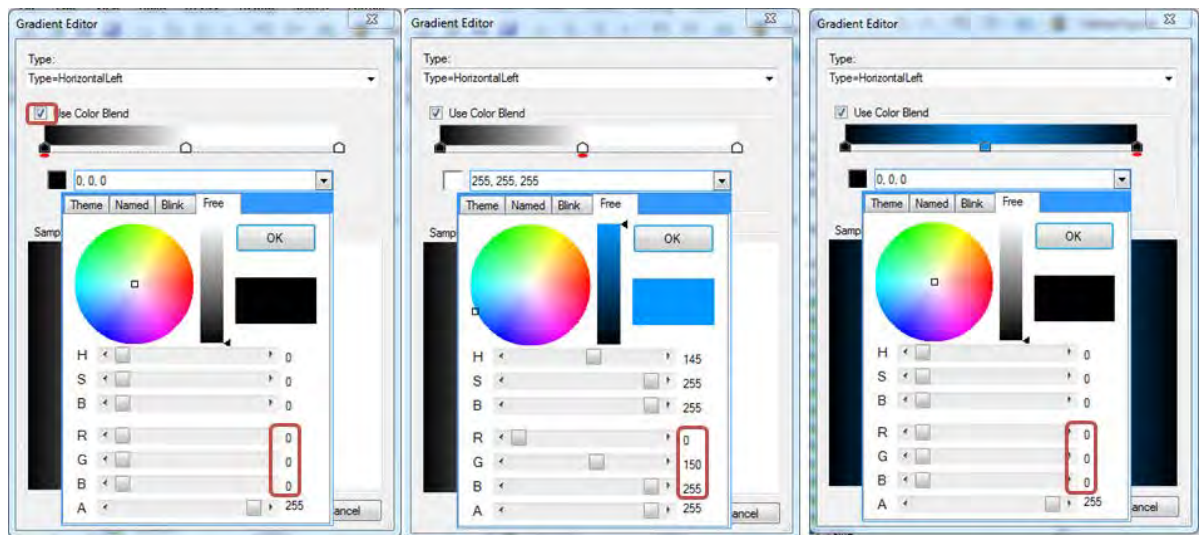


Рис. 9. Создание Цилиндра (шаг 3)

11) Должен получиться прямоугольник, представленный на рис. 10.

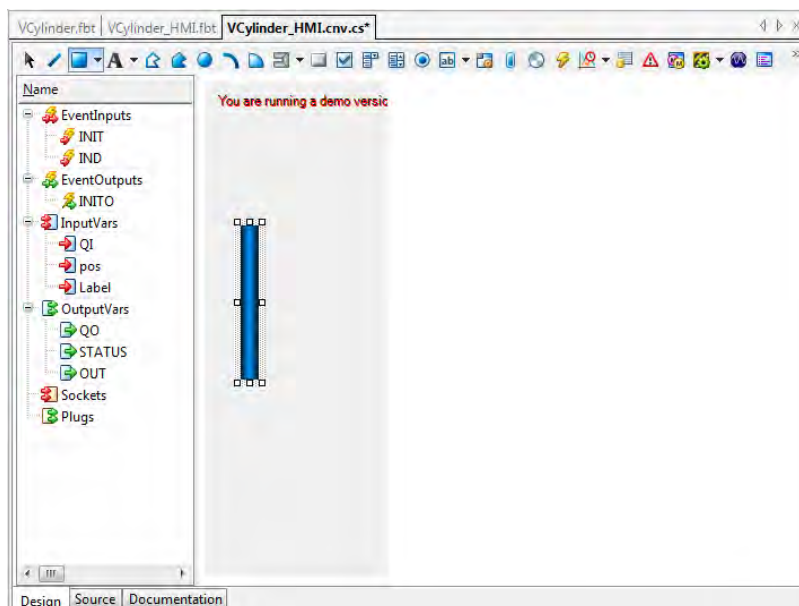


Рис. 10. Создание Цилиндра (шаг 4)

12) Два раза скопируйте и вставьте созданные прямоугольники и измените значения параметров, как показано на рис. 11.

+	Location	24,262
+	Size	29,15

+	Location	7,114
+	Size	65,13

Рис. 11. Создание Цилиндра (шаг 5)

13) Должна получиться фигура, показанная на рис. 12.

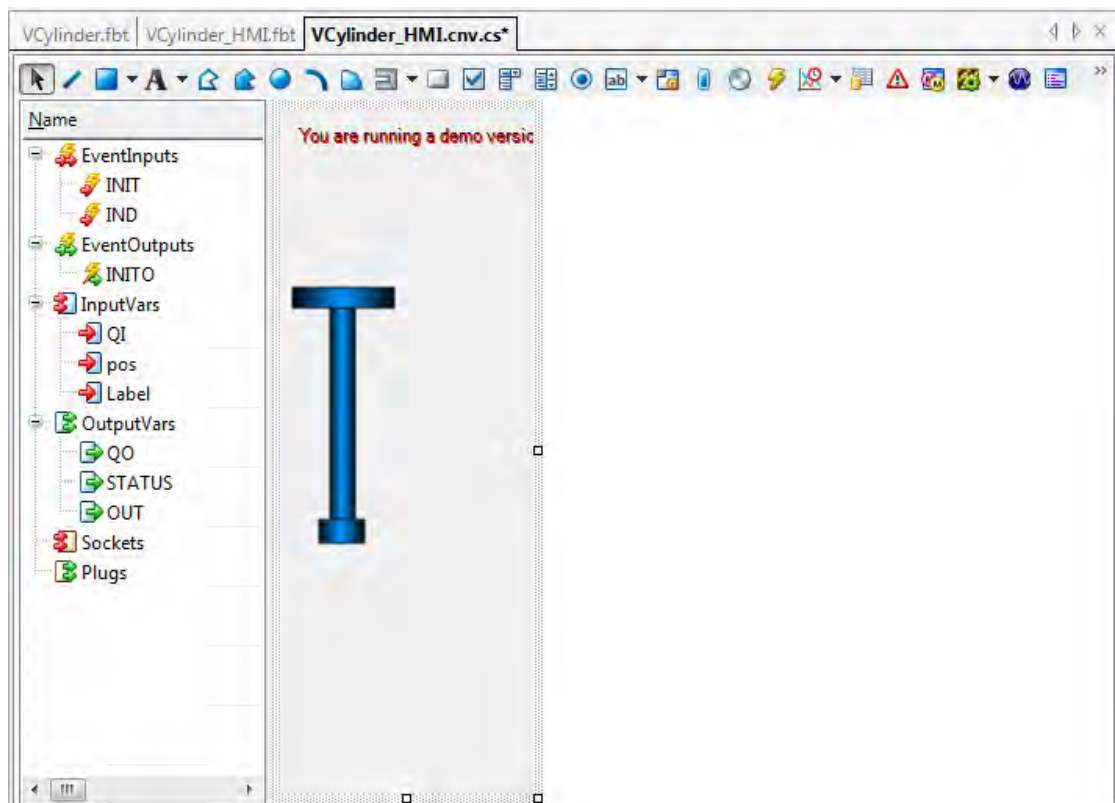


Рис. 12. Создание Цилиндра (шаг 6)

14) Создайте прямоугольник с закругленными краями, как показано на рис. 13.

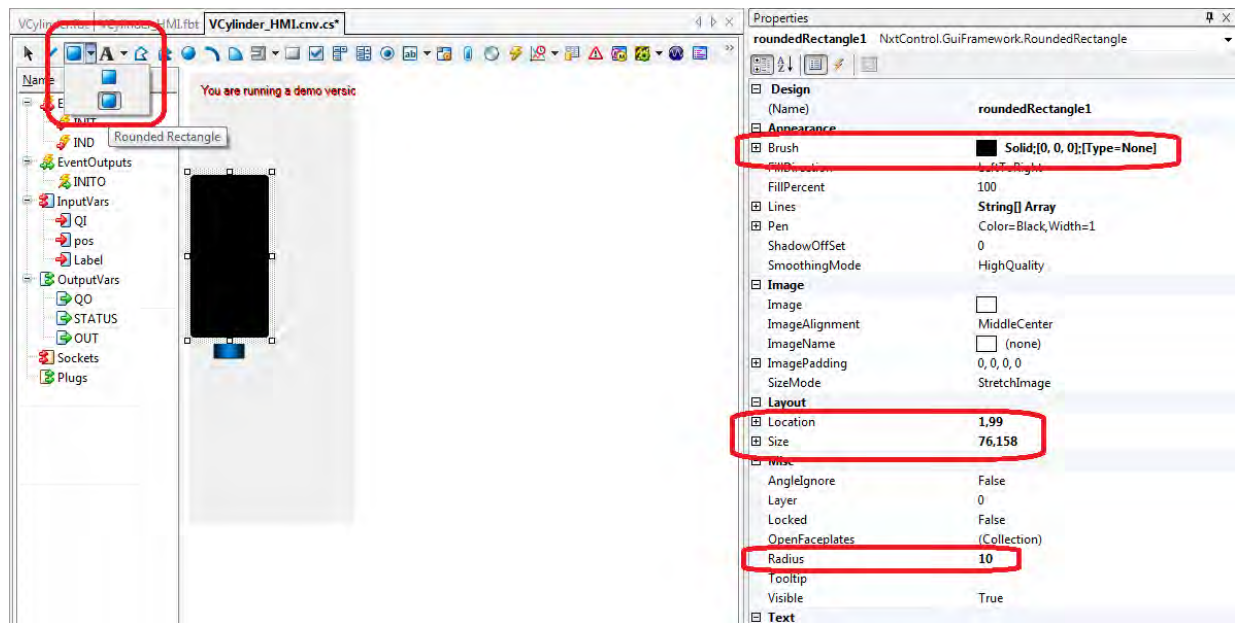


Рис. 13. Создание Цилиндра (шаг 7)

15) Теперь нужно поместить новый прямоугольник на задний план (рис. 14).

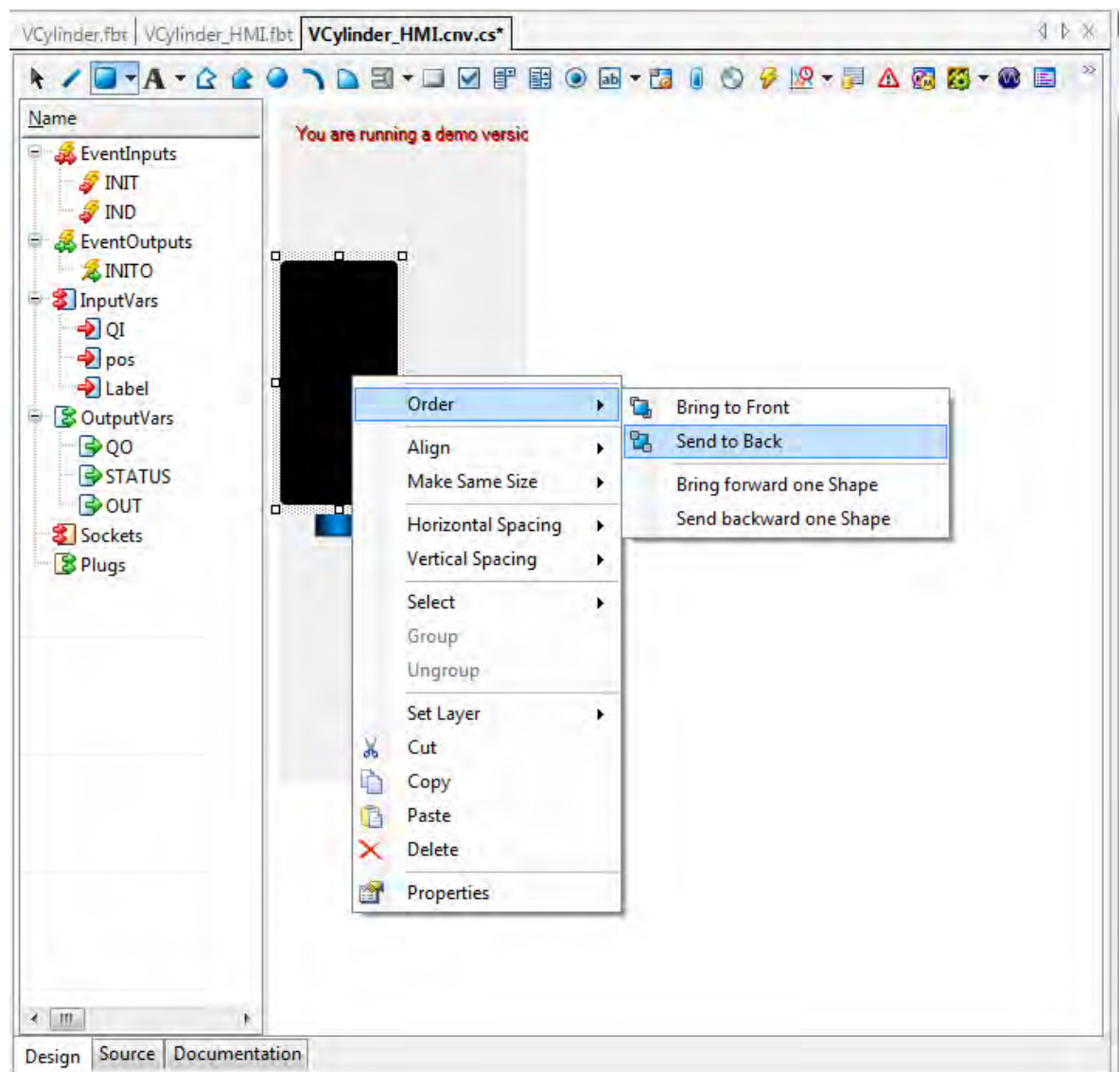


Рис. 14. Создание Цилиндра (шаг 8)

16) С использованием технологии «Скопировать-Вставить» создайте другой прямоугольник и измените его свойства в соответствии с рис. 15.

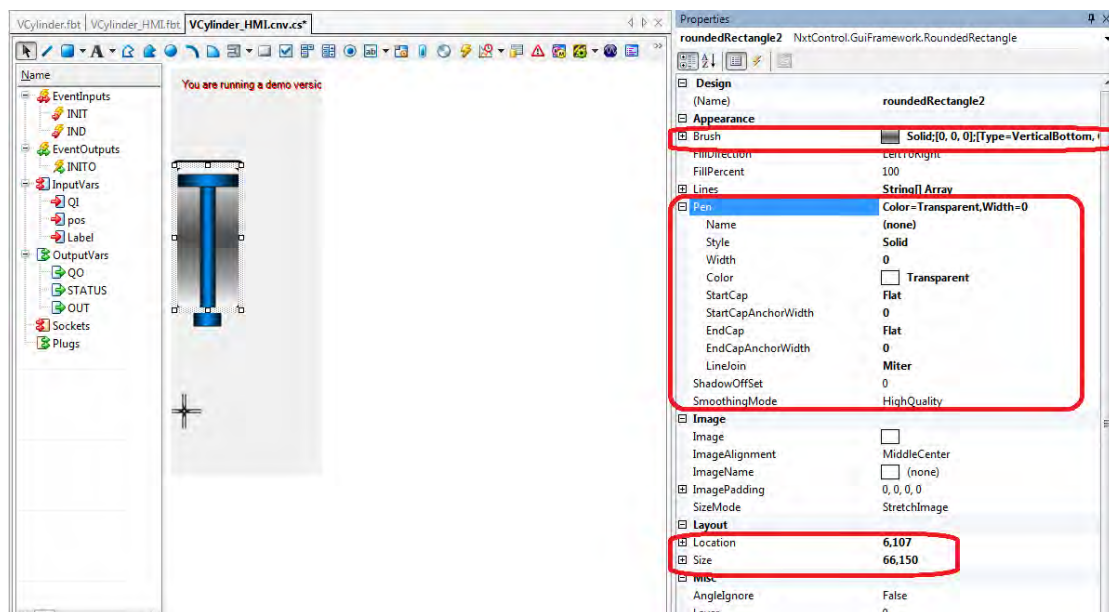


Рис. 15. Создание Цилиндра (шаг 9)

17) Настройте инструмент «Кисть», как показано на рис. 9, но с некоторыми изменениями. Вместо HorizontalLeft (горизонтальный левый) установите VerticalBottom (вертикальный нижний). Будет пять точек установки со значениями:

- a. 255, 255, 255
- b. 154, 154, 154
- c. 78, 78, 78
- d. 154, 154, 154
- e. 255, 255, 255

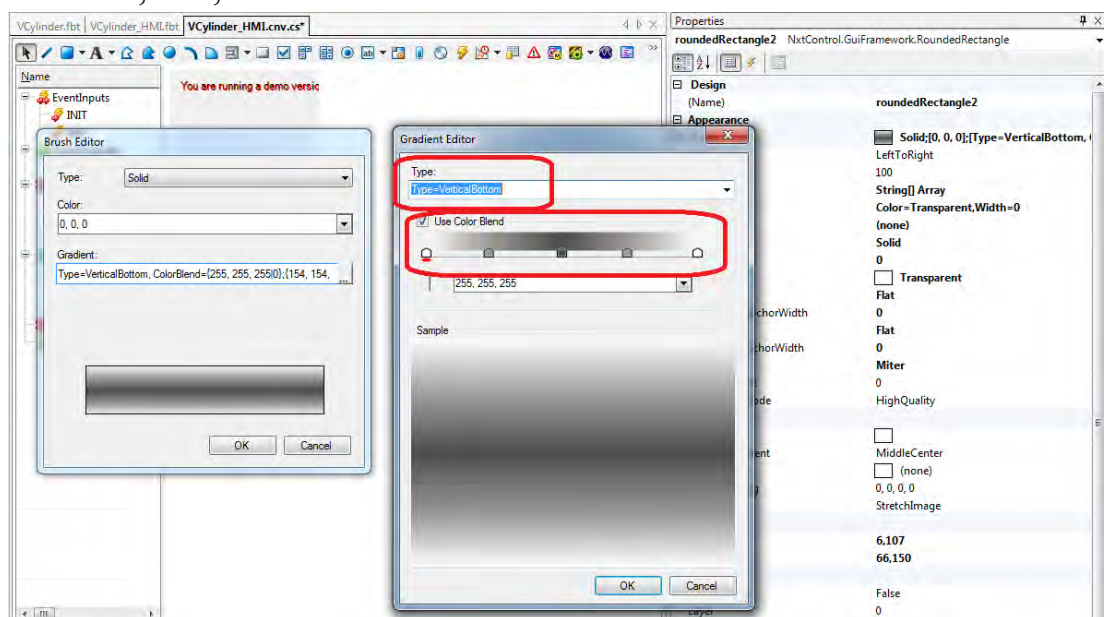


Рис. 16. Создание Цилиндра (шаг 10)

18) Нам нужно переместить новый прямоугольник назад на три слоя. Для этого три раза выберем пункт меню Send backward one Shape (рис. 17).

