



МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА
Институт информационных технологий
Кафедра вычислительной техники

ЛАБОРАТОРНАЯ РАБОТА №1
по дисциплине «Программное обеспечение мехатронных и робототехнических систем»

Тема: «Отладка программного обеспечения робототехнических систем с использованием виртуального моделирования »

Студент группы КРБО-03-17.

Зеленский Д.П.

Руководитель:

Морозов А.А. _____

Москва
2020

Цель работы: получение навыков моделирования объекта управления в промышленных системах автоматического управления и создание функциональных блоков. Задание: создать виртуальную систему управления (рис. 1.1), включающую: модель объекта управления (рис. 1.2), ПИ-регулятор и обратную связь. Передаточная функция

$$W = \frac{1}{k_e \cdot (T_M s + 1)}$$

объекта:

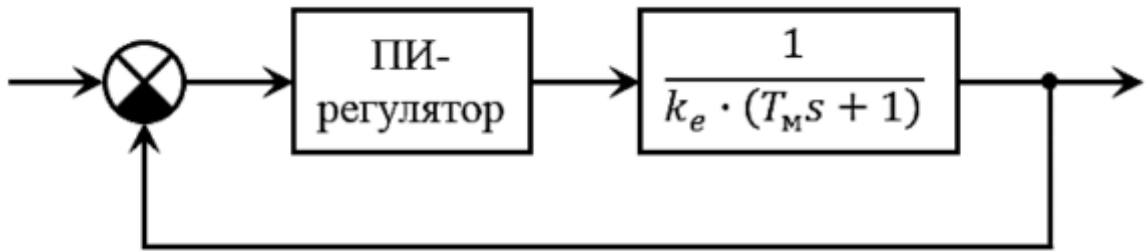


Рис. 1.1 – Структура системы управления

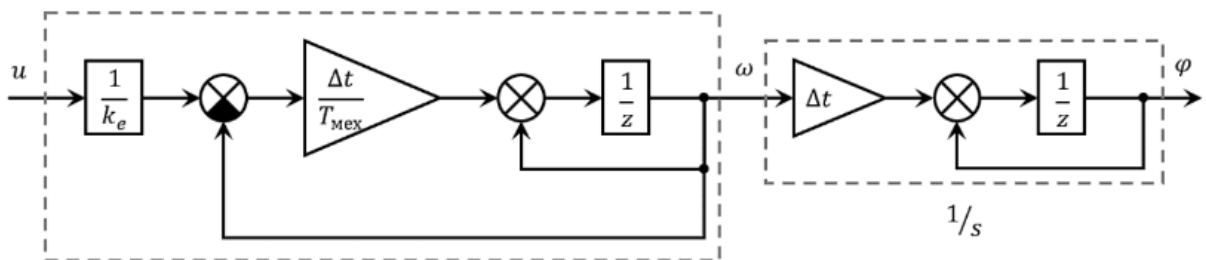


