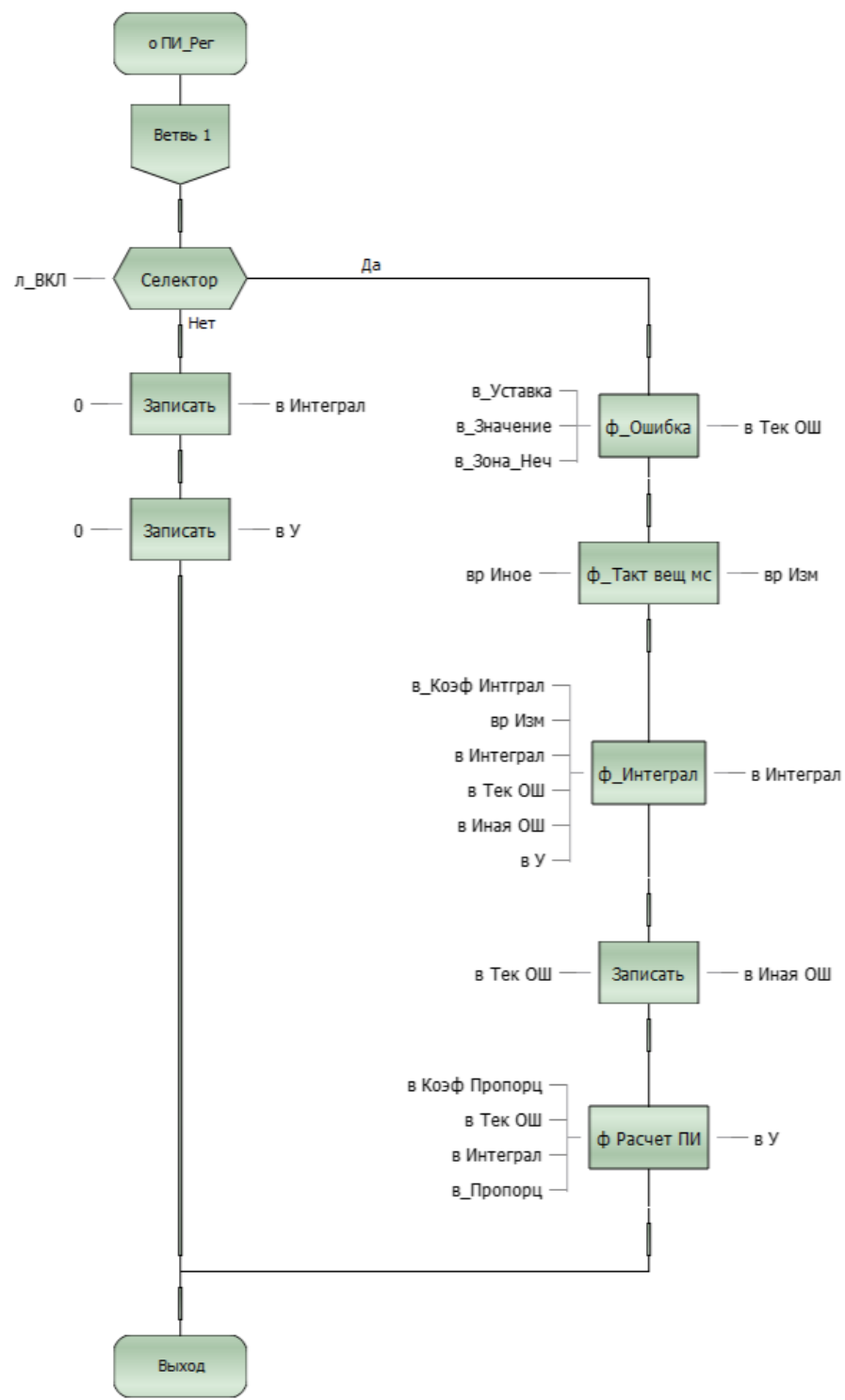


Сегодня я хочу поделиться успехами в эксплуатации среды разработки Дракон компании ООО АМГТ-Ресурс (Автор Алексей Степанов)

Задача: необходимо написать код автоматического Водопитателя. Водопитатель, по факту, это насос, который поддерживает давление в системе, путем плавного регулирования оборотов насоса. В случае невозможности, на полных оборотах насоса, компенсировать расход жидкости, подает сигнал утечки оператору.

Для работы по плавному регулированию нужно написать компонент ПИ – регулятор (Пропорционально Интегральный)

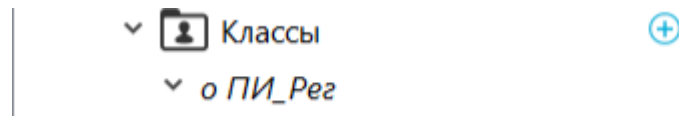
Его я создал в первую очередь в нашей среде:



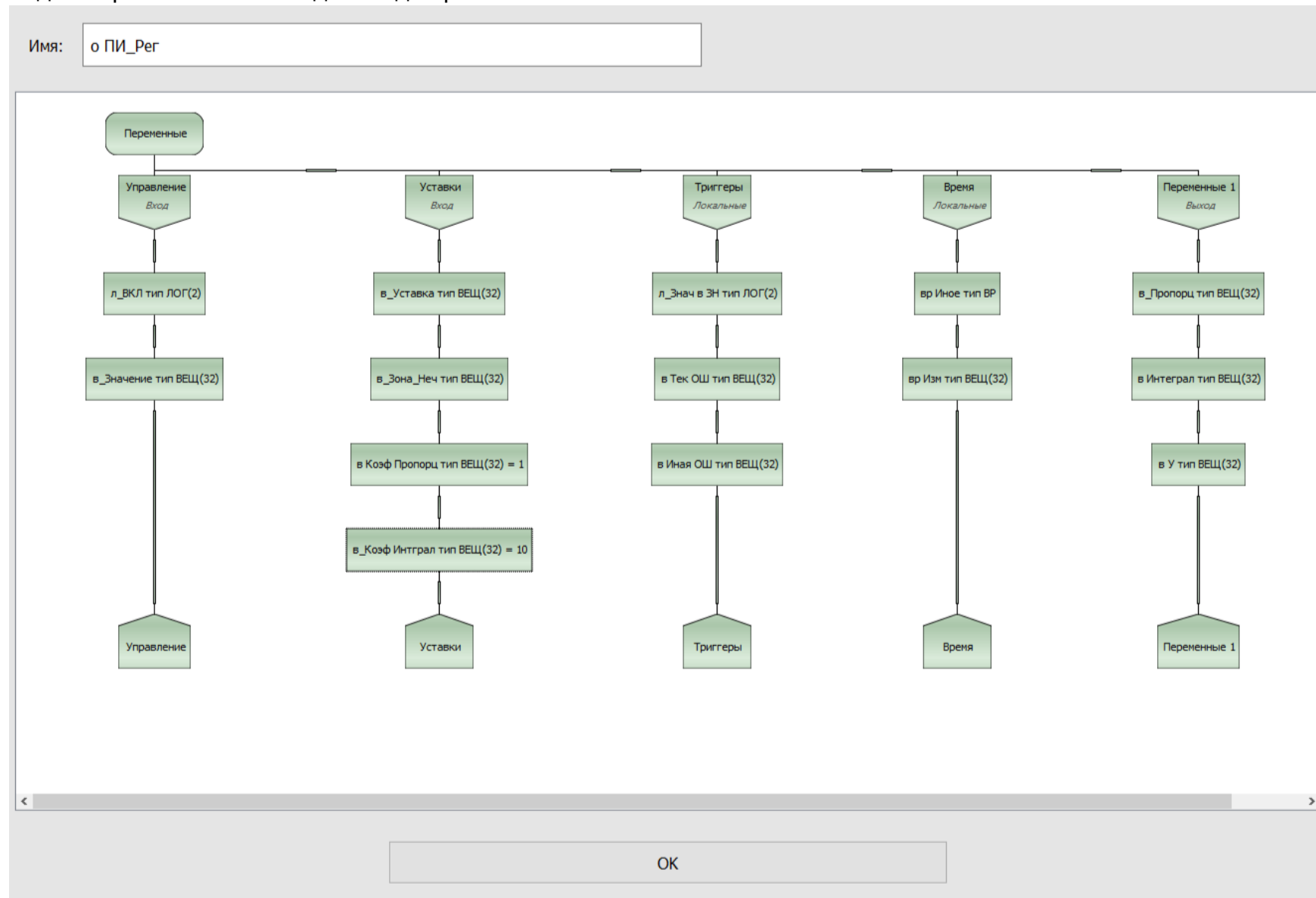
Вот такой компактный регулятор получился.