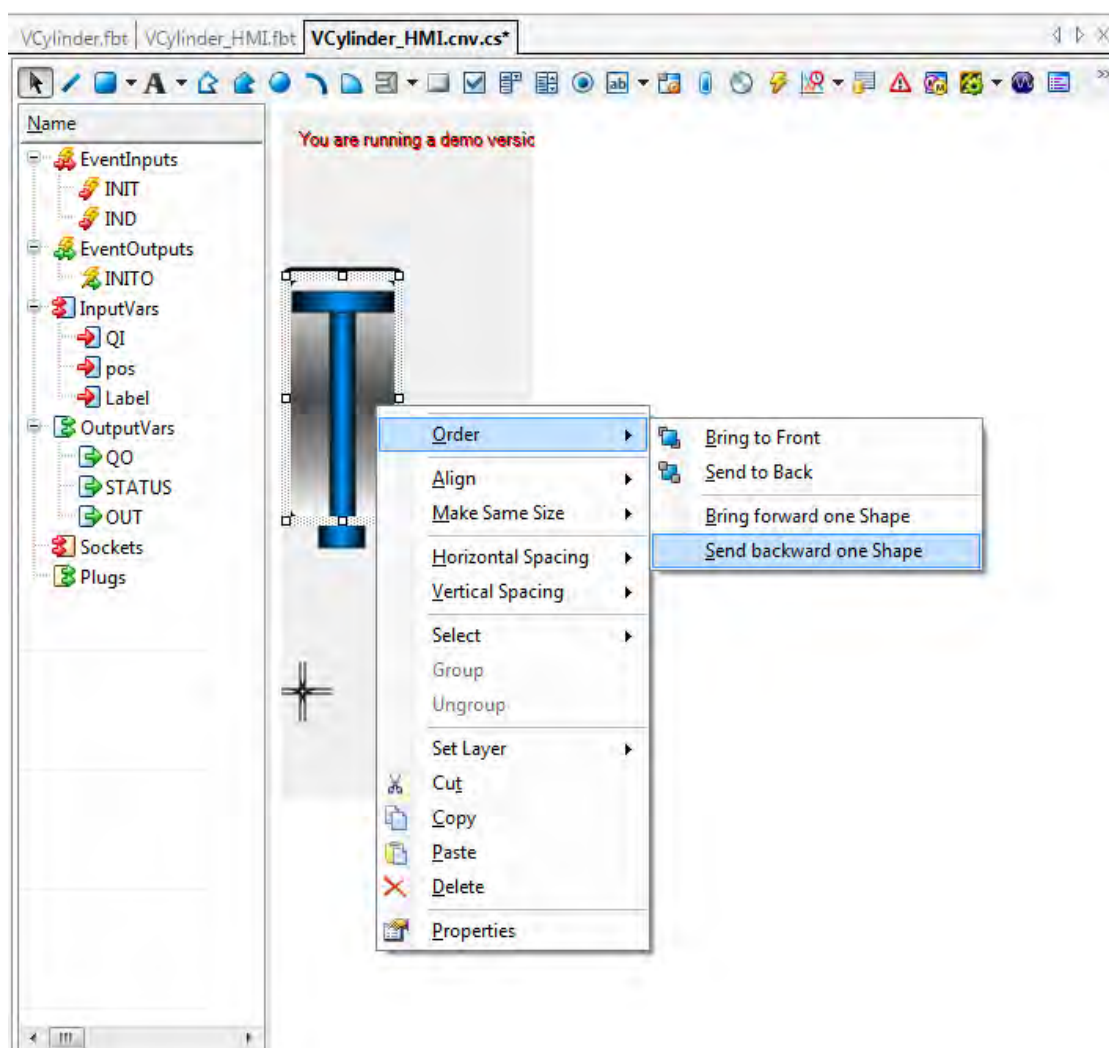


Рис. 17. Создание Цилиндра (шаг 11)

19) Выберем последние прямоугольники с закругленными краями и сгруппируем их. Для этого выделим эти фигуры, щелкнем на них правой кнопкой мыши и выберем пункт Group. Затем поместим группу на задний слой (примерно так же, как показано на рис. 14).

20) Аналогично создадим еще два прямоугольника с закругленными краями, делая и вставляя копию последнего прямоугольника, попутно устанавливая значения свойств. Тип будет HorizontalLeft вместо VerticalBottom, точки установки будут следующими:

- a. 255, 255, 255
- b. 234, 22, 30
- c. 178, 14, 18
- d. 234, 22, 30
- e. 255, 255, 255

Радиус будет равным '7', а размеры будут равны 66, 9. Положение (Location) для первой фигуры будет 6, 245, а второй - 6, 105.

Цилиндр должен выглядеть так, как показано на рис. 18.

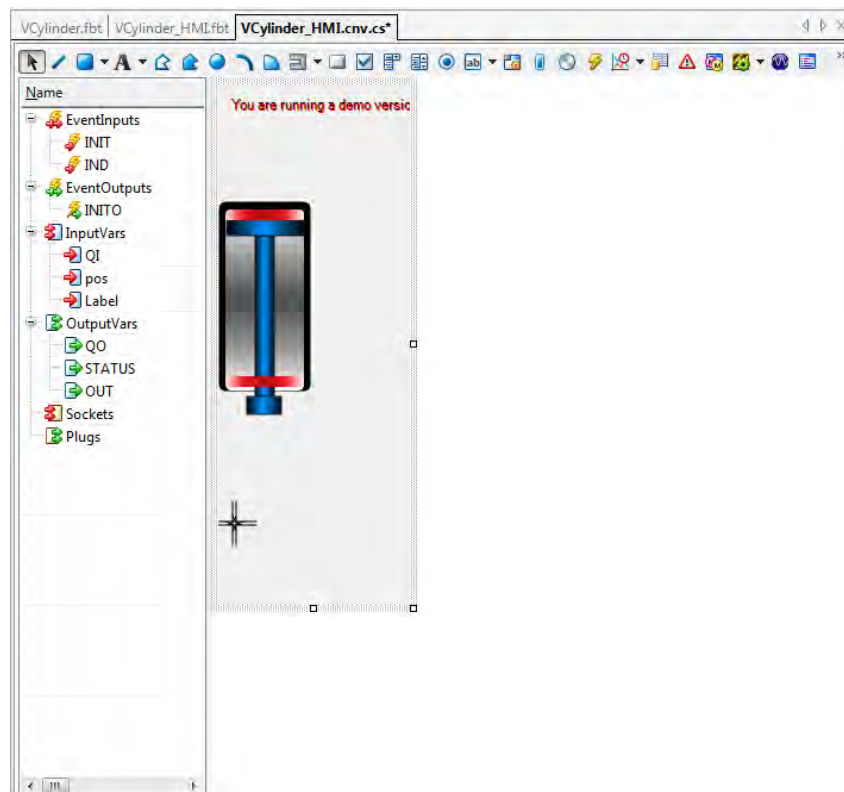


Рис. 18. Создание Цилиндра (шаг 12)

22) Последним элементом для комплексной фигуры «Цилиндр» будет многоугольник (Polygon). Создадим его, как показано на рис. 19. Настроим «Кисть» следующим образом: тип – HorizontalLeft, значения пяти точек установки:

- a. 255, 255, 255
- b. 114, 114, 114
- c. 210, 210, 210
- d. 114, 114, 114
- e. 255, 255, 255

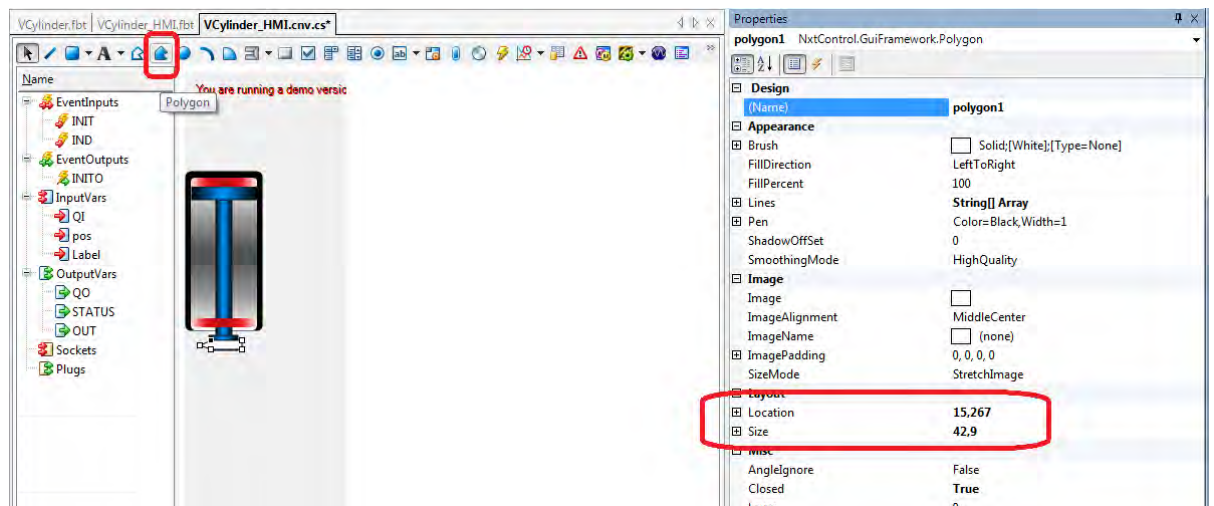


Рис. 19. Создание Цилиндра (шаг 13)

23) Теперь сгруппируем три прямоугольника с синим градиентом, два - с красным и один многоугольник в одну группу. После группирования Цилиндр будет выглядеть как на рис. 20.

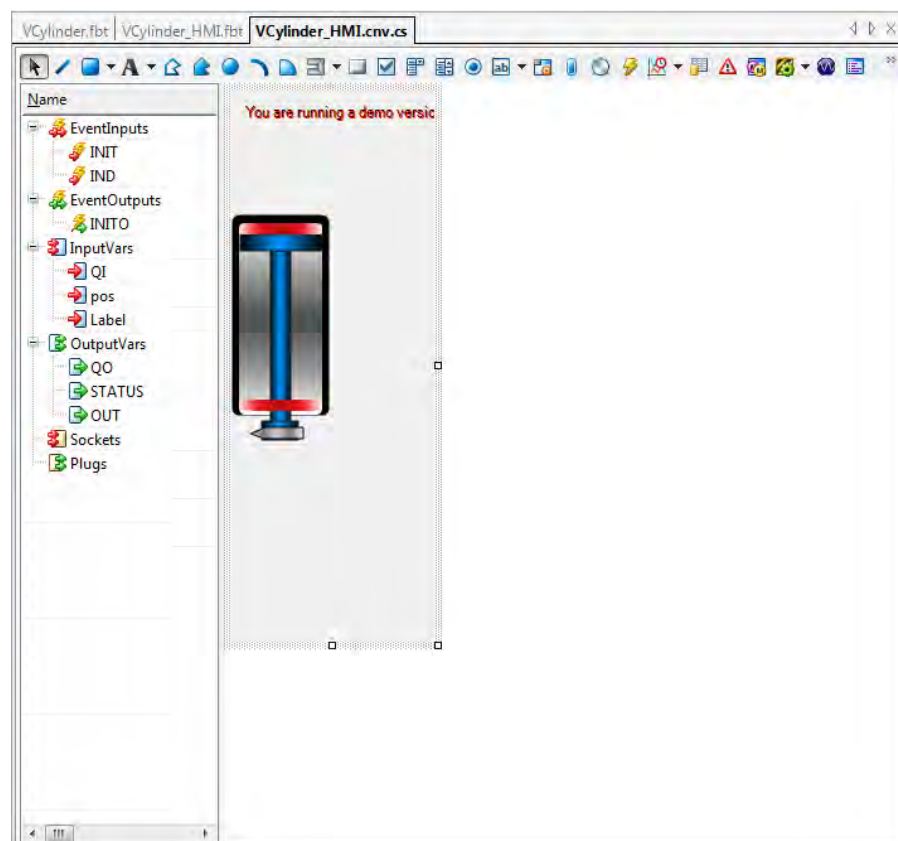


Рис. 20. Создание Цилиндра (шаг 14)

24) Создадим Метку (Label) и назовем ее Name со свойствами, показанными на рис. 21.

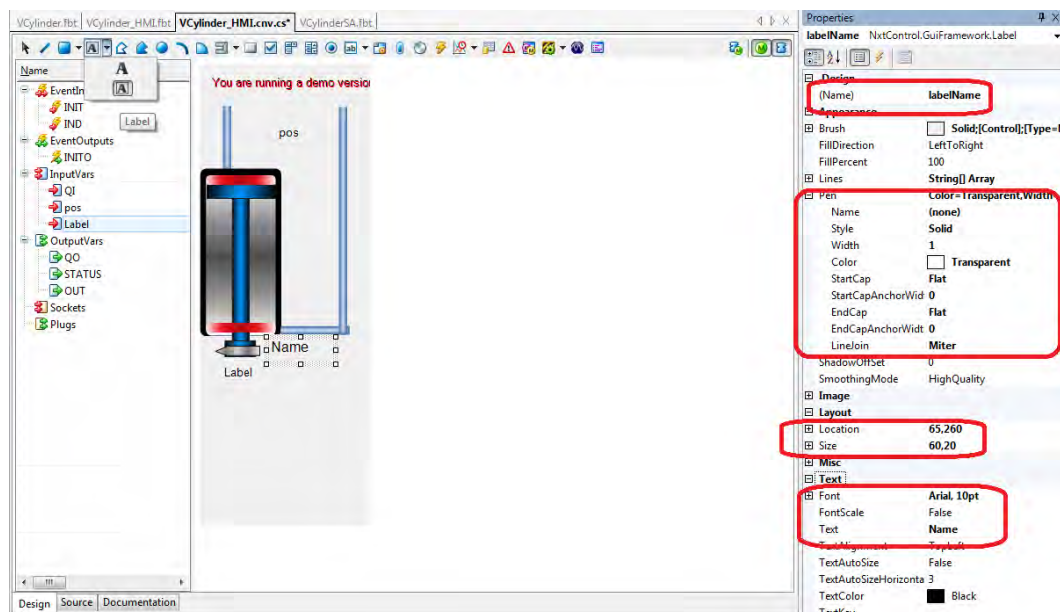


Рис. 21. Создание Цилиндра (шаг 15)

25) Остались последние три элемента для человеко-машинного интерфейса. Необходимо создать Трубу (Pipe) - рис. 22 показывает, как это сделать.

