МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра САУ

ПРОЕКТ

по дисциплине «Микроконтроллеры для систем управления»

Тема: разработка макета программно-аппаратного комплекса для управления температурой в печах для варки стекла на основе микроконтроллера ATmega2560

Студент гр. 6492 _____ Огурецкий Д.В.

Преподаватель _____ Голик С.Е.

Санкт-Петербург

2018

ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ

Студент Огурецкий Д.В.
Группа 6492
Тема работы: **разработка макета программно-аппаратного комплекса для управления температурой в печах для варки стекла на основе микроконтроллера ATmega2560**

Цель работы:
Разработать макет системы управления с обратной связью на основе PID регулятора для управления и контроля температуры в камере печи для варки стекла
Решаемые задачи:

1. Установка и поддержание желаемой температуры в печи;
2. Контролируемое изменение температуры по заданной линейной функции.
3. Вывод информации о температуре и цикле нагрева на TFT дисплей
4. Установка входящих параметров для контроля температуры с TFT дисплея.

Аннотация

Разработан макет программно-аппаратного комплекса для управления температурой в печах для варки стекла на основе микроконтроллера ATmega2560. Для контроля и поддержания температуры в камере печи использован PID алгоритм, который управляет симисторной сборкой для подачи киловаттной мощности на рассеивающий тепло элемент печи. Обратная связь для PID алгоритма осуществляется посредством оцифровки сигнала с термопары микросхемой CJMCU-MAX31856. Для ввода/вывода информации был разработан пользовательский интерфейс на основе TFT дисплея.

Оглавление

Введение

Исходя из поставленной задачи была разработана система управления температурой в печке. Она представляет собой несколько устройств, связанных вместе посредством различных интерфейсов, которые передают информацию друг другу и обрабатывают её:

1. Микроконтроллер
2. Модуль термопары
3. Термопара (единственный датчик)
4. Схема управления сетевым напряжением, подаваемым на печку
5. Сенсорный дисплей
6. Печь

Основное устройство — это микроконтроллер, включающий в себя различные периферийные устройства. Он является связующим звеном между остальными устройствами.

Задачи, которые нужно было решить:

1. выбрать подходящее устройство
2. выбрать способы связи между устройствами
3. написать программу для устройств
4. создать реальный прототип.

Для данной работы я использовал знания, приобретенные на предмете: программирование и основы алгоритмизации ,информатика техническая, аналоговая электроника, случайные процессы в САУ, ТАУ, ТОЭ, и основная дисциплина МПУ.

1. Создание комплекса

Выбор устройств и их описание

Система представляет собой несколько устройств, связанных вместе посредством различных интерфейсов, которые передают информацию друг другу и обрабатывают её:

1. Микроконтроллер
2. Модуль термопары
3. Термопара (единственный датчик)
4. Схема управления сетевым напряжением, подаваемым на печку
5. Сенсорный дисплей
6. Макет печи

1. Микроконтроллер

Основное устройство — это микроконтроллер, включающий в себя различные периферийные устройства. Он является связующим звеном между остальными устройствами.

Изначально предполагалось загружать проект на платформу Arduino Nano 3.0 с микроконтроллером ATmega328 , но из-за недостаточного объёма его FLASH-памяти было решено перейти на микроконтроллер ATmega2560 на плате Arduino Mega 2560 R3 .

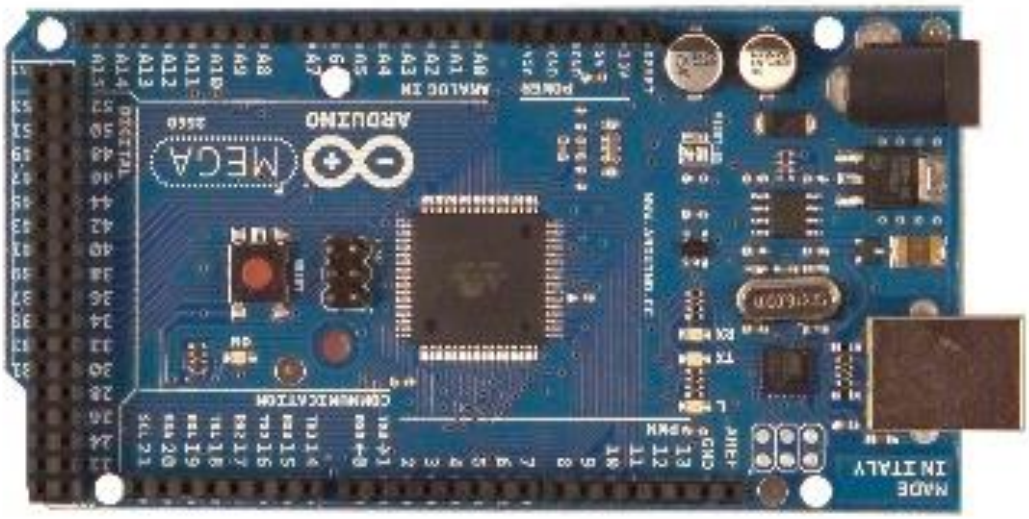


Рис.1а(так выглядит плата Arduino mega2560 с периферийными устройствами)

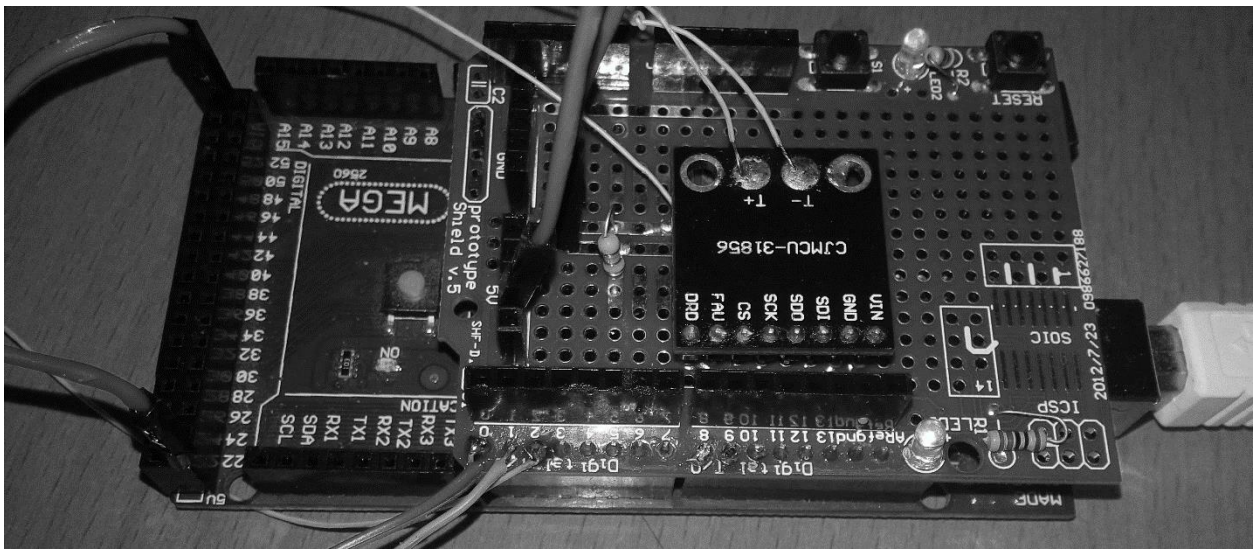


Рис.1б(так выглядит плата Arduino mega2560 с периферийными устройствами и модулем термопары)

2. Модуль термопары

CJMCU-MAX31856 Thermocouple Module High Precision Development Board (Модуль высокоточной обработки термопары) от Maxim Integrated позволяет быстро и несложно реализовать готовый измерительный тракт для работы с термопарами (K, J, N, R, S, T, E и B типов) на одной микросхеме, т.к. содержит в себе все необходимые и достаточные функциональные блоки:

- защиту входных линий;
- цепь усиления и нормирования сигнала;
- АЦП 19-Bit, 0.0078125°C;
- источник опорного напряжения;
- компенсацию холодного спая;
- встроенную линеаризацию.

Можно было возложить выполнение всех этих функций на микроконтроллер, т.к. у него 16 аналоговых входов, каждый из которых может представить аналоговое напряжение в виде 10-битного числа (1024 значений). Разрядность АЦП — 10 бит., но напряжение холодного спая термопары, например типа K, при максимальной температуре 1372 °C равно 54.886 мВ. Необходимо усиливать данное напряжение для его измерения, поэтому в данном случае я решил использовать специальное устройство. Также АЦП имеет большую разрядность 19-Bit, что даёт лучшую точность порядка 0.0078125°C.

Рис.2 (**CJMCU-MAX31856)**

3. Термопара (единственный датчик)

Термопара представлена типа K , её характеристики:

Тип термопары IEC (МЭК)	Материал положительного электрода	Темп. коэффициент, µV/°C	Материал отрицательного электрода	Температурный диапазон °C (длительно)	Температурный диапазон °C (кратковремен)
K	Хромель Cr—Ni	40…41	Алюмель Ni—Al	0 до +1100	−180 до +1300

Максимальное напряжение выхода 52мВ. Соответственно нужно усиливать сигнал, мы это делаем с помощью **модуля высокоточной обработки термопары**.

Рис.3 (термопара)

4. Схема управления сетевым напряжением, подаваемым на печь

Эта схема должна включать и выключать подачу напряжения от сети на нагревательный элемент. Для этого, учитывая , что сетевое напряжение переменное с частотой 50 Гц и с действующим значением $U_д$=220 В, используем симистор, управляемый управляющим сигналом.

