

Способы обмена данными расчетного модуля с внешним миром

Милое Дитя!

После того, как мы создали модель объекта управления, разработали алгоритм управления и завершили задачу комплексного моделирования получив ожидаемое качество управления перед нами стоит задача программирования управляющего контроллера. Транслировать в СИ-код алгоритм управления и далее собрать из него динамически разделяемую библиотеку (расчетный модуль) для среды выполнения алгоритмов NordWind нам помогут инструменты генерации СИ-кода ПО SimInTech. Но алгоритм управления не должен работать в изоляции на контроллере он должен получать результаты измерений датчиков, а после завершения расчетного цикла передавать управляющие воздействия на органы управления. Для этого необходим обмен данными с устройствами ввода вывода (УВВ) управляющего контроллера. Эта задача может быть реализована двумя методами:

1. Разработка блоков чтения и записи с УВВ на основании блока “Язык программирования”.
2. Использование динамической библиотеки обмена данными с УВВ.

Рассмотрим по подробнее каждый из этих методов.

Метод 1

Разработка блоков обмена данными с УВВ на основании библиотечного блока “Язык программирования”. В справочной системе ПО SimInTech существует информация о предназначение блока, при желании можно ознакомиться. Если кратко, то блок предназначен для реализации алгоритмов, формализованных с помощью встроенного языка программирования иногда это, бывает удобнее, чем создавать алгоритм из набора стандартных блоков.

В режиме моделирования блок, созданный на основе “Язык программирования” реализует созданный в нем алгоритм, а при генерации СИ-кода алгоритм, разработанный на встроенном языке программирования, транслируется в СИ-код в случае активизации свойства блока “**autotranslate**”. Существует возможность создать шаблон генерации произвольного СИ-кода в блоке “Язык программирования”, для этого необходимо добавить секцию “**initialization...end**” и в ней связать входные и выходные переменные с СИ-кодом, размещенным в текстовых полях **header**, **data**, **code** секции “**initialization...end**”. На рисунке 1 представлен библиотечный блок язык программирования. Блок находится в тулбоксе “Динамические”

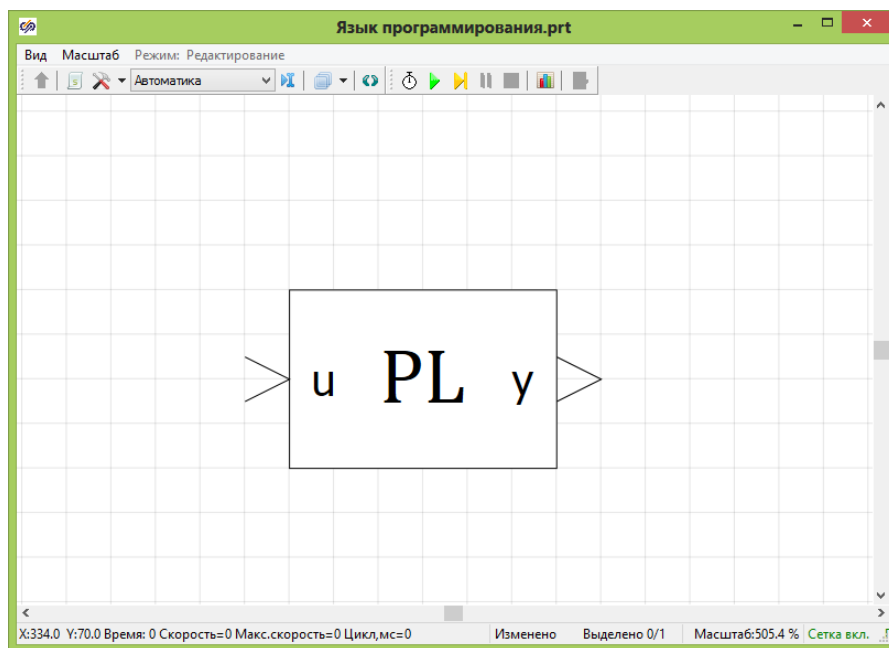


Рисунок 1

Войдем внутрь блока Рисунок 2

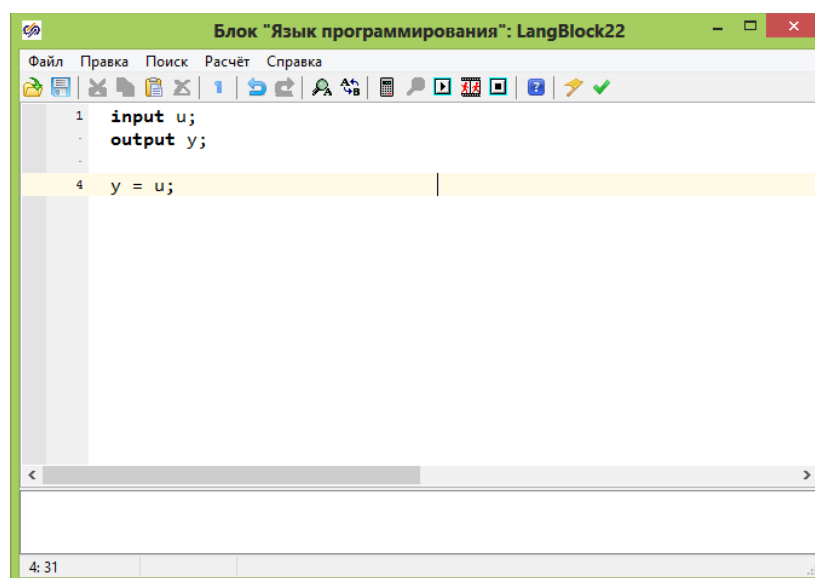


Рисунок 2

Внутри блока Рис.2 определены входные «**input u**» и выходные «**output y**» переменные, а код блока реализует простой алгоритм «**y = u**». В случае использования блока в составе модели в режиме моделирования входная переменная будет присвоена выходной. Если мы пожелаем получить СИ-код модели, то выражение «**y = u**» будет транслировано на язык СИ, для отмены трансляции алгоритма в СИ-код в “Свойствах объекта” необходимо деактивировать свойство “**autotranslate**” см. Рисунок 3.

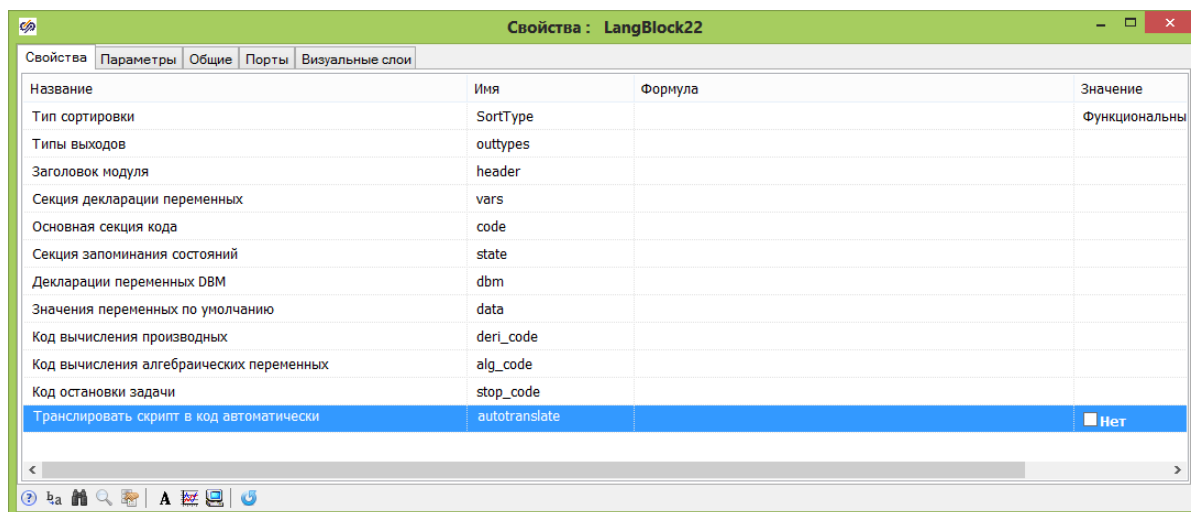
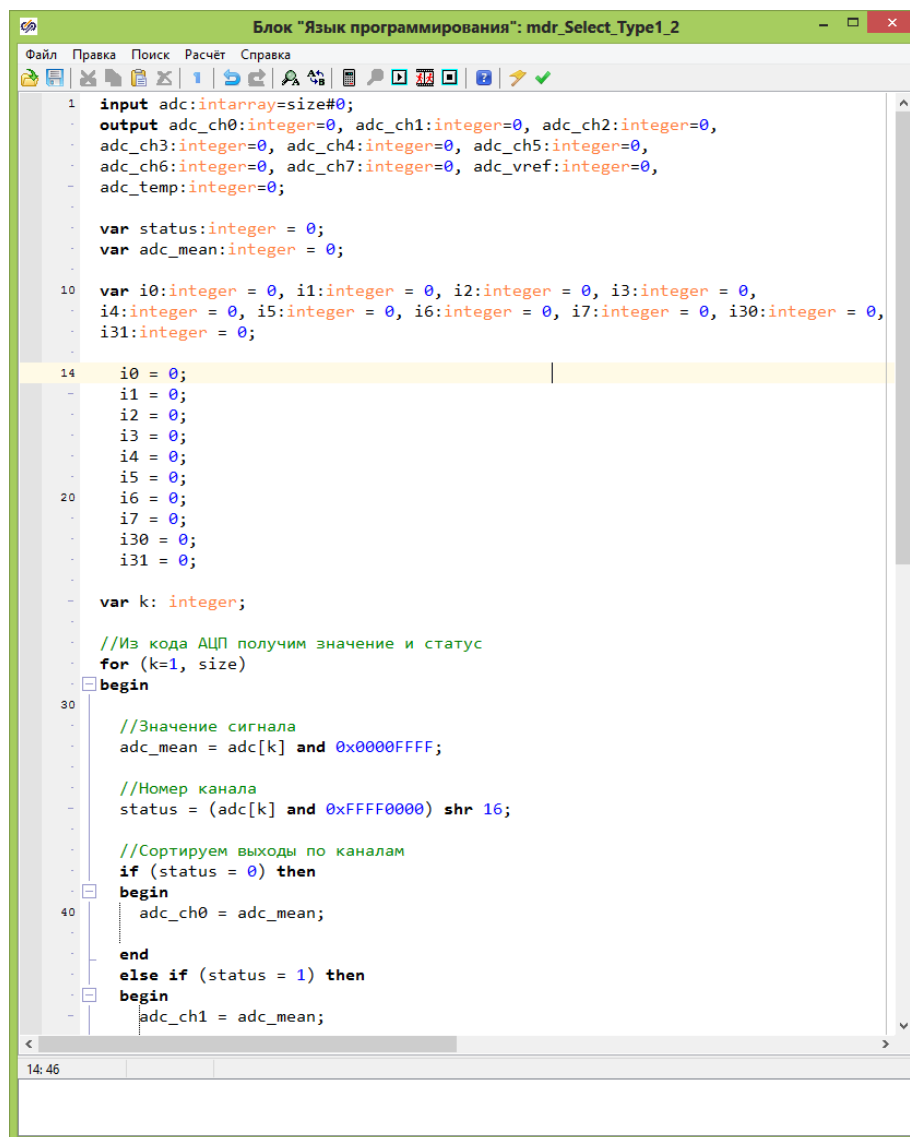