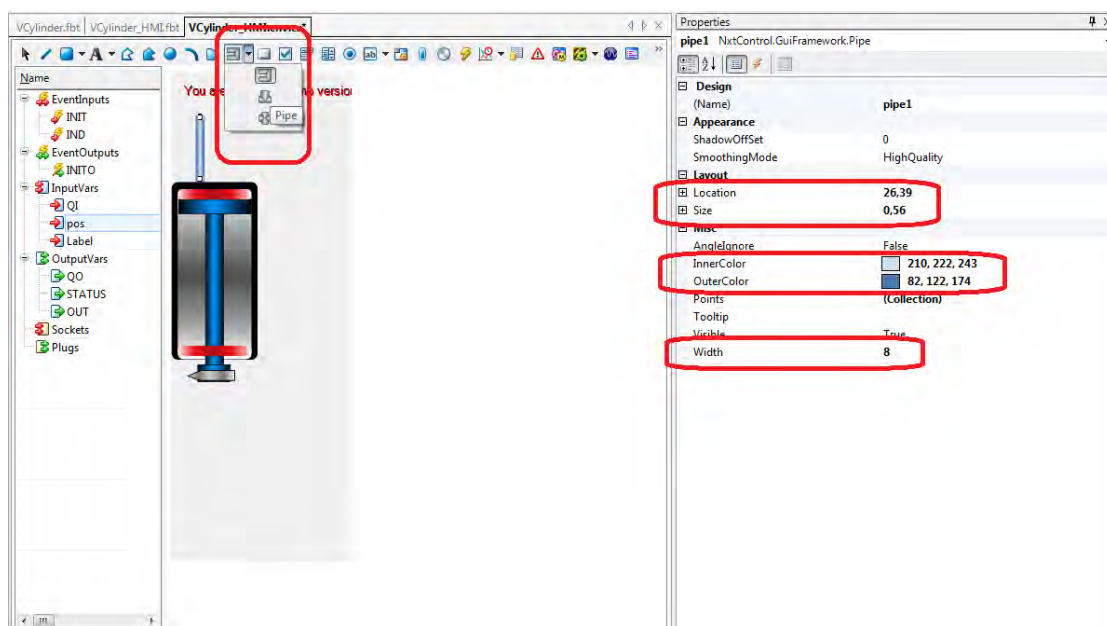


Рис. 22. Создание Цилиндра (шаг 16)

26) Аналогичным образом создадим еще две трубы Pipe1 и Pipe2 со следующими отличиями:

- a. Pipe1
 - i. Location – 136, 40

- ii. Size – 0, 211
- b. Pipe2
 - i. Location – 77, 252
 - ii. Size – 60, 0

27) Теперь создание человеко-машинного интерфейса завершено, он должен выглядеть так, как показано на рис. 23.

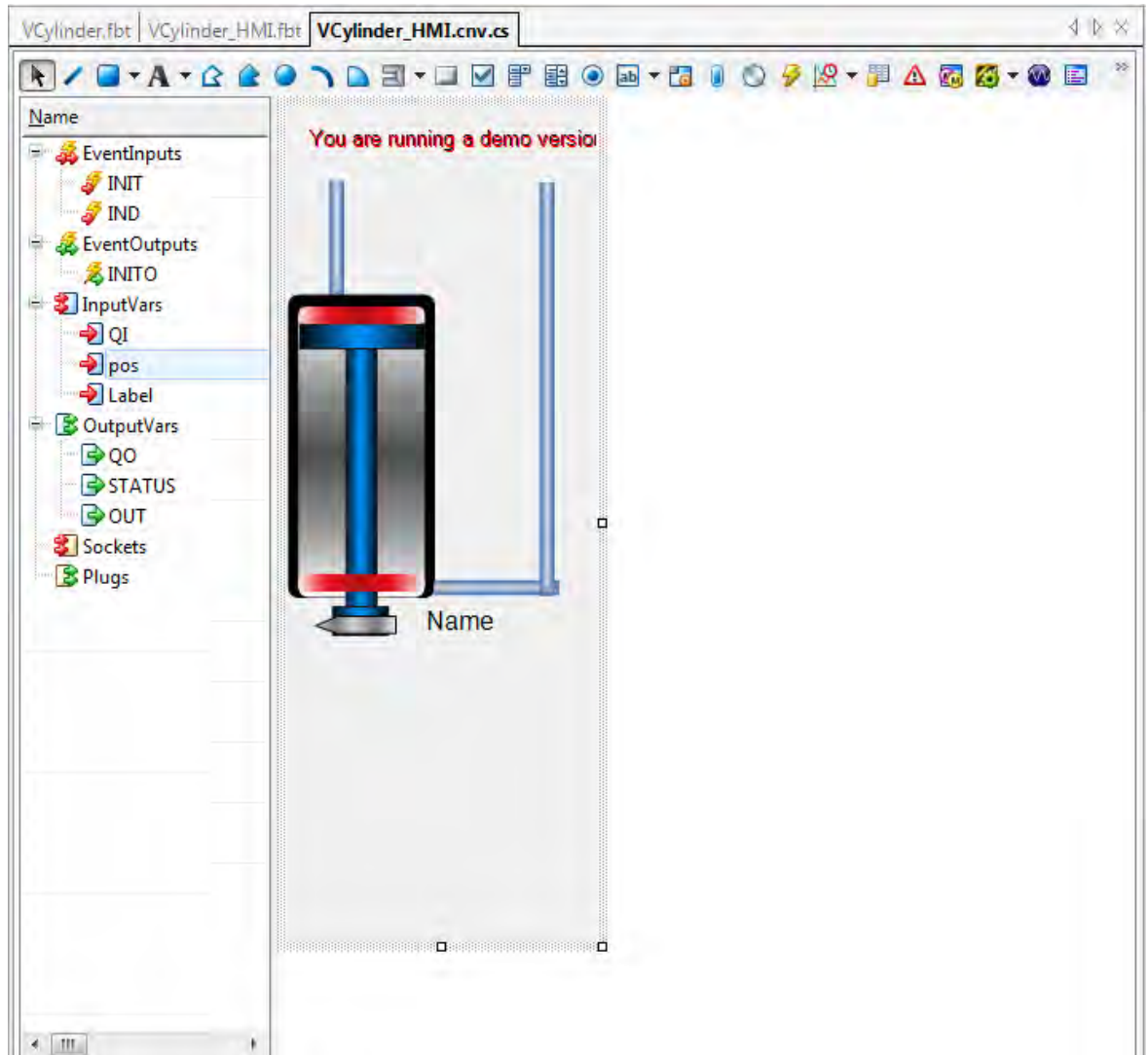


Рис. 23. Создание Цилиндра (шаг 16)

28) Чтобы облегчить программирование (кодирование), которое будет рассмотрено в следующем разделе, переименуем некоторые элементы. Группу, которая была создана в пункте 23, переименуем в grpInnerPart. Верхний красный прямоугольник переименуем в isMin, а нижний - в isMax.

2.2. Добавление логики в CAT HMI

Существуют два входа, на которые должен реагировать HMI: pos и Label. Для того, чтобы добавить функциональность для обработки изменения информации на этих входах, нужно проделать следующее.

Перетащите pos в рабочую область редактора. Когда Вы это сделаете должно возникнуть диалоговое окно, как показано на рис. 24. Выберите опцию Execute.

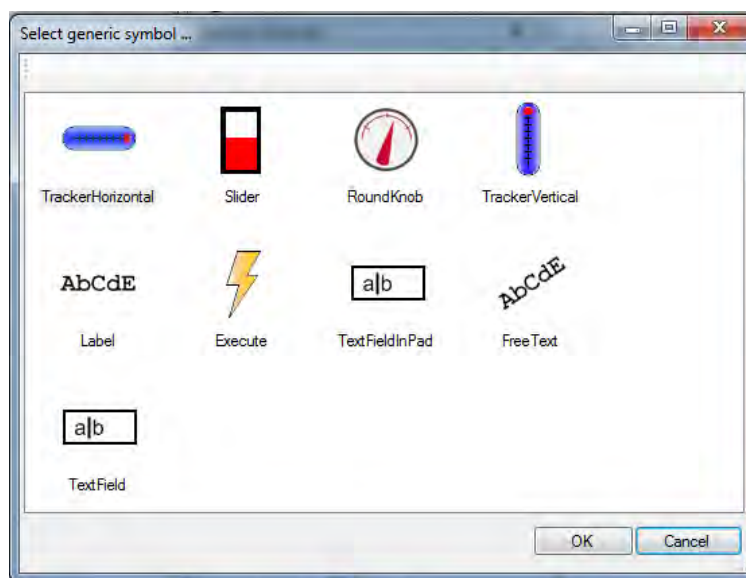


Рис. 24. Логика CAT (шаг 1)

Дважды щелкните мышью на только что добавленном элементе pos. Перед вами будет открыт редактор кода языка C#. Нажмите Yes в возникшем окне. Появится окно, приведенное на рис. 25. Для переключения представлений используйте вкладки Design и Source.

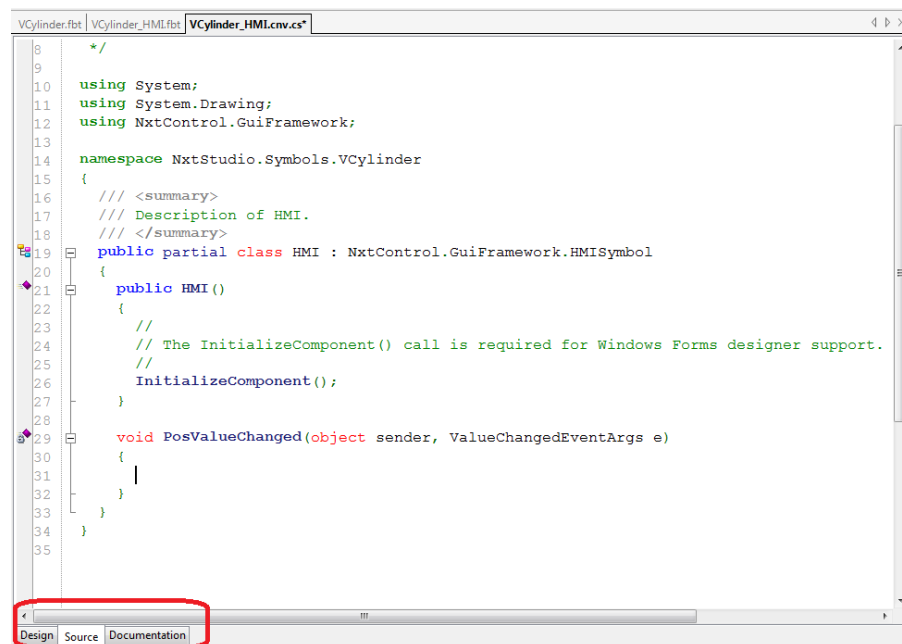


Рис. 25 Логика CAT (шаг 2)

Добавьте следующий код в функцию *PosValueChanged*:

```

NxtControl.Drawing.PointF newPos = grpInnerPart.Location;
float position = (float)e.Value;
double relpos = 115.0d/100.0d * position +
115.0d;

if (relpos > 220.0d) {
    isMax.Visible = true;
    isMin.Visible = false;
} else if (relpos < 125.0d) {
    isMax.Visible = false;
    isMin.Visible = true;
} else {
    isMax.Visible = false;
    isMin.Visible = false;
}
newPos.Y = relpos;
grpInnerPart.Location = newPos

```

Выполните похожие действия для входа Label. Код для функции *LabelValueChanged* будет следующим:

```

labelName.Text=(string)Label.Value;

```

2.3. Составной САТ-блок

Внимательно изучите САТ-блок *HCylinderSA* («Горизонтальный цилиндр»). Нам необходимо создать данный блок по аналогии с ранее созданным САТ- блоком «Вертикальный цилиндр». Для этого необходимо выполнить следующие действия.

- 1) Создайте новый элемент САТ и назовите его *VCylinderSA*.
- 2) Раскройте этот элемент и переименуйте *sDefault* в *HMI*.
- 3) Правой кнопкой мыши щелкните на элементе *SubCATs* и добавьте САТ-блок *Vcylinder*, созданный ранее, как показано на рис. 26.

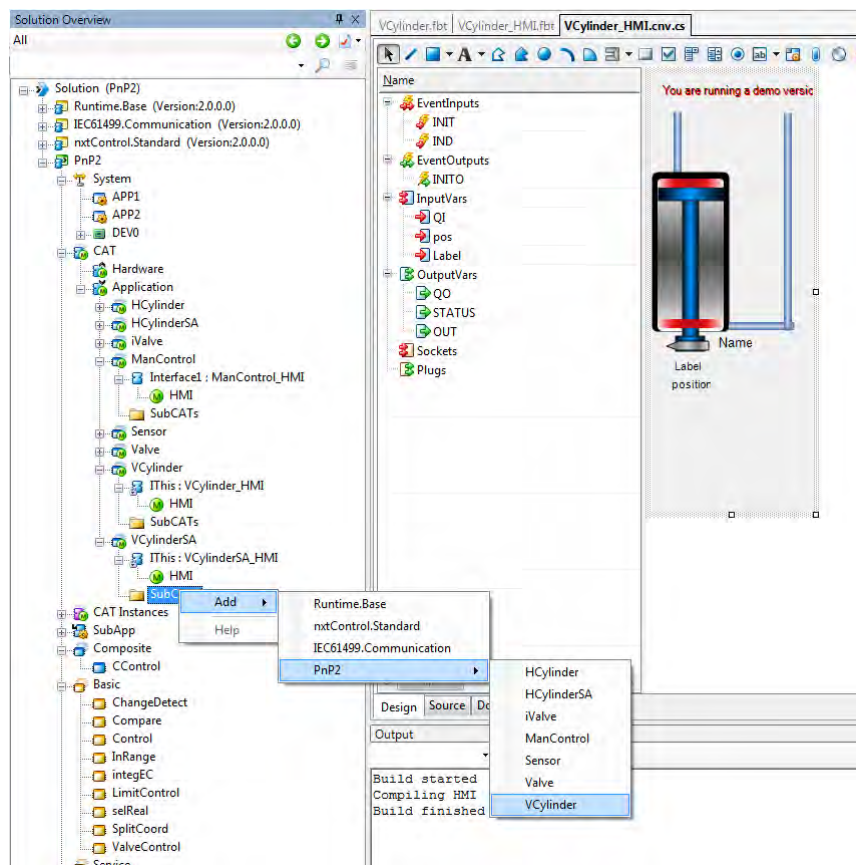


Рис. 26. Создание составного САТ-блока (шаг 1)

- 4) Аналогично добавьте два экземпляра САТ-блока *Valve* («Клапан») и два экземпляра САТ-блока *Sensor* («Датчик»). В итоге объект *SubCATs* после добавления в него пяти САТ-блоков должен выглядеть, как показано на рис. 27. Следует отметить, что добавленные на рис. 27 объекты были

переименованы. Их имена по умолчанию были VCylinder, Sensor1, Sensor2, Valve1 и Valve2.

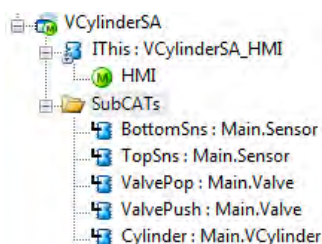


Рис. 27. Создание составного САТ-блока (шаг 2)

5) Откройте САТ-блок VCylinderSA и измените его интерфейс так, как показано на рис. 28.

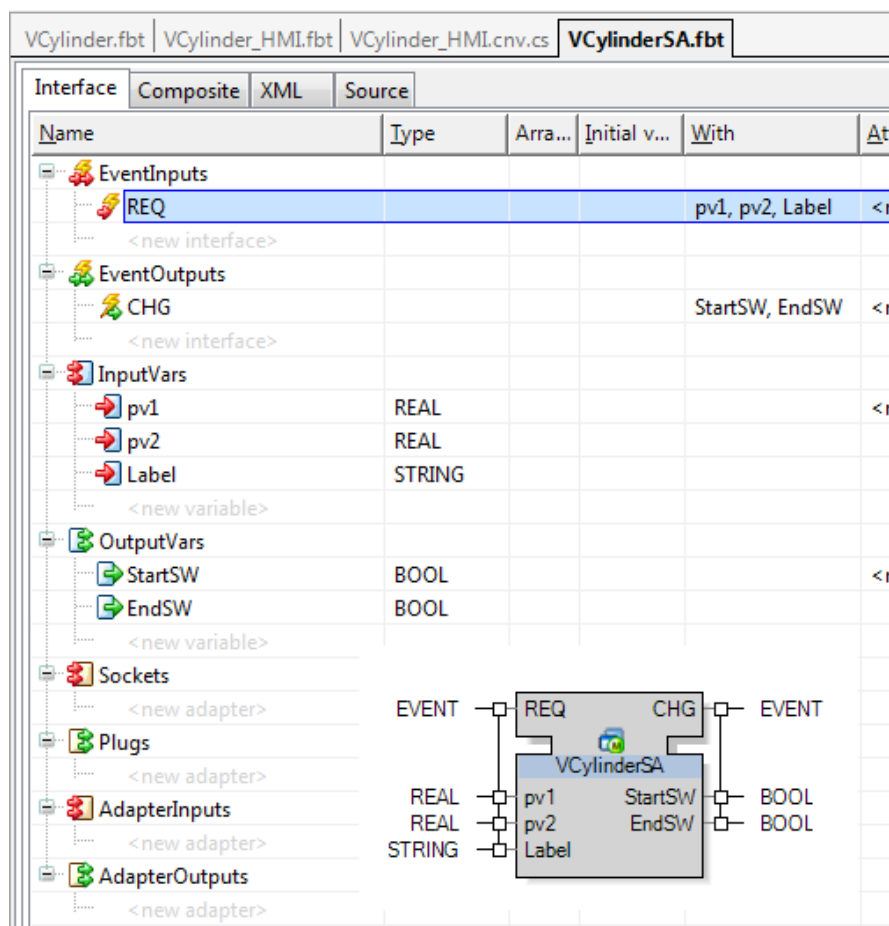


Рис. 28. Создание составного САТ-блока (шаг 3)

6) Переключитесь на вкладку Composite и проведите связи, как показано на рис. 29. Имейте в виду, что блоки нужно перетащить из секции SubCATs.