Рис.4а(симистор)

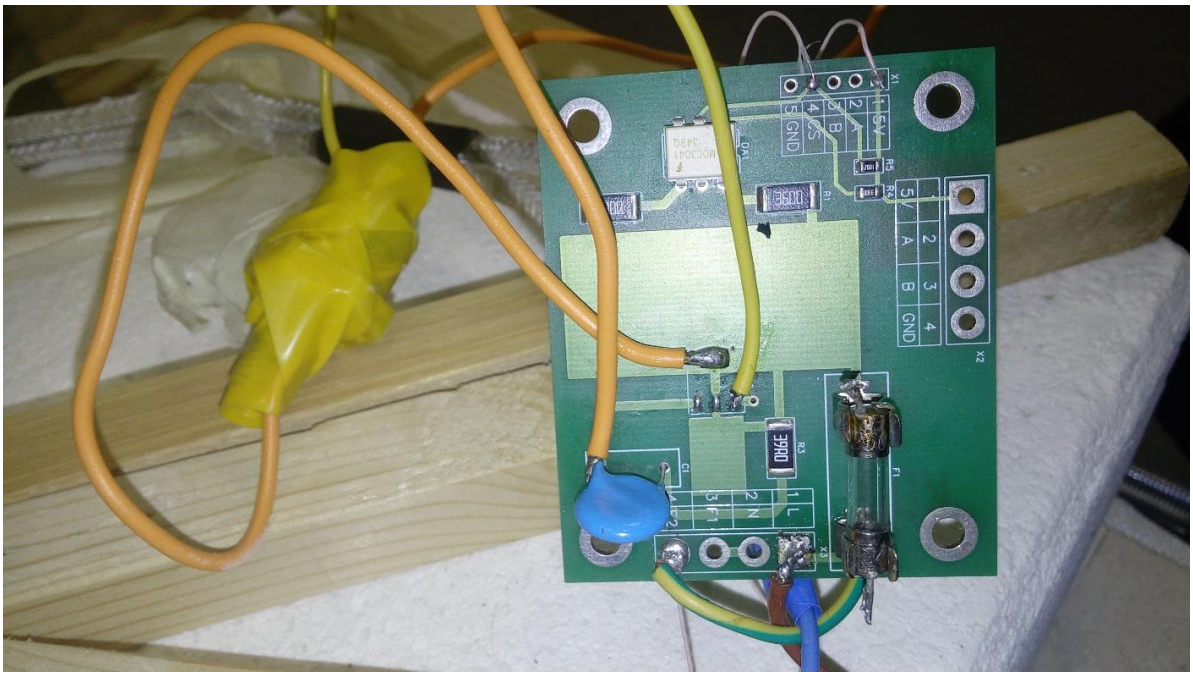


Рис.5(Схема управления сетевым напряжением, подаваемым на печку)

С помощью управляющего сигнала можно открывать и закрывать симистор, и соответственно подачу напряжения на печку. Когда мы подаём лог.1 (5В) симистор закрыт, когда лог.0(0В) открыт.

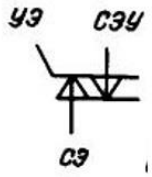


Рис.5,б(УГО симистора)

Поясним работу симистора и тиристора, взяв выдержку из учебника [1].

На рисунке 6 можно видеть, как работает тиристор, симистор отличается лишь тем, что ток может проходить как в прямом , так и в обратном направлении. Включение происходит подачей напряжения на УЭ положительной или отрицательной относительно СЭУ полярности. Выключение происходит при изменение полярности напряжения на основных электродах. [1]

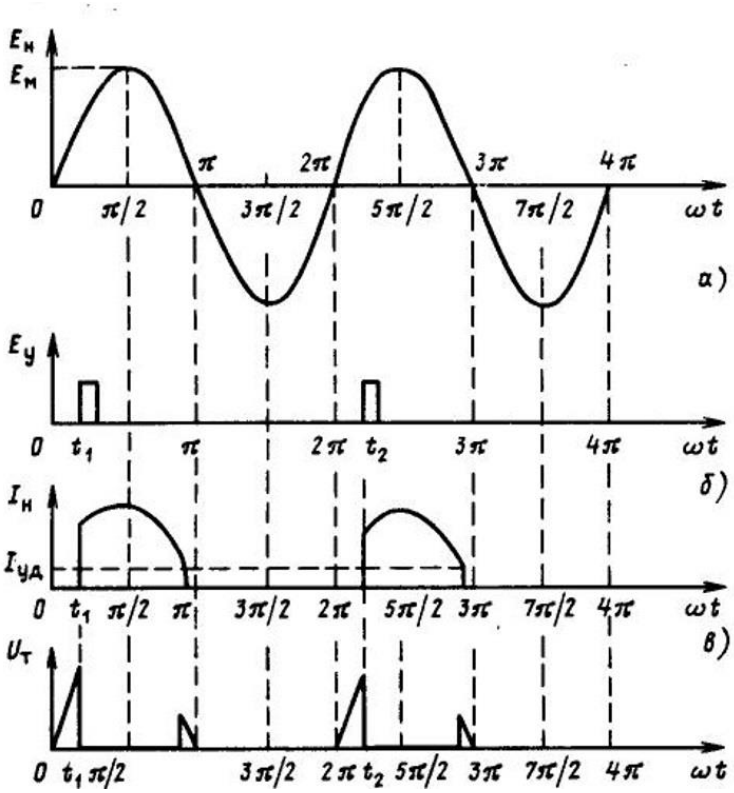


Рис.6(пояснение работы тиристора)

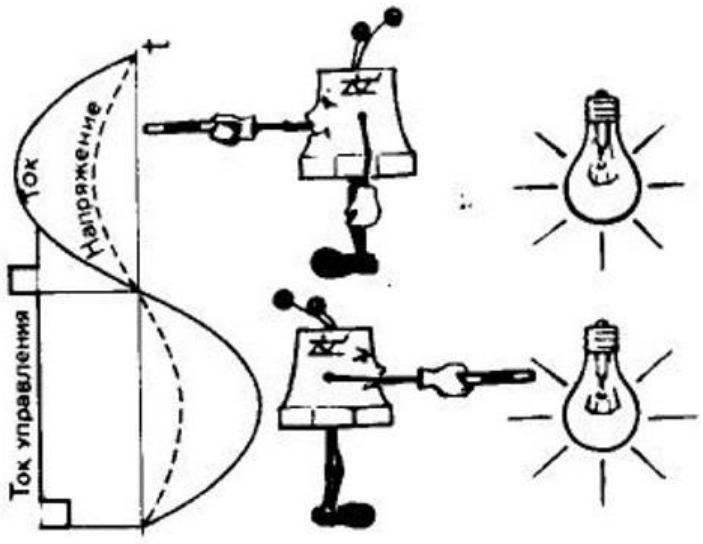


Рис.7(пояснение работы симистора)

В данной работе я использовал симистор ,так как его допустимый ток приблизительно 10 А, а у нас максимальный ток будет

$U_m = \sqrt{2} * 220 = 311\ V$

$R_{НЭ}$=100 Ом — сопротивление нагревательного элемента

$I_{m=}U_m/R_{НЭ}$=311/100=3.11 А следовательно амплитуда тока, протекающего через симистр меньше, чем допустимый ток по справочнику.

5. Сенсорный дисплей

Сенсорный дисплей выступает в роли удобного человеко-машинный интерфейс (HMI), который обеспечивает интерфейс управления и визуализации между человеком и процессом, машиной, приложением или устройством. Также он должен быть легко программируемый и иметь понятную среду разработки, набор команд. Под такие требования подходит экран Nextion NX8048T050 - 5.0" LCD TFT HMI Интеллектуальный Сенсорный Дисплей. Также он имеет сравнительно невысокую цену для дисплея с данными характеристиками и является резистивным сенсорным экраном.

Рис.8(Внешний вид дисплея)

6. Макет печи

Сама печь представляет собой небольшой короб.
Внутри находится нагревательный элемент, представляющий собой провод с малым сопротивлением (≈100 Om) и большой температурой плавления.

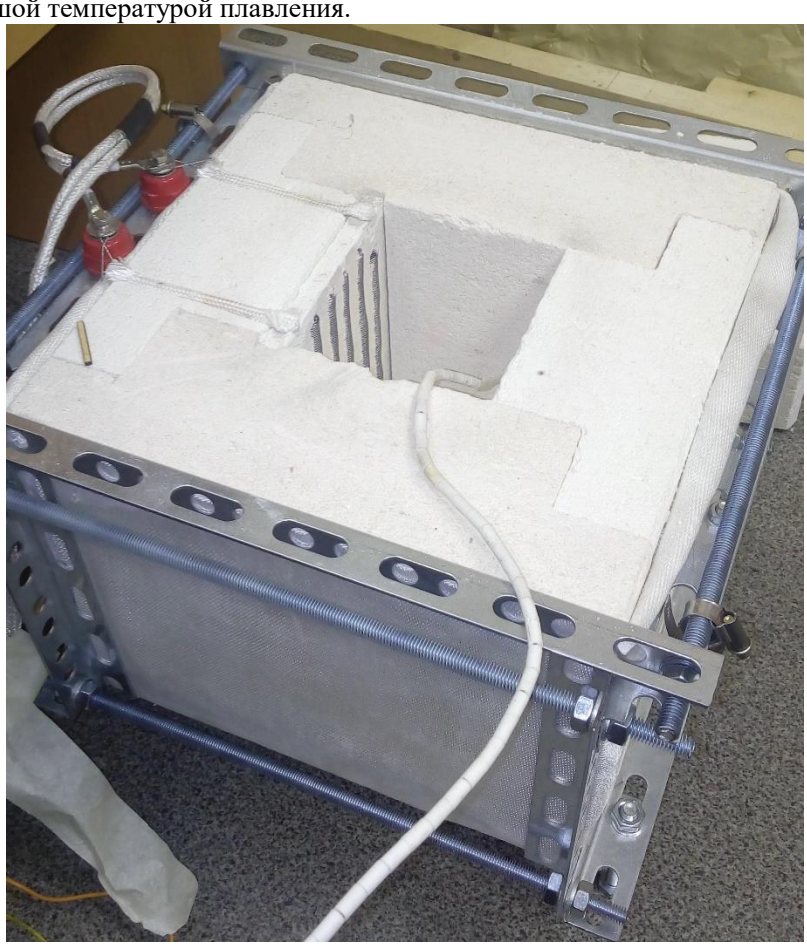


Рис.9а(печь, внешний вид)

Рис.9б(нагревательный элемент)

2. Принципиальная схема устройства

Схема было разработана в бесплатной программе diptrace , в ней были учтены требования ГОСТ насколько позволяет функционал программы.

Перед тем, как составить схему нужно проверить допустимый ток питания для каждого устройства, питание микроконтроллера и дисплея составляет 5 В, а для питания модуля термопары нужно поставить резистор ограничивающий ток , т.к. у него есть допустимый ток питания.

Исходя из этого резистор R1 равен 1,3 кОМ т.е. ток питания 3,8 мА.