Рисунок 3

В блоке “Язык программирования” можно добавить произвольное количество входов и выходов и реализовать математический алгоритм любой сложности, в качестве примера можно привести библиотечный блок из тулбокса “**MILANDR 1986BE9X 1986BE1T**” “Распаковка массива значений АЦП на значения и номера каналов (Тип 1)” см. Рисунок 4



```
1 input adc:intarray=size#0;
- output adc_ch0:integer=0, adc_ch1:integer=0, adc_ch2:integer=0,
- adc_ch3:integer=0, adc_ch4:integer=0, adc_ch5:integer=0,
- adc_ch6:integer=0, adc_ch7:integer=0, adc_vref:integer=0,
- adc_temp:integer=0;

- var status:integer = 0;
- var adc_mean:integer = 0;

10 var i0:integer = 0, i1:integer = 0, i2:integer = 0, i3:integer = 0,
- i4:integer = 0, i5:integer = 0, i6:integer = 0, i7:integer = 0, i30:integer = 0,
- i31:integer = 0;

14 i0 = 0;
- i1 = 0;
- i2 = 0;
- i3 = 0;
- i4 = 0;
- i5 = 0;
20 i6 = 0;
- i7 = 0;
- i30 = 0;
- i31 = 0;

- var k: integer;

- //Из кода АЦП получим значение и статус
for (k=1, size)
begin
30 //Значение сигнала
- adc_mean = adc[k] and 0x000FFFFF;

- //Номер канала
- status = (adc[k] and 0xFFFF0000) shr 16;

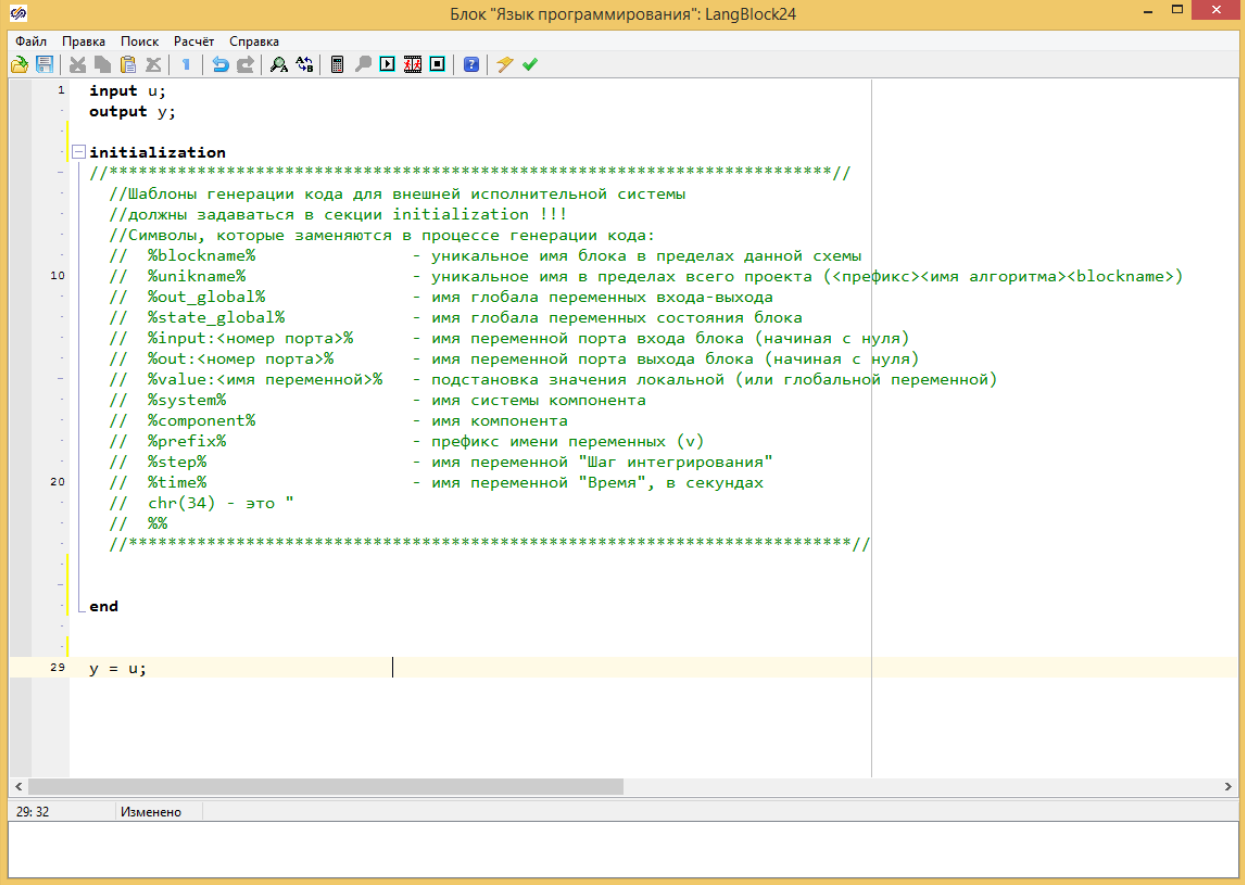
- //Сортируем выходы по каналам
if (status = 0) then
begin
40 - adc_ch0 = adc_mean;
- end
- else if (status = 1) then
begin
- adc_ch1 = adc_mean;
```

Рисунок 4

При генерации СИ-кода алгоритм см. Рис. 4 будет транслирован в Си-код.

Для обмена данными с УВВ удобно подготовить СИ-код в секции **“initialization...end”** и связать его с входными и выходными переменными, в этом случае при генерации СИ кода генератором Си-кода будет использоваться заранее подготовленный шаблон. Создадим простой блок обмена данными с УВВ на основе блока **“Язык программирования”**. Для этого необходимо:

1) Необходимо создать секцию “**initialization...end**” см. Рис. 5



The screenshot shows the LangBlock24 programming environment. The main window displays a code block with the following structure:

```
1  input u;  
   output y;  
  
  initialization  
  //*****  
  //Шаблоны генерации кода для внешней исполнительной системы  
  //должны задаваться в секции initialization !!!  
  //Символы, которые заменяются в процессе генерации кода:  
  // %blockname%           - уникальное имя блока в пределах данной схемы  
10 // %unique%              - уникальное имя в пределах всего проекта (<префикс><имя алгоритма><blockname>)  
  // %out_global%          - имя глобала переменных входа-выхода  
  // %state_global%        - имя глобала переменных состояния блока  
  // %input:<номер порта>%  - имя переменной порта входа блока (начиная с нуля)  
  // %out:<номер порта>%    - имя переменной порта выхода блока (начиная с нуля)  
  // %value:<имя переменной>% - подстановка значения локальной (или глобальной переменной)  
  // %system%              - имя системы компонента  
  // %component%           - имя компонента  
  // %prefix%              - префикс имени переменных (v)  
  // %step%                - имя переменной "Шаг интегрирования"  
20 // %time%                - имя переменной "Время", в секундах  
  // chr(34) - это "  
  // %  
  //*****  
  
end  
  
29 y = u;
```

The status bar at the bottom indicates "29: 32" and "Изменено".

Рисунок 5

2) В созданной секции можно использовать три текстовых поля **header**, **data**, **code**, **state** каждое поле предназначено:

- **header** в этом поле необходимо определять глобальные переменные и структуры, код помещенный в эту секцию после генерации окажется в файле **prog.h**;
- **data** в этом поле следует располагать Си-код который должен вызываться один раз, при запуске расчета удобно располагать в нем функции инициализации UBB, код помещенный в эту секцию после генерации окажется в файле **init.inc**;
- **code** в этом поле необходимо располагать основной СИ-код алгоритма, который будет вызываться на расчет при каждом шаге интегрирования, порядок вызова блока при расчете будет определено алгоритмом сортировки блоков при генерации СИ-кода, код помещенный в эту секцию после генерации окажется в файле **prog.inc**;

- **state** в этом текстовом поле необходимо располагать СИ-код целью которого является возвращение значений переменных в исходное состояние исполнение СИ-кода секции будет происходить после завершения расчета, код помещенный в эту секцию после генерации окажется в файле **state.inc**.

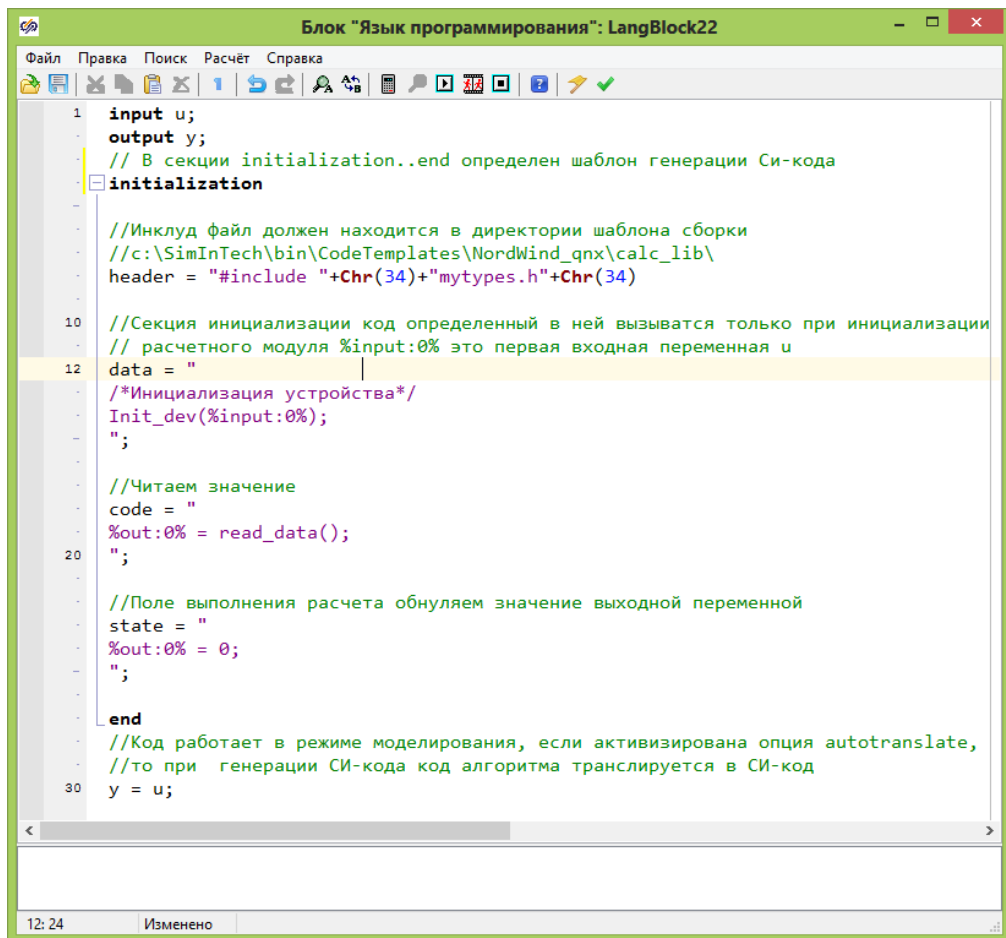
Текстовые поля начинаются и заканчиваются кавычками “...” внутри кавычек можно расположить наш Си-код, реализующий обмен с УВВ. Закомментированный текст см. Рис.5 стоит скопировать из готового библиотечного блока, он может быть для нас полезен тем что, содержит полезную справочную информацию:

- способ кодировки входных и выходных переменных блока для использования их в СИ-коде;
- наименование префиксов, которые можно использовать при определении переменных в СИ-коде, (например, использование префикса “%unikname%” при формировании переменных гарантирует то что после генерации СИ-кода имя переменной будет уникально и удастся избежать конфликта имен).

Примечание! Перечисленные файлы (**prog.h**, **init.inc**, **prog.inc**, **state.inc**) располагаются в директории **c:\SimInTech\bin\CodeTemplates\NordWind_qnx\calc_lib** шаблона сборки Си-кода для исполняемой среды **NordWind** они создаются генератором СИ-кода каждый раз при запуске пользователем процесса генерации СИ-кода для выбранного шаблона **NordWind_qnx** в настройках инструмента генерации СИ-кода. После завершения генерации СИ-кода и сборки расчетного модуля (динамической библиотеки **so**) содержимое директории копируется в директорию исходных кодов определенную пользователем при настройке инструмента генерации Си-кода.