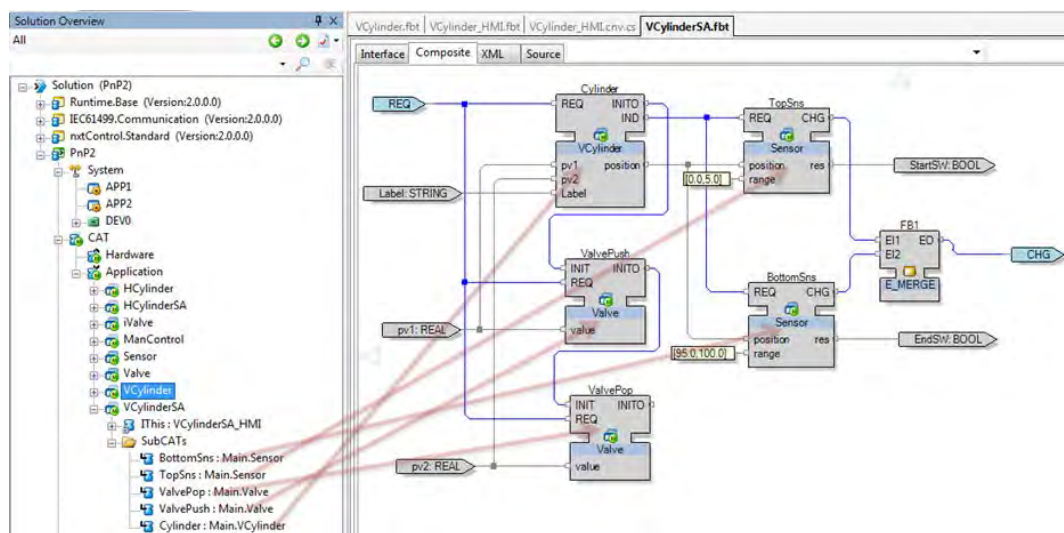


Рис. 29. Создание составного САТ-блока (шаг 4)

Теперь дважды кликните на HMI-части VCylinderSA и, так же как и на предыдущем шаге, перенесите все элементы из секции SubCATs в рабочую область редактора, и используйте следующие параметры:

Блок	Свойства
Actual Canvas (имеющее имя HMI в окне свойств)	Size -> 228, 441
Cylinder	Location-> 0,39
TopSns	Location-> 80,114
BottomSns	Location-> 80,227
ValvePush	Location-> 5,10
ValvePop	Location-> 118,10

8) И, наконец, создайте прямоугольник со следующими свойствами (рис. 30).

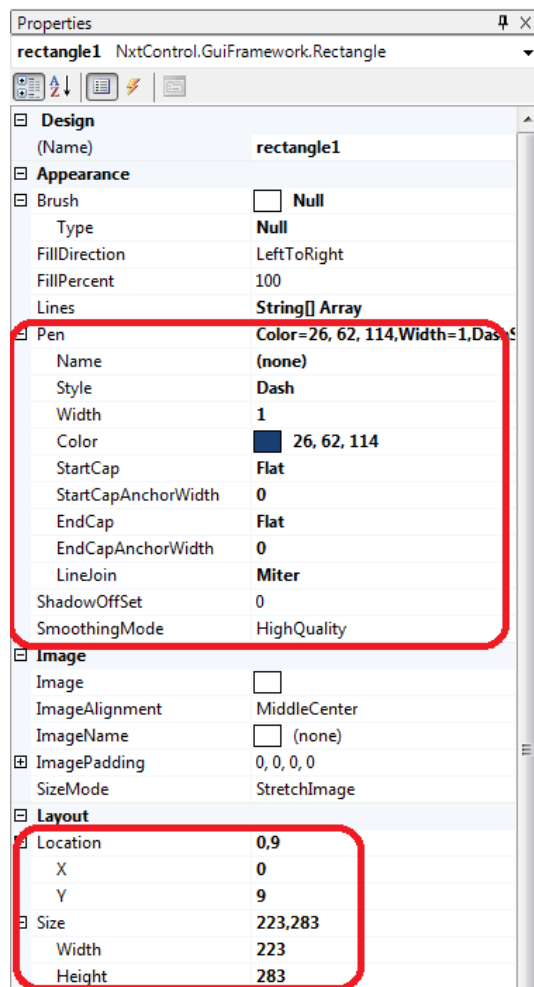


Рис. 30. Создание составного САТ-блока (шаг 5)

9) Окончательный вариант должен выглядеть, как показано на рис. 31.

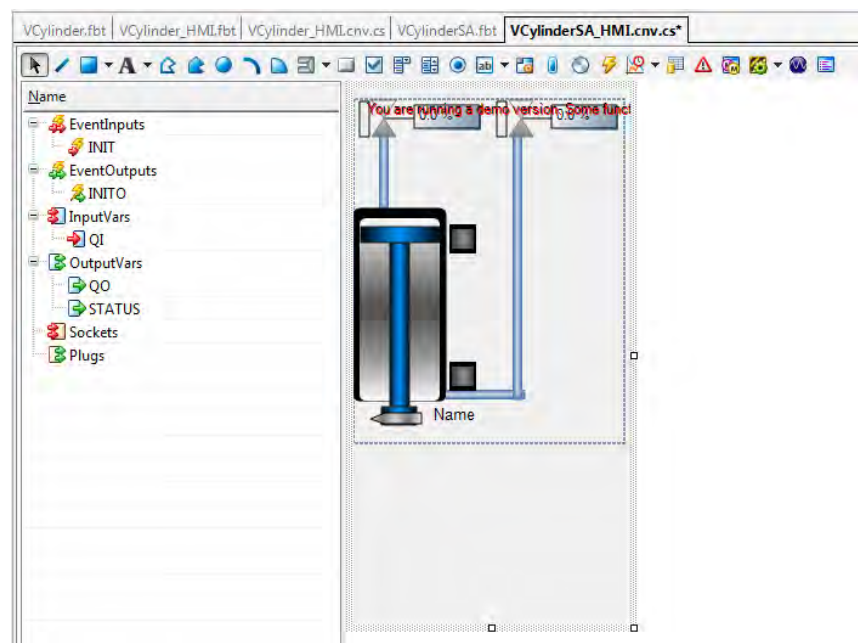


Рис. 31. Создание составного САТ-блока (шаг 6)

2.4. Расширение приложений

Проект шаблона уже имеет несколько блоков в элементе APP1, который находится под элементом System в дереве решений. К нему нужно добавить только что созданный VCylinderSA. Для того, чтобы сделать это, просто перетащите CAT-блок VCylinderSA в систему System, как показано на рис. 32, и создайте соответствующие связи.

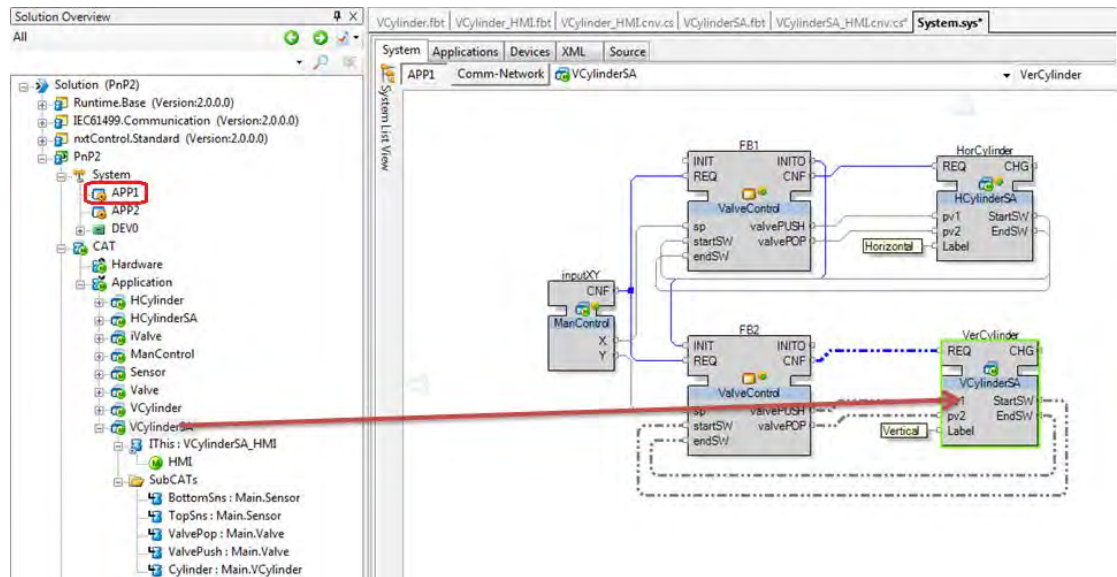


Рис. 32. Добавление CAT-блока VCylinderSA в приложение APP1

Вы можете заметить пунктирные линии. Это связано с тем, что вновь добавленный блок не отображен ни на одно Устройство/Ресурс. Отобразите его на DEV0.RES0 (для этого нажмите правой кнопкой мыши на указанный блок -> Mapping -> DEV0.RES0).

2.5. Расширение рабочей области

Как только мы расширили приложение, сразу возникает необходимость расширить рабочую область. Перетащите созданный экземпляр в рабочую область *MainCanvas* под *Canvases* -> 800x600.

Обратите внимание, что Вам нужно перенести блок из CAT Instances -> Application, а не из CAT -> Application (рис. 33). Если после перетаскивания ничего не появилось, то сохраните изменения и перезапустите nxTStudio.

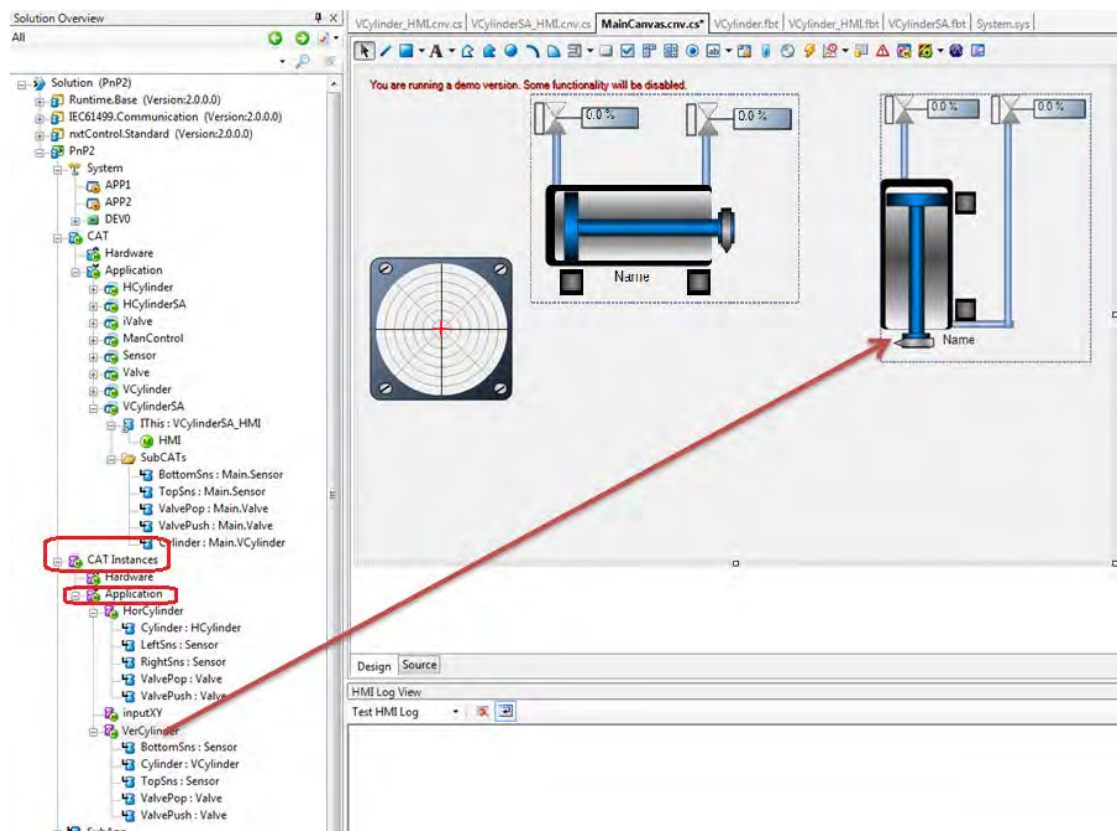


Рис. 33. Расширение рабочей области

Следуя процедуре развертывания из лабораторной работы № 1, запустите приложение, управляйте движением цилиндров с помощью джойстика

2.6. Полезная информация по реализации HMI на основе CAT-блоков с использованием C#

1) Входные события в CAT_HMI

Входная переменная QI по умолчанию связана с событийным входом INIT. В случае, если интерфейс по данным необходимо расширить, новые информационные входы должны быть ассоциированы с событийным входом, отличным от INIT. Можно добавить событийный вход, например, с именем REQ, и ассоциировать новые информационные входы с ним. Если желательно изменить эти информационные входы при инициализации, вход INIT реализующего CAT может быть связан с обоими входами INIT и REQ этого CAT-блока HMI-интерфейса.

2) Выходные события в CAT_HMI

Выходное событие INITO по умолчанию выдается внутри метода public_HMI, который определяется следующим образом:

```
public HMI()  
{  
    InitializeComponent();  
    //Insert additional initialisation code HERE  
}
```

Этот метод вызывается, когда CAT-блок HMI-интерфейса вызывается событием INIT. Событие INITO выдается после выполнения этого метода. Другой код для инициализации рекомендуется добавлять после вызова InitializeComponent(). В случае, если в CAT-блоке HMI необходимо определить новые информационные выходы, и их значения должны быть определены во время инициализации, они не могут быть ассоциированы с INITO. Следовательно, новые информационные выходы могут быть ассоциированы с другим событийным выходом. Однако, это событие не может быть выдано в данном коде инициализации.

3) Формат метода FireEvent

Если выходное событие (назовем его EO) CAT-блока HMI связано с информационными выходами DO1, DO2, Don, то соответствующий формат метода, генерирующего событие, определяется как FireEvent_EO(value1, value2, ..., value n), где параметры-значения value соответствуют типам данных выходов DO1, ..., DOn. Однако, если событие ассоциировано с выходными переменными по умолчанию QO и STATUS, соответствующие значения в списка параметров FireEvent_EO будут опускаться. Выходные переменные по умолчанию всегда первые в этом списке и не могут быть удалены. Например, если событийный выход IND ассоциирован с тремя информационными выходами QO, STATUS и NEWDATA, то метод, выдающий событие IND и устанавливающий NEWDATA в значение 3, будет выглядеть как FireEvent_IND(3).

4) Ограничения

-- Событийный выход INITO САТ-блока НМІ не может быть связан с другими выходами этого САТ-блока, отличными от INITO;

-- Событийный выход INITO САТ-блока НМІ не может быть ассоциирован с другими информационными выходами, отличными от QO и STATUS.

Лабораторная работа № 3

Изучение коммуникационных блоков

Цель работы

Изучить работу коммуникационных блоков в среде NxtStudio.

Задание

Реализовать в среде NxtStudio приложение, которое бы позволяло передавать сообщения между разными устройствами с использованием коммуникационных блоков.

Ход работы

Для передачи информации между устройствами в среде NxtStudio существует библиотека «IEC61499.Communication», которая входит в комплект с демо-версией программы. В ней реализованы сервисные интерфейсные функциональные блоки следующих типов:

- **Publish** («писатель»), осуществляет рассылку данных всем блокам-читателям, имеющим тот же IP-адрес и порт подключения;
- **Subscribe** («читатель»), принимает данные, рассылаемые соответствующим «писателем»;
- **Client**, может как отправлять данные на блок-сервер, так и получать их от него;
- **Server**, может отправлять данные блоку-клиенту и получать их от него.

Таким образом связка блоков «Publish – Subscribe» предназначена для осуществления широковещательной рассылки между устройствами, а «Client – Server» для соединения «point-to-point».

В данной лабораторной работе для передачи данных между устройствами воспользуемся блоками Publish, Subscribe.

