|  |
| --- |
| float setpoint;  float setkp = 9;  float setki = 0.0039;  float setkd = 0;  void setup(){  Output\_Init();  Display\_Init();  Serial\_Init();  Thermocouple\_Init();  zero\_cross\_init  }  void loop(){  Thermocouple\_Handler();  PID\_Koefisien(setkp, setki , setkd, 1);  PID\_Calculate(setpoint, Thermocouple\_GetTemperature());    int d;  d = PID\_ReadResult();  if(d < 0){d = 0;}  else if(d > 100){d = 100;}  Output\_SetDutyCycle(d); //Nilai Duty Cycle % (PWM)  Output\_periode(1000);  Output\_Handler();  Display\_Handler();  Serial\_Handler();  zero\_handler  }  void SetSetpoint(float sp){  setpoint = sp;  }  void SetKp(float kp){  setkp = kp;  }  void SetKi(float ki){  setki = ki;  }  void SetKd(float kd){  setkd = kd;  }  float GetSetpoint(){  return setpoint;  }  #include <LiquidCrystal\_I2C.h>  LiquidCrystal\_I2C lcd(0x27, 16, 2);  uint32\_t displayTimer;  void Display\_Init(){  lcd.begin();  }  void Display\_Handler(){  if(millis() - displayTimer >= 1000){  lcd.setCursor(0,0);  lcd.print(" Temp. Pwr.");  lcd.setCursor(0,1);  lcd.print(Thermocouple\_GetTemperature(),2);  lcd.print("|");  lcd.print(GetSetpoint(), 0);  lcd.print(" ");  lcd.setCursor(12,1);  lcd.print(Output\_GetDutyCycle());  lcd.print("% ");  displayTimer = millis();  }  }  #define DRIVER\_OUT 5  #define LED\_PIN 13  uint32\_t periode;  uint8\_t dutyCycle;  uint32\_t ledTimer;  uint8\_t Output\_GetDutyCycle(){  return dutyCycle;  }  void Output\_Init(){  pinMode(DRIVER\_OUT, OUTPUT);  pinMode(LED\_PIN, OUTPUT);  }  void Output\_periode(uint8\_t period){  periode = period;  }  void Output\_SetDutyCycle(uint8\_t duty){  dutyCycle = duty;  }  void Output\_Handler(){  uint32\_t tOn, tOff;  if (dutyCycle == 0){  tOn = 0;  digitalWrite(DRIVER\_OUT, LOW);  digitalWrite(LED\_PIN, LOW);  }  else {tOn = periode \* (dutyCycle / 100.0);  tOff = periode - tOn;  if  (millis() - ledTimer < tOn){  digitalWrite(DRIVER\_OUT, HIGH);  digitalWrite(LED\_PIN, HIGH);  }  else if  (millis() - ledTimer < tOn + tOff){  digitalWrite(DRIVER\_OUT, LOW);  digitalWrite(LED\_PIN, LOW);  }  else {  ledTimer = millis();  }  }  }  float kp, ki, kd, ts;  float se, de, le, u;  uint32\_t pidTimer;  void PID\_Koefisien(float Kp, float Ki, float Kd, float Ts){  kp = Kp;  ki = Ki;  kd = Kd;  ts = Ts;  }  void PID\_Reset(){  se = 0;  de = 0;  le = 0;  u = 0;  pidTimer = millis();  }  float PID\_ReadResult(){  return u;  }  void PID\_Calculate (float sp, float av){  if (millis() - pidTimer >= ts \* 1000){  float e = sp - av;    // Delta error  de = e - le;    // rumus PID  u = (kp \* e) + (ki \* se \* ts) + (kd \* de / ts);    // Integral Error  se = se + e;    // Nilai Error sebelumnya  le = e;  pidTimer = millis();  }    }  uint32\_t serialTimer;  uint32\_t serialRxTimer;  char serialRxBuffer[22];  uint8\_t serialRxPtr;  void Serial\_Init(){  Serial.begin(115200);  }  void Serial\_Handler(){  if(Serial.available()){  char data = 0;  memset(serialRxBuffer, 0, 22);  serialRxPtr = 0;  serialRxTimer = millis();  do{  if(Serial.available() && serialRxPtr < 20){  data = Serial.read();  serialRxBuffer[serialRxPtr++] = data;  }  }while(data != '\n' && millis() - serialRxTimer < 1000);  }  if(serialRxPtr > 0){  if(strncmp(serialRxBuffer, "SP:", 3) == 0){  SetSetpoint(atof(serialRxBuffer + 3));  PID\_Reset();  }  if(strncmp(serialRxBuffer, "KP:", 3) == 0){  SetKp(atof(serialRxBuffer + 3));  PID\_Reset();  }  if(strncmp(serialRxBuffer, "KI:", 3) == 0){  SetKi(atof(serialRxBuffer + 3));  PID\_Reset();  }  if(strncmp(serialRxBuffer, "KD:", 3) == 0){  SetKd(atof(serialRxBuffer + 3));  PID\_Reset();  }  if(strncmp(serialRxBuffer, "RST", 3) == 0){  SetKp(9);  SetKi(0.0038);  SetKd(0);  PID\_Reset();  }  memset(serialRxBuffer, 0, 22);  serialRxPtr = 0;  }    if (millis() - serialTimer >= 1000){  Serial.println(Thermocouple\_GetTemperature());  serialTimer = millis();  }  }  #include <SPI.h>    #define MAX6675\_CS 10  #define MAX6675\_SO 11  #define MAX6675\_SCK 12  uint32\_t thermocoupleTimer;  float thermocoupleData;  void Thermocouple\_Init()  {  pinMode(MAX6675\_CS, OUTPUT);  pinMode(MAX6675\_SO, INPUT);  pinMode(MAX6675\_SCK, OUTPUT);  }    void Thermocouple\_Handler() {  if(millis() - thermocoupleTimer >= 500){  thermocoupleData = readThermocouple();  thermocoupleTimer = millis();  }  }  float Thermocouple\_GetTemperature(){  return thermocoupleData;  }    float readThermocouple() {  uint16\_t v;    digitalWrite(MAX6675\_CS, LOW);    // Read in 16 bits,  // 15 = 0 always  // 14..2 = 0.25 degree counts MSB First  // 2 = 1 if thermocouple is open circuit  // 1..0 = uninteresting status    v = shiftIn(MAX6675\_SO, MAX6675\_SCK, MSBFIRST);  v <<= 8;  v |= shiftIn(MAX6675\_SO, MAX6675\_SCK, MSBFIRST);    digitalWrite(MAX6675\_CS, HIGH);  if (v & 0x4)  {  // Bit 2 indicates if the thermocouple is disconnected  return NAN;  }    // The lower three bits (0,1,2) are discarded status bits  v >>= 3;  // The remaining bits are the number of 0.25 degree (C) counts  return v\*0.25;  int heater = 5;  int zero\_cross = 2;  volatile int count = 0;  uint32\_t zeroTimer;  int last\_CH1\_state = 0;  volatile bool zero\_cross\_detected = false;  uint32\_t maximum\_firing\_delay = 10000;  uint32\_t fireDelay;  volatile uint32\_t zcTimer;  void zero\_cross\_init(){  pinMode (heater,OUTPUT);  pinMode (zero\_cross,INPUT\_PULLUP);  attachInterrupt(digitalPinToInterrupt(zero\_cross), zero\_cross\_ISR\_handler, RISING);  }  float pwr;  void zero\_set\_power(float p){  pwr = p;  fireDelay = ((100 - pwr) \* 1.8) \* (0.01 / 180);  pwr = (pwr > 100 ? 100 : pwr);  fireDelay = maximum\_firing\_delay - ((pwr > 0 ? (float)pwr / 100 : 0) \* maximum\_firing\_delay);  }  float zero\_get\_power(){  return pwr;  }  void zero\_handler(){  if(micros() - zcTimer >= fireDelay){  digitalWrite(heater,HIGH);  }  }  void zero\_handler(){  uint32\_t period = 500;  uint32\_t ton = (pwr / 100.0) \* period;  if(millis() - zcTimer < ton){  digitalWrite(heater, HIGH);  }  else if(millis() - zcTimer < period){  digitalWrite(heater, LOW);  }  else{  zcTimer = millis();  }  }  uint32\_t zero\_GetFiringDelay(){  return fireDelay;  }  void zero\_cross\_ISR\_handler(){  digitalWrite(heater, LOW);  zcTimer = micros();  }  int Get\_zero\_cross\_detected(){  return count;  } |