Пример № 1

Самый простой вариант реализации блока обмена данными с УВВ Рис 6.



```
1  input u;
   output y;
   // В секции initialization..end определен шаблон генерации Си-кода
   initialization
   //Инкалд файл должен находится в директории шаблона сборки
   //c:\SimInTech\bin\CodeTemplates\NordWind_qnx\calc_lib\
   header = "#include "+Chr(34)+"mytypes.h"+Chr(34)

10  //Секция инициализации код определенный в ней вызывается только при инициализации
   // расчетного модуля %input:0% это первая входная переменная u
12  data = "
   /*Инициализация устройства*/
   Init_dev(%input:0%);
   ";

   //Читаем значение
   code = "
   %out:0% = read_data();
20  ";

   //Поле выполнения расчета обнуляем значение выходной переменной
   state = "
   %out:0% = 0;
   ";

   end
   //Код работает в режиме моделирования, если активизирована опция autotranslate,
   //то при генерации Си-кода код алгоритма транслируется в Си-код
30  y = u;
```

Рисунок 6.

В текстовом поле **“header”** определен файл заголовок **“mytypes.h”** содержащий реализацию функций обмена данными с УВВ, для кодировки **“(кавычек)”** обозначающих начало и конец имени файла заголовка используем служебный символ **“+Chr(34)+”**. Необходимо расположить файл заголовок в шаблон сборки Си кода в директорию **c:\SimInTech\bin\CodeTemplates\NordWind_qnx\calc_lib**.

В текстовом поле **“data”** определена функция инициализации (**“Init_dev()”**), функция будет вызвана один раз перед запуском цикла расчета, в качестве формального параметра функции передается значение входной переменной **“u”** в текстовом поле **“data”** она кодируется **“%input:0%”** см. Рис. 6 строка 12. Си-код из текстового поля **“code”** будет вызываться на исполнение при каждом такте выполнении расчетного цикла в порядке выполнения блока обмена данными с УВВ. Результат работы функции присвоены выходной переменной **“y”** в текстовом поле **“code”** выходная переменная **“y”** кодируется **“%out:0%”**.

В текстовом поле **“state”** происходит обнуление значения выходной переменной, вызов Си-кода из этой секции будет произведен после завершения расчетного цикла.

Пользователь самостоятельно может добавить в шаблон сборки любую библиотеку, для этого необходимо в директории шаблона сборки `c:\SimInTech\bin\CodeTemplates\NordWind_qnx\` отредактировать файл **“build.bat”** добавив ее в строку сборки расчетного модуля.

Пример № 2

Использование свойств блока для настройки его работы.

Рассмотрим блок инициализации обмена данными по протоколу **“Modbus RTU”** см. Рис.7

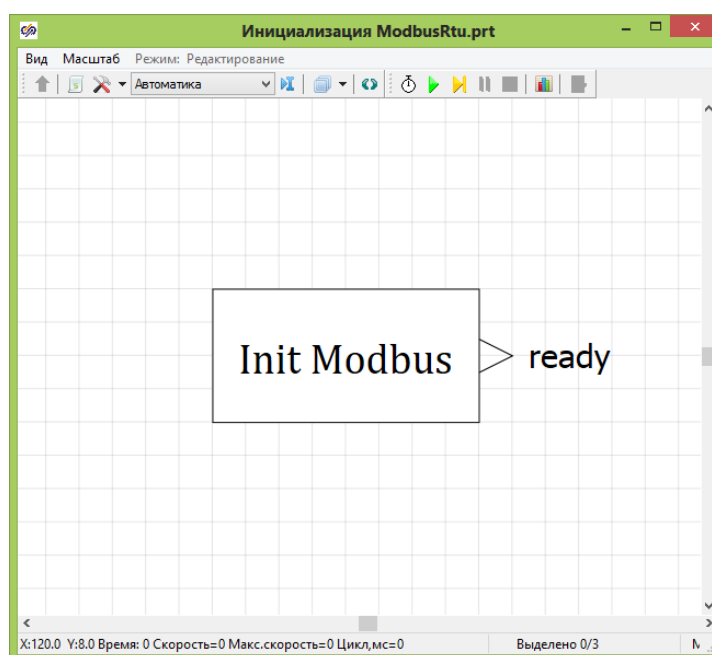


Рисунок 7

Откроем окно свойств блока см. Рис.7 и посмотрим, как можно в коде шаблона использовать свойства блока для настройки его работы. Имена свойств блока можно использовать в алгоритме блока, реализованном на встроенном языке программирования в качестве переменных и для генерации СИ-кода, причем нет необходимости описывать их тип внутри блока, он выбирается при создании свойств. Для активизации окна свойств необходимо навести курсор мыши на блок и правой кнопкой мыши из выпадающего меню выбрать опцию **“Свойства объекта”** см. Рис. 8

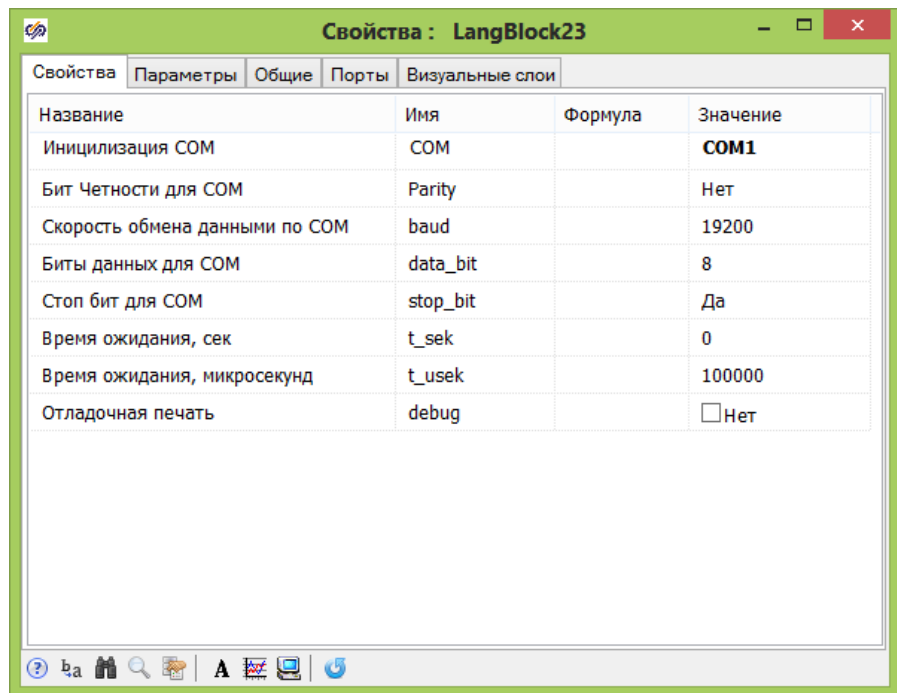


Рисунок 8

На Рисунке 8 мы видим окно со списком свойств блока, которые настраивают работу протокола блока **“Modbus RTU”**:

1. Выбор устройства последовательного интерфейса **“COM”**.
2. Бит четности для оценки целостности принятых данных по последовательному интерфейсу **“Parity”**.
3. Скорость обмена данными по последовательному интерфейсу **“baud”**.
4. Ширина поля данных передаваемых по последовательному интерфейсу **“data_bit”**.
5. Использование стоп бита при передаче данных **“stop_bit”**.
6. Время ожидания данных сек **“t_sek”** и микросекундах **“t_usek”**.
7. Вывод отладочной печати при приеме передачи данных **“debug”**.

Рассмотрим способы создания списка свойств для блока **“Modbus RTU”**, предварительно удалив перечисленные выше свойства, мы создадим их заново.

Для этого необходимо:

- 1) Правой выделить блок мышкой и нажать на правую кнопку, в выпадающем окне активизировать опцию **“Изменить блок...”** см. Рис. 9.

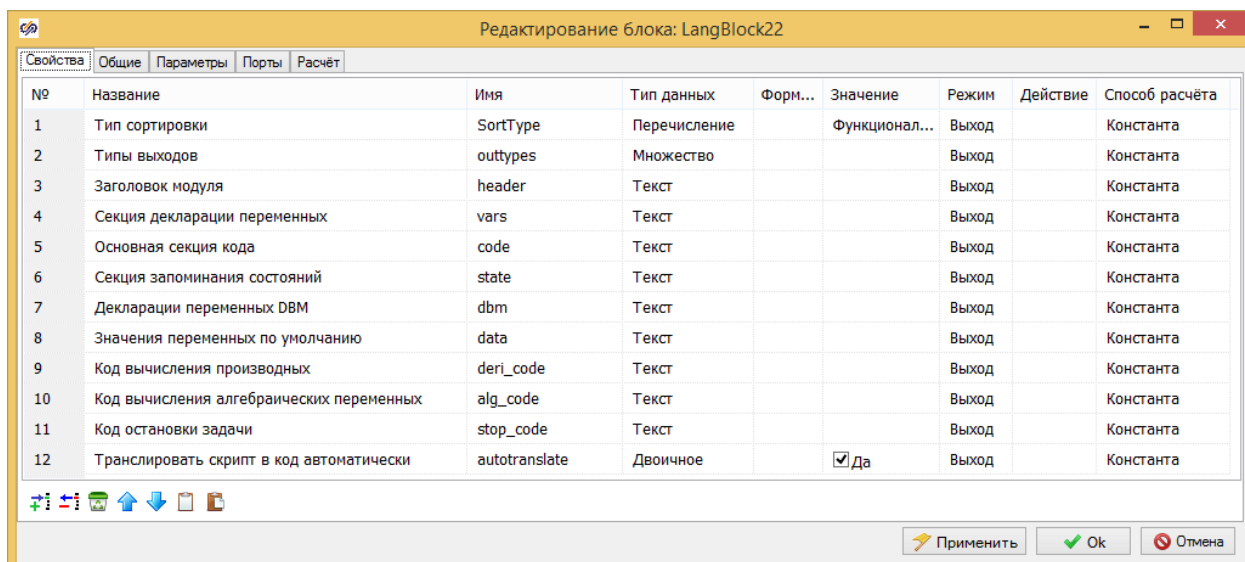


Рисунок 9

На Рис. 9 в окне свойств перечислен список из 12 свойств (они изначально существуют и необходимы для нормальной работы блока, удалять их нельзя!) к этому списку мы можем добавить свойства необходимые для настройки алгоритма работы блока и генерации Си-кода. Будет удобно для настройки свойств блока пользователем сделать 12 свойств блока невидимыми, а видимыми оставить только созданные свойства см. Рис. 8. Для этого необходимо выделить эти свойства см. Рис 10.

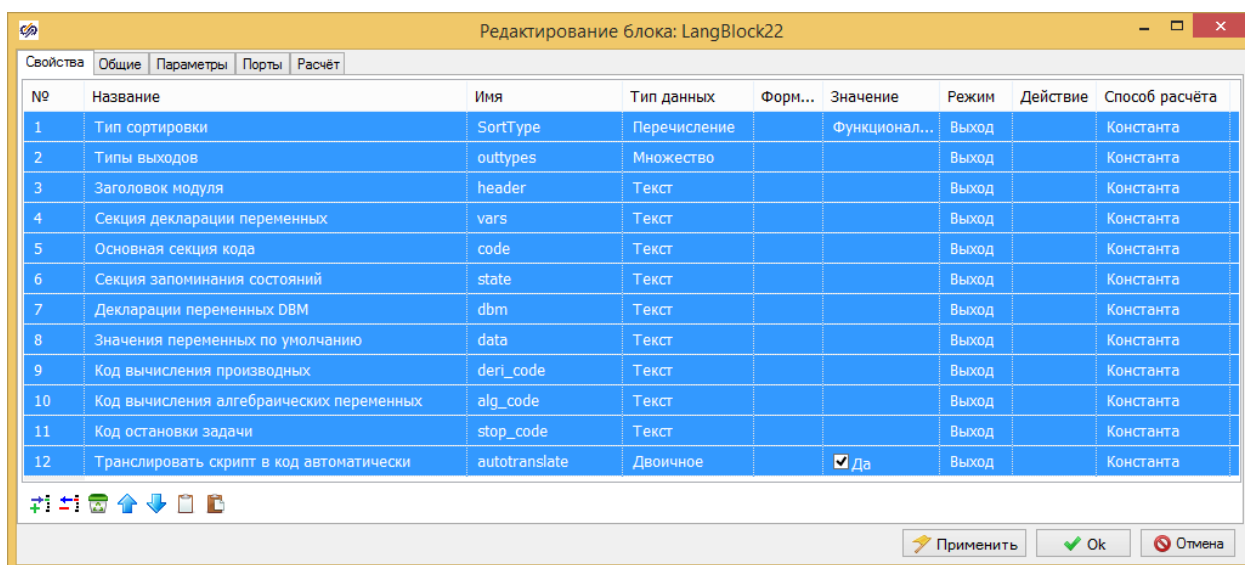


Рисунок 10

И используя правую кнопку мыши выбрать в выпадающем меню опцию “Скопировать имя” скопировать см. Рис 10 список свойств.