Рис. 1.2 – Структура объекта управления

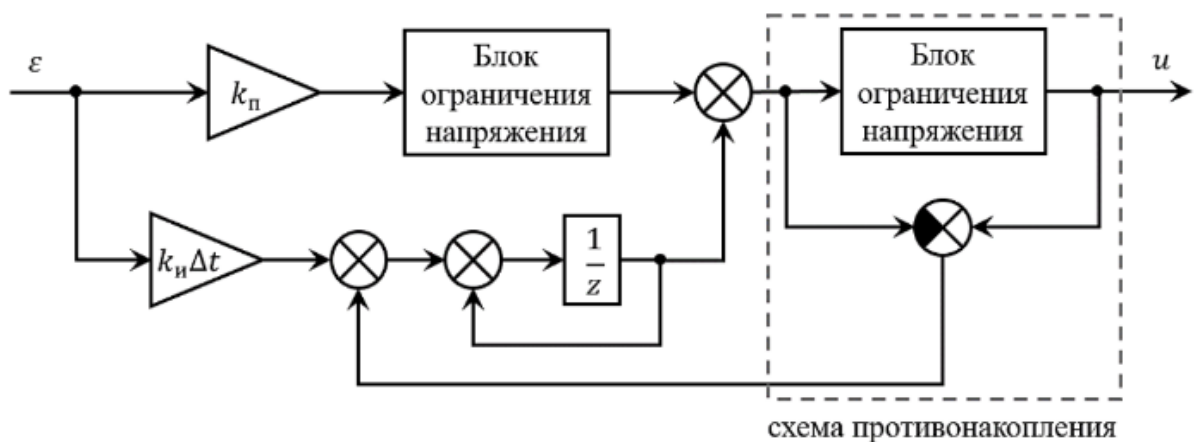


Рис. 1.3 – Структура ПИ-регулятора

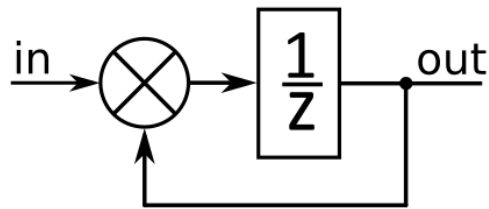


Рис. 1.4. Структура блока интегратора.

Порядок выполнения:

1. Создать проект и добавить в него перечень объектов для реализации требуемых моделей;

2. Создать новый проект в среде Automation Studio без конфигурации оборудования (см. указания 1).

3. Создать в проекте объекты (см. указания 1).:

А) ANSI C Program;

Б) ANSI C Library «MotorControl».

5. В библиотеке создать 3 функциональных блока (см. указания 3) и дать им имена:

А) «FB_Motor», в котором далее реализуем модель двигателя постоянного тока;

Б) «FB_Regulator», выполняющий роль модели ПИ-регулятора.

В) «FB_Integrator», представляющим из себя модель интегрирующего звена.

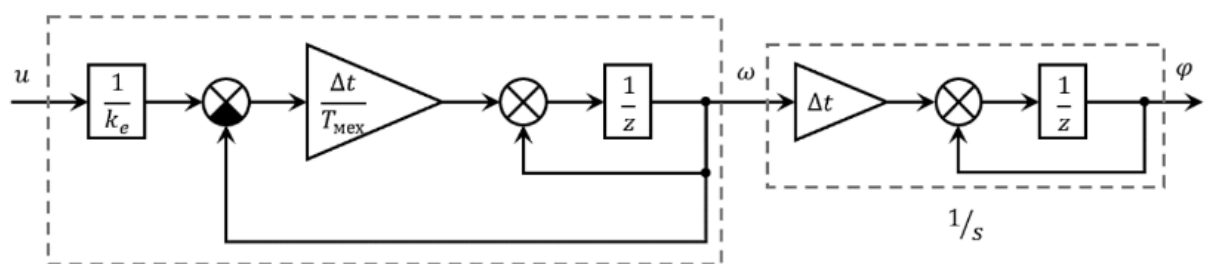


Рис. 2. Структура модели объекта управления.