Использовал процедурный подход, т.е класс – это сама процедура, все остальное это классические функции. Методы, вставки и свойства не использовал.

1. Создал класс (процедуру) с именем ПИ\_рег



## 2. Задал переменные необходимые для работы



Перед именем каждой переменной, я ставлю префикс, маленькими буквами, который обозначает тип переменной  
в – Вещественные

л- Логические

вр – Время

ф – Функция

н -натуральное

и т.д

Имена переменных можно задавать как на русском, так и на английском, любыми регистрами (нижний, верхний), допускаются цифры, нижнее подчеркивание, и пробелы.

Те же правила действуют для имен объектов: Программа, Класс, Метод , Вставка, Свойство, Функция, Список глобальных переменных,

Назначение переменных я объяснять не буду, из названий и так понятно.

Единственное наверно непонятное имя «Иной» - я выбрал это слово, вместо слова «Предыдущий», только по тому что он короче, и в принципе отражает суть.

### 3. Создание функций.

Функции — это блоки кода, выполняющие определенные операции. Если требуется, функция может определять входные параметры, позволяющие вызывающим объектам передавать ей аргументы. При необходимости функция также может возвращать значение как выходное. Функции полезны для инкапсуляции основных операций в едином блоке, который может многократно использоваться. В идеальном случае имя этого блока должно четко описывать назначение функции. Функция может вызываться или вызываться из любого количества мест в программе. Значения, передаваемые функции, являются аргументами, типы которых должны быть совместимы с типами параметров в определении функции. Нет практического ограничения на длину функции, но хороший дизайн предназначен для функций, выполняющих одну хорошо определенную задачу. Сложные алгоритмы лучше разбивать на более короткие и простые для понимания функции, если это возможно.



Первая функция предназначена для вычисления ошибки, с учетом зоны не чувствительности регулятора, если она равна не 0

Имя:

Тип:

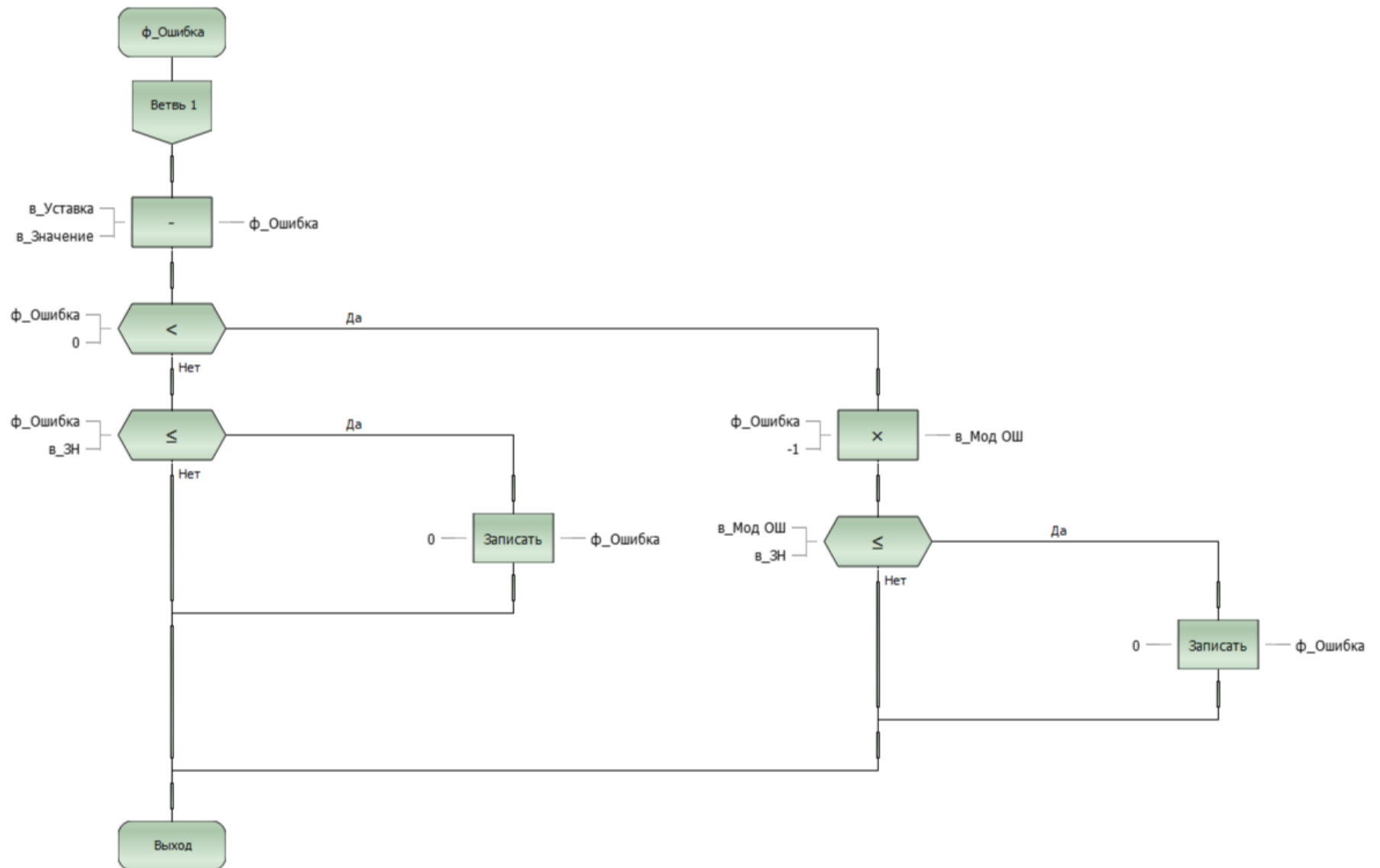
```
graph TD; Root[Переменные] --- Path1[Параметры Вход]; Root --- Path2[Возрат Вход-Выход]; Root --- Path3[Переменные Локальные]; Path1 --- V1[в_Уставка тип ВЕЩ(32)]; V1 --- V2[в_Значение тип ВЕЩ(32)]; V2 --- V3[в_ЭН тип ВЕЩ(32)]; V3 --- P1[Параметры]; Path2 --- R1[Возрат]; Path3 --- V4[в_Мод ОШ тип ВЕЩ(32)]; V4 --- P2[Переменные];
```