2) В окне свойства блока переходим на вкладку “Общие” см. Рис. 11

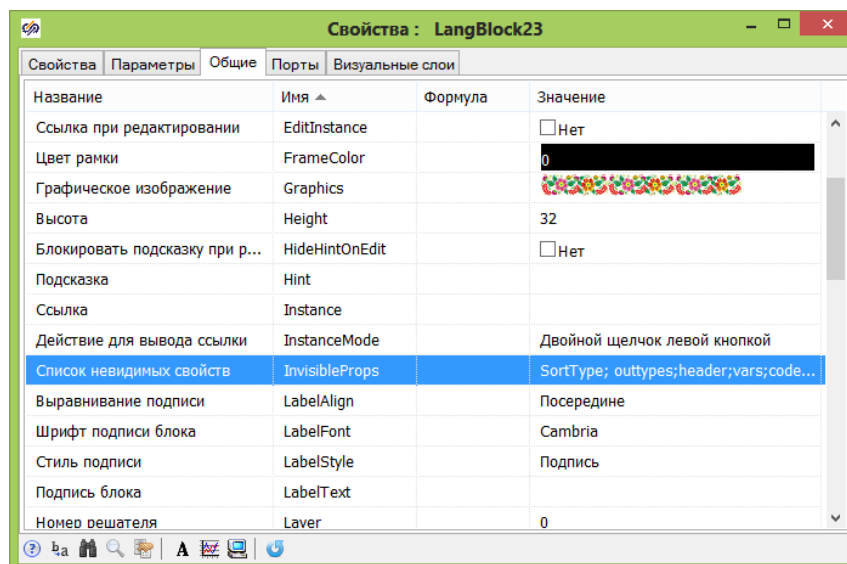


Рисунок 11

и копируем скопированный ранее список свойств в другое свойство блока **“Список невидимых свойств”** см. Рис 11. Если мы опять перейдем на вкладку **“Свойства”**, то выделенный список останется на своем месте, картина изменится в том случае если, выделив блок курсором мыши выберем опцию **“Свойства объекта”** см. Рис. 12.

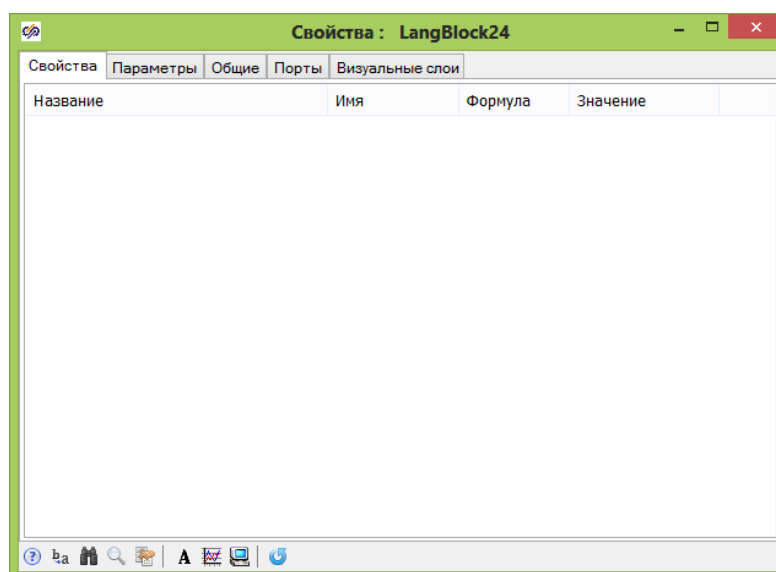


Рисунок 12

Список свойств для настройки работы блока пустой, теперь необходимо сформировать список свойств блока, настраивающих работу.

3) Для добавление новых свойств необходимо выполнить действия пункт 1) и используя кнопки, расположенные в нижнем левом углу окна свойств см. Рис 9

добавить новое свойство блока. После добавления нового свойства блока необходимо определить название, имя свойства его тип см. Рис 13.

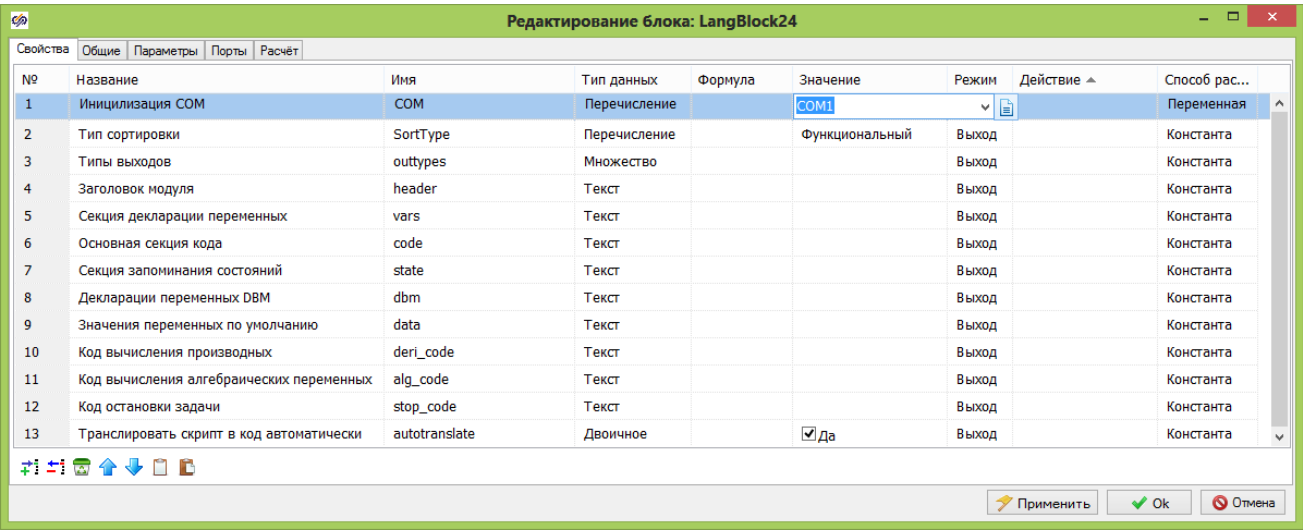


Рисунок 13

Для создания блока обмена данными по протоколу Modbus добавили выбор устройства последовательного интерфейса с именем “COM” см. Рис. 13, тип данных “Перечисление” — это целое значение, которого будет меняться в заданном диапазоне. Для определения возможных значений “Перечисление” необходимо определить список возможных значений, для этого необходимо указателем мыши нажать на “Альбом” в столбце “Значение” окна свойств блока см. Рис. 13 в результате появится область ввода значений см. Рис. 14

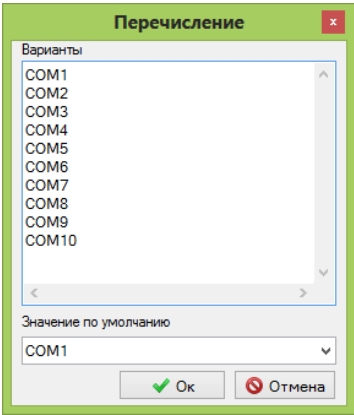
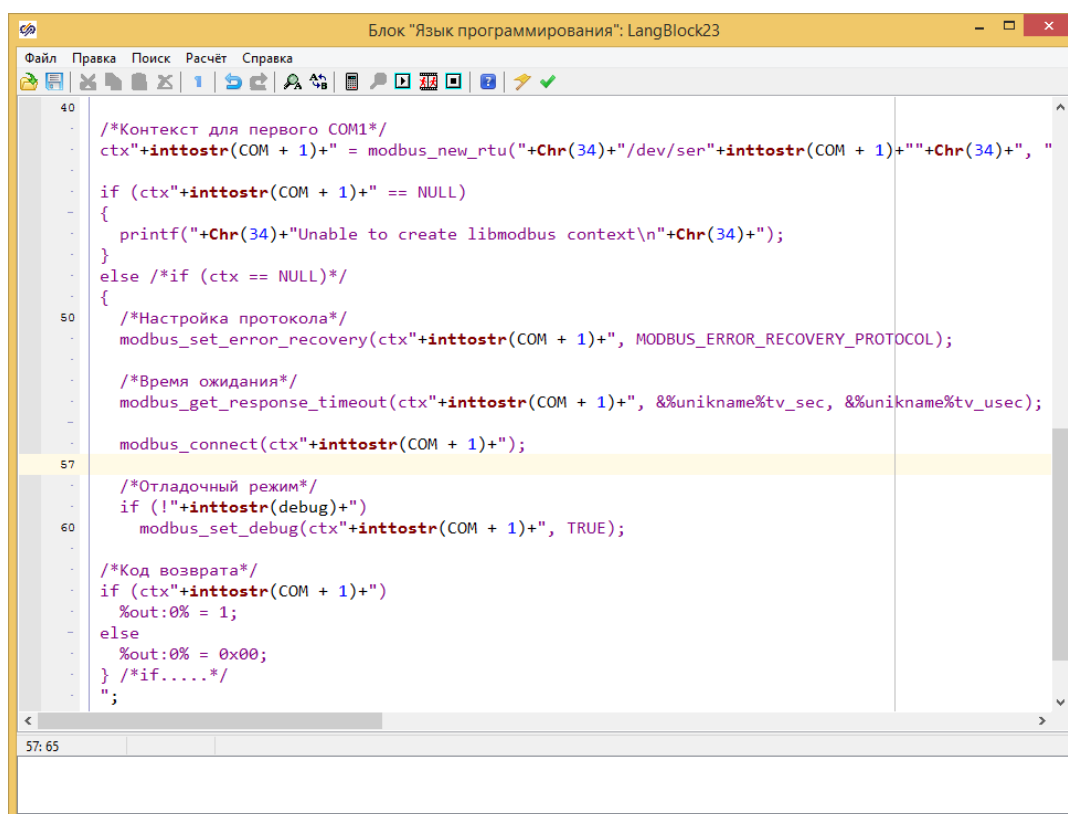


Рисунок 14

В области ввода мы перечислили все возможные значения “COM” иных значений “COM” принимать не будет (значения свойства будут варьироваться от 0 до 9). При выборе каждого следующего значения свойства “COM” будет увеличиваться на единицу, внутри блока это свойство можно использовать, как целую переменную и использовать ее при разработке алгоритма блока с помощью скриптового языка. Это свойство можно использовать при генерации Си-кода, например, в текстовом поле “data” в функции

инициализации устройства можно сформировать имя устройства последовательного интерфейса в зависимости от выбора, сделанного пользователем см. Рис. 15.



```
40  /*Контекст для первого COM1*/
    ctx"+inttostr(COM + 1)+" = modbus_new_rtu("+Chr(34)+"/dev/ser"+inttostr(COM + 1)+" "+Chr(34)+", "
    if (ctx"+inttostr(COM + 1)+" == NULL)
    {
        printf("+Chr(34)+"Unable to create libmodbus context\n"+Chr(34)+");
    }
    else /*if (ctx == NULL)*/
    {
        /*Настройка протокола*/
        modbus_set_error_recovery(ctx"+inttostr(COM + 1)+" , MODBUS_ERROR_RECOVERY_PROTOCOL);

        /*Время ожидания*/
        modbus_get_response_timeout(ctx"+inttostr(COM + 1)+" , &%unikname%tv_sec , &%unikname%tv_usec);

        modbus_connect(ctx"+inttostr(COM + 1)+" );
57
        /*Отладочный режим*/
        if (!"+inttostr(debug)+" )
        modbus_set_debug(ctx"+inttostr(COM + 1)+" , TRUE);
60
        /*Код возврата*/
        if (ctx"+inttostr(COM + 1)+" )
            %out:0% = 1;
        else
            %out:0% = 0x00;
        } /*if.....*/
    };
57: 65
```

Рисунок 15

Функция “**inttostr ()**” преобразует целое значение к текстовому см. Рис. 15. Для того, чтобы получить в СИ-коде строку определяющую устройство последовательного интерфейса “**/dev/ser№**”, где **№** номер устройства, необходимо в любом из текстовых полей шаблона генерации СИ-кода использовать следующую конструкцию:

" +Chr (34)+"/dev/ser"+inttostr(COM + 1)+" "+Chr(34)+",

где **" +Chr(34)+"** кодирует кавычки , **" +inttostr(COM + 1)+"** преобразует целое значение в текстовое представление при генерации СИ-кода. Остальные свойства блока добавляем по аналогии см. Рис. 16.