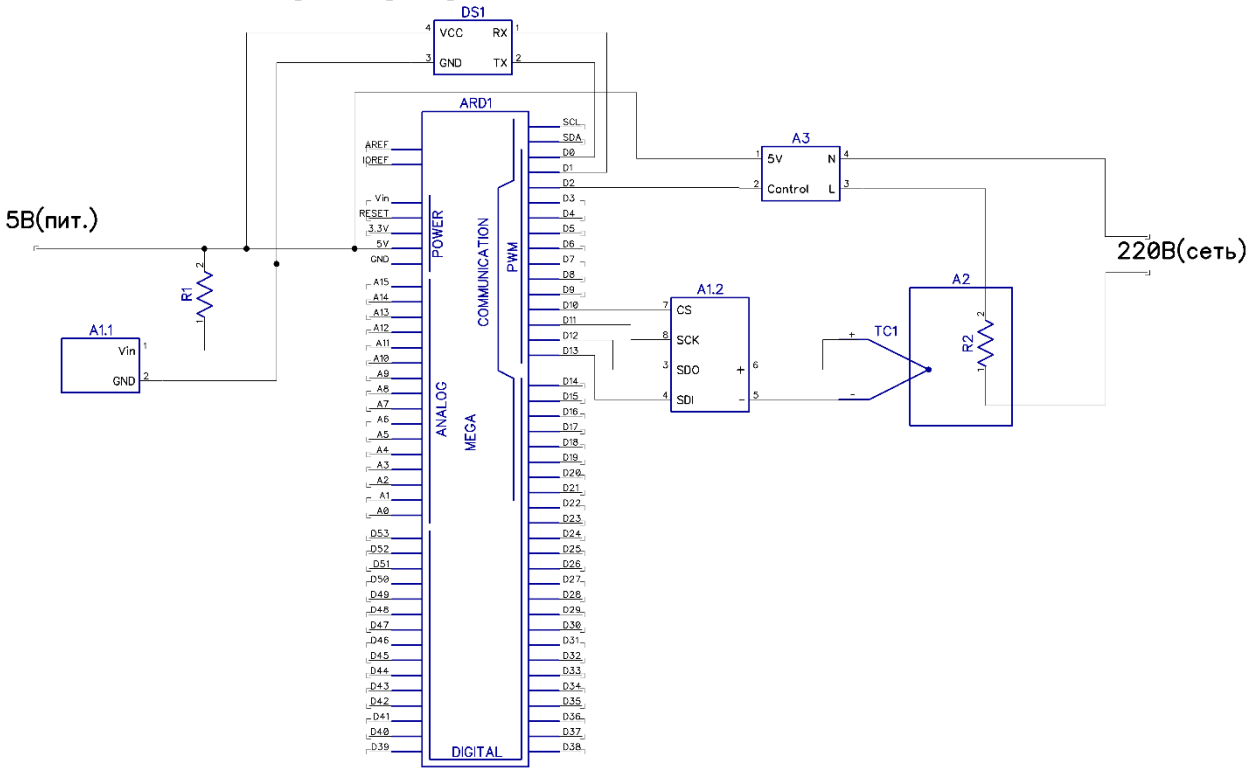


Рис.9(принципиальная схема)

Краткий перечень элементов в табл.1:

Позиционное обозначение	Описание
A1	CJMCU-MAX31856 (Модуль высокоточной обработки термопары)
A2	печь
A3	симисторная сборка
R2	Нагревательный элемент печки(представленный в виде резистора
R1	Резистор ограничивающий ток питания модуля термопары
ARD1	Arduino Mega 2560 R3 с ATmega2560
DS1	Nextion NX8048T050 - 5.0" LCD TFT HMI Интеллектуальный Сенсорный Дисплей
TC1	Термопара типа K

3. Написание кода программы

Программа была написана для микроконтроллера в Visual Studio Code и отлажена в Arduino IDE, а для дисплея в NextionEditor (IDE для написания HMI) [2].
Здесь стоит отметить, что программный код микроконтроллера можно разделить на несколько частей, каждая из которых реализует конкретную задачу. Также я разбил код на два файла, для того, чтобы уменьшить его:

1. Заголовочный файл “stove.h”
2. Файл самой программы “stove.ino”

Имеется файл, для прошивки дисплея “stove.HMI”
Структура программы такая, имеется код, выполняющийся при загрузке MCU (функция void setup()) и код , которые постоянно выполняется в бесконечном цикле (функция void loop ()).
Функция void setup() содержит операции по инициализации всех устройств, необходимые для дальнейшей работы с ними. Её будем рассматривать последовательно , на каждом этапе описание отдельных функциональных блоков кода.
Функция void loop () имеет следующую структуру (рис.10):
Описание назначения функций:

имя функции	Назначение функции
movingAverage	скользящее среднее
myPID.Compute()	ПИ алгоритм, расчёт выходной переменной
checkMAX31856()	обработка оповещений от термоконтроллера
Timer3.setPwmDuty(PIN_OUTPUT, Output)	выставляем скважность выходного сигнала
jumpderivative()	обработке экстренных скачков производной
senddata()	отправка данных на экран
reachingSetpoint()	проверка: достигли ли мы установленного значения?
exitpause()	проверка выполнения операций перед выходом из паузы
changeLinearly()	алгоритм слежения за целевой функцией
readNextioncommand	обработка приходящих данных
checkCommand(String ins)	парсинг сообщение от экрана

4. Флаги программы:

Эти флаги используются для своих целей.
uint8_t flag_pause_exit = 0; //флаг выхода из паузы
uint8_t flag_first_run_interval = 0 ; //флаг(показатель) первого запуска интервала 1-уже запущен 0- не запущен
uint8_t flag_point_switch = 0 ; //разрешение на изменение температуры по заданному алгоритму
uint8_t flag_reaching_Setpoint = 1; //флаг достижение установившего значения
Основные переменные программы:
//точки изменения температуры
uint16_t temp_point[10];   //содержит информацию о температуре в точке
uint16_t time_point[10];//содержит информацию о времени нахождении на интервале точки
float32_t time_step[10] ;//время ,через которое меняется установочная температура (в сек) для каждого участка
uint8_t interval = 0; // номер участка 0-8
uint8_t compare = 0; //больше или меньше температура в начале участка, чем температура в конце участка
//1 температура в начале участка больше 0 - температура в начале участка меньше
uint32_t lastsecond = 0 ;//время крайнего вызова изменения setpoint (такого значения переменной хватит на 50 дней)

uint8_t mistake_id = 0; //номер ошибки

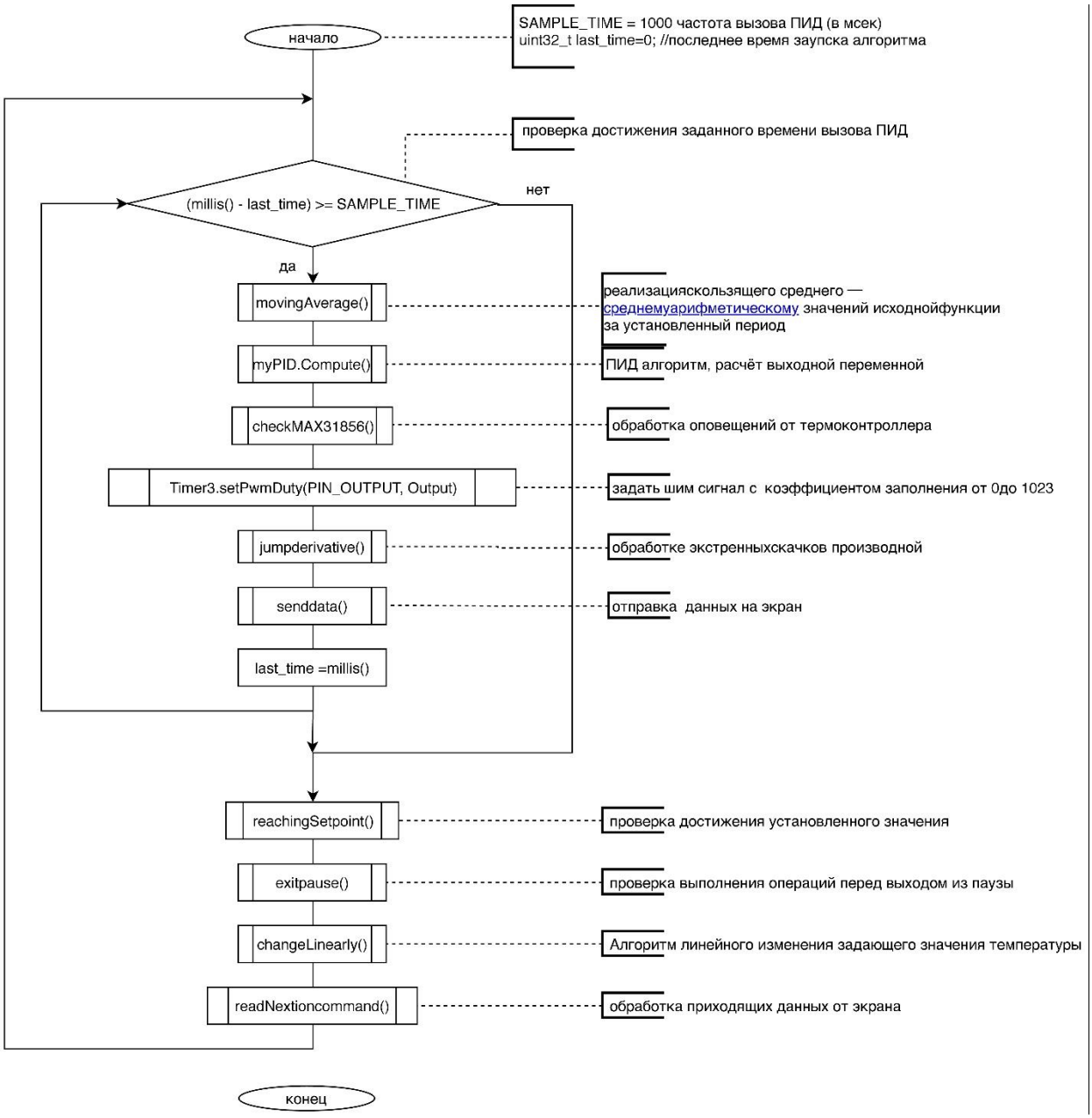


Рис.10(блок-схема функции void loop ())

5. Настройка передачи данных с модуля термопары на MCU

Передача осуществляется по интерфейсу SPI, для её осуществления использована готовая библиотека для работы с CJMCU-MAX31856, в нём реализован программный SPI. Программная реализации более удобна ввиду того, что можно выбирать любые пины в качестве управляющих.