OK

Декларируем необходимые переменные, декларировать их можно как сразу, так и по ходу разработки Дракон схемы, через окно «ассистента»:

Имя:	<input type="text" value="в_Мод ОШ"/>	Тип:	<input type="text" value="Вещественное (32)"/>	▼	...	T
Начальное значение:	<input type="text"/> ...					
Перечисление:	<input type="text" value="ИСТИНА/ЛОЖЬ"/>					
Комментарий:	<input type="text"/>					
<input type="button" value="Готово"/>			<input type="button" value="Отмена"/>			

Далее разрабатываем дракон схему функции «ф\_Ошибка»:



На сегодняшний день встроенных операторов достаточно чтобы собирать необходимые алгоритмы. Специфические операторы будут содержаться в подключаемых библиотеках.  
В данном случае у меня не было функции «Модуль числа», ее заменил алгоритмом с умножением на -1.



Создаем функцию, Которую я назвал ф\_Такт вещ мс

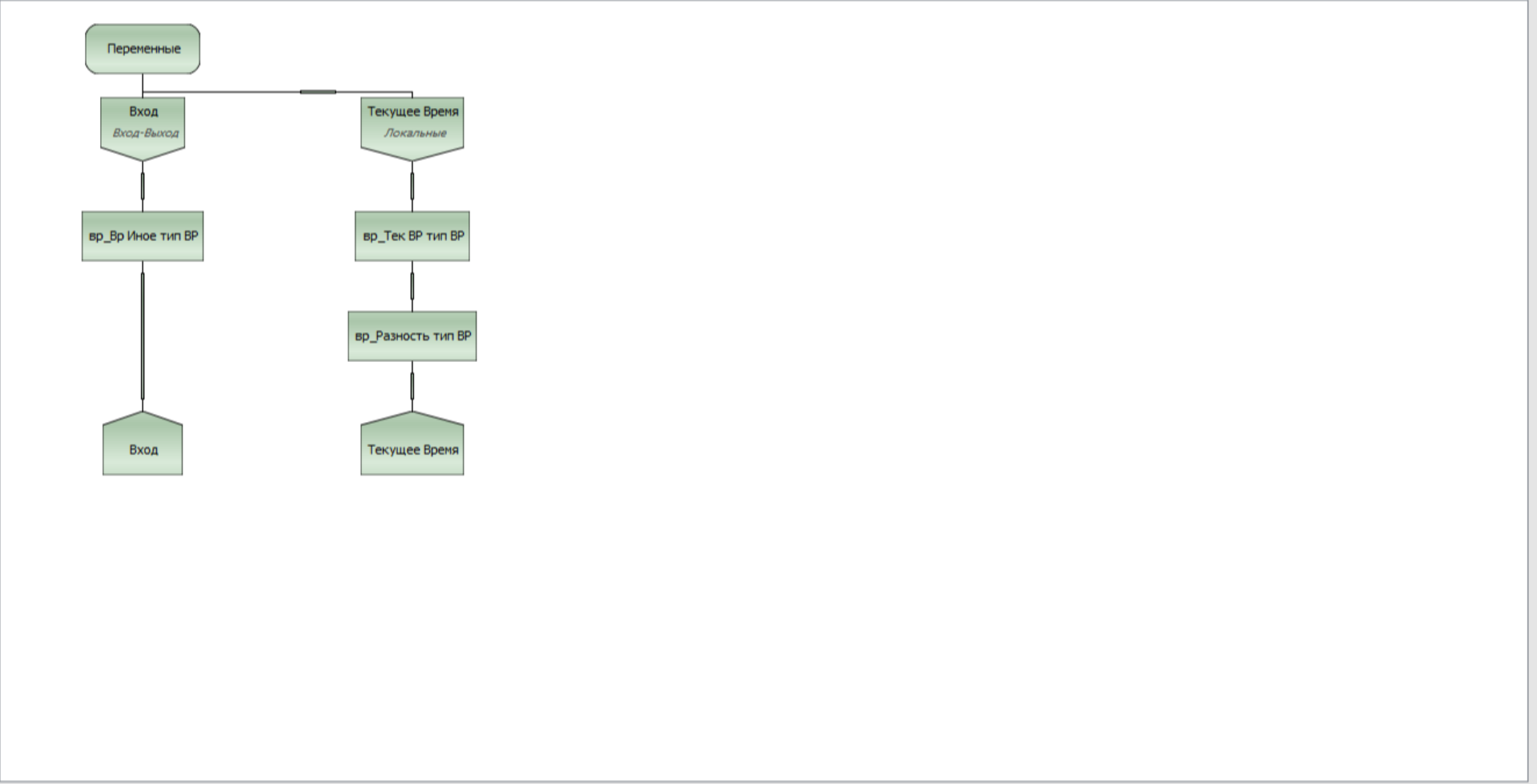
Задача этой функции, вычислять время между вызовами, преобразовывать его в вещественное, в миллисекундах.



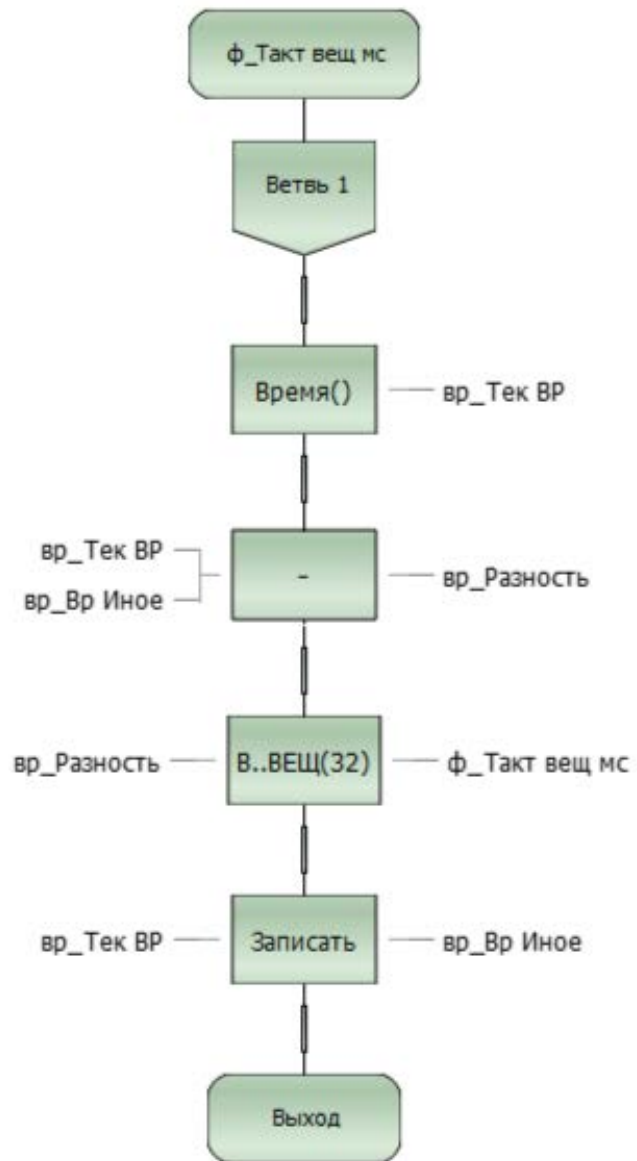
Определяем переменные для функции ф\_Такт вещ мс:

