МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«МИРЭА – Российский технологический университет»**

**РТУ МИРЭА**

---

Институт искусственного интеллекта (ИИИ)
Кафедра проблем управления

# ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 3

**по дисциплине «Программное обеспечение мехатронных и робототехнических систем»**


Выполнили студенты группы КРБО-03-22: Миргазев М. А.
Попов Е. О.
Рамазанов Р. И.

Проверил доцент кафедры ПУ: Морозов А. А.


Москва 2025

**Цель работы:** получение навыков разработки программного обеспечения системы управления одной степенью подвижности учебного робота УРТК на базе технологий B&R Automation Studio и библиотеки Acp10sdc, включая создание оси, настройку параметров привода, конфигурирование конечного автомата управления и выполнение симуляции и экспериментов на реальном стенде.

**Задание:** требуется разработать функциональный блок, основанный на библиотеке Acp10sdc, предназначенный для управления одной осью УРТК.

Функционал должен включать:

- обработку концевых датчиков;
- выполнение процедуры реферирования к датчику, расположенному со стороны двигателя;
- переход оси в состояние «отключено» после успешного реферирования;
- возможность управления из среды тестирования или от кнопок стенда.

## ОСНОВНАЯ ЧАСТЬ

### Описание действий и используемого аппаратного обеспечения

В ходе работы был создан проект в среде B&R Automation Studio. При создании проекта учитывались требования лабораторного стенда УРТК. В конфигурацию симуляции были добавлены необходимые модули и интерфейсы контроллера:

- 5LS182.6-1
- X20BC0083
- X20MM4456
- X20DI9371
- X20D09322
- 80VD100PS.C02X-01

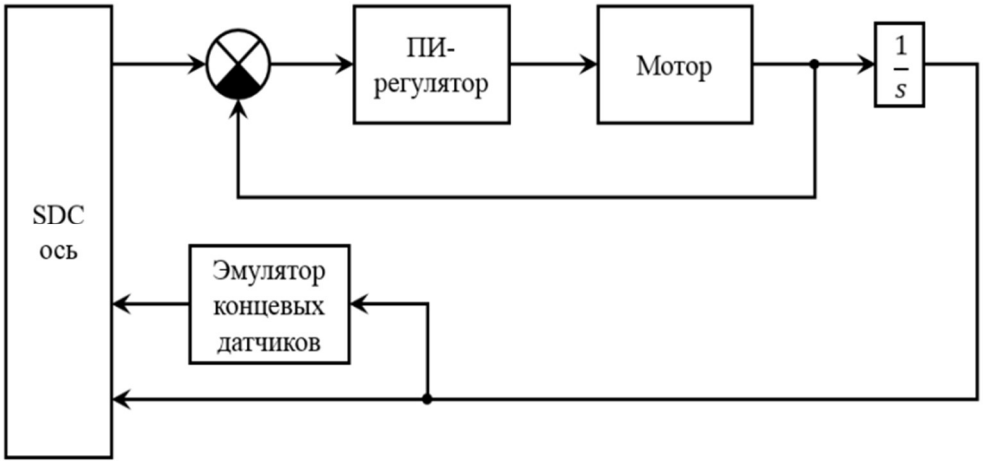


Рисунок 1 – Структурная схема стенда и аппаратной конфигурации

После создания проекта были добавлены необходимые библиотеки для обеспечения корректной работы SDC-оси:

- Acp10sdc
- Acp10man
- Acp10par
- Acp10_MC
- Acp10sim
- AsIOTime

В разделе Global Variables были автоматически сформированы структуры, необходимые для работы SDC-оси. На их основе была создана отдельная ось Axis_X, определяющая аппаратную конфигурацию, интерфейсы привода, энкодера и цифровых входов.

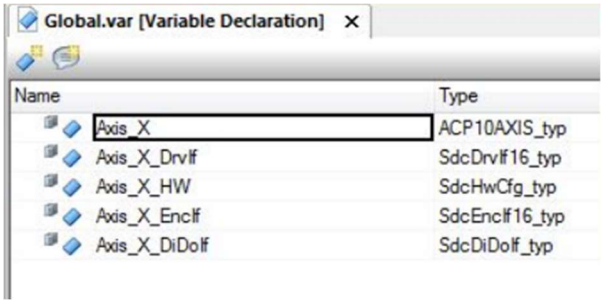


Рисунок 2 – Глобальные переменные оси X

Для оси в проект добавлен модуль инициализации ACP10 Axis, где были заданы параметры, соответствующие реальным характеристикам оси УРТК: состояния концевых выключателей, ускорения, замедления, коэффициенты регулятора, параметры реферирования.