Рассмотрим процесс подключения этой библиотеки (из файла stove.h)

```
#define SCK    11
#define CS     10
#define SDI    13 //
#define SDO    12 //
```

```
#define CR0_INIT  (CR0_AUTOMATIC_CONVERSION + CR0_OPEN_CIRCUIT_FAULT_TYPE_K  +
CR0_NOISE_FILTER_50HZ )
#define CR1_INIT  (CR1_AVERAGE_2_SAMPLES + CR1_THERMOCOUPLE_TYPE_K)
#define MASK_INIT (~(MASK_VOLTAGE_UNDER_OVER_FAULT +
MASK_THERMOCOUPLE_OPEN_FAULT))
MAX31856 *temperature;
```

Здесь происходит определение пинов, управляющих значений для регистров, в которых указывается тип термопары и создание указателя на объект класса.MAX31856.

(из файла stove.ino)

```
// Создание объекта MAX31856 с определением пинов
  temperature = new MAX31856(SDI, SDO, CS, SCK);
  // Инициализация решистров
  temperature->writeRegister(REGISTER_CR0, CR0_INIT);
  temperature->writeRegister(REGISTER_CR1, CR1_INIT);
  temperature->writeRegister(REGISTER_MASK, MASK_INIT);
  // Ожидание отправки всех байтов
  delay(200);
```

6. checkMAX31856()

Функция, реализующая обработка оповещений от термоконтроллера checkMAX31856()

```
void checkMAX31856(void)//--------------------обработка оповещений от термоконтроллера----------------

{

//--------------------обработка оповещений от термоконтроллера-----------------
    if((Input == FAULT_OPEN)&&(flag_point_switch == 1)) // No thermocouple
    {//pause
        mistake_id = 1; //запись номера ошибки
        Output = 1023 ; //выключаем нагрев
        myPID.SetMode(MANUAL); //ручной режим ПИД
        flag_point_switch = 0; //останавливаем работу на  интервале
        t_ALG_last_run =t_ALG_last_run + now() - t_ALG; // сохранение время работы алгоритма
        t_INT_last_run =t_INT_last_run + now() - t_INT; // сохранение время работы интервала
        Serial.print((String)"page FAULT_OPEN"+char(255)+char(255)+char(255)); //страница с
оповещением
        Serial.print((String)"main.bt1.val=1"+char(255)+char(255)+char(255)); //изменение состояния
индикатора паузы
        Serial.print((String)"main.ERROR.en=1"+char(255)+char(255)+char(255)); //включаем мигание экрана
    }
    if((Input == FAULT_VOLTAGE)&&(flag_point_switch == 1)) //  Under/over voltage error.  Wrong
thermocouple type?
    {//pause
        mistake_id = 2;//запись номера ошибки
        Output = 1023 ; //выключаем нагрев
        myPID.SetMode(MANUAL); //ручной режим ПИД
        flag_point_switch = 0;//останавливаем работу на  интервале
```

```
      t_ALG_last_run =t_ALG_last_run + now() - t_ALG; // сохранение время работы алгоритма
      t_INT_last_run =t_INT_last_run + now() - t_INT; // сохранение время работы интервала
      Serial.print((String)"page FAULT_VOLTAGE"+char(255)+char(255)+char(255)); //страница с
оповещением
      Serial.print((String)"main.bt1.val=1"+char(255)+char(255)+char(255)); //изменение состояния
индикатора паузы
      Serial.print((String)"main.ERROR.en=1"+char(255)+char(255)+char(255)); //включаем мигание экрана
   }
   if((Input == NO_MAX31856)&&(flag_point_switch == 1)) // MAX31856 not communicating or not
connected
   {//pause
      mistake_id = 3;//запись номера ошибки
      Output = 1023 ; //выключаем нагрев
      myPID.SetMode(MANUAL); //ручной режим ПИД
      flag_point_switch = 0;//останавливаем работу на  интервале
      t_ALG_last_run =t_ALG_last_run + now() - t_ALG; // сохранение время работы алгоритма
      t_INT_last_run =t_INT_last_run + now() - t_INT; // сохранение время работы интервала
      Serial.print((String)"page NO_MAX31856"+char(255)+char(255)+char(255)); //страница с
оповещением
      Serial.print((String)"main.bt1.val=1"+char(255)+char(255)+char(255)); //изменение состояния
индикатора паузы
      Serial.print((String)"main.ERROR.en=1"+char(255)+char(255)+char(255)); //включаем мигание экрана
   }
   if((Input == FAULT_OPEN)||(Input == FAULT_VOLTAGE )||(Input == NO_MAX31856)) { Input = 0 ;};
//вывод нулевой температуры
   if((Input != FAULT_OPEN)&&(Input != FAULT_VOLTAGE )&&(Input != NO_MAX31856))
   Serial.print((String)"main.ERROR.en=0"+char(255)+char(255)+char(255)); //выключаем мигание экрана,
если нет оповищения об ошибке
   //-------------------------------------------------------------------------
}
```

7. movingAverage() реализация скользящего среднего

Простое скользящее среднее, или **арифметическое скользящее среднее** (англ. *simple moving average*, англ. *SMA*) численно равно среднему арифметическому значений исходной функции за установленный период и вычисляется по формуле:

$$SMA_t = \frac{1}{n}\sum_{i=0}^{n-1} p_{t-i} = \frac{p_t + p_{t-1} + \cdots + p_{t-i} + \cdots + p_{t-n+2} + p_{t-n+1}}{n},$$

(из файла stove.h) описание переменных

```
//Для реализации скользящего среднего
const int numReadings = 5;// Использование константы вместо переменной позволяет задать размер для
массива.
unsigned int readings[numReadings];      // данные, считанные с входного аналогового контакта
byte index = 0;                  // индекс для значения, которое считывается в данный момент
unsigned int total = 0;                  // суммарное значение
```

(из файла stove.ino) реализцаия самого алгоритма

Изначально происходит заполнение окна усреднения, это нужно, чтобы в ПИД алгоритм поступили корректные данные.

```
// инициализация массива окна усредения (инициализируем первые 4 значений, т.к. 5 мы измерим сразу
  //-------------------------------------------------
 while ( index < (numReadings-1))
 {
  readings[index] =  temperature->readThermocouple(CELSIUS);
  total= total + readings[index];
  index ++;
 }
 readings[index]=0; // значение последенго показания равно 0
 //так как в основном алгоритме мы вычитаем самое раннее значения из окна усреднения
 //-------------------------------------------------
```

Сама функция

```
void  movingAverage(void)//скользящее среднее

{

   //----усредение----------------------

   total= total - readings[index]; // вычитаем самое раннее значения из окна усреднения

   readings[index] = temperature->readThermocouple(CELSIUS); //считывание показания

   // добавляем его к общей сумме:

   total= total + readings[index];

   // продвигаемся к следующему значению в массиве:

   index ++;

   // если мы в конце массива...

   if (index >= numReadings)

     // ...возвращаемся к началу:

     index = 0;

   // вычисляем среднее значение:

   Input = total / numReadings;

   //----------------------------------------

}
```

8. myPID.Compute() ПИД алгоритм и ШИМ

Это самая главная функция в MCU , она реализует расчёт скважности выходного ШИМ сигнала. Данная функция была взята из готовой библиотеки и в ней изменен ПИД на ПИ алгоритм и еще некоторые исправления.
Для включения симистора, необходимо подавать на его управляющий электрод импульс, для этого используется ШИМ той же частоты, что и частота вызова ПИД. Поясню, я выбрал период вызова ПИД 1 сек, и период ШИМ такой же. То есть за 1 сек на нагревательный элемент поступит энергии от 0 до 100 % (или от 0 до 50 периодов, или от 0 до 100 полупериодов) .

С помощью управляющего сигнала можно открывать и закрывать симистор, и соответственно подачу напряжения на печку. Когда мы подаём лог.1 (5В) симистор закрыт, когда лог.0(0В) открыт. Вместо подачи 0 и 1 мы

подаём периодический сигнал с коэфф. заполн. в относительных единицах Output (0 min , 1023 max).При этом нужно инвертировать этот коэфф. Заполн., так как при max к.з. на выходе лог.1 и симистр закрыт, при min к.з. наоборот. Это реализовано в алгоритме.

Для начала рассмотрим переменные , которые используются: (из файла stove.h) описание переменных

```
//определение пременных для ПИ
unsigned int Setpoint = 400; //установочная темепратура
unsigned int Input; //настоящая температура
int Output = 1023;  //  коэфф. Зап. изначально максимально
double Kp=8; //коэффициент пропорнционального звена
double Ki=1; //коэффициент интегрирующего звена
#define SAMPLE_TIME 1000 //// частота вызова ПИ (в мсек)
//по умолчанию(SetOutputLimits(0, 255);   inAuto = false;)
PID myPID(&Input, &Output, &Setpoint, Kp, Ki, REVERSE); //создание объекта класса myPID REVERS указывает на то,
//что направление изменения выходного сигнала идёт в обратном направлении
#define PIN_OUTPUT 2 //выход , решулирующий температуру
uint32_t last_time= 0; //последнее время заупска алгоритма
```

(из файла stove.ino) инициализация

```
  //при первом запуске происходит также инициализация Initialize()
  myPID.SetMode(MANUAL);
  myPID.SetSampleTime(SAMPLE_TIME);
  myPID.SetOutputLimits(0, 1023);
```

Инициализация таймера, т.е ШИМ:

```
//------PWM----------------
  Timer3.initialize((long) SAMPLE_TIME * 1000);         // инициализировать timer1, и установить период равный периоду вызова ПИД в мксек
  Timer3.pwm( PIN_OUTPUT,  Output );                // задать шим сигнал  с  коэффициентом заполнения от 0 до 1023
```

(из файла PID_v2.h) Переменные которые используются

```
int dInput  = 0;
    int outputSum = 0;
    unsigned int lastInput = 0;
    byte SampleTime; //in sec
    int outMin, outMax;
```