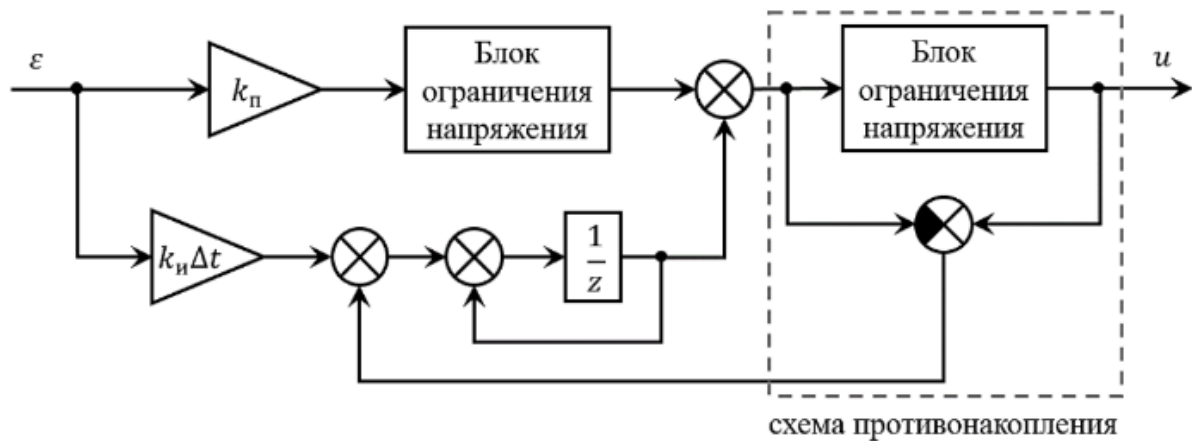


Рис. 3. Структура ПИ – регулятора.

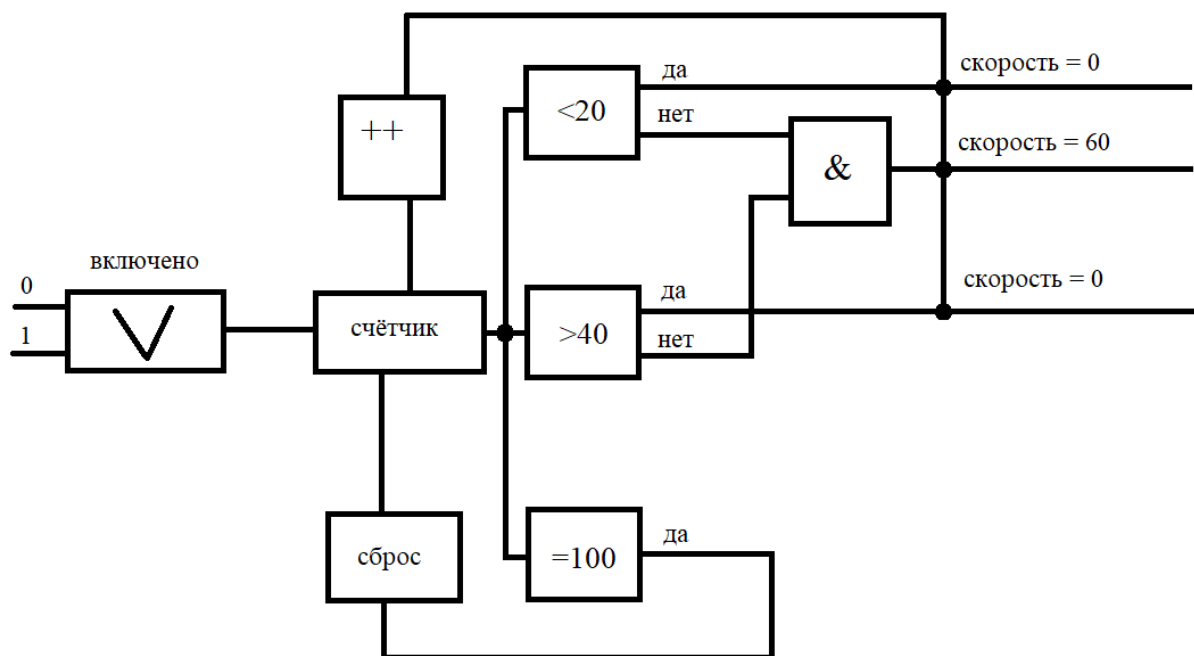


Рис. 4. Структурная схема конечного автомата.

Система имеет два состояния: работа двигателя с регулятором и без регулятора. Класс MotorCon содержит три функциональных блока: FB_Motor, FB_Integrator и FB_Regulator, отвечающие за обработку входного сигнала.