После создания проекта необходимо подключить соответствующую библиотеку функциональных блоков. Для этого нужно кликнуть правой

кнопкой мыши по названию проекта и перейти во вкладку References. Данное окно представлено на рисунке 1. Если необходимая библиотека установлена в среду NxtStudio, то она будет отображаться в верхней таблице, а если библиотека уже привязана к проекту, то в нижней. Для привязки библиотеки необходимо выбрать её в верхней таблице и нажатием кнопки «Add» добавить к проекту.

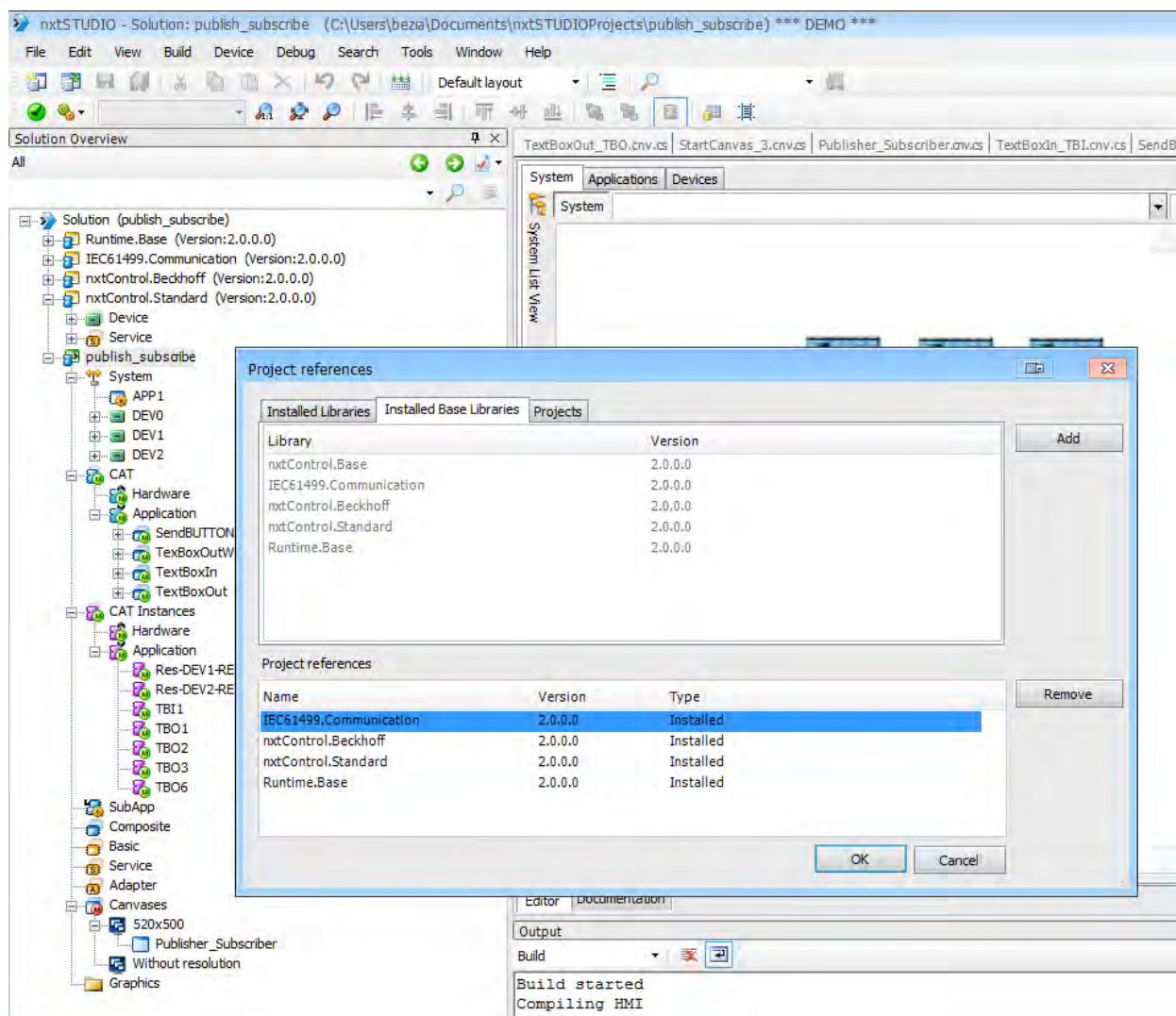


Рисунок 1 – Подключение библиотеки элементов к проекту

После подключения библиотеки в нашем проекте появится вкладка со всеми блоками данной библиотеки. Блоки библиотеки представлены в виде сервисных интерфейсных блоков, т.е. их нельзя редактировать и от нас скрыта реализация их внутренней логики. Пример блока представлен на рисунке 2.

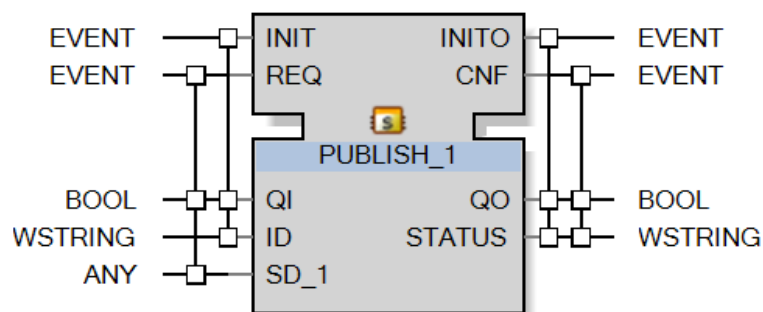


Рисунок 2 – СИФБ Publish_1

С назначением портов блоков можно ознакомиться во вкладке «Help» соответствующей библиотеки.

Для демонстрации работы коммуникационных блоков создадим приложение, включающее в себя 3 устройства. На первом устройстве будет располагаться блок-писатель и блок-читатель, на двух других по одному блоку-читателю.

Для ввода и вывода передаваемых данных создадим соответствующие САТ-блоки. Итоговое приложение примет вид, как на рисунке 3.

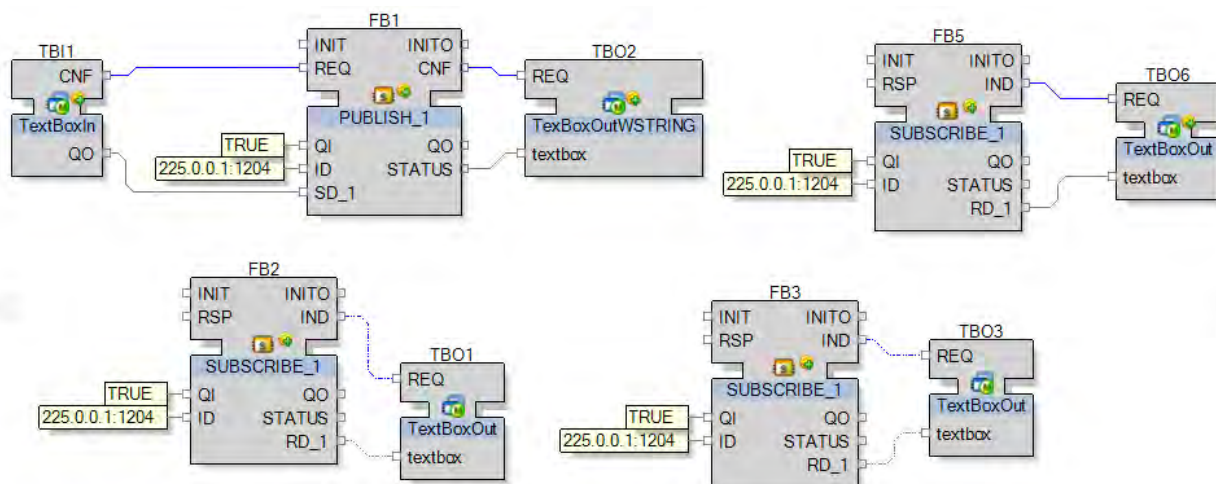


Рисунок 3 – Итоговое приложение

Блоки **FB2**, **FB3**, **FB5** – блоки-читатели.

Блок **FB1** – блок-писатель.

Блок **TBI1** – САТ-блок для ввода передаваемых данных.

Блоки **TBO1**, **TBO3**, **TBO2**, **TBO6** – САТ-блоки для отображения переданных данных.

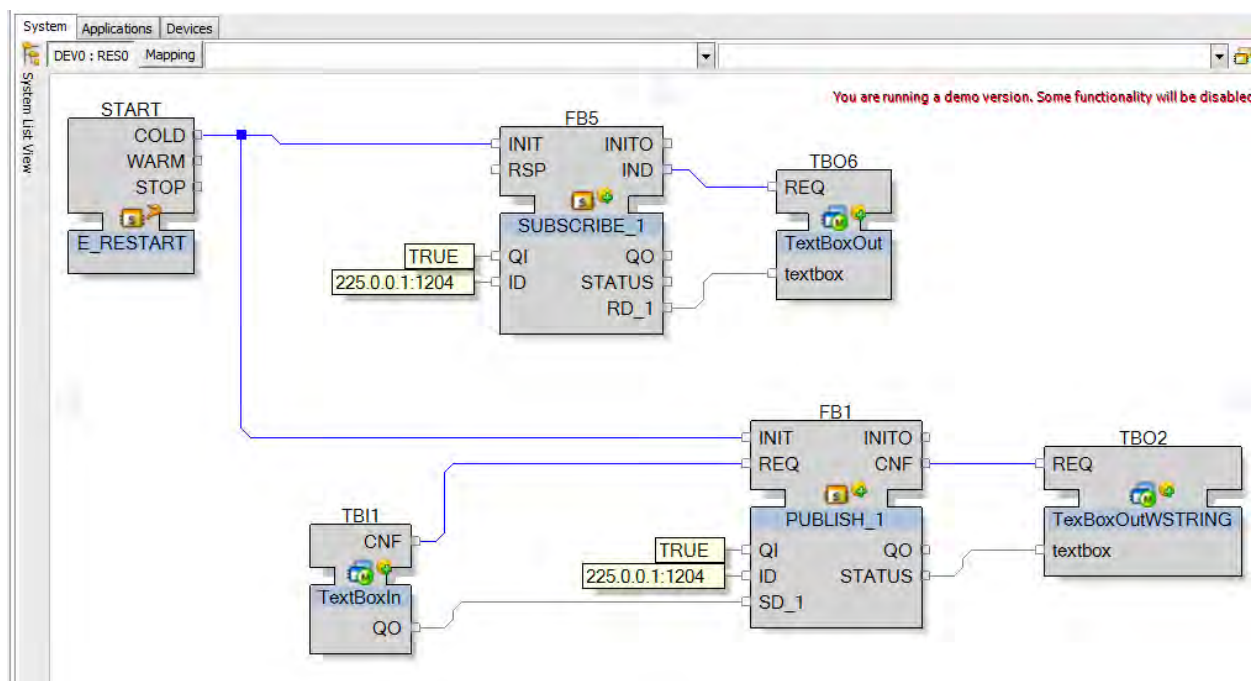


Рисунок 4 – Блоки, расположенные на ресурсе RES0 устройства DEV0

Из рисунка 3 видно, что блоки Publish и Subscribe никак не соединены друг с другом, т.к. связь между ними устанавливается непосредственно во время их инициализации при начале работы устройств DEV0, DEV1, DEV2. Для установки соединения друг с другом данные блоки должны иметь одно и тоже значение поля ID. При этом, если планируется широковещательная рассылка (более чем на один блок-читатель), то IP-адрес в поле ID должен быть в диапазоне адресов начиная с 224.0.0.0 до 239.255.255.255. Следует учитывать, что диапазон адресов с 224.0.0.0 до 224.0.0.255 зарезервирован для системных нужд.

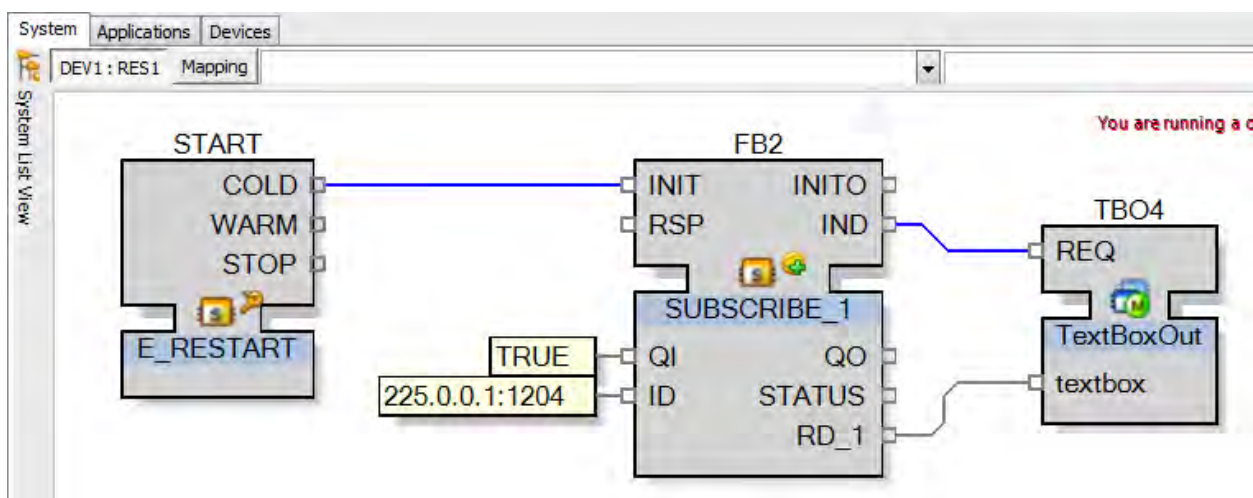


Рисунок 5 – Блоки, расположенные на ресурсе RES1 устройства DEV1

После создания приложения и расположения всех элементов на соответствующих устройствах необходимо реализовать человеко-машинный интерфейс для взаимодействия с программой.

Итоговый интерфейс выглядит в соответствии с рисунком 6.

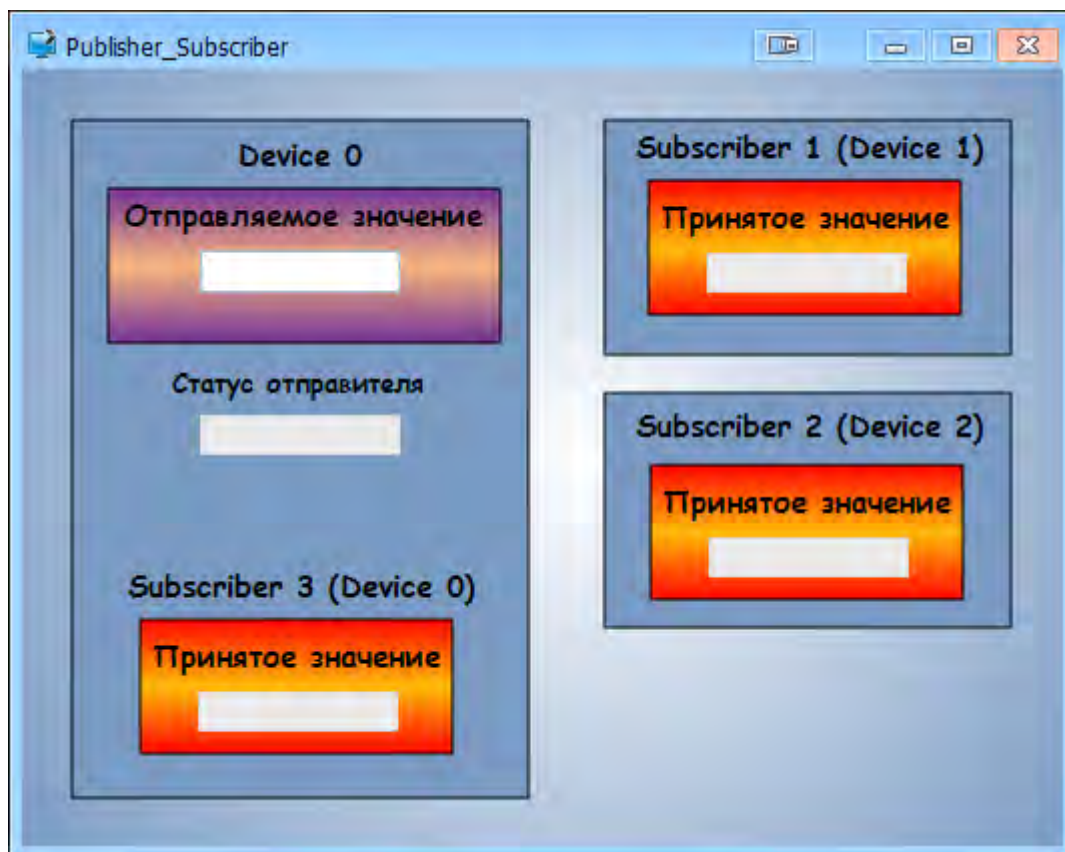


Рисунок 6 – Интерфейс программы

Результаты работы программы

Для проверки работоспособности созданного приложения был запущен эмулятор «Soft PLC» для каждого из устройств. При этом необходимо указать разные номера портов для каждого из устройств.