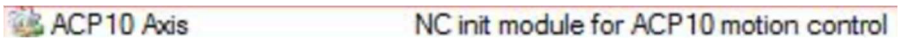


Рисунок 3 – Параметры инициализации оси X

Затем была создана таблица параметров ACOPOS (ACOPOS Parameter table) и заполнена ключевыми значениями: максимальная скорость, число импульсов на оборот датчика и т. д.

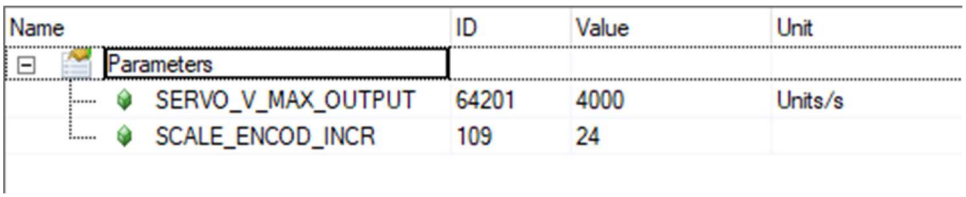


Рисунок 4 – Параметры оси

Далее в реальной конфигурации (Configuration View) выполнен мэппинг SDC-оси в таблице NC Mapping Table. Параметры мэпинга представлены на Рисунке ниже.

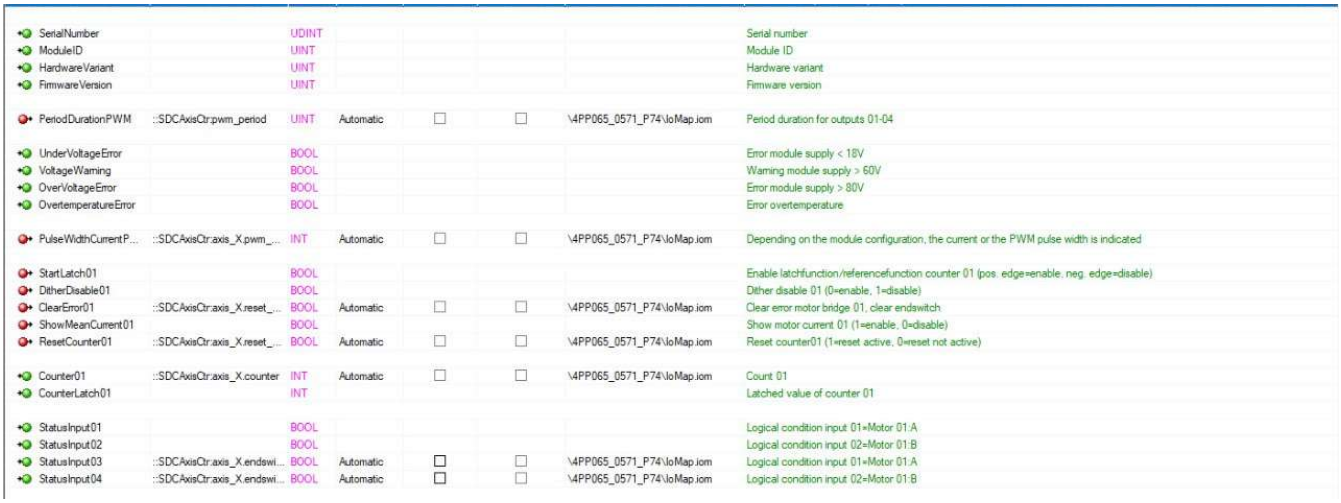


Рисунок 5 – Мэппинг оси X

После конфигурирования оси была выполнена программная часть, включающая разработку собственных функциональных блоков и конечного автомата управления.

### Структура конечных автоматов программного обеспечения

Программное обеспечение управления осью УРТК основано на трёх независимых конечных автоматах, каждый из которых реализует свою часть алгоритма. В этом пункте рассматривается именно структурная организация автоматов как формальных моделей, то есть набор их состояний и последовательность переходов между ними. На данном уровне описания внимание сосредоточено не на содержании логики, а на том, как автомат устроен как структура.

