Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего образования
«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
(ФГАОУ ВО «ЮФУ»)

Институт компьютерных технологий и информационной безопасности

# ЛАБОРАТОРНАЯ РАБОТА №8
# **«Системы с автоматическим управлением»**

по курсу: «Введение в инженерную деятельность»

Выполнил
студент группы КТбо1-7 _____ С. А. Бекезин

Принял
ассистент _____ С. С. Лихтин

Таганрог 2022

**СОДЕРЖАНИЕ**

# ВВЕДЕНИЕ

Целью работы является изучение алгоритмов управления устройствами с использованием обратной связи.

Задачами работы являются:

1) Настроить стабилизатор напряжения.
2) Запрограммировать и настроить PID регулятор;
3) Написать программу для автоматического управления стабилизатором напряжения.

## ОСНОВНАЯ ЧАСТЬ

В лабораторной работе №5 мы уже работали с электродвигателем. Сейчас же мы подключили его к порту Voltage Regulator. Далее используя PID функцию, мы крутили потенциометр, а скорость вращения я мотора менялась. Этого удалось достичь благодаря данному коду

```
#include <iostream>
#include "stm32f4xx.h"
float Reg_P = 0.1; // Коэффициент усиления пропорциональной составляющей
float Reg_I = 0.05; // Коэффициент усиления интегрирующей составляющей
float Reg_D = 0.02; // Коэффициент усиления дифференцирующей составляющей
int mV;int mA;
int mD;

void SetAltFunc(GPIO_TypeDef* Port, int Channel, int AF)
{
 Port->MODER &= ~(3<<(2*Channel)); // Сброс режима
 Port->MODER |= 2<<(2*Channel); // Установка альт. Режима
 if(Channel<8) // Выбор регистра зависит от номера контакта
 {
 Port->AFR[0] &= ~(15<<4*Channel); // Сброс альт. функции
 Port->AFR[0] |= AF<<(4*Channel); // Установка альт. функции
 }
 else
 {
 Port->AFR[1] &= ~(15<<4*(Channel-8)); // Сброс альт. функции
 Port->AFR[1] |= AF<<(4*(Channel-8)); // Установка альт. функции
 }
}

int AnalogRead(int N) // Функция принимает номер канала для преобразования
{
 ADC1->SQR3 = N; // Выбран полученный из аргумента канал
 for(int a=0; a<100; a++) { asm("NOP"); } // Ожидать больше 100 тактов
```

```
 ADC1->CR2 |= ADC_CR2_SWSTART; // Начать преобразование
 while(!(ADC1->SR & ADC_SR_EOC)) { asm("NOP"); } // Ждать установки бита конца операции
 return ADC1->DR; // Вернуть результат преобразования
}




int PID(int Value, int Current) // Функция ПИД регулятора
{
 static int l=0;
 static float i=0;
 float p, d;
 int e;

 e=Value-Current; // Текущая ошибка регулирования

 p=e*Reg_P; // Вычисление пропорциональной составляющей
 i+=e*Reg_I; // Вычисление интегрирующей составляющей с накоплением
 d=(e-l)*Reg_D; // Вычисление дифференцирующей составляющей

 l=e; // сохраняем текущую ошибку регулирования

 return (int)(p+i+d); // Возвращаем управляющее воздействие
}


extern "C" void TIM2_IRQHandler()
{

 TIM2->SR=0;

 mV = AnalogRead(3)*11966/4096; // Значение напряжения на выходе в милливольтах
 mA = AnalogRead(6)*3300/4096; // Значение тока на выходе в миллиамперах