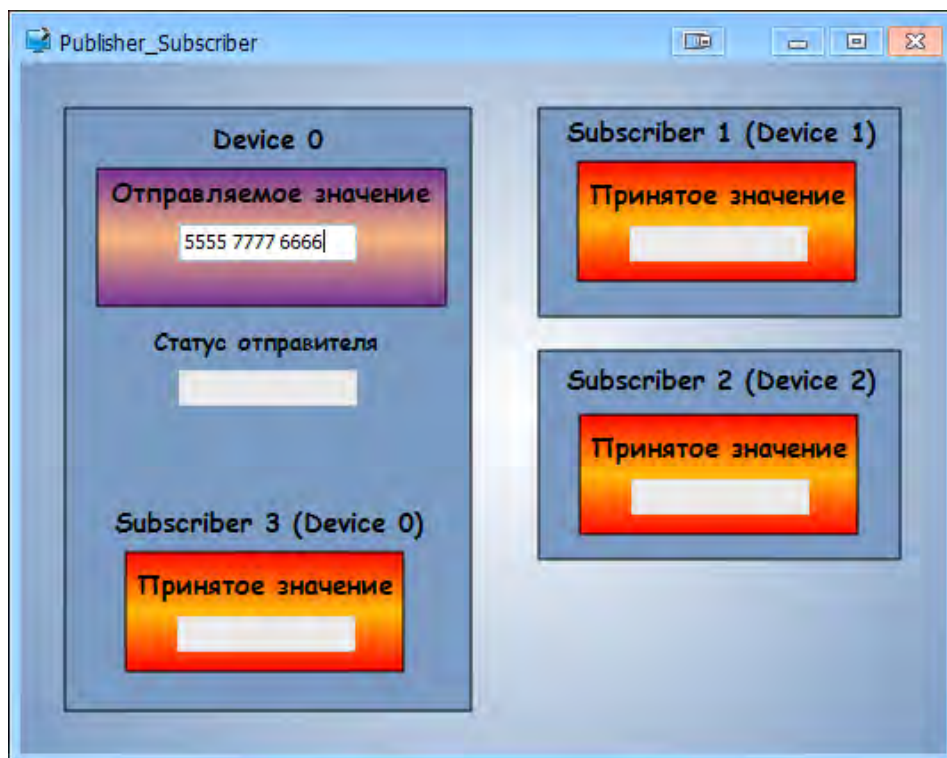


Рисунок 7 – Состояние интерфейса перед отправкой сообщения

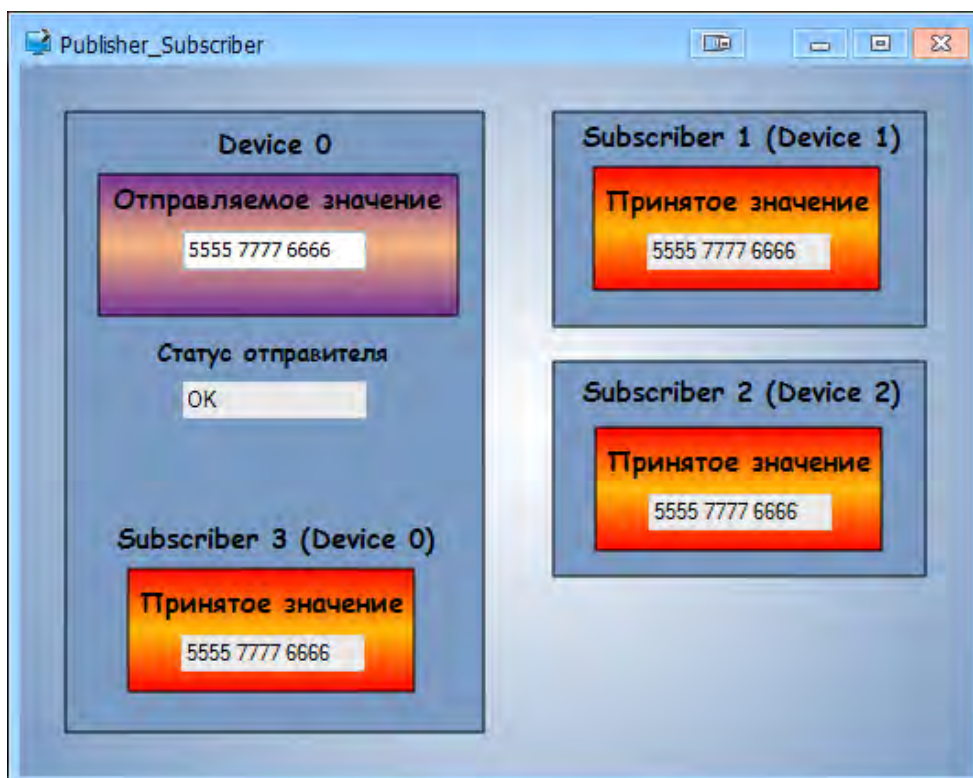


Рисунок 8 – Интерфейс после отправки сообщения

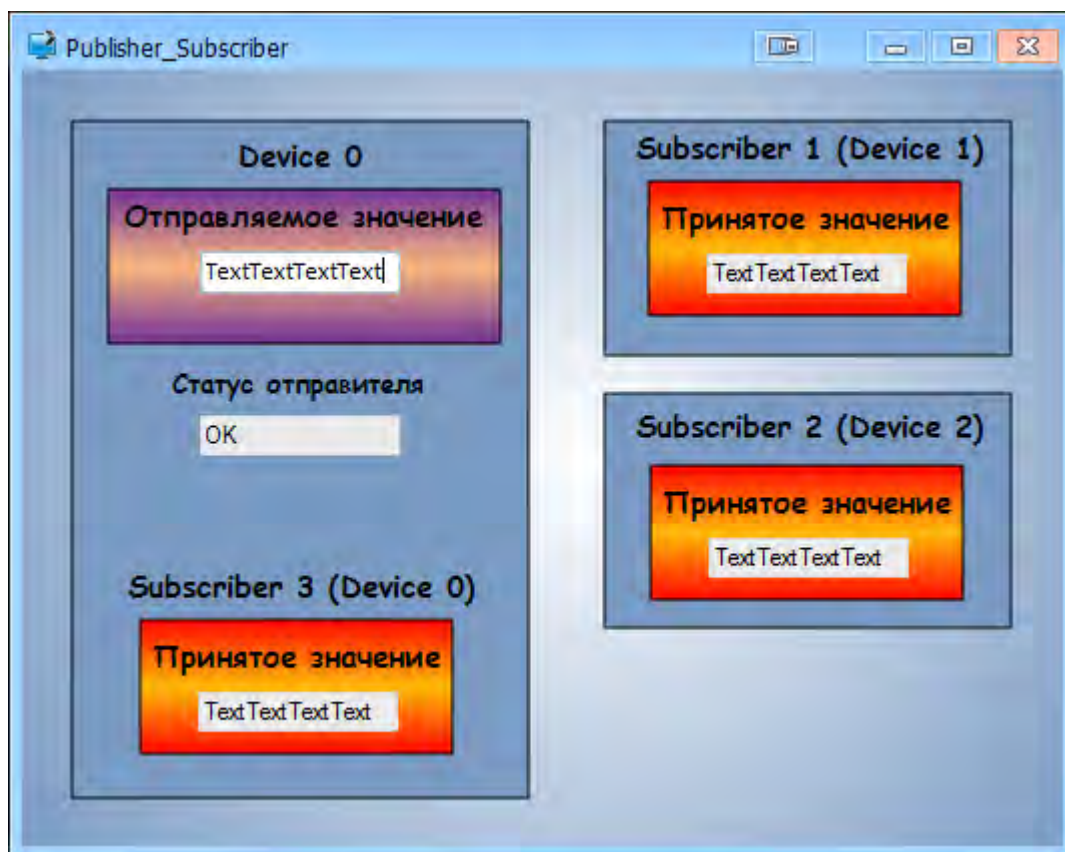


Рисунок 9 – Интерфейс после отправки следующего сообщения

В результате работы приложения можно сделать вывод, что передача данных между устройствами с использованием коммуникационных блоков реализована и сеть функциональных блоков работоспособна.

Вывод

В ходе выполнения данной лабораторной работы был изучен принцип подключения библиотек к проектам в среде NxtStudio и построена сеть функциональных блоков, выполняющая передачу данных между устройствами с использованием коммуникационных блоков библиотеки «IEC61499.Communication».

Лабораторная работа № 4

Создание интерфейса «человек-машина» для системы охлаждения дата-центра

Цель работы: 1) ознакомиться с принципами работы системы охлаждения дата-центров; 2) получить практические навыки разработки интерактивного человеко-машинного интерфейса (ЧМИ) на основе САТ-блоков в среде NxtStudio для управления, мониторинга и имитации работы локального и общего кулеров.

Примечание: данная лабораторная работа состоит из ряда последовательных заданий, выполнение которых приводит к созданию целевого проекта в среде NxtStudio.

Задание 1. Подготовка к лабораторной работе

1.1. Создайте новый проект (HMI and IEC Solution).

1.2. Назовите проект ServerRoom и сохраните его.

1.3. Распакуйте архив Lab4.zip¹.

1.4. Импортируйте САТ-блок (ServerRoomFB.20150215-1529.cat.zip²) в свой проект. Нажмите правой кнопкой мыши на САТ и выберите Import. Данный САТ-блок содержит шаблон интерфейса «человек-машина», в который не включены изображения кулеров и кнопок (рис. 4.1). В данной работе этот САТ-блок надо усовершенствовать для возможности его полноценного использования.

¹ Данный архив можно скачать по адресу:

https://drive.google.com/open?id=0B1_0eFPTb3VXdXR0NjNmcGxFSUE

² Этот файл можно найти в архиве Lab4.zip

1.5. Скопируйте папку SR-Images³, содержащую графические элементы интерфейса, в папку HMI вашего проекта. Следует отметить, что данная папка содержит в том числе как статичные, так и анимационные изображения вентиляторов. В дальнейшем каждое изображение должно быть прикреплено к соответствующему элементу в рабочей области HMI CAT-блоков.

1.6. Нажмите на изображении прямоугольника под надписью Local Fan в левом верхнем углу экрана. Имя этого компонента rect_LF, вы можете найти его в Панели свойств (Properties Pane).

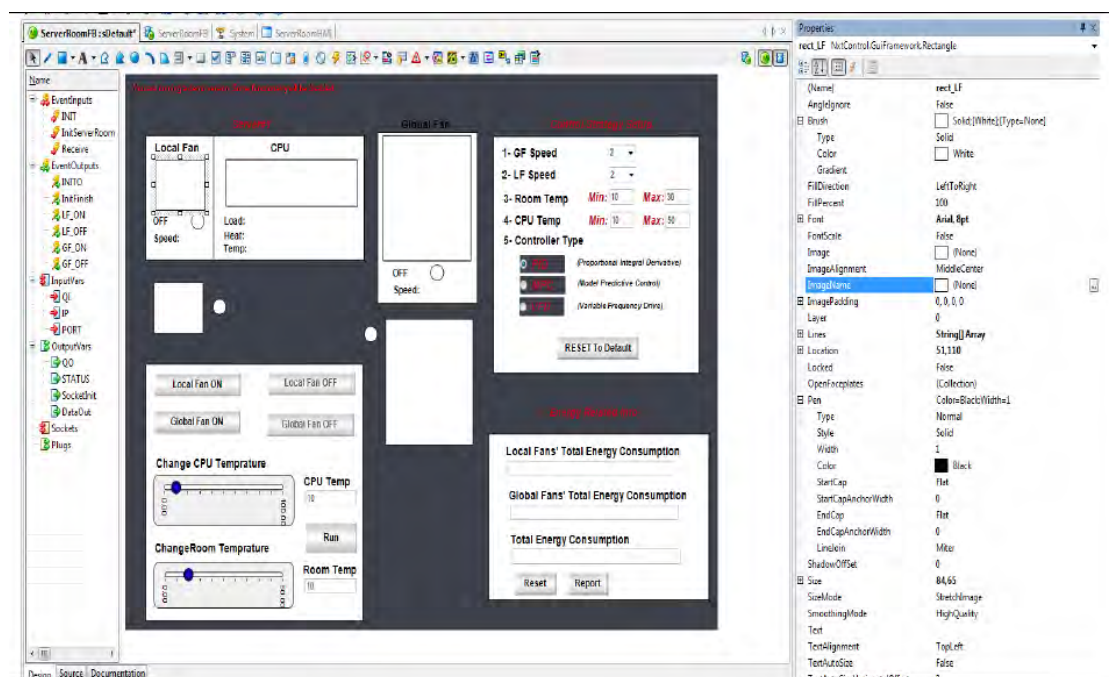


Рис. 4.1. Шаблон интерфейса «человек-машина», не содержащий изображения вентиляторов и кнопок

1.7. Для компонента rect_LF выберете свойство Image Name и щелкните мышью на поле ввода значений для изменения имени файла.

1.8. Откроется диалоговое окно редактора списка изображений (ImageList Editor). В списке Image Lists выберите HMI, затем кликните на кнопку Add («Добавить»), откроется диалоговое окно Catalogue Name. Введите в нем имя каталога, например, Images (рис. 4.2).

³ Эту папку можно найти в архиве Lab4.zip

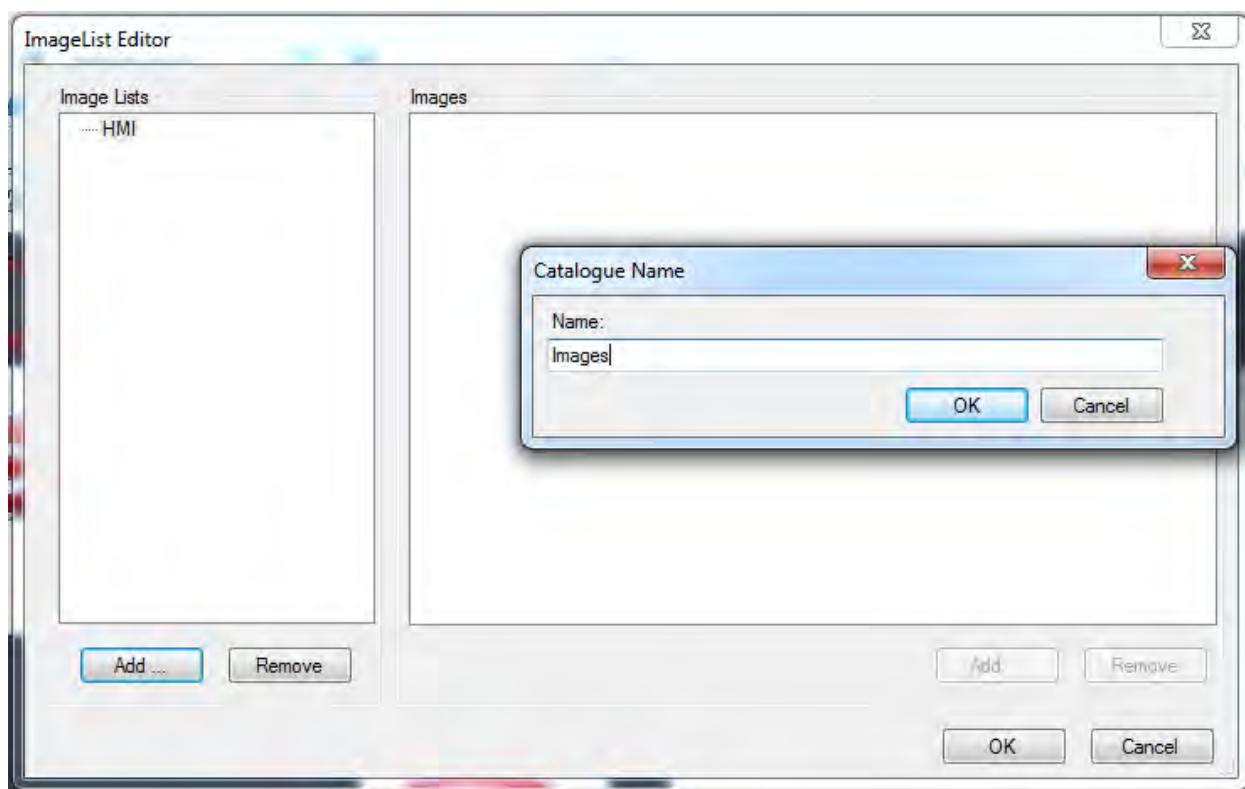


Рис. 4.2. Создание списка изображений в проекте

1.9. Под надписью HMI щелкните на только что созданном списке Images, затем нажмите на кнопку Add в правом нижнем углу. Откроется диалоговое окно для навигации по файловой системе, с помощью которого нужно найти ранее скопированную папку SR-Images. Выберите все файлы изображений из этой папки и нажмите Open (рис. 4.3). Добавленные изображения можно просмотреть в редакторе списка изображений (рис. 4.4).