Имя:

Тип:

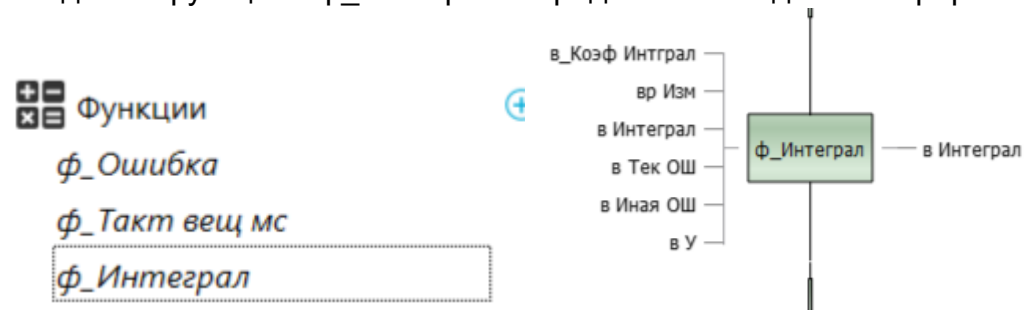


И создаем алгоритм



Здесь хочу обратить внимание на встроенную функцию «Время()», которая возвращает текущее время системы в миллисекундах, и встроенный оператор конверсии типов. В данном случае конверсия любого входящего типа, в тип Вещественное(32 бита)

Создание функции «ф\_Интеграл» -предназначена для интегрирования по времени между вызовами.



## Декларация переменных.

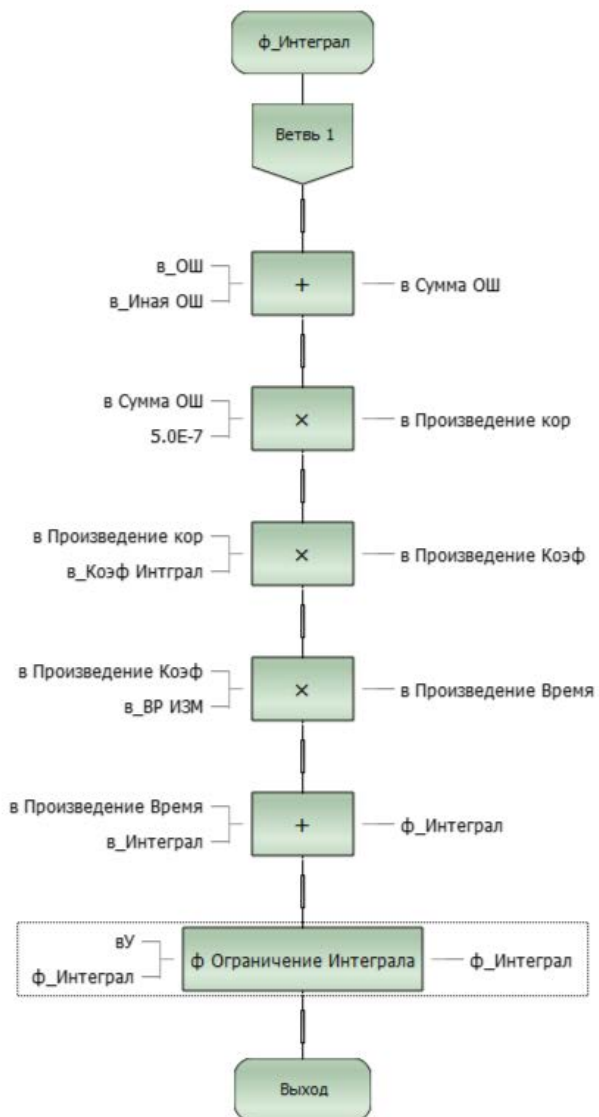
Имя:

Тип:

```
graph TD; Root[Переменные] --- Ustavki[Uставки  
Вход]; Root --- Teкущие_данные[Текущие данные  
Вход]; Root --- Локаль[Локаль  
Локальные]; Ustavki --- Ustavki_Var[в_Коеф Интграл тип ВЕЩ(32)]; Teкущие_данные --- Teкущие_данные_Var1[в_ВР ИЗМ тип ВЕЩ(32)]; Teкущие_данные --- Teкущие_данные_Var2[в_Интеграл тип ВЕЩ(32)]; Teкущие_данные --- Teкущие_данные_Var3[в_ОШ тип ВЕЩ(32)]; Teкущие_данные --- Teкущие_данные_Var4[в_Уная ОШ тип ВЕЩ(32)]; Teкущие_данные --- Teкущие_данные_Var5[вУ тип ВЕЩ(32)]; Локаль --- Локаль_Var1[в Сумма ОШ тип ВЕЩ(32)]; Локаль --- Локаль_Var2[в Произведение кор тип ВЕЩ(32)]; Локаль --- Локаль_Var3[в Произведение Коеф тип ВЕЩ(32)]; Локаль --- Локаль_Var4[в Произведение Время тип ВЕЩ(32)]; Ustavki_Var --- Ustavki_Box[Uставки]; Teкущие_данные_Var1 --- Teкущие_данные_Box[Текущие данные]; Teкущие_данные_Var2 --- Teкущие_данные_Box; Teкущие_данные_Var3 --- Teкущие_данные_Box; Teкущие_данные_Var4 --- Teкущие_данные_Box; Teкущие_данные_Var5 --- Teкущие_данные_Box; Локаль_Var1 --- Локаль_Box[Локаль]; Локаль_Var2 --- Локаль_Box; Локаль_Var3 --- Локаль_Box; Локаль_Var4 --- Локаль_Box;
```

OK

Схема функции:



Это функция задумывалась как вычислительная, но потом я добавил в нее еще одну функцию по ограничению интегрирования, чтобы не допустить переполнения интегральной части

Функция «ф Ограничение интеграла»

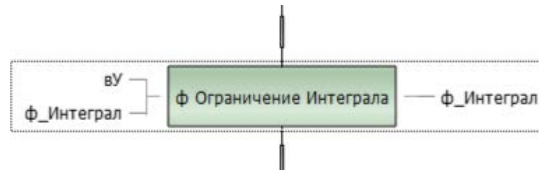
Функции

$\phi_{\text{Ошибка}}$

$\phi_{\text{Такт вец мс}}$

$\phi_{\text{Интеграл}}$

$\phi_{\text{Ограничение Интеграла}}$



Декларация переменных:

Имя:

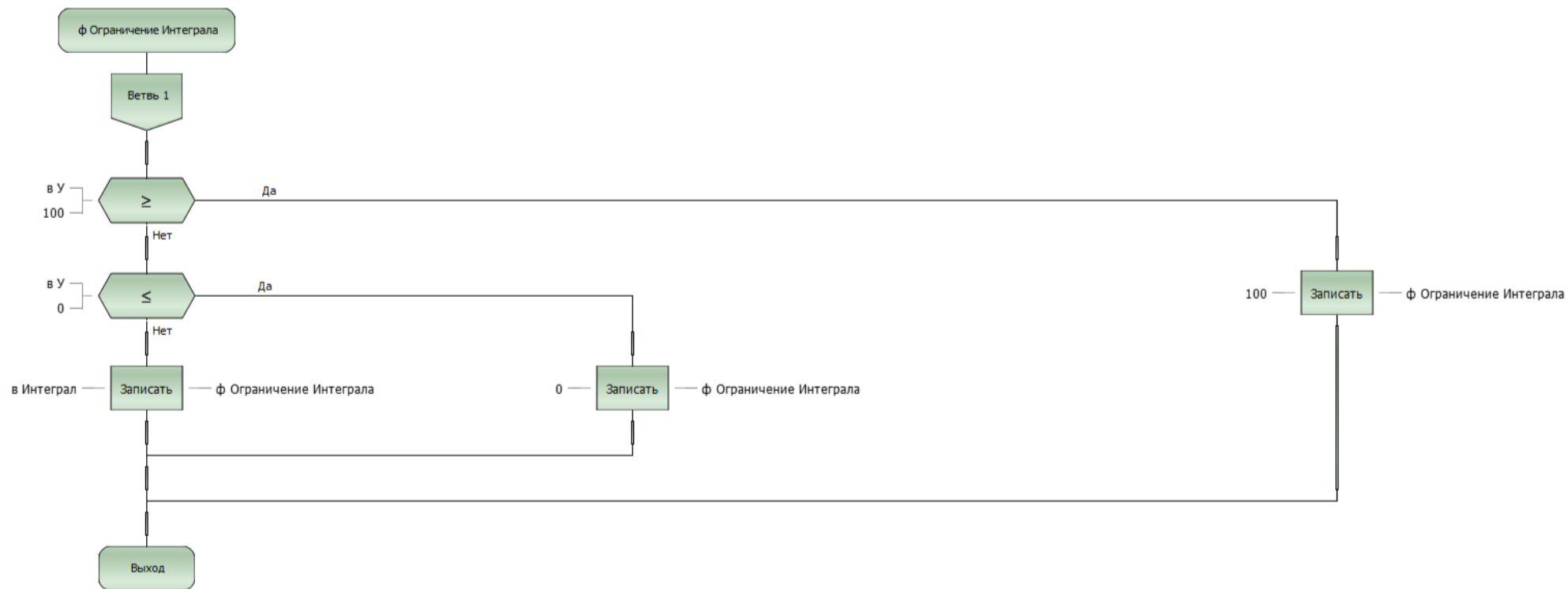
Тип:

```
graph TD; A[Переменные] --- B[Вход  
Вход]; B --- C[в У тип ВЕЩ(32)]; C --- D[в Интеграл тип ВЕЩ(32)]; D --- E[Вход];
```

OK

Собственно, сама схема:





Ну и последний модуль это «ф Расчет ПИ», задача этой функции посчитать  $Y$ , и ограничить его в пределах  $0 \dots 100$

Функции

*ф\_Ошибка*

*ф\_Такт вещ мс*

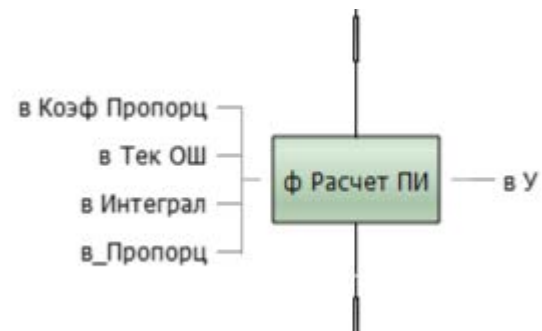
*ф\_Интеграл*

*ф Ограничение Интеграл*

*ф Расчет ПИ*

Программы

Планировщик

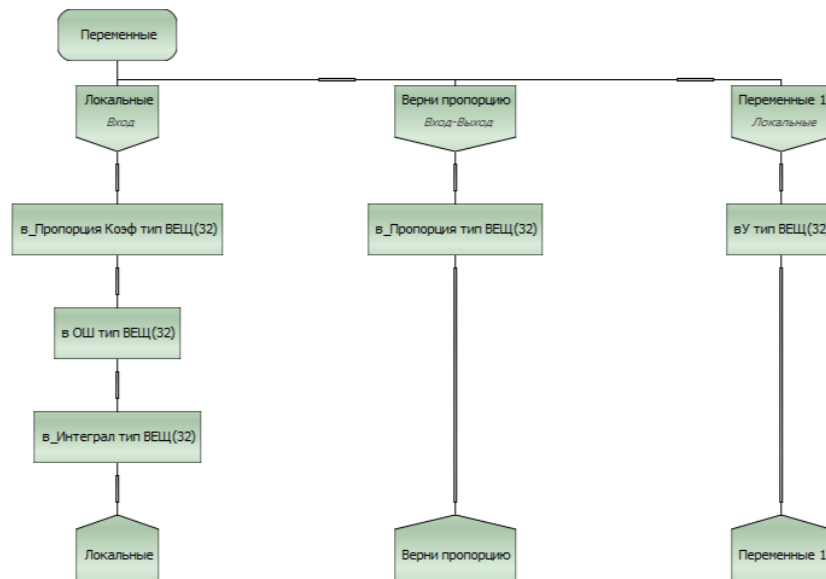


Декларация переменных:

Имя: ф Расчет ПИ

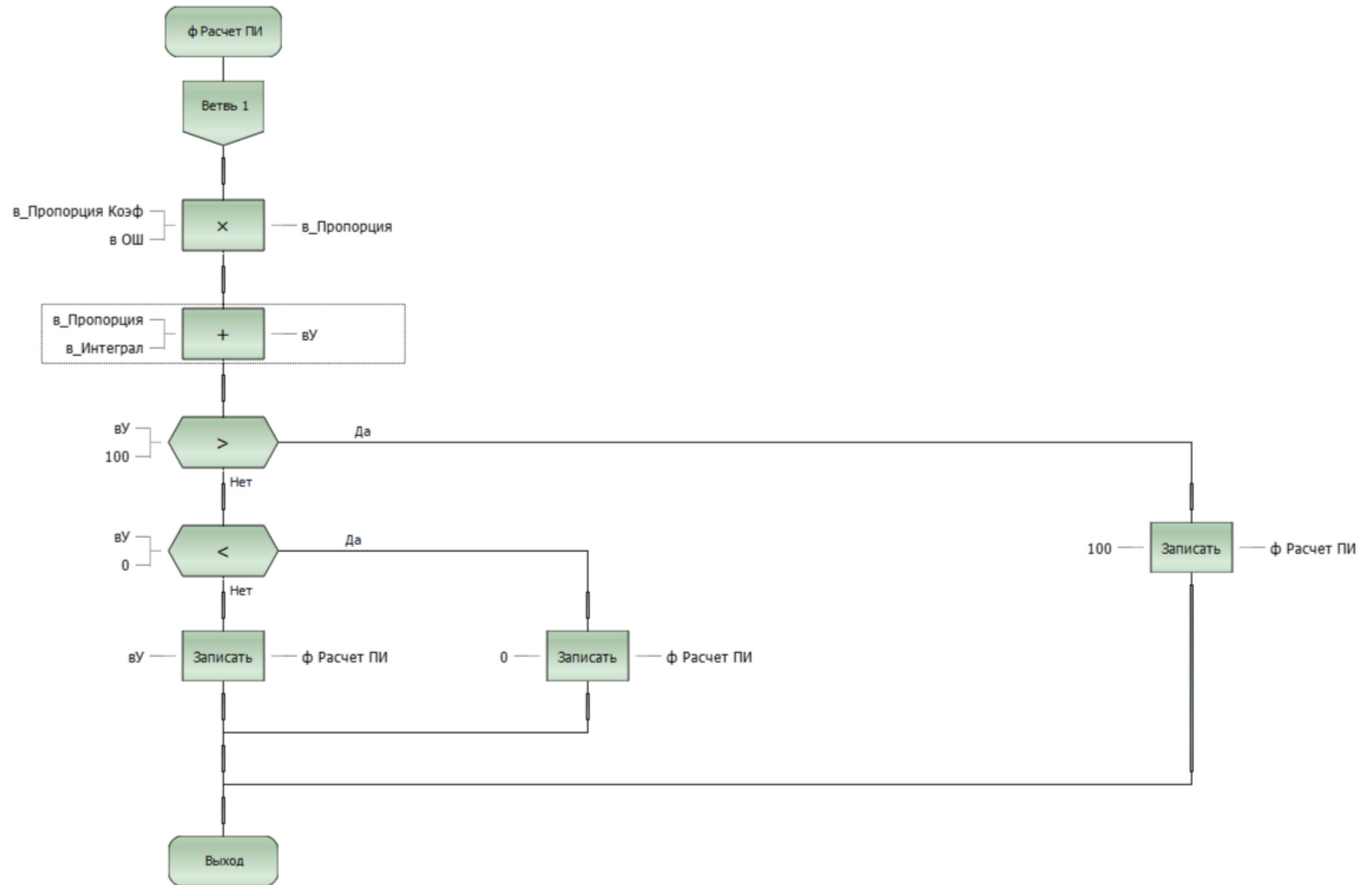
Тип: Вещественное (32)