Конечный автомат функционального блока FB_power включает четыре состояния, объединённые в последовательность подготовки оси к работе. Его структура формирует линейную цепочку переходов от исходного состояния ожидания, далее к состоянию включения питания, затем к реферированию и, наконец, к завершению выполнения. Переходы между состояниями предопределены и не допускают возврата назад, что делает автомат строго последовательным по своей топологии.

Автомат FB_move построен как компактная трёхсоставная модель. Он начинается со структурного состояния Idle_move, затем переходит в состояние Move, выполняющее движение, и завершает свой структурный цикл состоянием Done_move. После достижения конечной точки он возвращается в начальное состояние, что обеспечивает циклический характер структуры автомата.

Самым развитым по структуре является автомат функционального блока FB_Trajectory. Он содержит девять состояний, образующих разветвлённую последовательность. Первые два состояния отвечают за структурную подготовку двух осей, после чего автомат проходит через четыре последовательно соединённых состояния, соответствующих этапам движения по траектории. Затем

он переходит в состояние остановки и отключения питания и завершает цикл состоянием фиксации завершения.

**Функциональное содержание состояний конечных автоматов**

Состояние Idle в автомате FB_power представляет собой момент полного бездействия, когда функциональный блок не воздействует на привод и лишь ожидает поступления команды на запуск. Переход в состояние Power означает, что функциональный блок приступает к включению питания оси: активируется блок MC_Power, привод проверяет свою готовность, и после подтверждения автомат передаёт управление в состояние Home. В состоянии Home осуществляется процедура реферирования, привод движется к концевому датчику, фиксирует базовую позицию, и при успешном выполнении операции автомат переходит в состояние Done, которое служит для вывода системы в корректное завершающее состояние и подготовки автомата к следующему циклу.

Состояние Idle_move в автомате FB_move отражает момент, когда движение не инициировано, и ось находится в неподвижном состоянии. Вход в состояние Move приводит к формированию команды перемещения: задаются параметры длины, скорости, ускорения и вызывается MC_MoveAdditive. Состояние Done_move служит для фиксации завершения выполнения команды, после чего автомат сбрасывает временные параметры и возвращается в исходное положение.

В автомате FB_Trajectory состояние Idle_Trajectory является точкой, из которой начинается полный цикл выполнения составной траектории. Состояние PowerY инициирует включение питания оси Y, а состояние PowerZ — включение оси Z. Эти состояния служат для подготовки системы к движению. Состояния side1, side2, side3 и side4 выполняют логически однородную функцию: каждое из них инициирует одно перемещение, соответствующее одной из сторон траектории. В них происходит вычисление параметров перемещения, вызов FB_move и ожидание завершения движения. После выполнения всех четырёх этапов движение прекращается, и автомат переходит в состояние Stop_Trajectory, в котором питание

обоих приводов отключается. Завершает работу автомата состояние Done_Trajectory, фиксирующее окончание выполнения траектории и возвращающее систему в состояние ожидания.

## Описание структуры программного обеспечения

Структура программного обеспечения представляет собой иерархическую архитектуру, построенную по принципу разделения обязанностей между уровнями управления. На базовом уровне находятся объекты SDC-оси, которые отвечают за взаимодействие с аппаратными модулями ACOPOS. На этом уровне формируются аппаратные параметры, назначаются типы интерфейсов, определяются таблицы параметров и производится аппаратное привязывание сигналов, связанных с энкодером, цифровыми входами и драйвером.

Над базовым уровнем располагается уровень функциональных блоков, реализующих конкретные алгоритмы управления. Здесь находится логика конечных автоматов, обеспечивающих питание, реферирование, движение и выполнение траектории. Каждый блок может использовать другие блоки более низкого уровня, передавая им команды и получая обратную информацию о выполнении операций.

На самом верхнем уровне структурируется программный модуль User_Interface, выполняющий роль координационного центра. Он объединяет работу всех функциональных блоков и связывает их с внешними командами, поступающими от пользователя или от инструментов Automation Studio, таких как Test и Watch. Именно через него осуществляется запуск алгоритмов, передача значений перемещений и сбор обратных данных о состоянии системы.