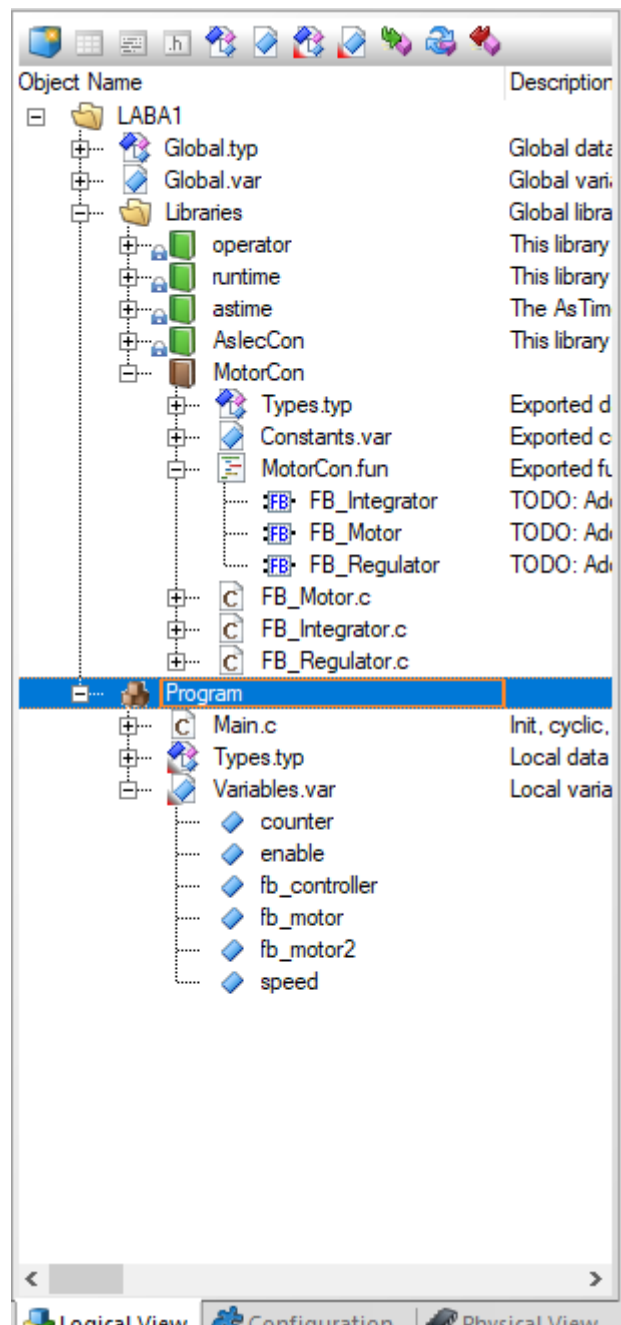


Рис. 5. Иерархическая структура проекта.

MotorCon::MotorCon.fun [Function and Function Block Declaration] x										
C:\MotorCon::FB_Motor.c [Ansi C] C:\MotorCon::FB_Integrator.c [Ansi C] Online Settings C:\Program::Main.c [Ansi C]										
Name	Type	& Reference	Scope	Constant	Retain	Replicable	Redundancy	Value	Desc	
FB_Motor							Unusable		TOD	
u	REAL		VAR_INPUT							
w	REAL		VAR_OUTPUT							
phi	REAL		VAR_OUTPUT							
integrator	FB_Integrator		VAR							
Tm	REAL		VAR							
Ke	REAL		VAR							
dt	REAL		VAR							
FB_Integrator							Unusable		TOD	
in	REAL		VAR_INPUT							
out	REAL		VAR_OUTPUT							
dt	REAL		VAR							
FB_Regulator							Unusable		TOD	
u	REAL		VAR_INPUT							
e	REAL		VAR_OUTPUT							
k_p	REAL		VAR							
k_i	REAL		VAR							
integrator	FB_Integrator		VAR							
iyOld	REAL		VAR							
max_abs_value	REAL		VAR							
dt	REAL		VAR							

Рис. 6. Структура функциональных блоков FB_Motor, FB_Integrator и FB_Regulator.

На входе FB_Motor и FB_Regulator находится переменная **u** – входное напряжение двигателя, а на выходе – две у FB_Motor (угловая скорость **w** (омега) и **phi** – угловое смещение якоря) и **e** – противоЭДС – в регуляторе.