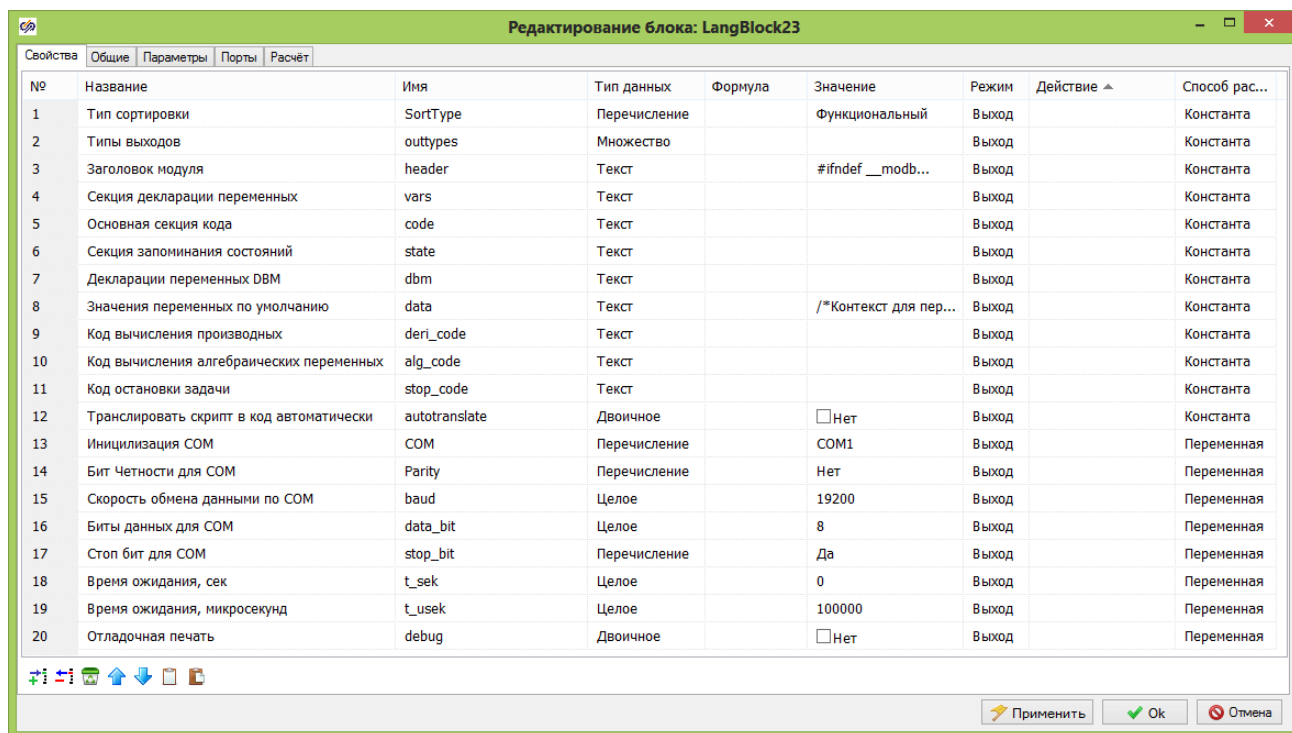


Рисунок 16

Свойство “**Parity**” имеет перечисляемый тип и может принимать только два значения (0 или 1) внутри блока это свойство является переменной типа integer. На основании значения этого свойства выберем символическую переменную используемую в качестве третьего аргумента функции инициализации протокола Modbus (‘Y’ или ‘N’), для выбора можно воспользоваться встроенным языком программирования см. Рис. 17 и на основании значения целой переменной “**Parity**” определить символ (‘Y’ или ‘N’), символические переменные будут храниться в строке “**Parity_str**”.

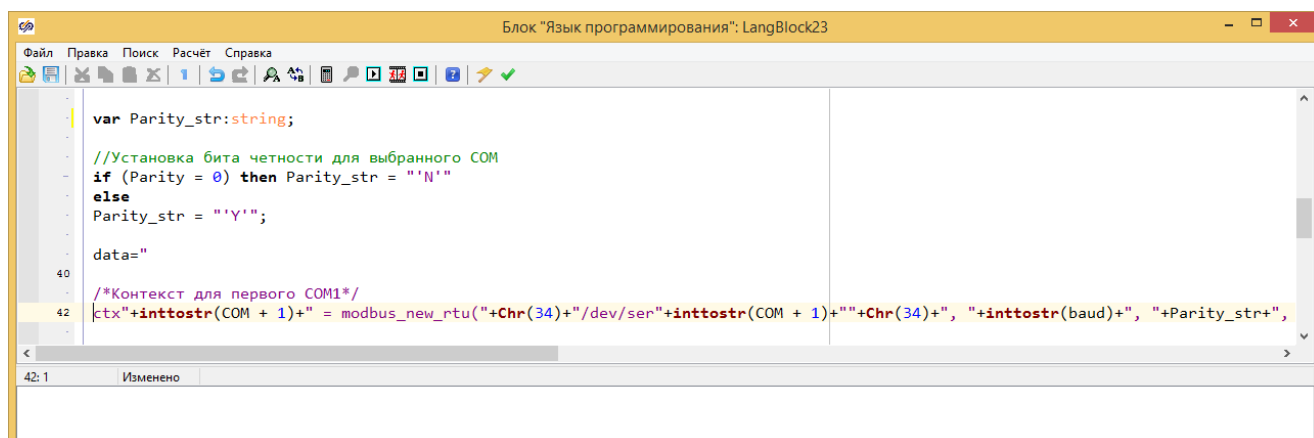


Рисунок 17

Для определения значения строковой переменной “**Parity_str**” используем алгоритм, анализирующий значение переменной-свойства “**Parity**” см. Рис. 17. Значение переменной может принимать два значения (0 или 1)

```

var Parity_str:string;
//Установка бита четности для выбранного COM
if (Parity = 0) then Parity_str = "N"
else
Parity_str = "Y";

```

В результате работы алгоритма строка **Parity_str** может принять два значения (`Y` или `N`), для правильного преобразования в текст при генерации СИ-кода необходимо строковую переменную в текстовом поле “**data**” представить в виде выражения: “**+Parity_str+**” (для преобразования из строкового типа в текстовый необходимо использовать окаймить переменную кавычками, а в начале и конце переменной добавит символ `+`, “**+string_var+**”), см. Рис. 17. Для свойства “**baud**” выбрали целый тип, поэтому переменная может принимать целые значения для ее преобразования в текстовый формат используем функцию **inttostr ()** см. Рис 17 (для превращения целого значения в строку окаймляем кавычками функцию и добавляем символ `+` в начале и конце, логика следующая: функция **inttostr ()** из целого типа создает строковой тип, а для преобразования строки в текст необходимы кавычки и знак плюс).

Пример № 3

Управление генерацией СИ-кода с помощью скриптового языка программирования.

В зависимости от значения свойств разрабатываемого блока можно реализовывать разные алгоритмы генерации СИ-кода. Рассмотрим эту возможность на примере блока приема данных по протоколу Modbus “**Read AO**” см. Рис. 18. Выделим мышкой блок и нажмем правой кнопкой и перейдем на опцию “**Изменить блок...**”, это необходимо для того, чтобы узнать типы созданных свойств блока (они выделены) см. Рис. 19.

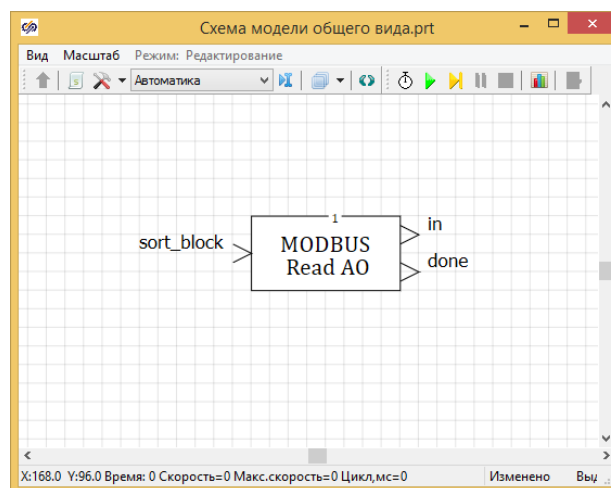


Рисунок 18

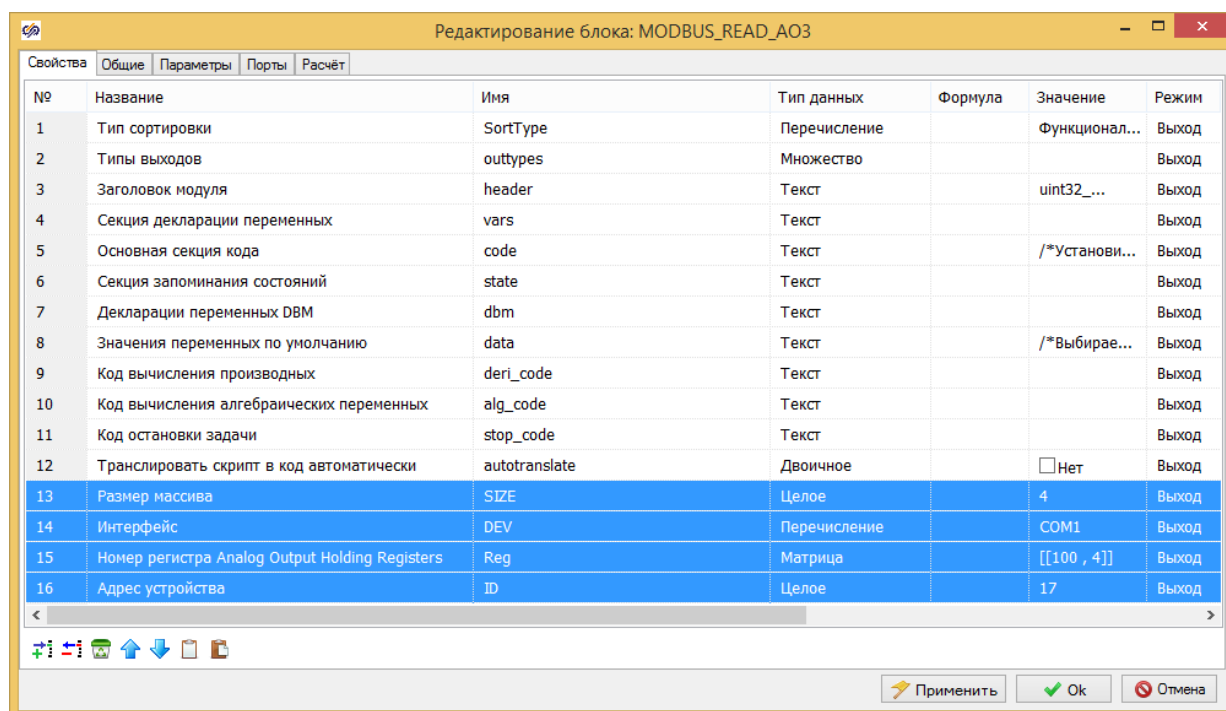


Рисунок 19

- Рассмотрим свойство с именем “**Reg**”, это свойство имеет тип матрица, этот тип удобен для того, чтобы пользователю было удобно задавать интервалы приема данных (начальный индекс и количество сигналов которые необходимо считать и разместить в таблице “АО” начиная с начального адреса, например, [[100, 4]] начиная с индекса 100 разместить в таблице “АО” 4 элемента). Внутри блока будет использоваться функция чтения регистров из библиотеки **libmodbus**:

int modbus_read_registers (modbus_t *ctx, int addr, int nb, uint16_t *dest), где

addr - является начальным индексом таблицы, а **nb** - количеством принятых значений.

