KENDALI PID DENGAN FITUR PERUBAHAN PARAMETER

Tugas EL4121 – Perancangan Sistem Embedded

Oleh:

13219033 - Sidartha Prastya. P

INSTITUT TEKNOLOGI BANDUNG
2022

DAFTAR ISI

DAFTAR GAMBAR

1 Spesifikasi

Pada tugas ini terdapat beberapa spesifikasi sebagai berikut.

- Sistem yang dikendalikan adalah kecepatan dengan fungsi transfer seperti pada tugas sebelumnya *(Supervisory Control)*.
- Pengendali menggunakan *Controller* PID.
- *Display* menggunakan LCD.
- Parameter Kp, Ki, dan Kd dapat ditampilkan di *display*.
- Parameter Kp, Ki, dan Kd dapat diubah menggunakan tombol *push button.*
- Algoritma perubahan parameter meniru seperti pada pengendali REX C100.
- Input kecepatan *(setpoint)* menggunakan potensiometer.
- Parameter-parameter sistem dicatat di komputer PC/Laptop (kecepatan, control value, waktu).
- Parameter Kp, Ki, dan Kd disimpan di EEPROM / Flash setelah diubah sehingga tidak hilang ketika listrik dimatikan.

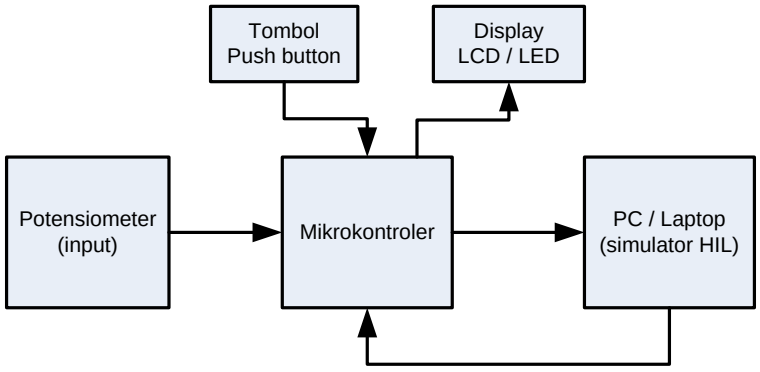
Adapun tombol yang diperlukan:

- SET (untuk masuk dan keluar mode *setting*)
- SHIFT : untuk berpindah digit ketika melakukan *setting.*
- ADD : untuk menambahkan nilai digit yang dipilih.

Adapun karakteristik tombol yang dibutuhkan, yaitu

- Tombol dilengkapi dengan *debouncing*.
- Apabila tombol ditekan sekali, maka nilai akan berubah sekali.
- Apabila tombol ditekan dan ditahan, maka nilai akan berubah secara terus menerus dengan perioda tertentu (fitur *repeat*).
- Untuk keluar dari mode *setting*, maka tombol SET harus ditekan lama.

Berikut adalah blok sistem yang digunakan:



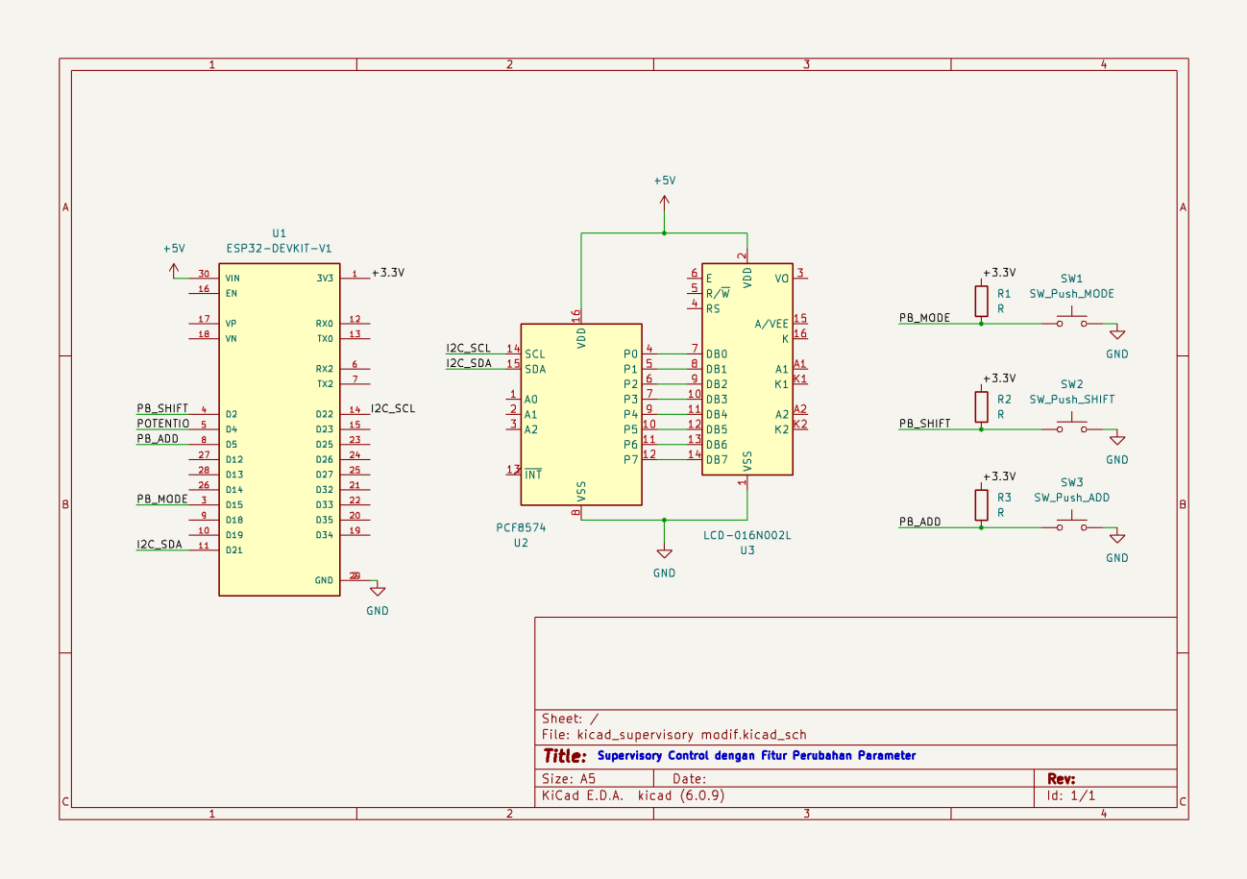
Gambar 1. Diagram Blok Sistem

2 Perancangan Hardware

Pada tugas ini digunakan ESP32 sebagai mikrokontroler yang berperan sebagai *controller* PID sekaligus bertugas untuk menerima input dan menampilkan parameter dari/ke pengguna. Secara ringkas, berikut adalah daftar komponen yang digunakan :

- ESP32 DOIT (1 buah)
- Push Button (3 buah)
- LCD *Display* (1 buah)
- I2C *Module* (1 buah)
- Resistor (3 buah)