Чтобы снять графики выходных переменных посредством инструмента Trace, необходимо его настроить.

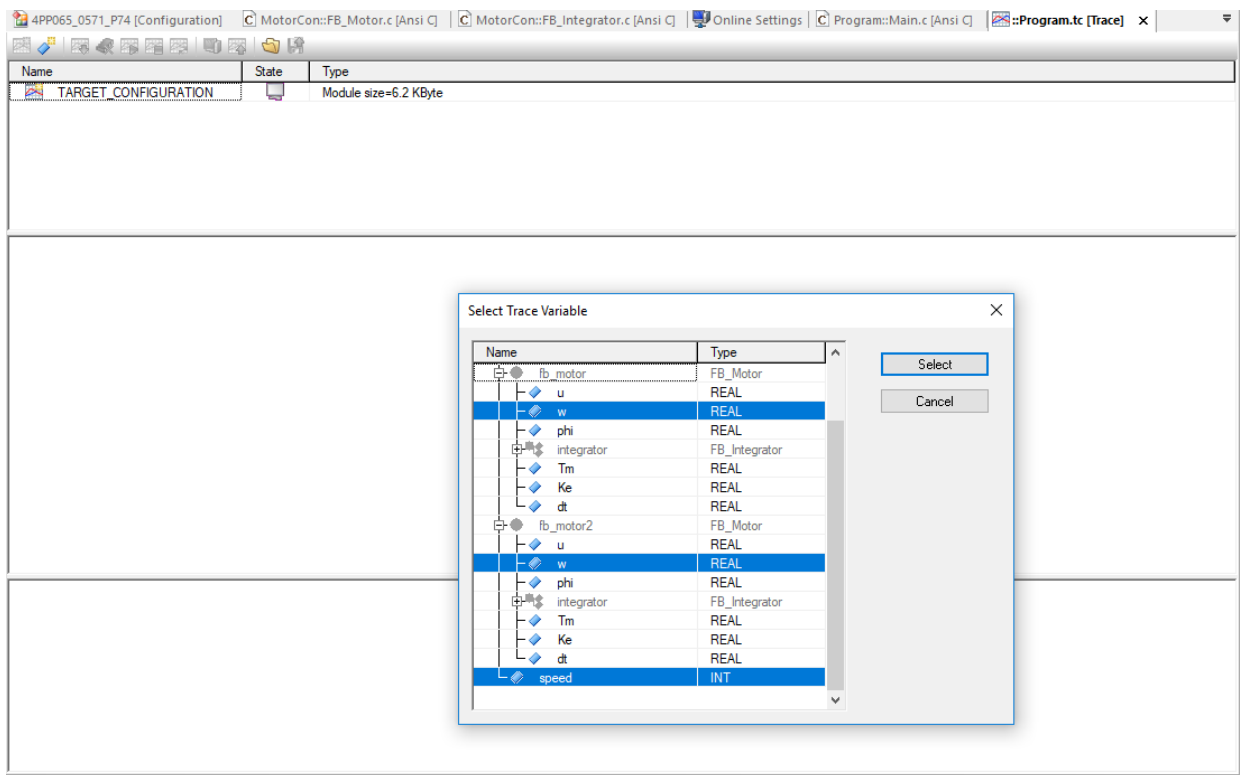


Рис. 7. Настройка вводных переменных для инструмента Trace.