 mD=AnalogRead(13);
```

```
 GPIOF->BSRRL = GPIO_BSRR_BS_11;
 GPIOF->BSRRL = GPIO_BSRR_BS_13;
for(int i=0;i<10000;i++){

}
 GPIOF->BSRRH = GPIO_BSRR_BS_11;
 GPIOF->BSRRH = GPIO_BSRR_BS_13;
for(int i=0;i<10000;i++){

}

TIM12->CCR1=PID(mD,mV);
}


int main()
{
 RCC->AHB1ENR |= RCC_AHB1ENR_GPIOFEN; // Порт I задействован
 GPIOF->MODER &= ~GPIO_MODER_MODER11;
 GPIOF->MODER &= ~GPIO_MODER_MODER13;
 GPIOF->MODER |= GPIO_MODER_MODER11_1;
 GPIOF->MODER |= GPIO_MODER_MODER13_1;
RCC->AHB1ENR |= RCC_AHB1ENR_GPIOHEN; // Порт H задействован
GPIOH->MODER &= ~GPIO_MODER_MODER12; // Сброс режима для PH12
GPIOH->MODER |= GPIO_MODER_MODER12_0; // Установка режима на выход PH12
GPIOH->BSRRL = 1<<12; // Установить значение HIGH (3.3V) для PH12
RCC->AHB1ENR |= RCC_AHB1ENR_GPIOHEN; // Порт H задействован
SetAltFunc(GPIOH, 6, 9); // Установка альт. режима AF9 для TIM12_CH1(PH6)
RCC->APB1ENR |= RCC_APB1ENR_TIM12EN; // Таймер 12 задействован (APB1 x2 = 84МГц)
TIM12->CR2 = TIM_CR2_MMS_0 | TIM_CR2_MMS_1; // Генерация пульса сравнения
TIM12->ARR = 1000 - 1; // Диапазон сравнения 100
TIM12->PSC = (84000000/1000/42000)-1; // Задан делитель на 40кГц
TIM12->CCMR1= TIM_CCMR1_OC1M_1 | TIM_CCMR1_OC1M_2 | TIM_CCMR1_OC1PE; // Режим PWM
1 для CH1
TIM12->CCER = TIM_CCER_CC1E; // Выход CH1 активен
TIM12->CR1 = TIM_CR1_CEN; // Таймер запущен
```

```
RCC->AHB1ENR |= RCC_AHB1ENR_GPIOCEN; // Порт C задействован
GPIOC->MODER &= ~GPIO_MODER_MODER3; // Сброс режима для PC3
GPIOC->MODER |= GPIO_MODER_MODER3; // Аналоговый вход для PC3
RCC->APB2ENR |= RCC_APB2ENR_ADC1EN; // АЦП задействован
RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN; // Порт A задействован
GPIOA->MODER &= ~(GPIO_MODER_MODER3 | GPIO_MODER_MODER6); // Сброс режима для PA3 и PA6
GPIOA->MODER |= GPIO_MODER_MODER3_0 | GPIO_MODER_MODER3_1; // Аналоговый вход PA3
GPIOA->MODER |= GPIO_MODER_MODER6_0 | GPIO_MODER_MODER6_1; // Аналоговый вход PA6
ADC1->CR2 = ADC_CR2_ADON; // АЦП активен
RCC->APB1ENR |= RCC_APB1ENR_TIM2EN; // Задействовать 2 таймер
TIM2->PSC = ((84000000)/100/1000)-1; // Задать делитель на 100Гц
TIM2->ARR = 1000-1; // Задать диапазон переполнения
TIM2->DIER = TIM_DIER_UIE; // Разрешить прерывания по переполнению
TIM2->CR1 = TIM_CR1_CEN; // Запустить таймер
NVIC_SetPriority(TIM2_IRQn, 2); // Задать приоритет прерывания
NVIC_EnableIRQ(TIM2_IRQn); // Прерывание активировано
while(1){}

}
```

Затем в зависимости от положения потенциометра менялась частота мигания светодиодов. В зависимости от коэффициентов усиления пропорциональной и интегрирующей составляющей менялась длительность циклов, отвечающая за работу светодиодов.

## ЗАКЛЮЧЕНИЕ

В ходе этой лабораторной работы мы познакомились с ПИД-регулятором. Полное название – Пропорционально-интегрально-дифференцирующий регулятор. Такое устройство используется в системах автоматического управления. Для его работы необходима обратная связь.