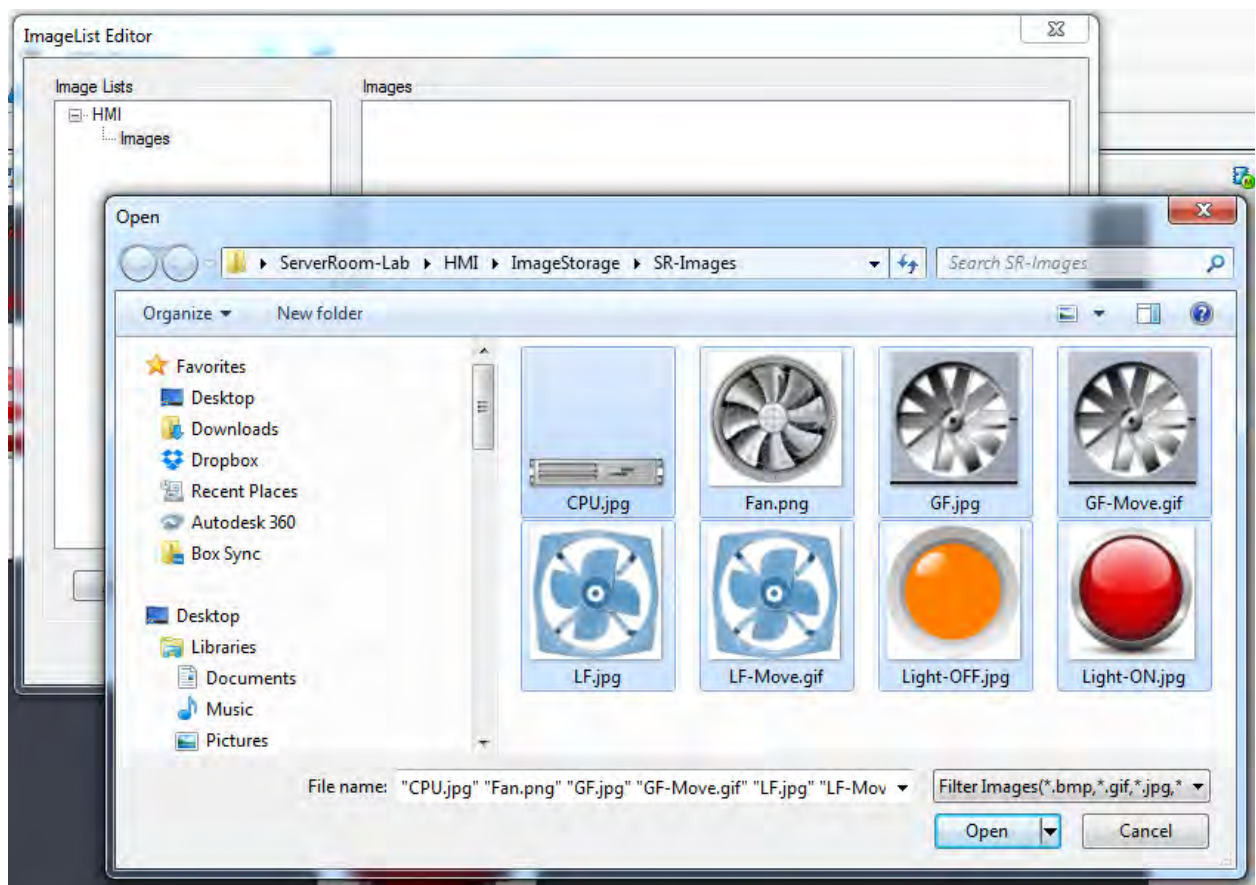


Рис. 4.3. Выбор изображений для добавления в созданный список

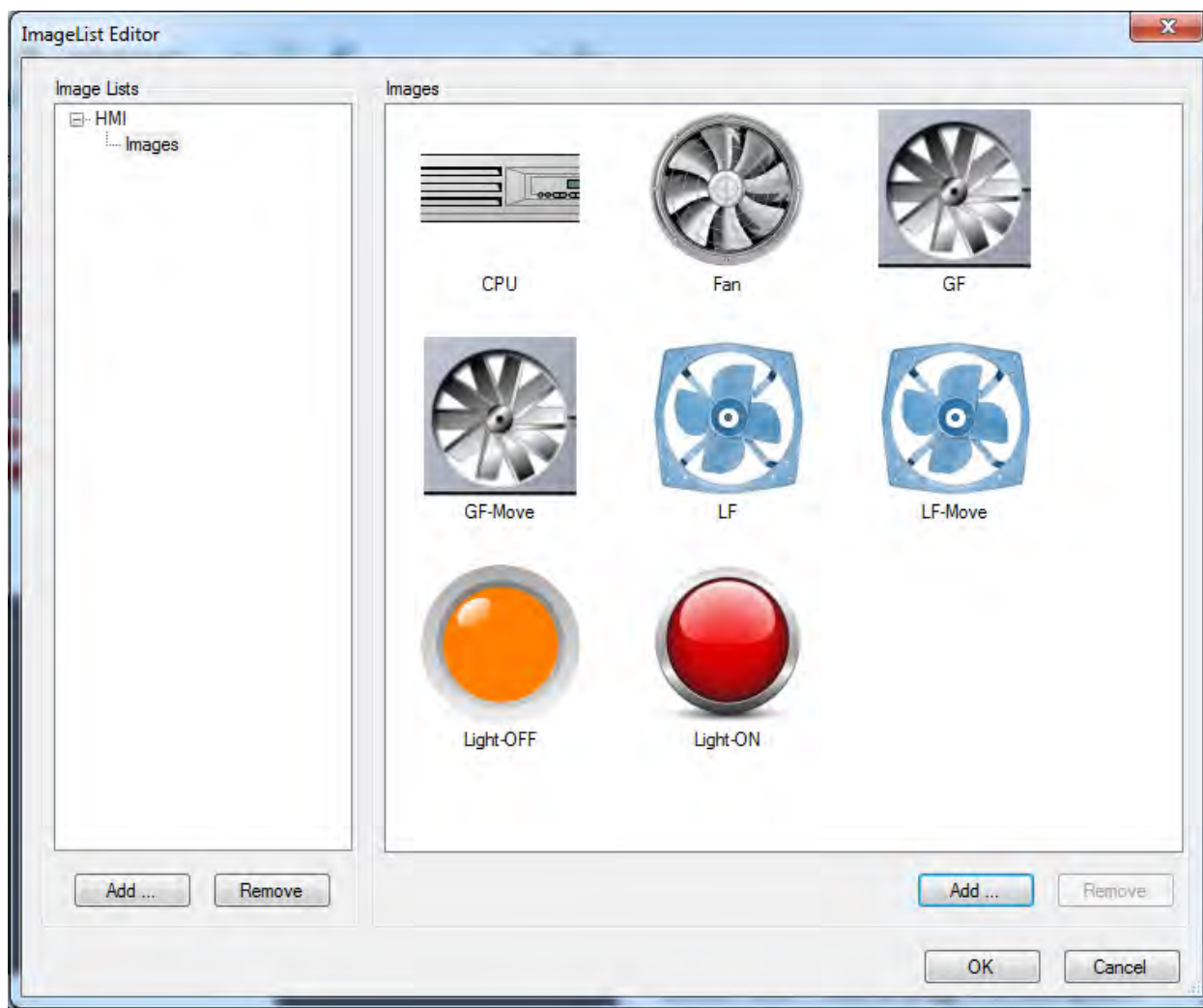


Рис. 4.4. Просмотр изображений в редакторе списка изображений

1.10. Выберите изображение с именем LF из списка изображений и нажмите на Ok (рис. 4.5). Это изображение представляет собой локальный кулер (вентилятор) в выключенном состоянии (состояние OFF).

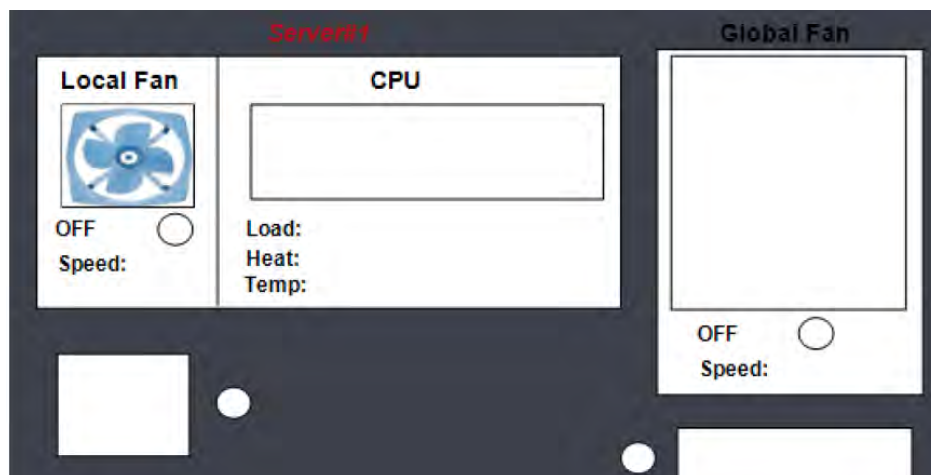


Рис. 4.5. Загрузка в пустой прямоугольник изображения кулера

1.11. Создайте новую рабочую область (Canvas) и назовите ее ServerRoomHMI. Для ее заголовка используйте то же самое имя.

1.12. Щелкните правой кнопкой мыши на пункте CAT Instances («Экземпляры CAT-блоков») и выберитеNewItem->ServerRoom->ServerRoomFB. Введите в появившемся окне имя вашего CAT-приложения -SRFBAPP.

1.13. Добавьте SRFBAPP в ранее созданную рабочую область ServerRoomHMI. Вы можете регулировать размер рабочей области (холста), чтобы она подходила вашему CAT-блоку.

1.14. Протестируйте ЧМИ. Для этого щелкните правой кнопкой мыши на пункте 1280x980 и выберите пункт Test HMI Runtime on Local Computer.

Задание 2: Подготовка графики

2.1. Теперь, когда среда установлена, добавьте следующие графические изображения для соответствующих компонентов ЧМИ, как это было описано в Задании 1 (рис. 4.6):

- a) свяжите изображение Light-OFF с компонентом LF_LightOFF;
- b) свяжите изображение LF-Move с компонентом rect_LFMove;
- c) свяжите изображение Light_ON с компонентом LF_LightON;
- d) свяжите изображение CPU с компонентом rect_CPU;
- e) свяжите изображение GF с компонентом rect_GF;
- f) свяжите изображение GF-Move с компонентом rect_GFMove;
- g) свяжите изображение Light-OFF с компонентом GF_LightOFF;
- h) свяжите изображение Light-ON с компонентом GF_LightON.

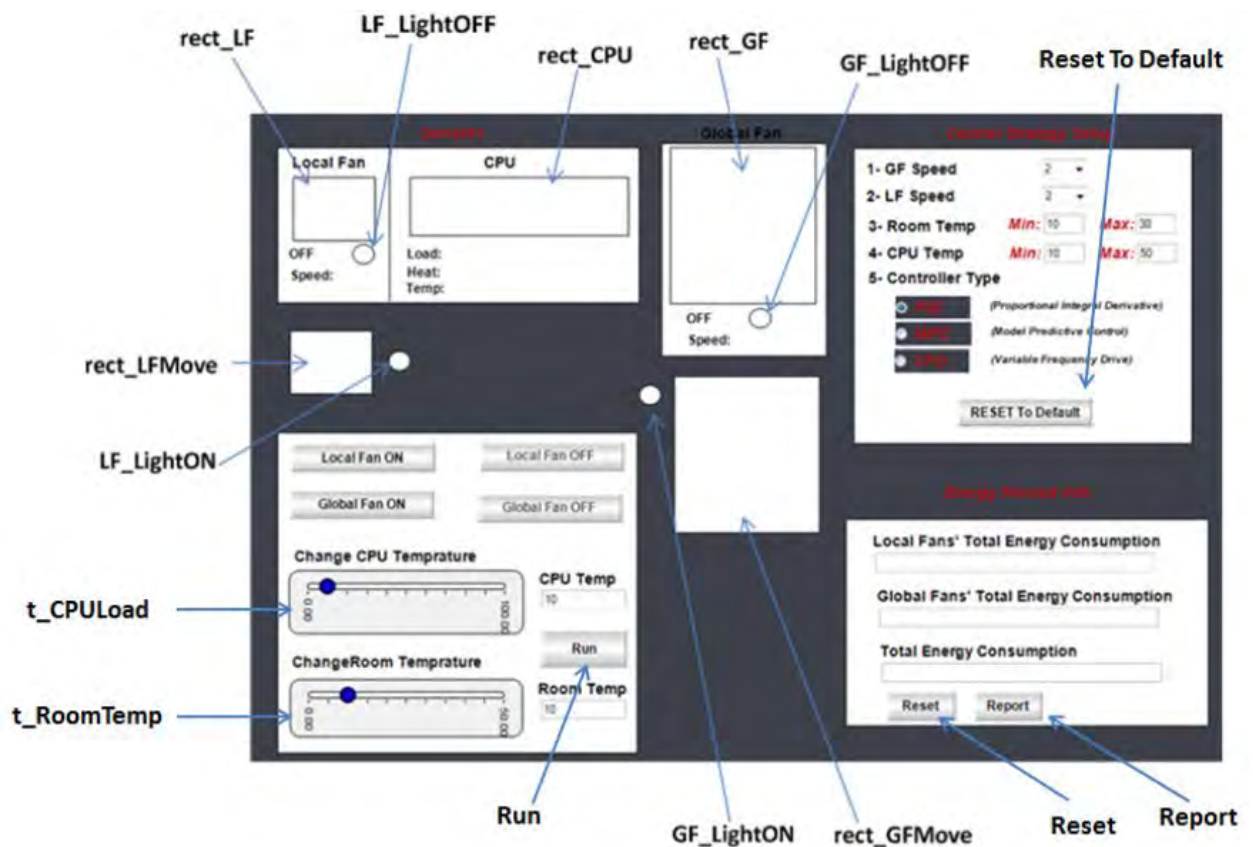


Рис. 4.6. Соответствие геометрических областей и компонентов ЧМИ

2.2. Снова проверьте ваш ЧМИ. Для этого щелкните правой кнопкой мыши на пункте 1280x980 и выберите Test HMI Runtime on Local Computer. Вы должны увидеть все изображения в вашем ЧМИ.

Задание 3: Разработка интерактивного САТ-блока для управления и мониторинга локального и общего кулера

Совет: щелкните на ServerRoomFB, затем найдите sDefault. Щелкните на нем два раза и выберите Source (Источник), чтобы найти код программы.

3.1. Напишите программу для кнопок Global Fan ON и Global Fan OFF так, чтобы они могли включать и выключать общий кулер. Вы можете использовать код, написанный для кнопок включения и выключения локального кулера Local Fan ON и Local Fan OFF. Примечание: измените следующие функции в исходном коде (Source Code) вашего САТ-блока.

Следует заметить, что для описания логики работы САТ-блоков используется язык программирования C#.

```
void GF_ON_BClick(object sender, EventArgs e)
{
// YOU HAVE TO COMPLETE THIS METHOD FOR TURNING ON THE GLOBAL FAN
}
void GF_OFF_BClick(object sender, EventArgs e)
{
// YOU HAVE TO COMPLETE THIS METHOD FOR TURNING OFF THE GLOBAL FAN
}
```

3.2. Можно заметить, что два объекта (типа Tracker), представляющие следящие системы - трекеры, уже включены в ЧМИ: t_CPULoad (под именем Change CPU Temperature) – для отслеживания температуры центрального процессора и t_RoomTemp (под именем Change Room Temperature) – для отслеживания температуры воздуха в помещении.

Код функции t_CPULoad() уже задан. Используя этот метод, измените метод **t_RoomTemp()**.

```
void T_RoomTempValueChanged(object sender, EventArgs e)
{
// Complete This Method Wherever Rquired
// Use These Variables:
/*
1- GF_State
2- rect_GF.Visible
3- rect_GFMove.Visible
4- GF_ON_OFF.Text
5- GF_LightON.Visible
6- GF_LightOFF.Visible
*/
RoomTemp_Text.Text = t_RoomTemp.Value.ToString();
if (t_RoomTemp.Value > double.Parse(RTMax.Text) &&
GF_State.Equals(0)){
this.FireEvent_GF_ON("GF_ON");
// Complete this if block using given variables and follow the
t_CPULoad () for Help
//...
Thread threadRoom = new Thread(new ThreadStart(moveRoomTemp));
threadRoom.Start();
}
if (t_RoomTemp.Value == double.Parse(RTMin.Text) &&
GF_State.Equals(1)){
this.FireEvent_GF_OFF("GF_OFF");
// Complete this if block using given variables and follow the
t_CPULoad () for Help
```

```
//...
GF_State = 0;
rect_GF.Visible = true;
rect_GFMove.Visible = false;
GF_ON_OFF.Text = "OFF";
GF_LightON.Visible = false;
GF_LightOFF.Visible = true;
}
}
```

3.3. Теперь, когда вы убедились, что ЧМИ работает правильно, необходимо выполнить следующие шаги, чтобы этот интерфейс стал выглядеть более привлекательно:

a) переместите компонент rect_LFMove непосредственно на компонент rect_LF, чтобы они действовали бы как один объект;

b) переместите компонент LF_LightON непосредственно на компонент LF_LightOFF, чтобы они действовали бы как один объект;

c) переместите компонент rect_GFMove непосредственно на компонент rect_GF так, чтобы они действовали бы как один объект;

d) переместите компонент GF_LightOn непосредственно на компонент GF_LightOFF, чтобы они действовали бы как один объект.