~ ... T

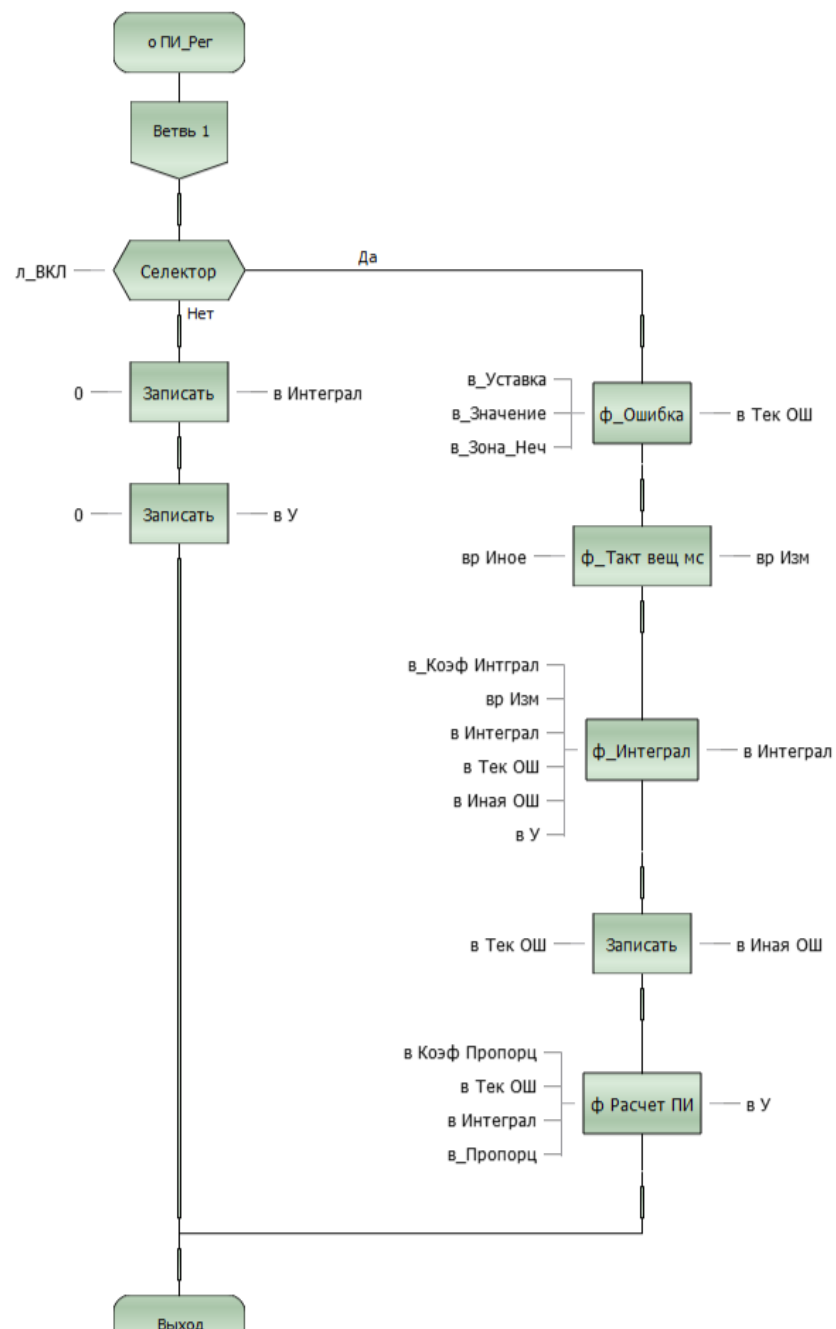


OK

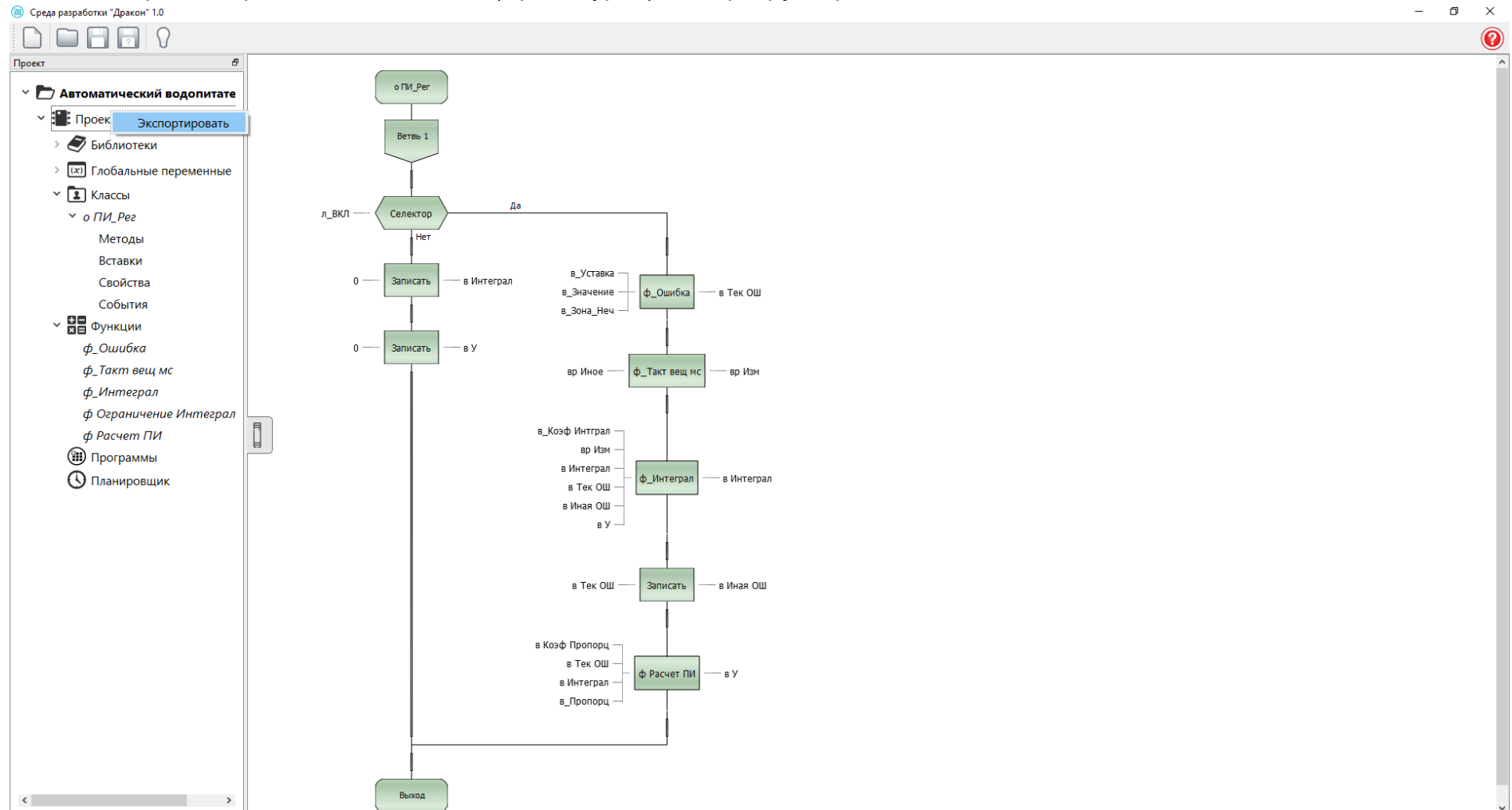
И схема:



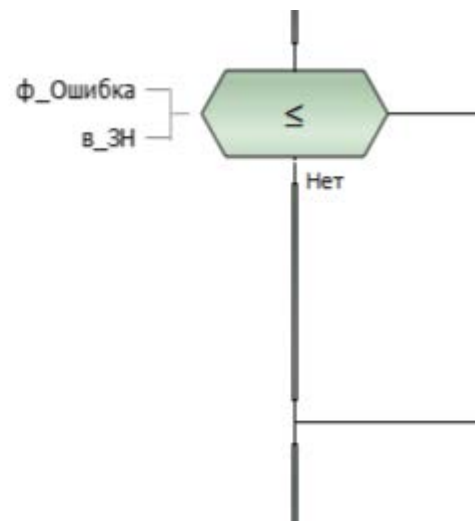
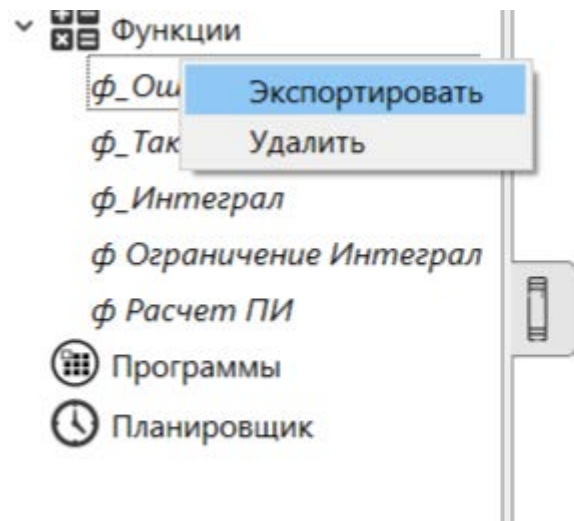