## Результаты экспериментальных исследований

Проверка работоспособности разработанного программного обеспечения проводилась на двух уровнях: в среде симуляции Automation Studio и на реальном

стенде УРТК. На этапе симуляции была выполнена инициализация SDC-оси и проверена функциональность всех трёх конечных автоматов. С помощью окна Watch отслеживались внутренняя логика состояний, изменения переменных и корректность переходов.

Особое внимание уделялось процедуре Homing, реализуемой в автомате FB_power. Симуляция показала, что ось корректно подъезжает к концевому датчику, фиксирует достигнутую референсную точку и завершает цикл. Таким же образом был протестирован функциональный блок FB_move, и система успешно достигала заданных позиций без колебаний и ошибок.

На этапе испытаний на реальном стенде УРТК работа системы полностью соответствовала результатам симуляции. Ось успешно проходила через процедуру реферирования, корректно реагировала на нажатие кнопок стенда, а конечные автоматы FB_power, FB_move и FB_Trajectory выполняли переходы точно в соответствии с логикой, заложенной в программное обеспечение. Для более глубокого анализа была использована система графического мониторинга NC Trace, с помощью которой были получены графики по параметрам ParID 111, 112, 113, 114 и 462.

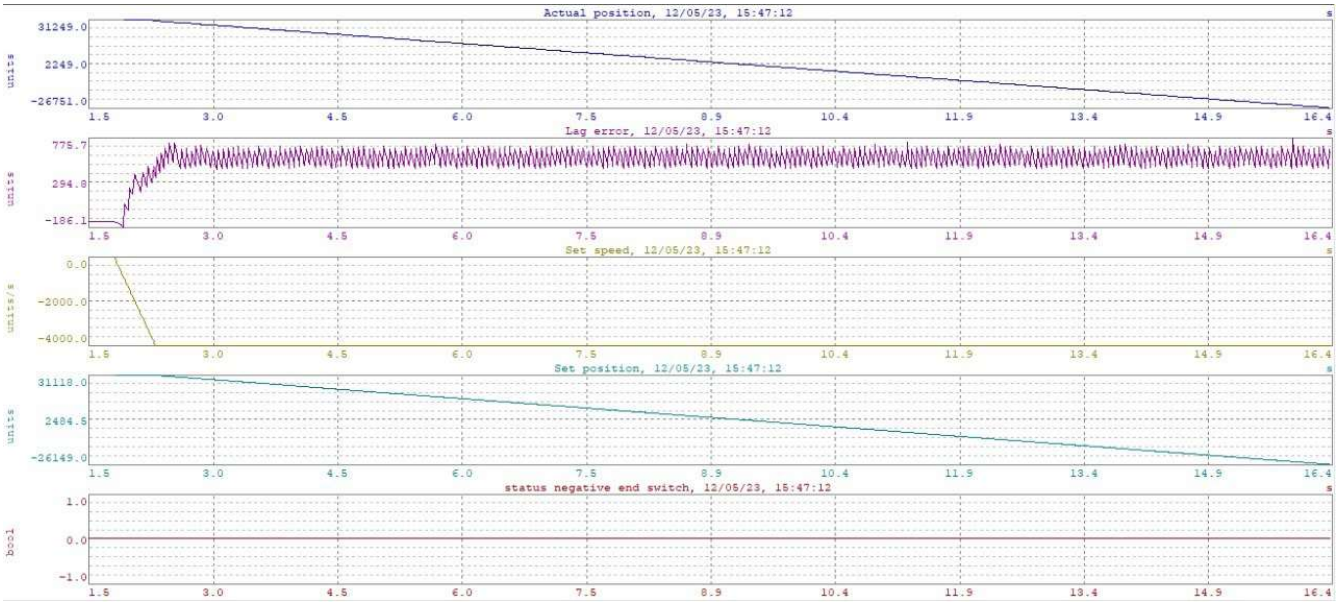


Рисунок 6 – Результат экспериментов

## Вывод

В ходе выполнения лабораторной работы была разработана и всесторонне исследована программная система управления одной степенью подвижности учебного робота УРТК. Проведённая работа позволила на практике освоить полный цикл создания управляющего ПО: от конфигурирования аппаратной части и структуры SDC-оси до разработки функциональных блоков с конечными автоматами и последующего тестирования их поведения в среде симуляции и на реальном стенде. В результате была создана модульная и надёжная система, использующая библиотеку Acp10sdc и основанная на принципах промышленного программирования для приводных систем.