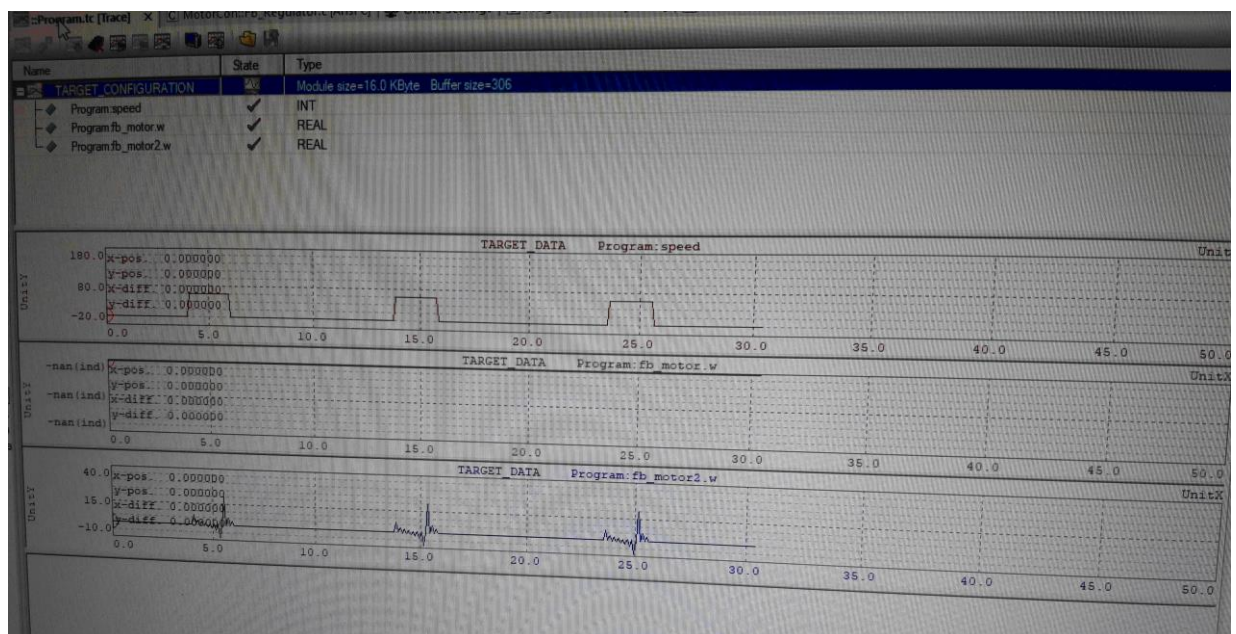


Рис. 8. Результаты экспериментальных исследований.

Вывод: в ходе проведения лабораторной работы было проведено ознакомление со способами отладки программного обеспечения робототехнических систем с использованием виртуального моделирования в программе Automation Studio.

Приложение 1. Листинг программного кода.

Program.Main:

```
#include <bur/plctypes.h>
#ifdef _DEFAULT_INCLUDES
    #include <AsDefault.h>
#endif
void _INIT ProgramInit(void)
{
    fb_motor.Tm = 0.1;
    fb_motor.u = 0;
    fb_motor.integrator.dt = 0.01;

    fb_motor2.dt = 0.01;
    fb_motor2.Ke = 0.24;
    fb_motor2.Tm = 0.1;
    fb_motor2.u = 0;
    fb_motor2.integrator.dt = 0.01;

    fb_controller.dt = 0.01;
    fb_controller.integrator.dt = 0.01;
    fb_controller.k_i = 7.1;
    fb_controller.k_p = 0.24 * 3;
    fb_controller.max_abs_value = 14.0;

    enable = 1;
    speed = 0;
    counter = 0;
}
void _CYCLIC ProgramCyclic(void)
{
    if (enable)
    {
        if (counter < 20)
        {
            speed = 0;
            counter++;
        }
        if (counter >= 20 && counter <= 40)
        {
            speed = 60;
            counter++;
        }
        if (counter > 40)
        {
            speed = 0;
            counter++;
        }
        if (counter == 100)
        {
            counter = 0;
        }
    }
}
```



```

    }
    fb_controller.u = speed - fb_motor.w;
    FB_Regulator(&fb_controller);
    fb_motor.u = fb_controller.e;
    fb_motor2.u = speed * fb_motor2.Ke;
    FB_Motor(&fb_motor);
    FB_Motor(&fb_motor2);
}
}
void _EXIT ProgramExit(void)
{
}