После создания необходимых функций, собрал все их по порядку в класс (Процедуру ПИ\_Per):



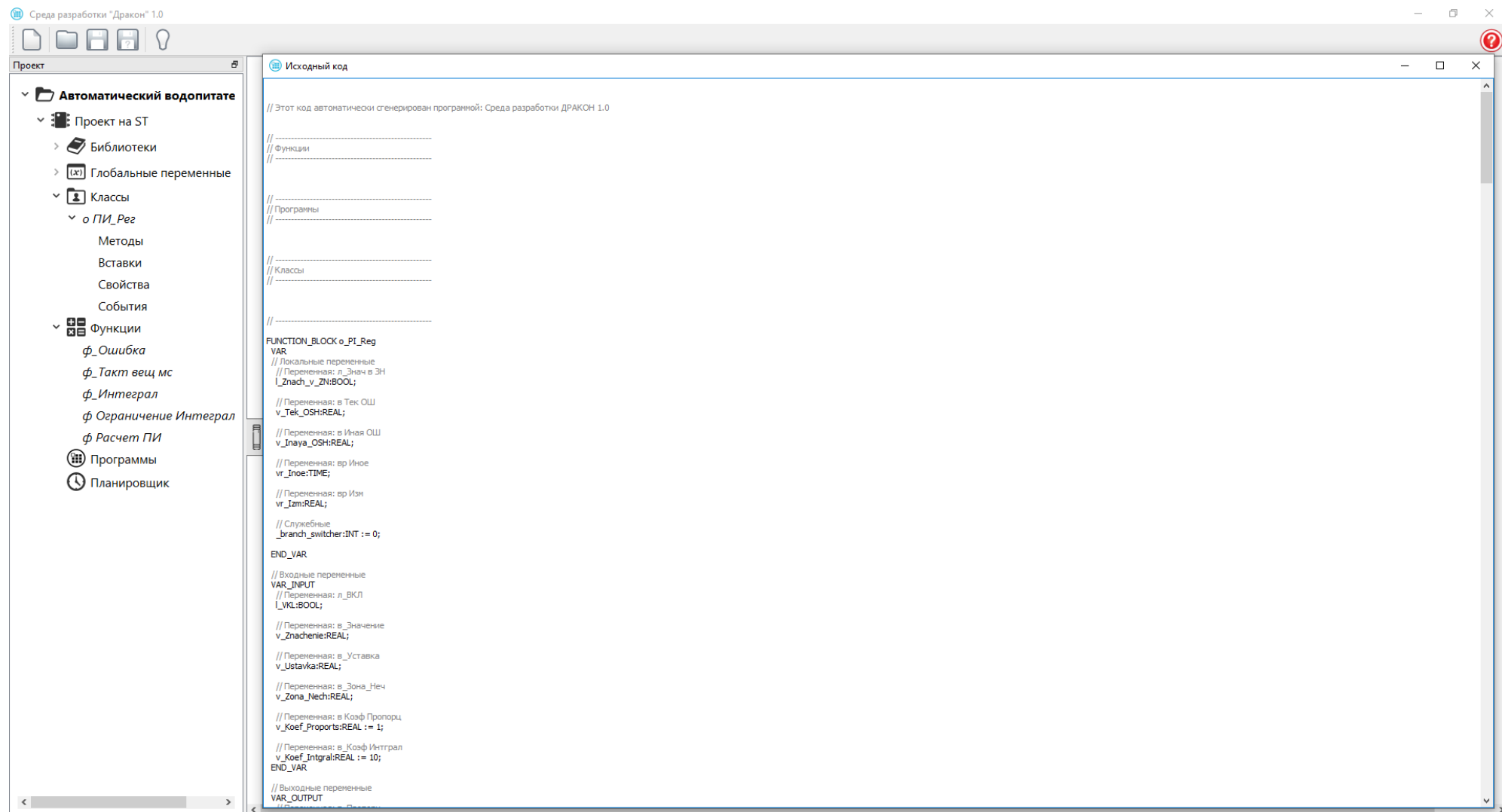
#### 4. После завершения работы над классом (Процедурой) экспортирую проект