3.4. После внесенных изменений, скомпилируйте заново рабочую область (можно использовать клавишу F8) и запустите ЧМИ. Это будет финальная версия вашего ЧМИ.

3.5. Запустите разработанный САТ-блок в рамках некоторого приложения на программном контроллере Soft PLC.

Задание 4. Расчет и вывод отчета о потреблении электроэнергии

4.1. Создайте две глобальные переменные и назовите их LF_EC_Total и GF_EC_Total.

4.2. Обновите программу так, что всякий раз, когда вы запускаете локальный кулер с использованием только одного трекера, значение константы C1 прибавлялось бы к значению переменной LF-EC-Total. Вы можете выбрать любое значение константы C1, которое имеет смысл.

4.3. Обновите программу так, что всякий раз, когда вы запускаете общий кулер с использованием только одного трекера, значение константы C2 прибавлялось бы к значению переменной GF-EC-Total. Вы можете выбрать любое значение константы C2, которое имеет смысл.

4.4. Измените следующий метод, чтобы выводить потребления электроэнергии в соответствующие текстовые поля (textbox). Необходимо выводить LF-EC-Total, GF-EC-Total и полное потребление энергии двумя этими вентиляторами.

```
void ReportClick(object sender, EventArgs e)
{
// YOU HAVE TO COMPLETE THIS METHOD TO DISPLAY ENERGY
CONSUMPTIONS IN APPROPRIATE TEXT BOX
}
```

4.5. Разработайте на языке C# полный набор функций (методов), определяющий логику работы САТ-блока. Должны быть реализованы следующие функции (методы):

- ❖ изменение цвета соответствующего светодиода при включении/выключении кулера;
- ❖ изменение соответствующего изображения (с анимацией/без анимации) при включении/выключении кулера;
- ❖ изменение значения температуры сервера/комнаты при перемещении ползунков трекеров. Если температуры достигают указанных предельных значений, то включаются соответствующие кулеры;
- ❖ добавление значения энергопотребления соответствующего кулера к глобальным переменным GF_EC_Total и LF_EC_Total при каждом автоматическом включении кулера;
- ❖ вывод отчета о потреблении электроэнергии при нажатии на кнопку Report.

4.5. Протестируйте приложение, включающее разработанный САТ-блок. На рис. 4.7 показано состояние системы после нажатия на кнопку Run и Report. Так как были указаны значений температур, выходящие за установленные ограничения, были включены оба кулера. После нажатия кнопки Report в соответствующие текстовые поля были выведены значения

энергопотребления кулеров. Примечание: для облегчения разработки САТ-блока можно воспользоваться листингом программ, приведенным в Приложении 1.

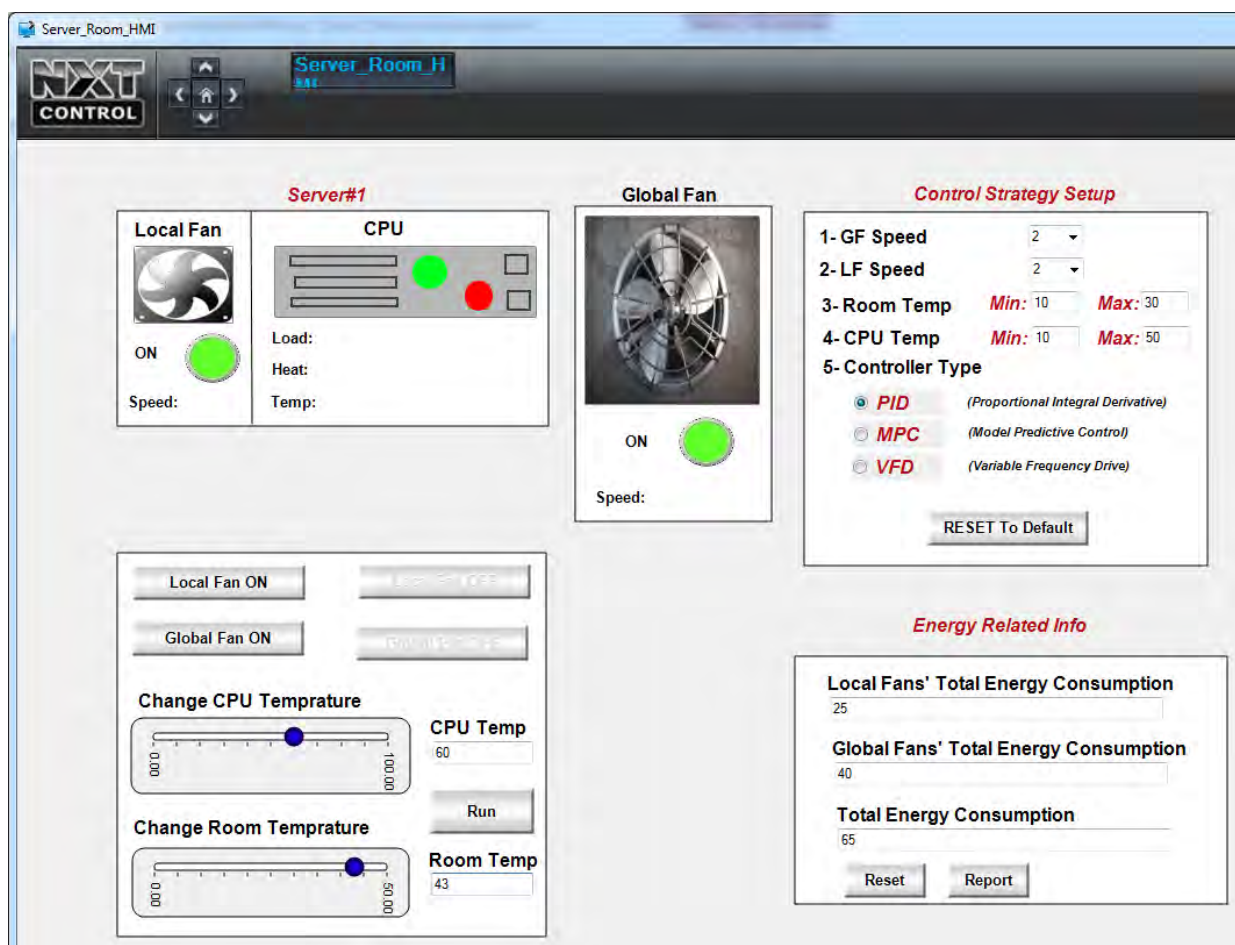


Рис. 4.7. Результаты работы программы

Задание 5. Разделение управляющего кода на два САТ-блока

5.1. Создайте два САТ-блока, первый из которых будет соответствовать локальному кулеру, а второй – общему. Управляющий код, который был разработан в ходе выполнения предыдущих пунктов, необходимо разделить на два этих блока.

5.2. Создайте приложение, содержащее эти два САТ-блока. Отобразите эти блоки на разные устройства. Соответственно, при запуске приложения используйте два программных контроллера Soft PLC.

Приложение 1. Листинг итогового САТ-блока для работы № 3

```
using System;
using System.Drawing;
using NxtControl.GuiFramework;
using System.Text;
using System.Windows.Forms;
using System.Threading;
using System.Net;
using System.Net.Sockets;
using System.Diagnostics;
namespace HMI.Main.Symbols.ServerRoomFB
{
    /// <summary>
    /// Description of sDefault.
    /// </summary>
    public partial class sDefault :
NxtControl.GuiFramework.HMISymbol
    {
        int LF_State = 0;
        int GF_State = 0;
        double GF_EC_Total = 0;
        double LF_EC_Total = 0;
        double C1 = 5;
        double C2 = 10;

        public sDefault()
        {
            //
            // The InitializeComponent() call is required for
Windows Forms designer support.
            //
            InitializeComponent();
            this.InitServerRoom_Fired +=new EventHandler
<ServerRoomFB.InitServerRoomEventArgs>(InitServerRoom);
            this.Receive_Fired += new EventHandler
<ServerRoomFB.ReceiveEventArgs> (Receive);
        }

        public void InitServerRoom(object Sender,
ServerRoomFB.InitServerRoomEventArgs sr){

        }

        public void Receive(object Sender,
ServerRoomFB.ReceiveEventArgs r){

        }

        /*---ON and OFF Buttons for Local and Global Fans
        /* Function to Turn the Local Fan On*/
        void LF_ON_BClick(object sender, EventArgs e)
```

```

    {
        this.FireEvent_LF_ON("LF_ON");
        LF_State = 1;
        rect_LF.Visible = false;
        rect_LFMove.Visible = true;
        LF_ON_OFF.Text = "ON";
        LF_LightON.Visible = true;
        LF_LightOFF.Visible = false;
        LF_ON_B.Enabled = false;
        LF_OFF_B.Enabled = true;
    }
/* Function to Turn the Global Fan Off*/
void LF_OFF_BClick(object sender, EventArgs e)
{
    this.FireEvent_LF_OFF("LF_OFF");
    LF_State = 0;
    rect_LF.Visible = true;
    rect_LFMove.Visible = false;
    LF_ON_OFF.Text = "OFF";
    LF_LightON.Visible = false;
    LF_LightOFF.Visible = true;
    LF_ON_B.Enabled = true;
    LF_OFF_B.Enabled = false;

}

/* Function to Turn the Global Fan On*/
void GF_ON_BClick(object sender, EventArgs e)
{
    this.FireEvent_LF_ON("GF_ON");
    GF_State = 1;

    rect_GF.Visible = false;
    rect_GFMove.Visible = true;
    LF_ON_OFF.Text = "ON";
    GF_LightON.Visible = true;
    GF_LightOFF.Visible = false;

    GF_ON_B.Enabled = false;
    GF_OFF_B.Enabled = true;

}
/* Function to Turn the Global Fan OFF*/
void GF_OFF_BClick(object sender, EventArgs e)
{
    this.FireEvent_LF_OFF("GF_OFF");
    GF_State = 0;

    rect_GF.Visible = true;
    rect_GFMove.Visible = false;
    LF_ON_OFF.Text = "OFF";
    GF_LightON.Visible = false;

```

```

GF_LightOFF.Visible = true;

GF_ON_B.Enabled = true;
GF_OFF_B.Enabled = false;

}

/* THIS METHOD RESPONDS TO CHANGES ON TRACKER WHICH IS
?CONNECTED TO A CPU
WHEN THE TRACKER IS SLIDE TO THE RIGHT AND EXCEEDS THE MAXIMUM
LIMIT OF THE CPU TEMPERATURE
ITS CORRESPONDING LOCAL FAN WILL BE ON TO REDUCE ITS
TEMPERATURE
AND TRACKER WILL SLIDE TO LEFT UNTIL CPU TEMPERATURE REACHES
ITS MINIMUM LIMIT */
void T_CPULoadValueChanged(object sender, EventArgs e)
{
    CPULoad_Text.Text = t_CPULoad.Value.ToString();

    if (t_CPULoad.Value > double.Parse(CPUTMax.Text) &&
LF_State.Equals(0)){
        this.FireEvent_LF_ON("LF_ON");
        LF_State = 1;
        rect_LF.Visible = false;
        rect_LFMove.Visible = true;
        LF_ON_OFF.Text = "ON";
        LF_LightON.Visible = true;
        LF_LightOFF.Visible = false;
        Thread threadCPU = new Thread(new
ThreadStart(moveCPULoad));
        threadCPU.Start();
        LF_EC_Total=LF_EC_Total+C1;
    }
    if (t_CPULoad.Value == double.Parse(CPUTMin.Text) &&
LF_State.Equals(1)){
        this.FireEvent_LF_OFF("LF_OFF");
        LF_State = 0;
        rect_LF.Visible = true;
        rect_LFMove.Visible = false;
        LF_ON_OFF.Text = "OFF";
        LF_LightON.Visible = false;
        LF_LightOFF.Visible = true;
    }
}

/* THIS METHOD RESPONDS TO CHANGES ON TRACKER WHICH IS
?CONNECTED TO THE ROOM

```

WHEN THE TRACKER IS SLIDE TO THE RIGHT AND EXCEEDS THE MAXIMUM
LIMIT OF THE ROOM TEMPERATURE
CORRESPONDING TO THE GLOBAL FAN WILL BE ON TO REDUCE ITS
TEMPERATURE
AND TRACKER WILL SLIDE TO LEFT UNTIL ROOM TEMPERATURE REACHES
ITS MINIMUM LIMIT */

```

void T_RoomTempValueChanged(object sender, EventArgs e)
{
    RoomTemp_Text.Text = t_RoomTemp.Value.ToString();

    if (t_RoomTemp.Value > double.Parse(RTMax.Text) &&
GF_State.Equals(0)){
        this.FireEvent_GF_ON("GF_ON");
        // Complete this if block using given variables and
follow the t_CPULoad () for Help
        GF_State = 1;
        rect_GF.Visible = false;
        rect_GFMove.Visible = true;
        GF_ON_OFF.Text = "ON";
        GF_LightON.Visible = true;
        GF_LightOFF.Visible = false;
        Thread threadRoom = new Thread(new
ThreadStart(moveRoomTemp));
        threadRoom.Start();
        GF_EC_Total=GF_EC_Total+C2;
    }

    if (t_RoomTemp.Value == double.Parse(RTMin.Text) &&
GF_State.Equals(1)){
        this.FireEvent_GF_OFF("GF_OFF");
        GF_State = 0;
        rect_GF.Visible = true;
        rect_GFMove.Visible = false;
        GF_ON_OFF.Text = "OFF";
        GF_LightON.Visible = false;
        GF_LightOFF.Visible = true;
    }
}

// Function for Run Button
void RunClick(object sender, EventArgs e)
{
    t_RoomTemp.Value =
double.Parse(RoomTemp_Text.Text);
    t_CPULoad.Value = double.Parse (CPULoad_Text.Text);
}

public void moveCPUload ()
{
    try
    {
        string[] parts = CPUload_Text.Text.Split('.');
        int y = int.Parse(parts[0]);
    }
}

```

```

        t_CPULoad.Value = Convert.ToDouble(y);
        Thread.Sleep(500);
        for (int
i=y;i>=Convert.ToInt32(double.Parse(CPUTMin.Text));i--)
        {
            t_CPULoad.Value = Convert.ToDouble(i);
            Thread.Sleep(100);
        }
    }catch (Exception ex){
    }
}
public void moveRoomTemp ()
{
    try
    {
        string[] parts = RoomTemp_Text.Text.Split('.');
        int y = int.Parse(parts[0]);
        Thread.Sleep(500);

        for (int
i=y;i>=Convert.ToInt32(double.Parse(RTMin.Text));i--)
        {
            t_RoomTemp.Value = Convert.ToDouble(i);
            Thread.Sleep(300);
        }
    }catch (Exception ex){
    }
}
}
/*----- Reseting the Control Strategy Setup to its
Default Value -----*/
/*-----*/
-----*/
void ReSetToDefaultClick(object sender, EventArgs e)
{
    GFSpeed.Text = "2";
    LFSpeed.Text = "2";
    RTMin.Text = "10";
    RTMax.Text = "30";
    CPUTMin.Text = "10";
    CPUTMax.Text = "50";
}
/*----- Creating Energy Report and Reseting the Text Boxes
void ReportClick(object sender, EventArgs e)
{
    GF_Total_EC.Text = GF_EC_Total.ToString();
    LF_Total_EC.Text = LF_EC_Total.ToString();
    Total_EC_All.Text = (GF_EC_Total +
LF_EC_Total).ToString();
}
void ResetClick(object sender, EventArgs e)
{

```

```
t_RoomTemp.Value = double.Parse(RTMin.Text);
RoomTemp_Text.Text = RTMin.Text;
t_CPULoad.Value = double.Parse (CPUTMin.Text);
CPULoad_Text.Text = CPUTMin.Text;
//ReportLog_listBox.Items.Clear();
LF_Total_EC.Clear();
GF_Total_EC.Clear();
Total_EC_All.Clear();
}
}
}
```

Литература

1. Vyatkin V. IEC 61499 Function Blocks for embedded and distributed control systems design. Third Edition - ISA, 2014. – 261 p.
2. Дубинин В.Н., Вяткин В.В. Модели функциональных блоков IEC 61499, их проверка и трансформации в проектировании распределенных систем управления: монография. – Пенза: Изд-во ПГУ, 2012. – 348 с.