Наиболее важную роль сыграло построение трёх отдельных конечных автоматов, каждый из которых реализует собственный этап управления. FB_power обеспечивает корректную подготовку привода, включая включение питания и выполнение процедуры реферирования. FB_move отвечает за точное и безопасное выполнение одиночных перемещений. FB_Trajectory объединяет оба предыдущих блока и реализует сложные последовательности движения. Благодаря чёткой структуре и детерминированной логике состояний каждый автомат функционирует предсказуемо, что особенно важно при управлении реальным оборудованием.

Проверка работоспособности программного обеспечения показала его устойчивость и корректность. На этапе симуляции были подтверждены правильность переходов между состояниями, соответствие динамики движения расчётным параметрам и отсутствие зависаний или ошибок в логике. Испытания на реальном стенде подтвердили, что система адекватно реагирует на изменения состояния концевых датчиков, корректно выполняет реферирование, плавно осуществляет движения и строго соблюдает последовательность шагов, заложенных в конечные автоматы.

**Приложение А (FB_power)**

```
#include <bur/plctypes.h>
#ifdef __cplusplus
	extern "C"
	{
#endif
	#include "Library.h"
#ifdef __cplusplus
	};
#endif
/* TODO: Add your comment here */
void FB_power(struct FB_power* inst) // Функциональный блок
запуска и референсирования осей
{
	switch (inst->State) // Машина состояний для ФБ
	{
		case Idle: // Состояние покоя
			{
				if(inst->enable)
				{
					inst->done = 0;
					inst->State = Power; // Переход к
состоянию запуска
				}

				break;
			}
		case Power: // Состояние запуска
```

```
            {
                inst->Power.Axis = inst->Axis;
                inst->Power.Enable = 1;

                inst->State = Home; // Переход к состоянию
референсирования

                break;
            }
        case Home: // Состояние референсирования осей
            {
                inst->Home.Axis = inst->Axis;
                inst->Home.Execute = 1;
                inst->Home.HomingMode = 1;
                inst->Home.Position = 0;
                if (inst->Home.Done)
                {
                    inst->State = Done; // Переход к
состоянию завершения
                }

                break;
            }
        case Done: // Состояние завершения работы
            {
                inst->done = 1;
                if(inst->done)
                {
                    inst->Home.Done = 0;
```

```
                        inst->enable = 0;
                        inst->State = Idle; // Переход в
состояние покоя после завершения
                    }

                    break;
                }
    }

    //Инициализация использованных функциональных блоков
    MC_Power(&inst->Power);
    MC_Home(&inst->Home);

}
```

**Приложение Б (FB_move)**

```
#include <bur/plctypes.h>
#ifdef __cplusplus
    extern "C"
    {
#endif
    #include "Library.h"
#ifdef __cplusplus
    };
#endif
/* TODO: Add your comment here */
void FB_move(struct FB_move* inst) // Функциональный блок
движения оси
```

```
{
     switch (inst->State) // Машина состояний
     {
          case Idle_move: // Состояние покоя
               {
                    if(inst->enable)
                    {
                         inst->Move.Done = 0;
                         inst->done = 0;
                         inst->State = Move; // Переход к
состоянию движения
                    }

                    break;
               }
          case Move: // Состояние движение
               {
                    // Передача необходимых параметров в
функциональный блок
                    inst->Move.Axis = inst->Axis;
                    inst->Move.Execute = 1;
                    inst->Move.Distance = inst->length;
                    inst->Move.Velocity = 3000;
                    inst->Move.Acceleration = 6000;
                    inst->Move.Deceleration = 6000;

                    if(inst->Move.Done)
                    {
                         inst->State = Done_move; // Переход к
```

```
состоянию завершения
                    }

                    break;
               }
          case Done_move: // Состояние завершения
               {
                    inst->done = 1;
                    if(inst->done)
                    {
                         inst->Move.Execute = 0;
                         inst->length = 0;
                         inst->enable = 0;
                         inst->State = Idle_move;
                    }

                    break;
               }
     }
     MC_MoveAdditive(&inst->Move); // Инициализация
Функционального блока движения
}
```