(из файла PID_v2.cpp) реализцаия самого алгоритма

```
bool PID::Compute()
{
   if (!inAuto) return false;
       /*Compute all the working error variables*/
       unsigned int input = *myInput;
       int error = *mySetpoint - input; //расчёт ошибки
       if(lastInput != 0)
       dInput = (input - lastInput); //расчёт производной, не делится ни на что, т.к. шаг учитывается в коэффициенте
       outputSum += (ki * error); //интегральная составляющая выхода (выход интегратора)


       if (outputSum > outMax) outputSum = (int)outMax;
       else if (outputSum < outMin) outputSum = (int) outMin;
```

```
    int output;
    output = kp * error;
    /*Compute Rest of PID Output*/
    output += outputSum ;

    if (output > outMax) output = (int)outMax;
    else if (output < outMin) output = (int)outMin;
    if(controllerDirection) //если controllerDirection== REVERSE =1 ,то выполняется
    {
       output = 1023 - output ; //так как у нас при большем коэф. заполн энергии поступает меньше
    }
    *myOutput = output;

    /*Remember some variables for next time*/
    lastInput = input;
    return true;
}
```

9. jumpderivative(void) обработке экстренных скачков производной

Эта функция необходима тогда, когда в печке по какой-то причине начинает резко расти температура. Соответственно значение производной много больше допустимой. Я взял 10 град./сек .
(из файла stove.h)

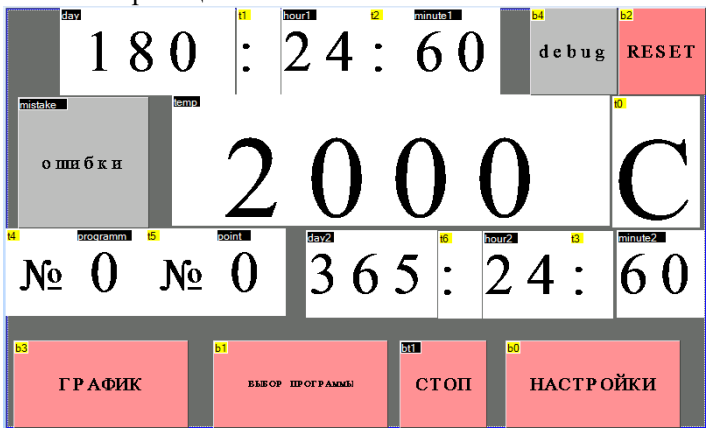
```
#define MAX_DERIVATIVE 10 //максимальное значение производной от температуры
```

(из файла stove.ino)

```
void jumpderivative(void)//----------обработке экстренных скачков производной------------------
{
  //----------обработке экстренных скачков производной------------------
  int dInput;
  dInput  = myPID.GetdInput(); //получаем значение производной (градус/сек)
  if((dInput>= MAX_DERIVATIVE)&&(flag_point_switch == 1))
  {//вход в паузу, если значение производной больше максимального и включен алгоритм
    Output = 1023 ; //выключаем нагрев
    myPID.SetMode(MANUAL); //ручной режим ПИД
    flag_point_switch = 0;//останавливаем работу на  интервале
    t_ALG_last_run =t_ALG_last_run + now() - t_ALG; // сохранение время работы алгоритма
    t_INT_last_run =t_INT_last_run + now() - t_INT; // сохранение время работы интервала
    Serial.print((String)"page jump_der"+char(255)+char(255)+char(255)); //страница с оповещением
    Serial.print((String)"main.bt1.val=1"+char(255)+char(255)+char(255)); //изменение состояния
индикатора паузы
  }
  //-----------------------------------------------------------------
}
```

10. Интерфейс дисплея

У дисплея Nextion есть собственный набор команд и собствнный редактор NextionEditor, в котором можно создавать интерфейс дисплея.
Ссылки на инструкции от производителя [2] ,[3]
Вот основные страницы , которые я создал
Главная страница.



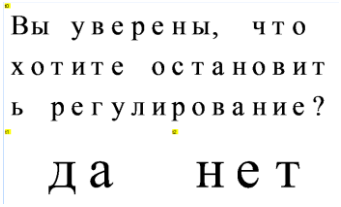
Здесь элементы day, hour1, minute1 отвечают за день, час, минуты с момента запуска алгоритма.

Day2,hour2,minute2 за день, час, минуты с момента запуска интервала

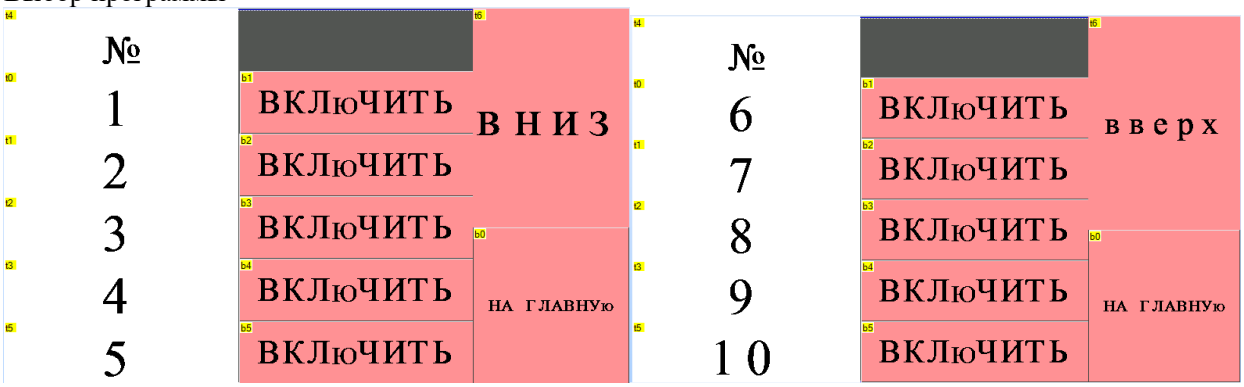
Program номер программы , point номер участка , temp температура

Здесь элемент ошибки появляется в случае ошибки, также при ошибках начинает мигать экран красным цветом. График пока в разработке.

По нажатию кнопки стоп, появляется уточняющее сообщение, при подтверждении которого, кнопка стоп меняет свой цвет на красный:

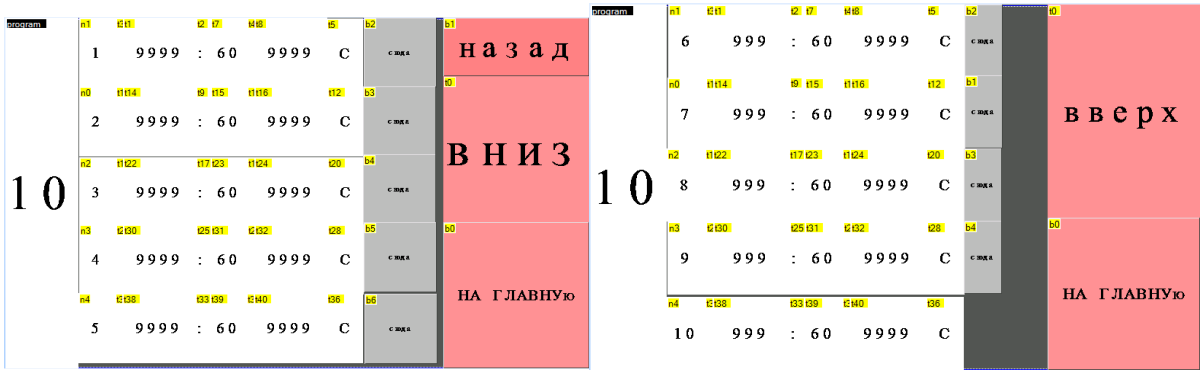


Выбор программы



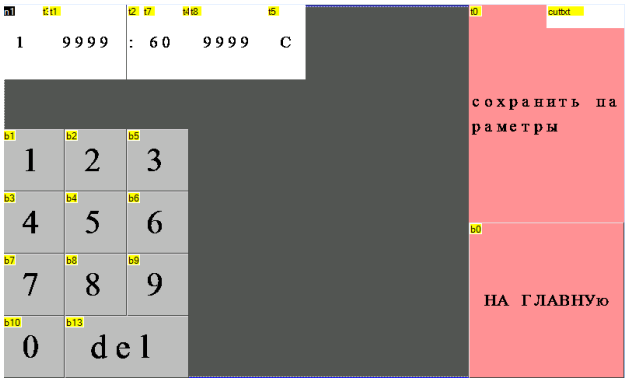
При нажатии на кнопку включить, происходит включение программы

Выбор точек алгоритма (выбор точек программы)

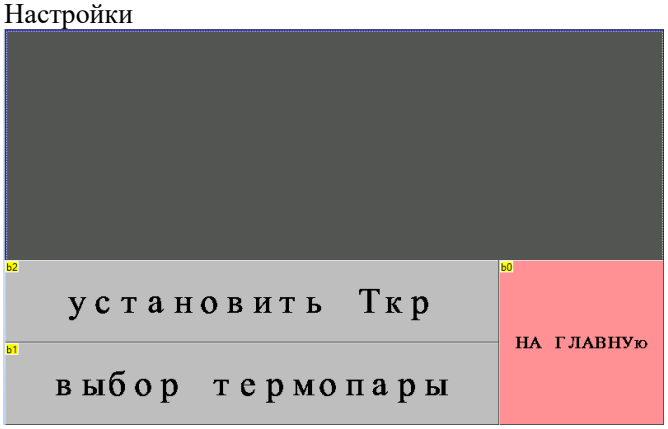


Здесь можно задать время в часах и минутах работы на определенном интервале, температуру, к которой мы должны прийти за это время. Также при нажатии кнопки «сюда», можно вернуться на данный участок и начать с данной температуры, таким образом будет выполняться участок, следующий за выбранным.