Разрабатываемый блок должен обладать возможностью задавать в “**Reg**” несколько интервалов приема данных приходящих по протоколу Modbus ([[addr1, nb1]; [addr2, nb2]; [addr3, nb3];]) рассмотрим использование скриптового языка для формирования аргументов функции чтения данных **modbus_read_registers**. Рассмотрим код скриптового языка блока см. Рис. 20. Необходимо сформировать строку “s” содержащую элементы матрицы “**Reg**” в виде одномерного массива (s = “ {addr1, nb1, addr2, nb2, addr3, nb3};”), порядок расположения элементов матрицы по строкам. Строка “s” должна содержать символы “ {” и “}” в начале и конце, а между ними элементы матрицы “**Reg**” через запятую, в самом конце строки после символа “}” необходимо расположить символ “;”. Строка необходима для инициализации массива СИ-коде

“%username%registr[[2]="+s+" “ массив содержащий последовательность адресов и

количества ожидаемых данных будет использоваться в качестве аргумента функции чтения данных по протоколу Modbus **modbus_read_registers()**.

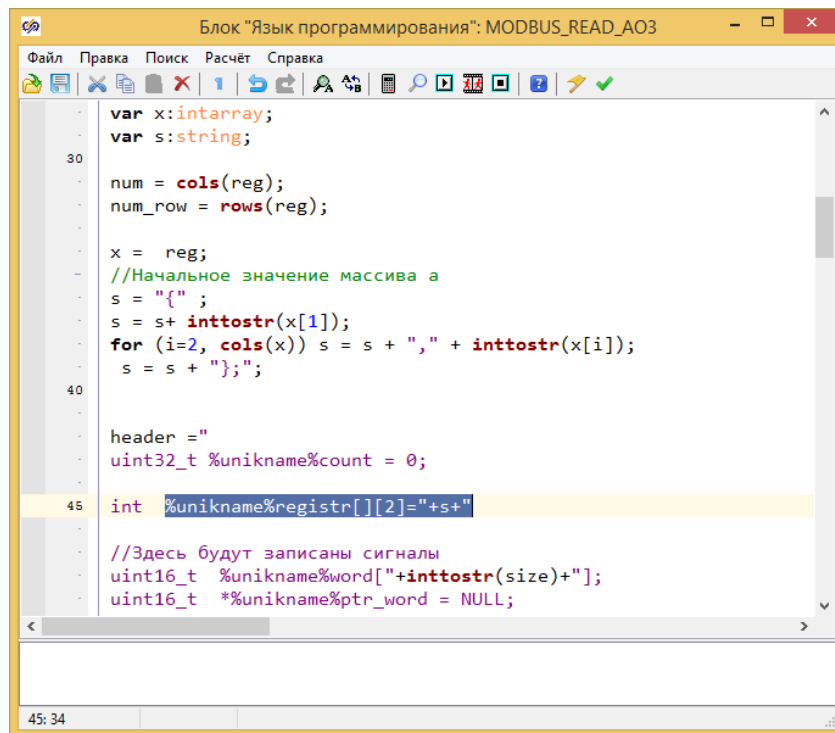


Рисунок 20

Для решения вышеописанной задачи необходимо:

- 1) Определить массив целочисленных элементов “**var x: intarray;**” и строковую переменную “**var s:string;**” после операции присвоения (**x = Reg**) элементов двухмерного массива “**Reg**” одномерному массиву “**x**” элементы хранящиеся в двухмерном массиве будут расположенными в одномерном массиве “**x**” по строкам см. Рис. 20.
- 2) Сформировать строку “**s**” содержащую элементы одномерного массива “**x**”, для этого используем следующий алгоритм:

```
s = "{";
```

```
s = s+ inttostr(x[1]);
```

```
for (i=2, cols(x)) s = s + "," + inttostr(x[i]);
```

```
s = s + "};";
```

В результате строка “**s**” содержит одномерный массив элементов (**s = “ {addr1, nb1, addr2, nb2, addr3, nb3};”**) и при генерации СИ-кода может инициализировать массив см. Рис. 21.

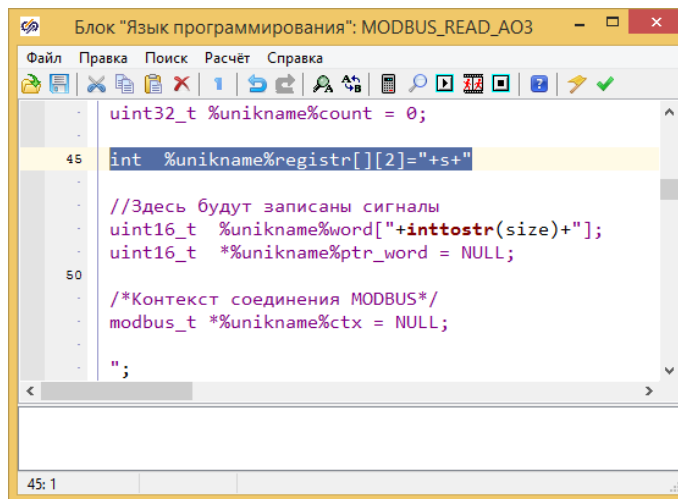


Рисунок 21

Массив `int %unickname%registr[][2]="+s+"` будет является аргументом при обращении к функции чтения регистров данных

```

%out:1% = modbus_read_registers(%unickname%ctx,
%unickname%registr[%unickname%count][0],
%unickname%registr[%unickname%count][1], %unickname%ptr_word);

```

- Рассмотрим свойство с именем “**DEV**” в зависимости от целочисленного значения свойства для генерации СИ-кода будут определяться текстовые поля **data** содержащие разный СИ-код. Код на скриптовом языке будет выглядеть следующим образом см. Рис. 22.

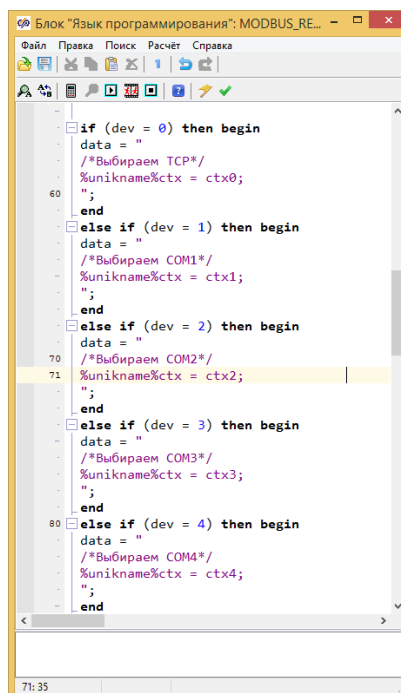


Рисунок 22

С помощью конструкции “**if then begin**” скриптового языка анализируя значение переменной “**dev**” выбираем содержание текстового поля “**data**”.

Существует особенность работы генерации СИ-кода при работе с массивами. Очень часто размер входного или выходного массива удобно определить через свойства блока, например, для рассматриваемого блока см. Рис. 23

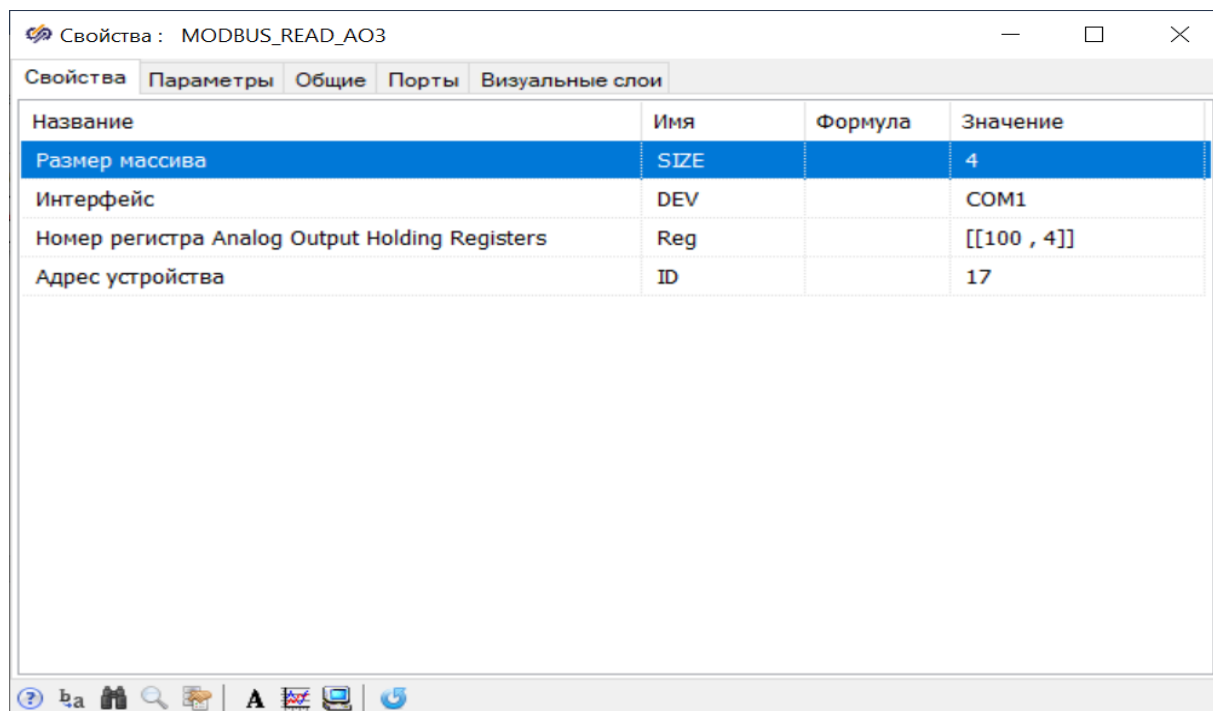


Рисунок 23

Это означает, что размер выходного массива может быть определен произвольным значением в том числе и единицей, в этом случае алгоритм генерации СИ-кода преобразует выходной массив к скалярной величине и если в шаблоне СИ-кода мы обращались к выходному массиву, как к векторной величине это может привести к ошибке при сборке расчетного модуля! Для этого необходимо воспользоваться скриптовым языком и в зависимости от размера выходного массива определить содержание текстового поля “**code**” см. приведенный ниже алгоритм, где “**in**” выходной массив размера “**size**”.

```

output in: intarray=size#0, done: integer;
if cols(in) > 1 then begin
code = "
.....";
end
else begin
code = "
.....";
end;

```

Существует еще один способ решения задачи! Создать указатель и присвоить ему адрес массива, а к указателю можно обращаться, как к массиву. В этом случае можно не использовать встроенный язык программирования.

Пример № 4

Рекомендация по созданию типового блока чтения данных с УВВ.

При получении значений сигналов с датчиков для алгоритма обработки важно не только значение измеряемой величины, но и достоверность полученного значения (ведь измерительное устройство может передать некорректный результат), а также время возникновения измеряемого сигнала. За оценку достоверности сигнала должен отвечать алгоритм драйвера устройства, драйвер должен оценить правильность значения сигнала (выход за границы диапазона, тайм-аут и т.д.) и свою оценку записать в переменную status (статус), а время возникновения сигнала драйвер должен записать в переменную time, это необходимо для архивирования сигнала. Будет полезным если блок чтения данных от УВВ будет передавать не только массив значений сигналов, но и массив статусов и времен возникновения сигнала. Приведем пример блока получения данных от УВВ см. Рис. 24.