**Приложение В (FB_Trajectory)**

```
#include <bur/plctypes.h>
#ifdef __cplusplus
extern "C"
{
```

```
#endif
#include "Library.h"
#ifdef __cplusplus
};
#endif
/* TODO: Add your comment here */
void FB_Trajectory(struct FB_Trajectory* inst) // Функциональный
блок управления УРТК
{
switch (inst->State) // Машина состояний
{
     case Idle_Trajectory: // Состояние покоя
          {
               if (inst->enable)
               {
                    inst->done = 0;
                    inst->State = PowerY; // Переход к запуску
оси Y
               }
               break;
          }
     case PowerY : // Запуск оси Y
          {
               inst->Power_AxisY.Axis = inst->AxisY; //
Объявление оси Y
               inst->Power_AxisY.enable = 1;
               if(inst->Power_AxisY.done)
               {
                    inst->State = PowerZ; // Переход к запуску
```

```
оси Z
                }
                break;
          }
     case PowerZ :
          {
                inst->Power_AxisZ.Axis = inst->AxisZ; //
Объявление оси Z
                inst->Power_AxisZ.enable = 1;
                if(inst->Power_AxisZ.done)
                {
                     inst->State = side1; // Переход к состоянию
движения по траектории 1-й стороны
                }
                break;
          }
     case side1 : // Состояние движения по траектории 1-й
стороны
          {
                inst->Move_Axis.Axis = inst->AxisY;
                inst->Move_Axis.length = inst->length*100;
                inst->Move_Axis.enable = 1;
                if(inst->Move_Axis.done)
                {
                     inst->State = side2; // Переход к состоянию
движения по траектории 2-й стороны
                }
                break;
          }
```

```
    case side2 : // Состояние движения по траектории 2-й
стороны
         {
              inst->Move_Axis.Axis = inst->AxisZ;
              inst->Move_Axis.length = inst->length*100;
              inst->Move_Axis.enable = 1;
              if(inst->Move_Axis.done)
              {
                   inst->State = side3; // Переход к состоянию
движения по траектории 3-й стороны
              }
              break;
         }
    case side3 : // Состояние движения по траектории 3-й
стороны
         {
              inst->Move_Axis.Axis = inst->AxisY;
              inst->Move_Axis.length = -inst->length*100;
              inst->Move_Axis.enable = 1;
              if(inst->Move_Axis.done)
              {
                   inst->State = side4; // Переход к состоянию
движения по траектории 4-й стороны
              }
              break;
         }
    case side4 : // Состояние движения по траектории 4-й
стороны
         {
```

```
            inst->Move_Axis.Axis = inst->AxisZ;
            inst->Move_Axis.length = -inst->length*100;
            inst->Move_Axis.enable = 1;
            if(inst->Move_Axis.done)
            {
                inst->State = Stop_Trajectory;
            }
            break;
        }
    case Stop_Trajectory : // Состояние остановки и выключения
УРТК
        {
            inst->Power_AxisY.Power.Enable = 0;
            inst->Power_AxisZ.Power.Enable = 0;
            if (!inst->Power_AxisY.Power.Status && !inst-
>Power_AxisZ.Power.Status)
            {
                inst->State = Done_Trajectory;
            }
        }
    case Done_Trajectory : // Состояние завершения работы ФБ
        {
            inst->done = 1;
            if(inst->done)
            {
                inst->Move_Axis.done = 0;
                inst->Move_Axis.done = 0;
                inst->enable = 0;
                inst->State = Idle_Trajectory;
```

```
            }
            break;
        }
}
// Инициализация использованных функциональных блоков
FB_move(&inst->Move_Axis);
FB_power(&inst->Power_AxisY);
FB_power(&inst->Power_AxisZ);
}
```

**Приложение Г (User_Interface)**

```
#include <bur/plctypes.h>
#ifdef _DEFAULT_INCLUDES
    #include <AsDefault.h>
#endif
void _INIT ProgramInit(void)
{
    // Объявление осей Y и Z по адресу
    Trajectory.AxisY = &AxisY;
    Trajectory.AxisZ = &AxisZ;

    // Мгновенное присваивание значения 0 при инициализации
переменной
    Trajectory.length = 0;
    Move_axis.length = 0;
}

void _CYCLIC ProgramCyclic(void)
{
    FB_Trajectory(&Trajectory); // Инициализация
функционального блока
}

void _EXIT ProgramExit(void)
{
}
```