Установка точки

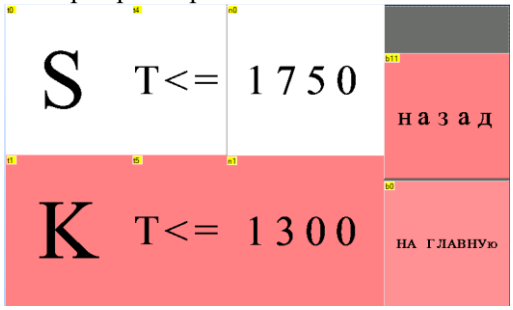


Настройки

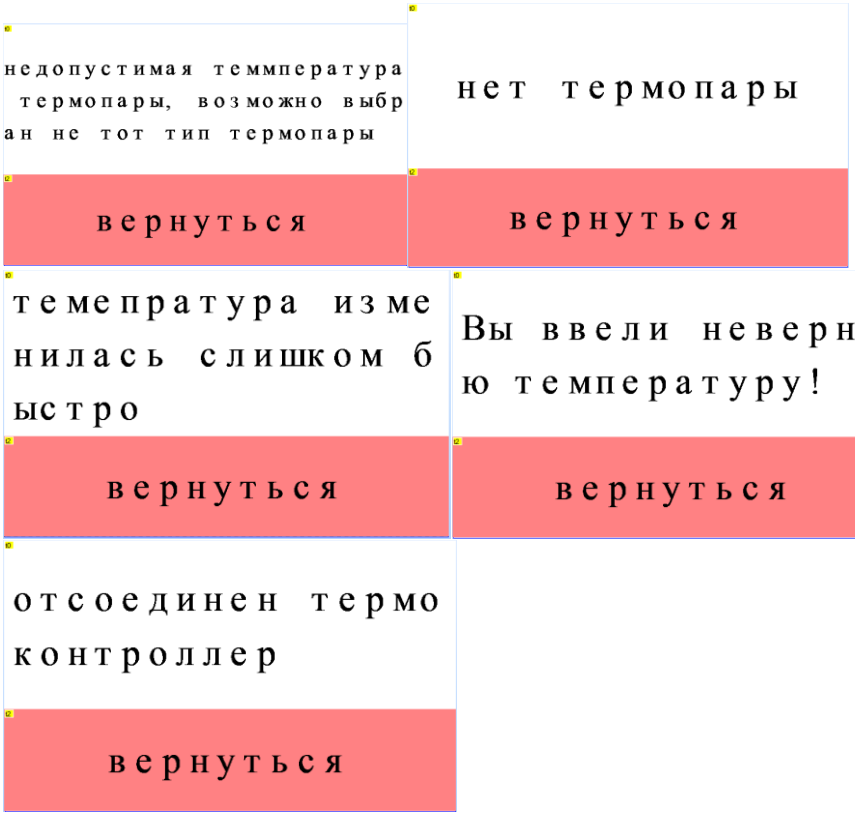


Установить Ткр пока в разработке.

Выбор термопары

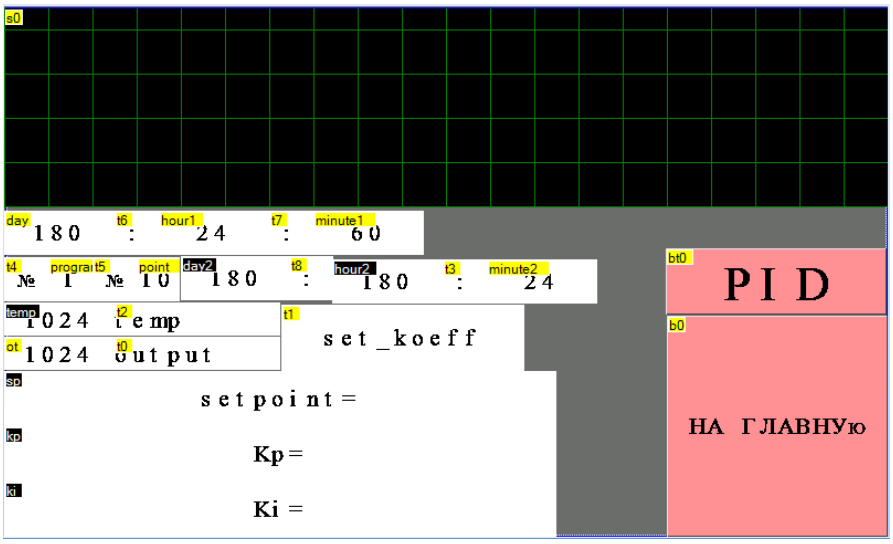


Возможные сообщение об ошибках (в дальнейшем убдет добалены новые). При возникновении какой-либо ошибки появляется соответствующее сообщение.



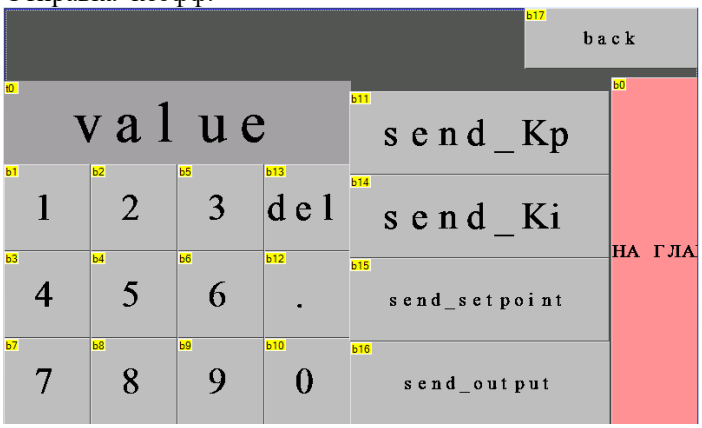
Затем начинает мигать главный экран и появляется кнопка ошибки, при переходе на которую появляется список ошибок:

Отладка (debug) ПИД алгоритма



Здесь дублируется время и температура с главного экрана, также отображается установочное значение setpoint, коэффициент пропорционального и интегрального звена., можно включить и отключить PID

алгоритм. Также сюда выводится бегущий график, отображающий температуру от 0 до 500 градусов и по времени от 0 до 800 сек ,так как разрешение экрана 800 pi по гор-ли.
Отправка коэфф.



11. Senddata() Реализация отправки данных на дисплей

Сначала происходит настройка скорости последовательного порта экрана и MCU: (из файла stove.ino)
Serial.begin(115200);

```
  Serial.print("baud=115200ÿÿÿ"); //установка скорости экрана

  t_MCU = now(); //записываем время перового запуска микроконтроллера
```

Затем отправляем некотрые необходимые данные:

```
//отправка на экран необходимых данны
   Serial.print((String)"debug.kp.txt=\""+"Kp="+Kp+"\""+char(255)+char(255)+char(255));
   delay(50);
   Serial.print((String)"debug.ki.txt=\""+"Ki="+Ki+"\""+char(255)+char(255)+char(255));
   delay(50);
   Serial.print((String)"debug.sp.txt=\""+"Setpoint="+Setpoint+"\""+char(255)+char(255)+char(255));
   delay(50);
```

Затем Senddata:

```
void senddata(void)//отправка  данных на экран
{
   //отправка  данных на экран
   Serial.print((String)"main.temp.val="+Input+char(255)+char(255)+char(255));
   delay(50);
   Serial.print((String)"debug.ot.val="+Output+char(255)+char(255)+char(255));
   delay(50);
   Serial.print((String)"debug.kp.txt=\""+"Kp="+Kp+"\""+char(255)+char(255)+char(255));
   delay(50);
   Serial.print((String)"debug.ki.txt=\""+"Ki="+Ki+"\""+char(255)+char(255)+char(255));
   delay(50);
   Serial.print((String)"debug.sp.txt=\""+"Setpoint="+Setpoint+"\""+char(255)+char(255)+char(255));
   delay(50);
   uint8_t temp = Input*180/500;
   Serial.print((String)"add 7,0,"+temp+char(255)+char(255)+char(255)); //рисуем график
   if(flag_point_switch)
   {
```

```
      //отправка времени с момента запуска алгоритма
      t_ALG_run = t_ALG_last_run + now() - t_ALG;//расчет времени работы алг. и инт.
      t_INT_run = t_INT_last_run + now() - t_INT;//------------------------------
      Serial.print((String)"main.day.val="+get_day(t_ALG_run)+char(255)+char(255)+char(255)); //(0-inf)
      Serial.print((String)"main.minute1.val="+minute(t_ALG_run)+char(255)+char(255)+char(255)); //(0-60)
      Serial.print((String)"main.hour1.val="+hour(t_ALG_run)+char(255)+char(255)+char(255)); //(0-24)
      //отправка времени с момента запуска интервала
      Serial.print((String)"main.day2.val="+get_day(t_INT_run)+char(255)+char(255)+char(255));//(0-inf)
      Serial.print((String)"main.hour2.val="+hour(t_INT_run)+char(255)+char(255)+char(255));//(0-24)
      Serial.print((String)"main.minute2.val="+minute(t_INT_run)+char(255)+char(255)+char(255)); //(0-60)
   }
}
```

12. readNextioncommand() обработка приходящих данных

Эта функция реализует приём и расшифровку сообщений от экрана.Для начала укажем формат данных , которые могут прийти с экрана:
(из файла stove.h)

```
//Расшифровка управляющих команд протокола передачи данных
//формат сообщения #12;
// # - признак начала сообщения
//1- байт номера команды
//2- байт номера действия или данные
// ; - признак конца команды
//номер команды
#define PID_SWITCH  2
#define SETPOINT 3
#define OUTPUT 4
#define KOEFF_PROPORTIONAL 5
#define KOEFF_INTEGRAL 6
#define TEMP_POINT 7  //точки установки температуры
#define POINT_SWITCH  8//включение выключение изменение температуры по заданному алгоритму
#define POINT_RUN  9  //пауза и выход из паузы
#define RETURN_INTERVAL  10 //вернуться на интервал
#define INTERVAL_HIGHLIGHING 11//подсвечивание интервала
#define MISTAKE 12 //отправка информации об ошибках
//номер действия
#define ON  1
#define OFF 0
```

(из файла stove.ino)

```
void readNextioncommand()// обработка приходящих данных
{
   static String  inStr = ""; // Это будет приемник информации от Nextion (здесь только //1- байт номера команды /2- байт номера действия или данные)
   static bool serialReadFlag = false; // А это флаг появление сообщения.

  /*****************************************************************/
   // обработка приходящих данных
   if (Serial.available())
```

```
  {
    uint8_t inn = Serial.read(); // читаем один байт
    if(serialReadFlag)
    { // Если установлен флаг приема - действуем
      if(inn == 59)// ASCII : ";" // Находим конец передачи ";"
      {
        if(inStr.length() > 0) // Проверяем длину сообщения и отправляем в "переработку"
        {
          checkCommand(inStr); // В этой функции будем парсить сообщение
        }
        serialReadFlag = false; // Сбрасываем флаг приема
      }
      else // А это нормальный прием
      {
        inStr += (char)inn; // Считываем данные
      }
    }
    else
    { // А здесь отлавливается начало передачи от Nextion
      if(inn == 35) // ASCII : "#"
      {
        serialReadFlag = true; // После # начинаем чтение при следующем заходе
        inStr = ""; // Но до этого очистим стринг приема
      }
    }
  }
  /*****************************************************************************/
}
```

Теперь подробнее о функции checkCommand(inStr) в следующем разделе.