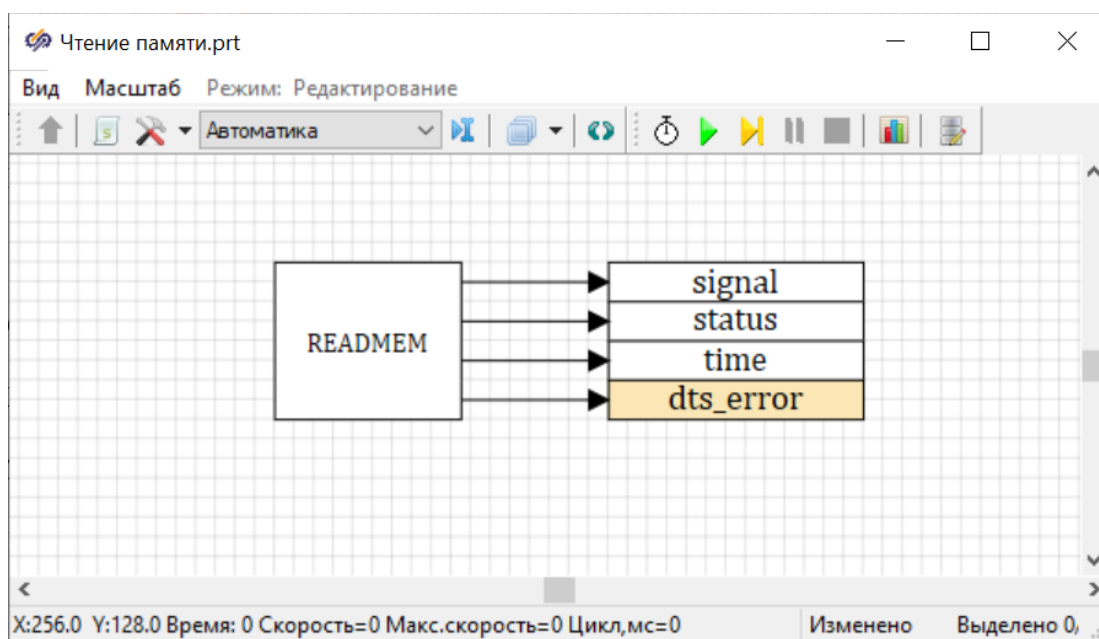


Рисунок 24

Для связи выходных массивов с СИ-кодом используется следующее правило:

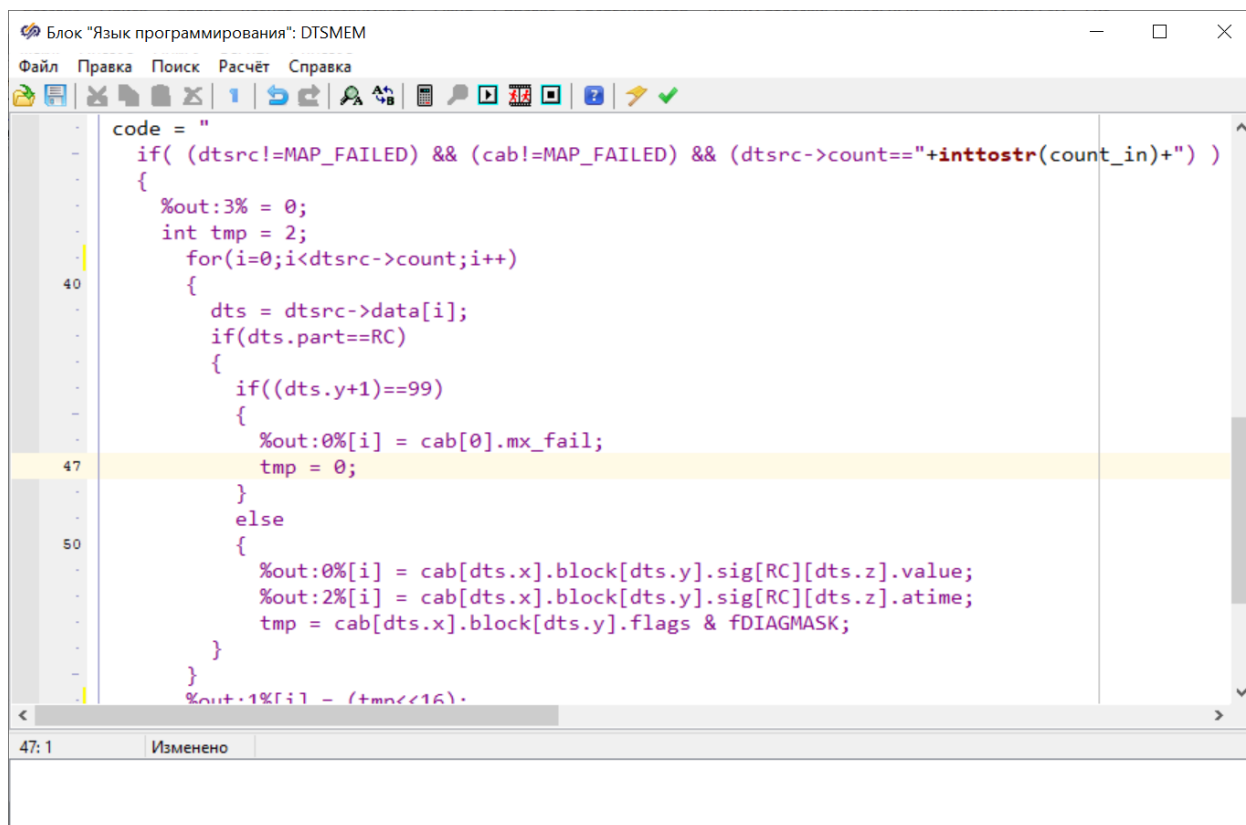
```
output ao: array = count_in#0, //%out:0%
```

```
st: intarray = count_in#0, //%out:1%
```

```
tm: array = count_in#0, //%out:2%
```

```
err: boolean = 1; //%out:3%
```

выходные переменная кодируются идентификатором “%out: index++%”, где начальное значение index равно 0 и увеличивается на единицу в порядке задания выходных переменных. Идентификатор может быть и скалярной переменной, и массивом см. Рис.25



```
code = "
-   if( (dtsrc!=MAP_FAILED) && (cab!=MAP_FAILED) && (dtsrc->count==" + inttostr(count_in) + " ) )
-   {
-       %out:3% = 0;
-       int tmp = 2;
-       for(i=0;i<dtsrc->count;i++)
40      {
-           dts = dtsrc->data[i];
-           if(dts.part==RC)
-           {
-               if((dts.y+1)==99)
-               {
-                   %out:0%[i] = cab[0].mx_fail;
47      tmp = 0;
-               }
-           }
50      else
-           {
-               %out:0%[i] = cab[dts.x].block[dts.y].sig[RC][dts.z].value;
-               %out:2%[i] = cab[dts.x].block[dts.y].sig[RC][dts.z].atime;
-               tmp = cab[dts.x].block[dts.y].flags & fDIAGMASK;
-           }
-       }
-       %out:1%[i] = (tmp<16);

```

Рисунок 25

Выходной массив значений сигналов необходимо записать в список сигналов базы данных проекта, для этого используется блок “**Запись в список сигналов**” (тулбокс сигналы) в нем приведен список имен сигналов базы данных проекта советуящий выходному массиву значений сигналов см. Рис. 26.

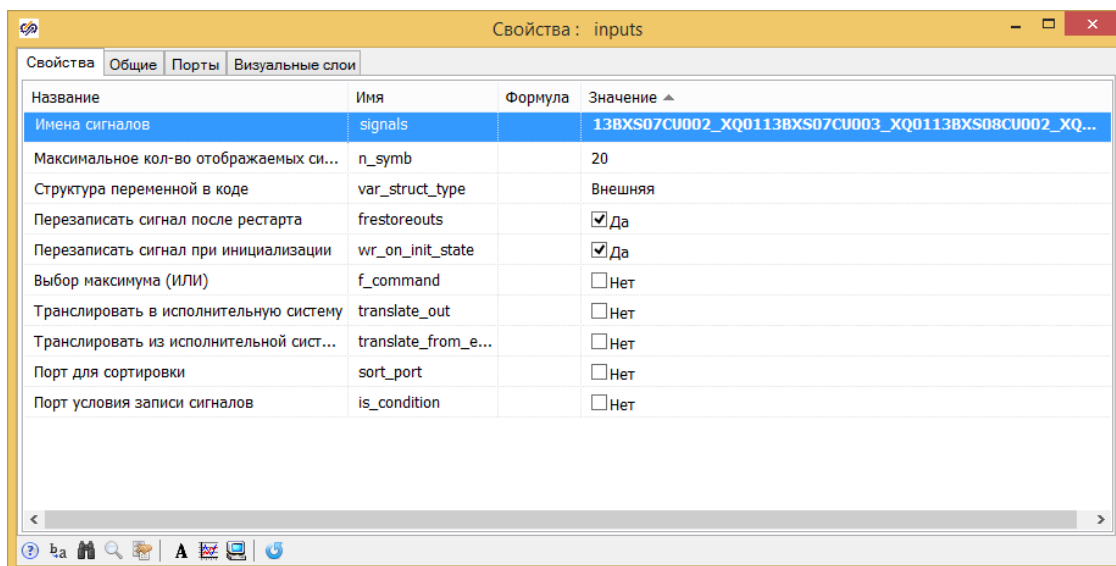


Рисунок 26

Для выходных массивов status и time нет соответствующих сигналов в базе данных проекта, но при генерации СИ-кода имена этих сигналов попадут в таблицу сигналов “*.extvars.table”, а среда выполнения алгоритмов “**NordWind**” выделяет области разделяемой памяти для хранения статусов и времен возникновения сигналов, поэтому при работе на контроллере значения статусов и времен возникновения сигналов будут отражены в этих областях памяти и будут иметь одинаковый индекс с массивами структур описывающих сигналы. Для выходных массивов status и time используется библиотечный блок “**Выходной контакт**” (тулбокс сигналы) в этом блоке создается список статусов и времен возникновения сигналов, в соответствии с правилом имя сигнала точка и поле расширение статус и время (signal_name.status, signal_name. time) см.Рис 27 и Рис 28.