```

FB_Motor:

```

#include <bur/plctypes.h>
#ifdef __cplusplus
    extern "C"
    {
#endif
#include "MotorCon.h"
#ifdef __cplusplus
    };
#endif
/* TODO: Add your comment here */
void FB_Motor(struct FB_Motor* inst)
{
    inst->integrator.in = (inst->u / (inst->Ke - inst->w)) * (inst->dt / inst->Tm); //integr's in
    (incoming signal u multiplied by 1/Ke subtracting w then multiplied by dt/Tm)

    //inst->integrator.out = inst->w;

    FB_Integrator(&(inst->integrator));          //calling integr
    inst->w = inst->integrator.out;                //from 1 integr

    inst->integrator.in = inst->w * inst->dt;      //integr's in (incoming signal w multiplied by dt)

    //inst->integrator.out = inst->phi;

    FB_Integrator(&(inst->integrator));          //calling integr

    inst->phi = inst->integrator.out;              //outcoming signal phi
}

```

FB_Integrator:

```
#include <bur/plctypes.h>
#ifdef __cplusplus
    extern "C"
    {
#endif
    #include "MotorCon.h"
#ifdef __cplusplus
    };
#endif
/* TODO: Add your comment here */
void FB_Integrator(struct FB_Integrator* inst)
{
    inst->out = inst->in + inst->dt;
}
```

FB_Regulator:

```
#include <bur/plctypes.h>
#ifdef __cplusplus
    extern "C"
    {
#endif
    #include "MotorCon.h"
#ifdef __cplusplus
    };
#endif
/* TODO: Add your comment here */
void FB_Regulator(struct FB_Regulator* inst)
{
    /*
        inst->integrator.in = inst->e * inst->k_i * inst->dt + inst->iyOld; //integr's in (incoming
        signal e multiplied by ki and dt adding old value)
        FB_Integrator(&(inst->integrator)); //calling integr

        inst->abs_in_1 = inst->e * inst->k_p; //limiter's in

        if(inst->abs_in_1 > inst->max_abs_value || inst->abs_in_1 < -inst->max_abs_value) //if in is
        equal or more than limit
        {
            if(inst->abs_in_1 > 0)
            {
                inst->abs_out_1 = inst->max_abs_value; //than limit the value
            }
            else
            {
                inst->abs_out_1 = -inst->max_abs_value; //than limit the value
            }
        }
    }
}
```

```

else                                //else if in is less than limit
{
    inst->abs_out_1 = inst->abs_in_1;    //than transmit the value to limiter's out
}

inst->abs_in_2 = inst->abs_out_1 + inst->integrator.out; //transmit summ of integral and
proport parts to limiter's in
if(inst->abs_in_2 > inst->max_abs_value || inst->abs_in_2 < -inst->max_abs_value) //if in
is equal or more than limit
{
    if(inst->abs_in_2 > 0)
    {
        inst->u = inst->max_abs_value;    //than limit the value and set to the outcoming
signal u
    }
    else
    {
        inst->u = -inst->max_abs_value;    //than limit the value and set to the outcoming
signal u
    }
}
else                                //else if in is less than limit
{
    inst->u = inst->abs_in_2;    //than transmit the value to limiter's out and set to the
outcoming signal u
}

inst->iyOld = inst->u - inst->abs_in_2;    //saving the current value

*/

inst->integrator.in=inst->e * inst->k_i * inst->dt + inst->iyOld;

FB_Integrator(&(inst->integrator));

inst->k_p=inst->e*inst->k_p;

inst->k_p=(inst->k_p > inst->max_abs_value || inst->k_p < - inst->max_abs_value)?((inst-
>k_p>0)?inst->max_abs_value:-inst->max_abs_value):inst->k_p;

inst->k_p+=inst->integrator.out;

inst->u=(inst->k_p > inst->max_abs_value || inst->k_p < - inst->max_abs_value)?((inst-
>k_p>0)?inst->max_abs_value:-inst->max_abs_value):inst->k_p;

inst->iyOld = inst->u - inst->k_p;
}

```