13. checkCommand(inStr) парсинг сообщение от экрана

Это одна из главных функций, реализующая расшифровку и выполнение команд от экрана. Реализуется в виде оператора множественного выбора switch, где в каждом где проверяется номер команды и далее происходит работа с данными, находящимися в перемнной last. Здесь используются переменные:

```
//точки изменения температуры
uint16_t  temp_point[10];   //содержит информацию о температуре в точке
uint16_t  time_point[10];//содержит информацию о времени нахождении на интервале точки
float32_t time_step[10] ;//время ,через которое меняется установочная температура (в сек) для каждого участка
uint8_t interval = 0; // номер участка 0-8
uint8_t mistake_id = 0; //номер ошибки
```

Она достаточно объёмная поэтому остановимся на пояснении наиболее важных из команд:

TEMP_POINT сохранение точек установки температуры

Формат сообщения на примере 1,20,600:2,40,300:3,60,300:4,0,0:
Запятая в роли разделителя параметров, двоеточие разделитель точек.

```
case TEMP_POINT  :
```

```
{
  // наше сообщение (пример) 1,20,600:2,40,300:3,60,300:4,0,0:
  String value = ""; //проверяемый символ(строка)
  int point_number = 0; //номер точки
  int parametrs_num = 0; //номер параметра
  //(0-номер точки,1-время в мин,2-температура)
  int i;//счетчик
  for(i = 0; i < last.length(); i++ )//проходим по всей строке
  {
    value += last[i];//запись следующего символа
    if(last[i] == ',')//запятая в роли разделителя
    {
      if(parametrs_num == 0)//номер точки
      {
        point_number = value.toInt();//запись номера точки
        parametrs_num ++;//переход следующий парметр
        value = ""; //удаляем значение
        continue;
      }
      if(parametrs_num == 1)//время
      {
        time_point[point_number-1] = value.toInt();//запись времени
        parametrs_num = 0; // достигли max-1 параметра
        value = ""; //удаляем значение
        continue;
      }
    }
    if(last[i] == ':')
    {
      temp_point[point_number - 1] = value.toInt(); //записываем температуру
      //расчет шага времени
      if (point_number == 1) //записываем шаг времени
      {//расчёт шага изменения температуры
        int16_t denominator = (temp_point[point_number - 1] - Input);
        if (denominator != 0)
        {
          time_step[point_number - 1] = (60 * time_point[point_number - 1]);
          time_step[point_number - 1] /= (denominator); //1 шаг времени
          time_step[point_number - 1] = abs(time_step[point_number - 1]);
        }
        else
          time_step[point_number - 1] = 0;//если температура постоянна на участке
                                         //шаг времени = 0
      }
      else
      {
        int16_t denominator = (temp_point[point_number - 1] - temp_point[point_number - 2]);
        if (denominator != 0)
        {
          time_step[point_number - 1] = 60 * (time_point[point_number - 1]);
```

```
                time_step[point_number - 1] /= denominator;
                time_step[point_number - 1] = abs(time_step[point_number - 1]);  //последующие шаги
времени
              }
              else
                time_step[point_number - 1] = 0;//если температура постоянна на участке
                                  //шаг времени = 0
            }
            value = ""; //удаляем значение

          }
        }
        break;
      }
```

POINT_RUN Вход и выход из паузы

```
      case POINT_RUN :  //9 пауза и выход из паузы слежения за заданной температурой
      {
        if(last.toInt() == ON)//выход из паузы
        {
          myPID.SetMode(AUTOMATIC);
          flag_pause_exit = 1; // флаг выхода из паузы (разрешение проверки условий выхода из паузы)
          //операции перед выходом из паузы
          flag_reaching_Setpoint = 0; //разрешение операции перед выходом из паузы : достижение
установленного значение
        }
        else //пауза
        {
          if(flag_point_switch == 1)
          { //вход в паузу возможен только, если алгоритм уже работает
            myPID.SetMode(MANUAL);
            Timer3.setPwmDuty(PIN_OUTPUT,  1023); //закрваем семистр, выключаем нагрев
            flag_point_switch = 0; //выключаем линейное изменение по функции
            t_ALG_last_run =t_ALG_last_run + now() - t_ALG; // сохранение время работы алгоритма
            t_INT_last_run =t_INT_last_run + now() - t_INT; // сохранение время работы интервала
          }
        }

        break;
      }
```

POINT_SWITCH включение новой программы слежения за заданной

```
case POINT_SWITCH :  //8 включение новой программы  слежения за заданной температурой
      {//сообщение  действие = "номер программы"
          t_ALG = now(); // время начала запуска  алгоритма
          Serial.print((String)"main.programm.val="+last.toInt()+char(255)+char(255)+char(255));//вывод
номера программы
          flag_point_switch = 1;
          Setpoint = Input ; // записываем текущую температуру
          interval = 0; //возвращение на 1-ый интервал
          Serial.print((String)"cle 24,0"+char(255)+char(255)+char(255)); //очистить график
```

```
        break;
    }
```

RETURN_INTERVAL возвращение на начало какого-либо интервала

```
case RETURN_INTERVAL:  //10 возвращение на начало какого-либо интервала
    {
      //условия для возвращения
      if(flag_point_switch == 1)//возвращение возможно только, если алгоритм уже работает
      {
        flag_reaching_Setpoint = 0; //разрешение операции перед возвращением на начало интервала : достижение установленного значение
        interval = last.toInt(); //запись номера интервала 0-8
        Setpoint = temp_point[interval] ; // записываем температуру начала интервала
        flag_first_run_interval = 0 ;//перед запуском интервала нужно проделать необходимые операции
        interval++; //начинать быдем с следующего интервала
        //если была нажата первая точка , то в Setpoint будет записано значение температуры  1 точки
        //и далее после достижения этой температуры, будут выполняться последующие интервалы
      }
      break;
    }
```

14. reachingSetpoint() проверка: достигли ли мы установленного значения Setpoint

Данная функция реализует ожидание достижения установленного значения Setpoint . Пока оно не установится, слежение за целевой функцией не будет выполняться, по нескольким причинам:

1. после паузы выключается нагрев и печь остывает, и для того чтобы продолжить выполнение функции после выхода из паузы, нужно сначала вернуться к последней температуре
2. При возвращении на какой-либо участок, нужно вернуться к нужной температуре

```
void reachingSetpoint(void)// проверка: достигли ли мы установленного значения?
{
  if((flag_reaching_Setpoint == 2)||(flag_reaching_Setpoint == 0)) // проверка: достигли ли мы установленного значения?
  {
    if(flag_reaching_Setpoint == 0)//первый запуск участка кода
    {
    //проверка больше или меньше температура в начале участка, чем установочное значение
          if(Input >= Setpoint)
          compare=1;
          else
          compare=0;
          flag_reaching_Setpoint == 2;
    }
    if(((compare==1)&&(Input <= Setpoint))||((compare==0)&&(Input >= Setpoint))) //достижение последнего установочного значения
    {
```

```
        flag_reaching_Setpoint = 1; // достигли
      }
    }
}
```

Здесь используется флаг: flag_reaching_Setpoint
Алгоритм работает так: если при первом запуске функции температура больше чем Setpoint, то мы ждём пока она станет не меньше , чем значение Setpoint,далее флаг достижения станет true.

15. Exitpause() проверка выполнения операций перед выходом из паузы

Эта функция необходима для выполнения, после выхода из паузы и достижение setpoint, перед тем как перейти к выполнению алгоритма, здесь и разрешается продолжить выполнение алогритма.
Здесь используется флаг flag_pause_exit

```
void exitpause(void)//проверка выполнения операций перед выходом из паузы

{
  if(flag_pause_exit == 1) //проверка выполнения операций перед выходом из паузы
  {
    if(flag_reaching_Setpoint == 1) //достигли установленного значения
    {
      t_ALG = now();//перезаписываем время последнего включения алгоритма
      t_INT = now();//и интервала
      flag_point_switch = 1; //разрешение выполнения алгоритма
      flag_pause_exit = 0; //выход из паузы осуществлен
    }
  }
}
```

16. changeLinearly() алгоритм слежения за целевой функцией

Это один из основных алгоритмов. Он выполняет слежение за целевой функцией. Суть метода такова, участок, на котором температура линейно изменяется разбивается на маленькие шаги, на которых изменяется setpoint. Поясним на рисунке.

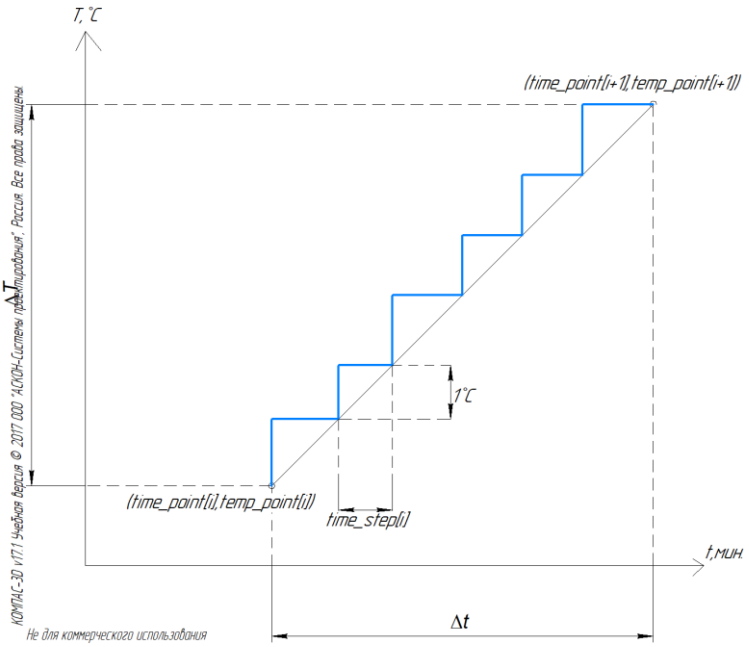


Рис.10(линейное изменение функции)