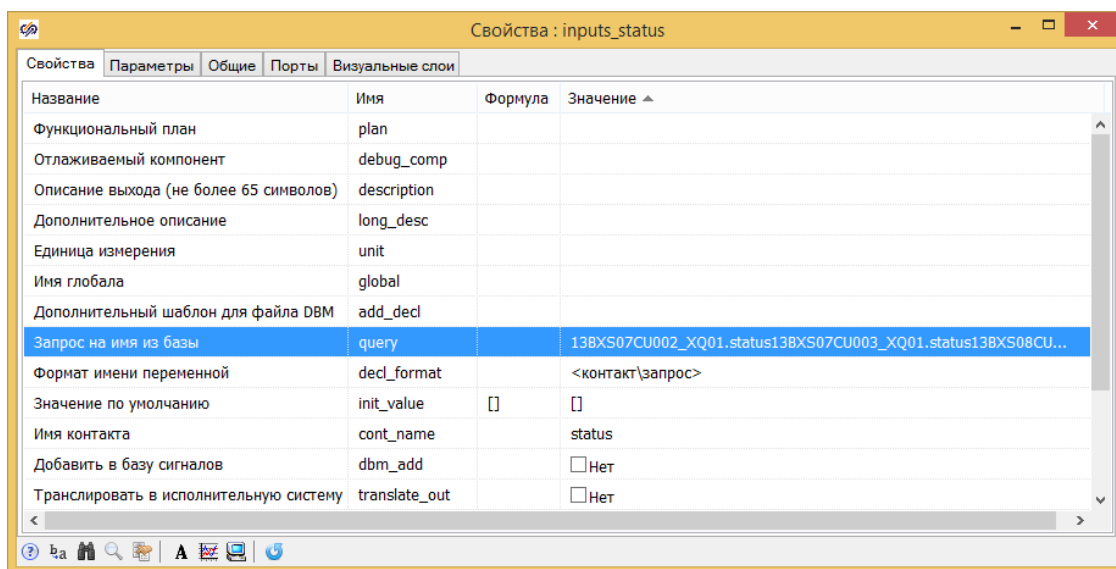


Рисунок 26

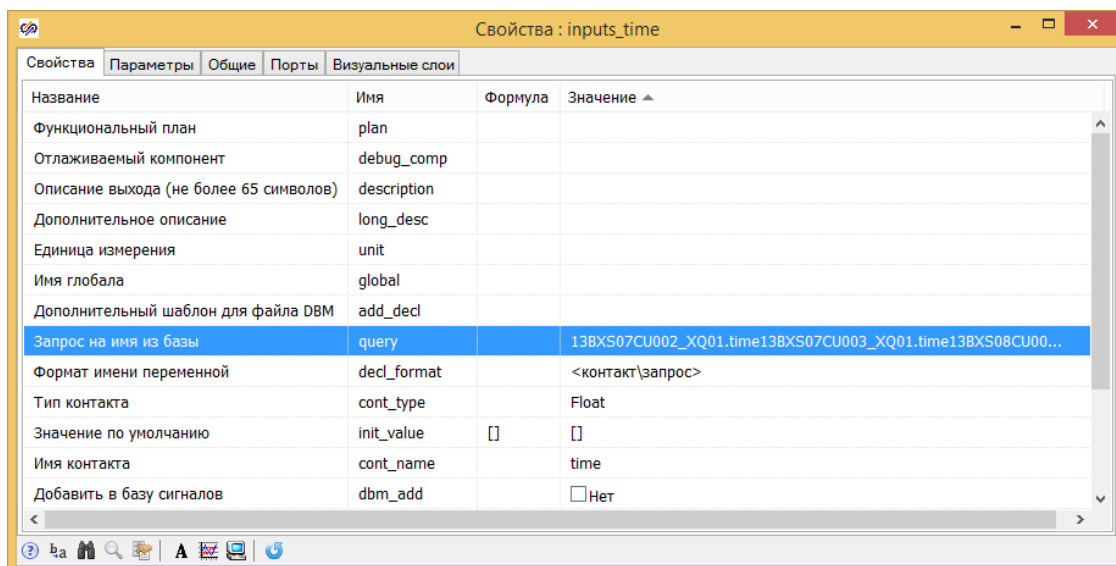


Рисунок 27

Часто бывает полезно подписать разработанный блок обмена данными с УВВ, это поможет сделать схему алгоритма более наглядной и читабельной. Для подписи блока необходимо:

- Выделить блок мышкой и нажать на правую кнопку, в выпадающем меню выбрать опцию **“Графическое изображение блока...”** см. Рис. 28

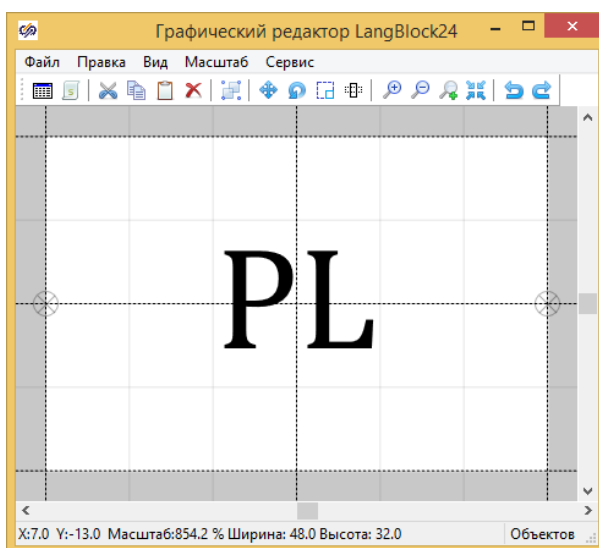


Рисунок 28

- Для замены текста **“PL”** на актуальный необходимо навести на него курсор мыши и два раза нажать левой клавишей в результате появится редактор, где можно набрать название блока см, Рис. 29.

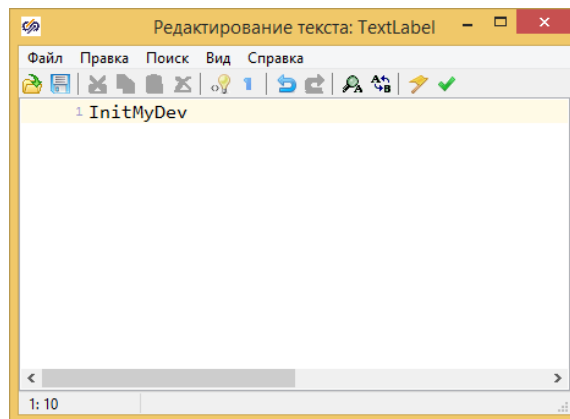


Рисунок 29

В результате на блоке появится надпись см. Рис. 30.

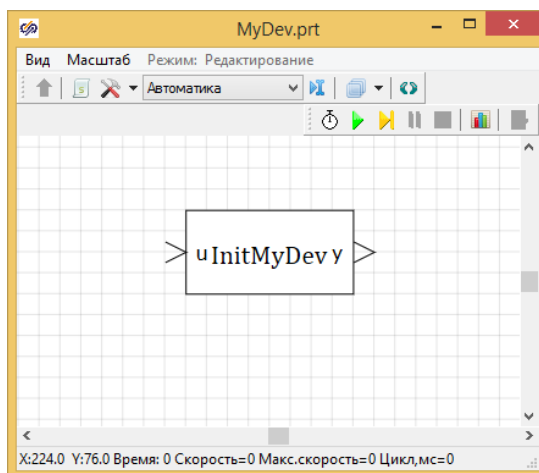


Рисунок 30

Можно изменить параметры шрифта и расположение надписи на блоке, для этого необходимо перейти к графическому изображению блока см. Рис. 28 курсором мыши активизировать надпись и нажать на правую кнопку мыши и выбрать опцию **“Свойства объекта”** в результате появится окно с помощью, которого можно выбрать оптимальные параметры надписи и размер шрифта см. Рис. 31.

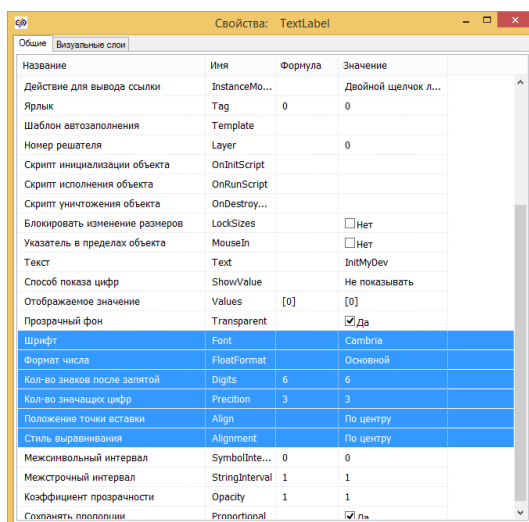


Рисунок 31

Для того, чтобы обозначения портов входа и выхода были снаружи блока необходимо активизировать блок мышью и нажать на правую кнопку и выбрать опции **“Свойства объекта”** в окне свойств во вкладке **“Общие”** в опции **“Режим показа подписей портов”** выбрать **“Значение: Снаружи блока всегда”** см. Рис. 32.

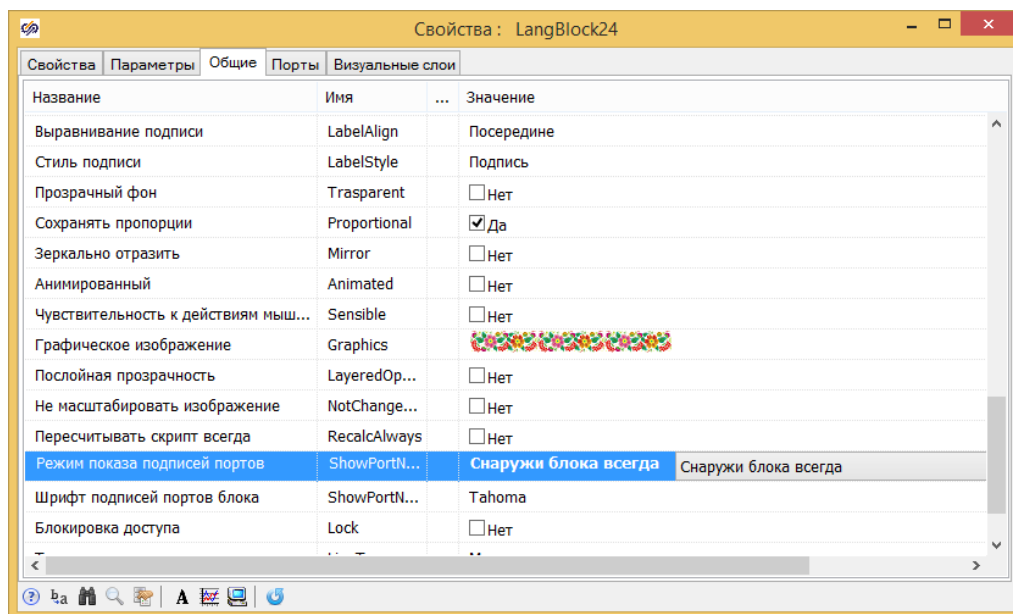


Рисунок 32

В результате изображение блока будет таким, как на Рис.33

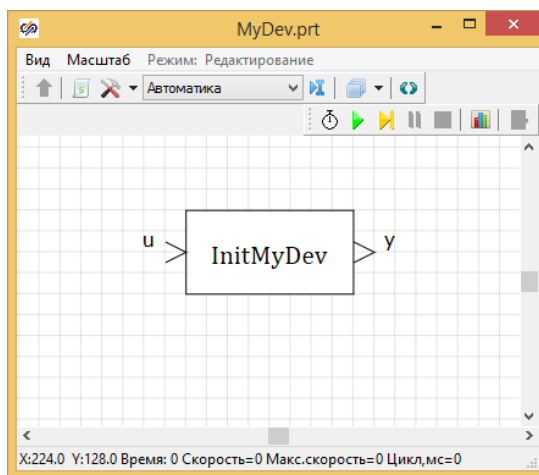


Рисунок 33

Метод 2

Для обмена данными с УВВ можно использовать динамически разделяемую библиотеку “**libio.so**”. Для сборки библиотеки необходимо использовать существующий проект библиотеки в IDE QNX Momentics он входит в комплект поставки ПО NordWind. Проект состоит из трех файлов:

- “**libio.c**” основной файл проекта содержит три функции:
 1. “**my_init ()**” предназначена для инициализации УВВ и вызывается один раз при инициализации библиотеки.
 2. “**read_io ()**” предназначена для чтения данных с УВВ и записи в базу данных NordWind, вызов происходит из среды выполнения алгоритмов NordWind перед началом запуска расчетного цикла.
 3. “**write_io ()**” предназначена для считывания значений управляющих сигналов из базы данных NordWind и передаче их УВВ, вызов происходит из среды выполнения алгоритмов NordWind после выполнения расчетного цикла.
- “**sdk_lib.c**”, “**sdk_lib.h**” содержат библиотеку доступа к базе данных NordWind, позволяя считывать и записывать значения сигналов по именам, эти функции удобно использовать в методах обмена данными с УВВ.

Для использования библиотеки “**libio.so**” ее необходимо поместить в рабочую директорию ПО NordWind в этом случае библиотека будет инициализирована средой исполнения алгоритмов “**NordWind**”, а функции обмена данными (**read_io** и **write_io**) будут вызываться на выполнение до и после завершения расчетного цикла алгоритмов соответственно. Программный код библиотеки выполняется в адресном пространстве “**NordWind**” и в случае возникновения ошибки в реализации функций обмена данными с УВВ произойдет прекращение работы среды исполнения алгоритмов “**NordWind**”. Существует возможность организовать обмен данными с УВВ с помощью отдельного приложения “**Driver_base**” входящего в комплект поставки ПО NordWind. Приложение “**Driver_base**” существует в виде проекта в IDE QNX Momentics:

1. **Driver.c** основной модуль приложения в нем реализована функция подключение к базе данных NordWind и пользователю предоставляется возможность используя вспомогательные функции чтения/записи базы данных реализовывать обмен данными между УВВ и базой данных.

2. “**sdk_lib+.c**”, “**sdk_lib+.h**” содержат библиотеку доступа к базе данных NordWind, позволяя считывать и записывать значения сигналов по именам, эти функции удобно использовать в методах обмена данными с УВВ.