В текущей версии можно экспортировать как весь проект, так и отдельные модули



Экспортируется в виде сплошного текста в отдельное окно



На сегодняшний день перенос кода в другую программу осуществляется копированием кода, я буду переносить в кодесис 3.5.17

- ..... f\_Integral (FUN)
- ..... f\_Ogranichenie\_Integrala (FUN)
- ..... f\_Oshibka (FUN)
- ..... f\_Raschet\_PI (FUN)
- ..... f\_Takt\_veshch\_ms (FUN)
- ..... o\_PI\_Reg (FB)

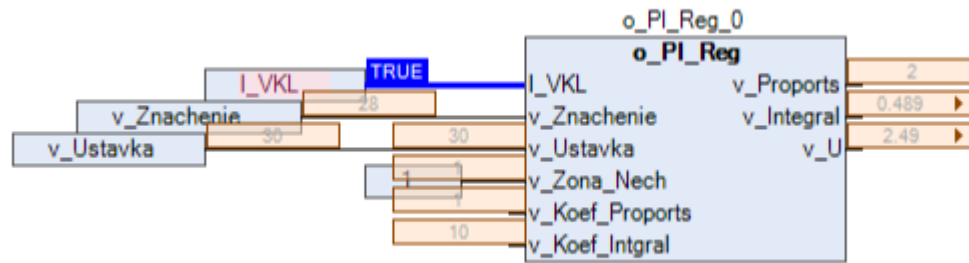


```

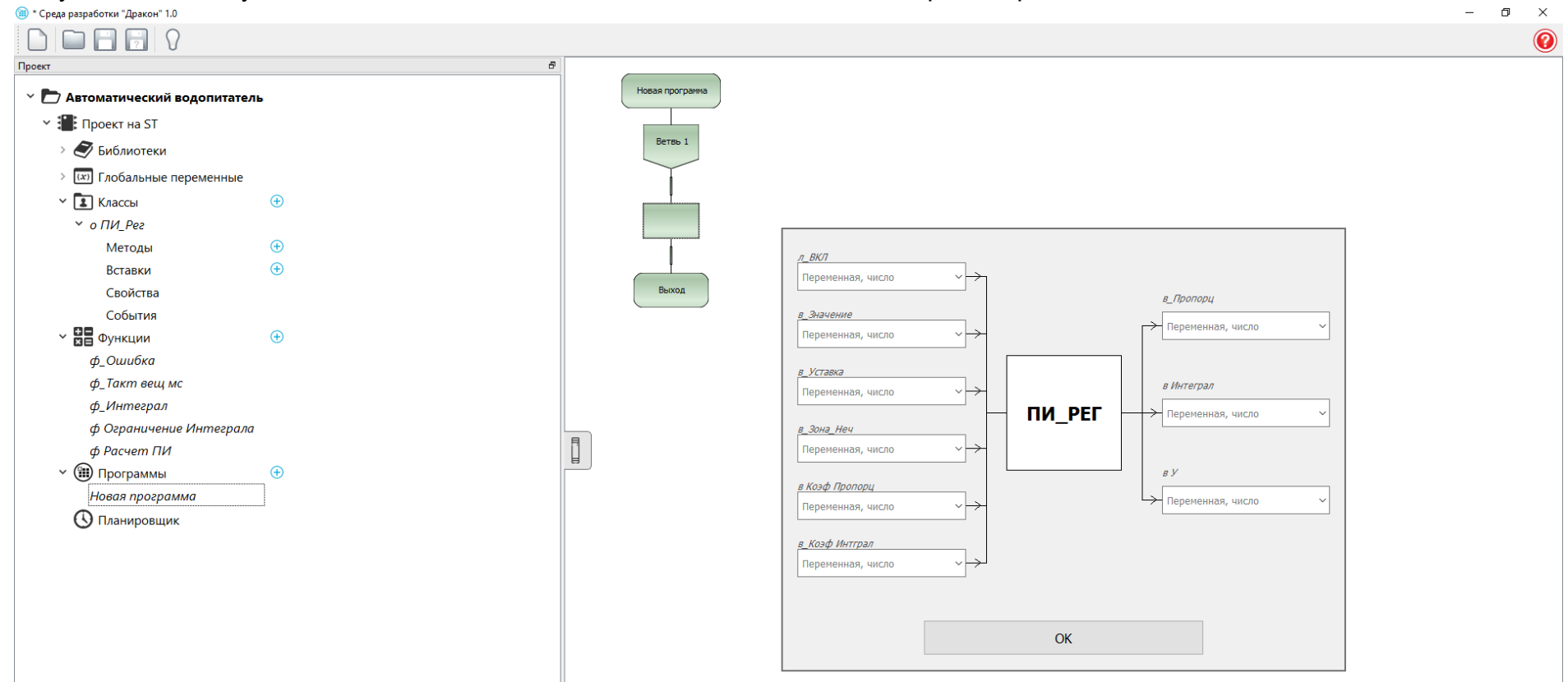
1 FUNCTION_BLOCK o_PI_Reg
2   VAR
3     // Локальные переменные
4     // Переменная: л_Знач в ЗН
5     l_Znach_v_ZN:BOOL;
6
7     // Переменная: в Тек ОШ
8     v_Tek_OSH:REAL;
9
10    // Переменная: в Иная ОШ
11    v_Inaya_OSH:REAL;
12
13    // Переменная: вр Иное
14    vr_Inoe:TIME;
15
16    // Переменная: вр Изм
17
18  _branch_switcher := 0;
19
20  // Алгоритм
21
22  WHILE _branch_switcher >= 0 DO
23    CASE _branch_switcher OF
24      0: // Ветвь 1
25        IF l_VKL
26        THEN
27          ; //
28          // BEGIN ACTION:
29          v_Tek_OSH := f_Oshibka( v_Ustavka, v_Znachenie, v_Zona_Nech );
30          // END ACTION:
31
32          // BEGIN ACTION:
33          vr_Izm := f_Takt_veshch_ms( vr_Inoe );
34          // END ACTION:
35
36          // BEGIN ACTION:
37          v_Integral := f_Integral( v_Koef_Intgral, vr_Izm, v_Integral, v_Tek_OSH, v_Inaya_OSH, v_U );
38          // END ACTION:
39
40          // BEGIN ACTION:
41          v_Inaya_OSH := v_Tek_OSH;
42          // END ACTION:
43
44          // BEGIN ACTION:
45          v_U := f_Raschet_PI( v_Koef_Proports, v_Tek_OSH, v_Integral, v_Proports );
46          // END ACTION:
47
48        ELSE

```

Далее компилирую в кодесис код и проверяю работу в режиме эмуляции:



Результат достигнут, имеем необходимый блок для создания более обширного проекта



Соответственно можно объявить экземпляры в другом классе или программе, и использовать его там.