Как видно функция на самом деле будет изменяться не линейно а по шагам и соответственно будет иметься max погрешность 1 ℃ .
Установочная температуру (setpoint) по шагам (для каждого участка шаг свой и высчитывается он в функции checkCommand()) изменяется через время time_step[i]=Δt/ΔT (сек/1 ℃), т.е. setpoint++.
Используемые переменные:
(из файла stove.h)

//точки изменения температуры

uint16_t temp_point[10]; //содержит информацию о температуре в точке

uint16_t time_point[10];//содержит информацию о времени нахождении на интервале точки

float32_t time_step[10] ;//время ,через которое меняется установочная температура (в сек) для каждого участка

uint8_t interval = 0; // номер участка 0-8

uint8_t compare = 0; //больше или меньше температура в начале участка, чем температура в конце участка

//1 температура в начале участка больше 0 - температура в начале участка меньше

uint32_t lastsecond = 0 ;//время крайнего вызова изменения setpoint (такого значения переменной хватит на 50 дней)

(из файла stove.ino) сам алгоритм

```
void changeLinearly(void)//алгоритм слежения за целевой функцией
{
  if(flag_point_switch) //алгоритм слежения за целевой функцией
  {
    if(flag_reaching_Setpoint == 1) //включаем только при достижении температуры установленного значения
    {
      for(int i = 0; i <= 9; i++)//проверяем на каком мы интервале находимся
      {
        if(interval == i)
        {
          if(flag_first_run_interval == 0) //операции перед включением интервала
          {
            flag_first_run_interval = 1 ; //интервал запущен
            t_INT = now(); //запись времени первого запуска интервала
            t_INT_last_run = 0 ;
```

```
    t_INT_run = t_INT_last_run + now() - t_INT;
    int intreval_1 = interval + 1 ;                                    //вывод номера интервала
    Serial.print((String)"main.point.val="+intreval_1+char(255)+char(255)+char(255));//вывод номера интервала
    //проверка больше или меньше температура в начале участка, чем температура в конце участка
    if(Input >= temp_point[i])
    compare=1;
    else
    compare=0;
    lastsecond = millis();
}
if(time_step[i] != 0) //линейное изменение температуры
{
    uint32_t time_after_step = millis()-lastsecond ; //время, пройденное после последнего шага (мс)
    if(  time_after_step >= (uint32_t)(time_step[i]*1000)  ) // прошло время шага
    {
       if(compare==0) //если нужно увеличивать температуру, то идем вверх на 1 градус
       Setpoint = Setpoint+1;
       else ////если нет, то идём  вниз
       Setpoint = Setpoint-1;
       if( get_minute(t_INT_run) >= time_point[i])//достижение установленного времени работы интервала
       {
          Setpoint = temp_point[i];//поддерживаем установленное значения температуры
          if(((compare==1)&&(Input <= temp_point[i]))||((compare==0)&&(Input >= temp_point[i])))
          {
             flag_first_run_interval = 0 ; //интервал закончен, разрешение операций перед включением интервала
             if(i==9)
             { // то есть закончился алгооритм
                flag_point_switch = 0; // запрещаем алгоритму вызываться во избежания повторения алгоритма
                            //для повторного вызова нужно перезапустить алгоритм через экран
             }
             interval++;//переход на следующий интервал по достижению установленного времени и температуры
          }
       }
       lastsecond = millis() ;
    }
}
else //температура постоянна на участке
{
    //Setpoint не изменяется, так как он установлен в предыдщем интервале, либо при запуске алгоритма
    if( get_minute(t_INT_run) >= time_point[i])//достижение установленного времени работы интервала
    {
       flag_first_run_interval = 0 ; //интервал закончен, разрешение операций перед включением интервала
       if(i==9)
       { // то есть закончился алгооритм
          flag_point_switch = 0; // запрещаем алгоритму вызываться во избежания повторения алгоритма
                      //для повторного вызова нужно перезапустить алгоритм через экран
       }
       interval++;//переход на следующий интервал по достижению установленного времени и температуры
    }
}
```

```
            break;
          }
        }
      }
    }


  }
}
```

17. Реализация расчёта времени:

Используем готовую библиотеку TimeLib.h, в которой прел=дставлен новый тип переменной time_t, хранящей секунды, по функции now() возвращает это время в сек с 1 января 1970 года (в секундах). Для реализации используются переменные (из файла stove.h)

```
//определение пременных для вывода времени
#include <TimeLib.h>
time_t t_MCU ; //записываем время перового запуска микроконтроллера
time_t t_ALG; // время запуска программы
time_t t_ALG_last_run = 0; //время работы алгоритма перед включением паузы
time_t t_ALG_run; // время работы алгоритма
time_t t_INT; // время запуска интервала
time_t t_INT_last_run = 0; //время работы интервала перед включением паузы
time_t t_INT_run = 0; // время работы интервала
```

Рис.5 поясняет работу на нём в скобочках указаны действия, которые нужно сделать в эти моменты времени:

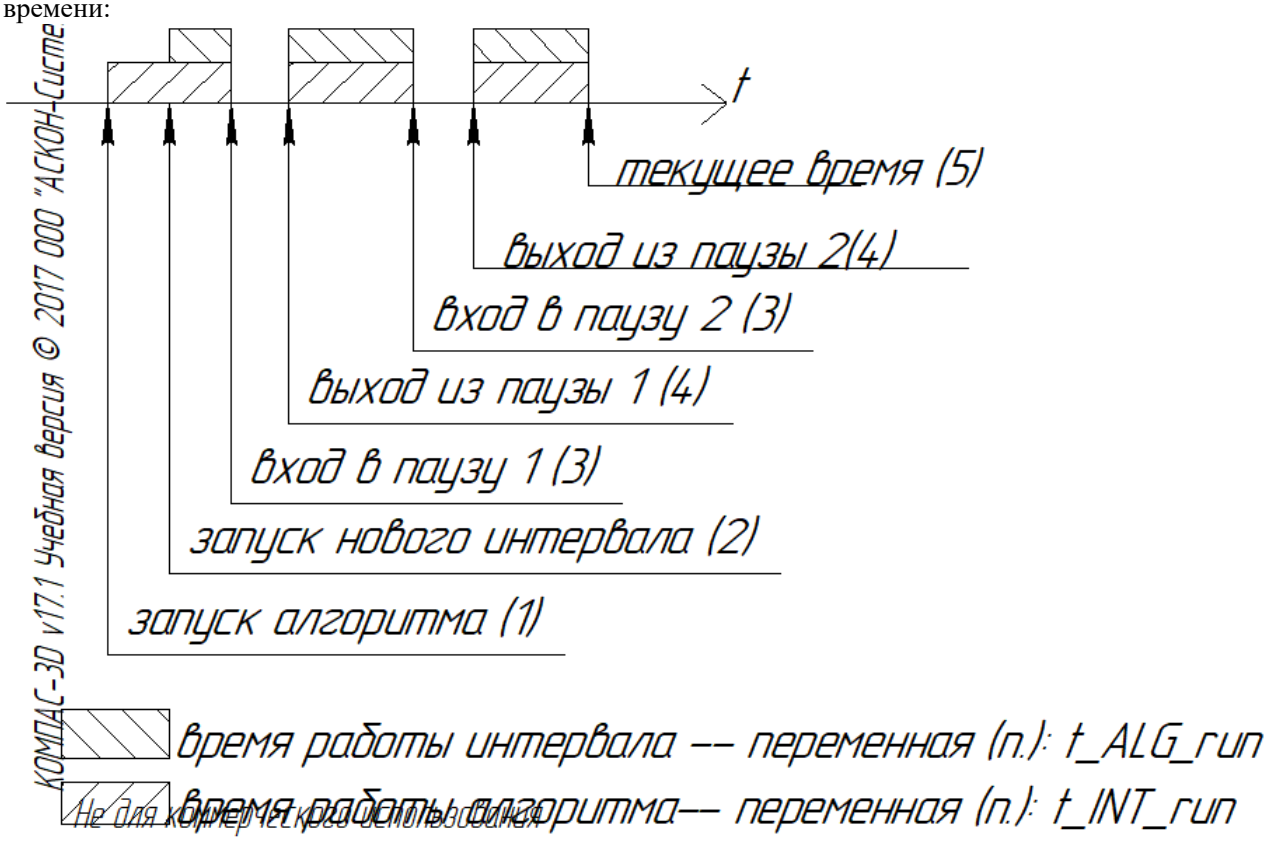


Рис.10 (пояснение работы)

1. t_ALG = now(); // запись времени начала запуска алгоритма
2. t_INT = now(); //запись времени первого запуска интервала
3. t_ALG_last_run =t_ALG_last_run + now() - t_ALG; // сохранение время работы алгоритма

t_INT_last_run =t_INT_last_run + now() - t_INT; // сохранение время работы интервала

4. Перезаписываем время начала алгоритма и интервала
   t_ALG = now();//перезаписываем время последнего включения алгоритма
   t_INT = now();//и интервала
5. t_ALG_run = t_ALG_last_run + now() - t_ALG;//расчет времени работы алг. и инт.
   t_INT_run = t_INT_last_run + now() - t_INT;//----------

Заключение

В результате создан макет системы управления с обратной связью на основе PID регулятора для управления и контроля температуры в камере печи для варки стекла.

Макет комплекса может выполнять следующие функции:

1. Установка и поддержание желаемой температуры в печи;
2. Контролируемое изменение температуры по заданной линейной функции.
3. Вывод информации о температуре и цикле нагрева на TFT дисплей
4. Установка входящих параметров для контроля температуры с TFT дисплея.

Также добавлены некоторые дополнительные возможности по настройке комплекса:

1. отображение ошибок на экран в случаи их возникновения.
2. выбор типа термопары.
3. Изменение коэффициентов ПИ регулятора.
4. построение графика с выводом температуры в реальном времени.
5. Запись данных на SD-карту
6. Расчёт времени регулирования и запись его в файл на SD-карту.

Список использованных источников

1. Евсеев Ю. А., Крылов С. С. Симисторы и их применение в бытовой электроаппаратуре, МОСКВА, ЭНЕРГОАТОМИЗДАТ, 1990.
2. https://nextion.itead.cc/resources/documents/instruction-set/ описание команд
3. https://nextion.itead.cc/editor_guide/ описание редакт