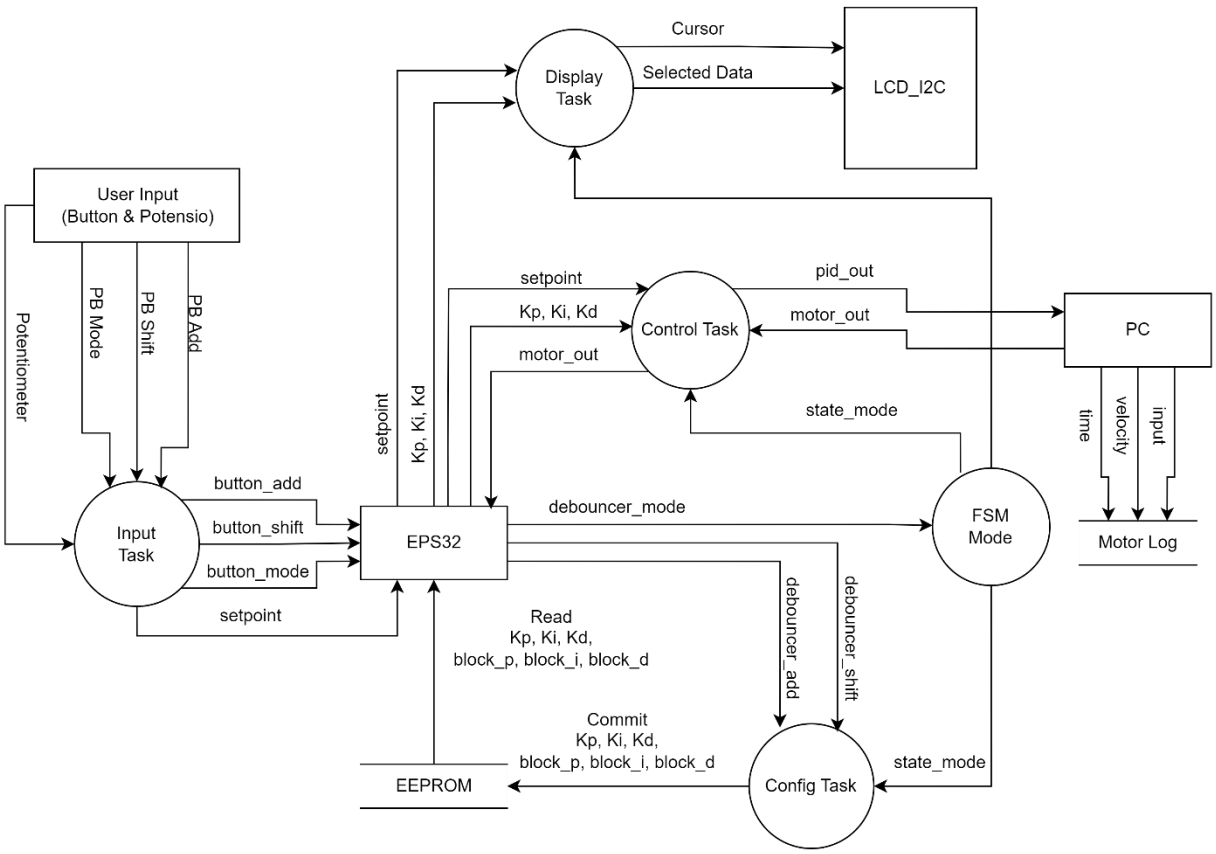
Berikut adalah skematik yang digunakan pada tugas ini.



Gambar 2. Skematik Rangkaian Sistem

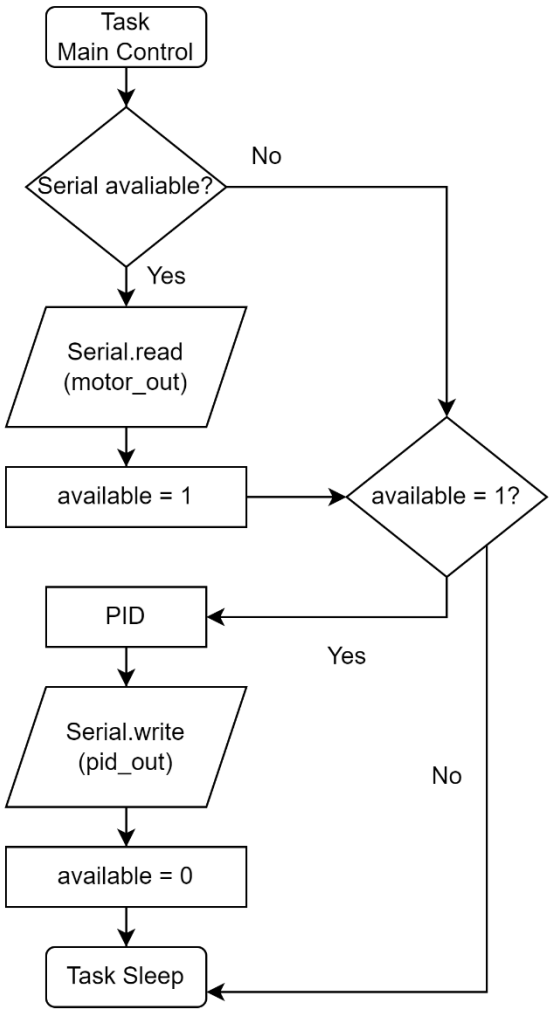
3 Perancangan Software

Pada tugas ini dibuat sistem dengan *Data Flow Diagram* (DFD) sebagai berikut.

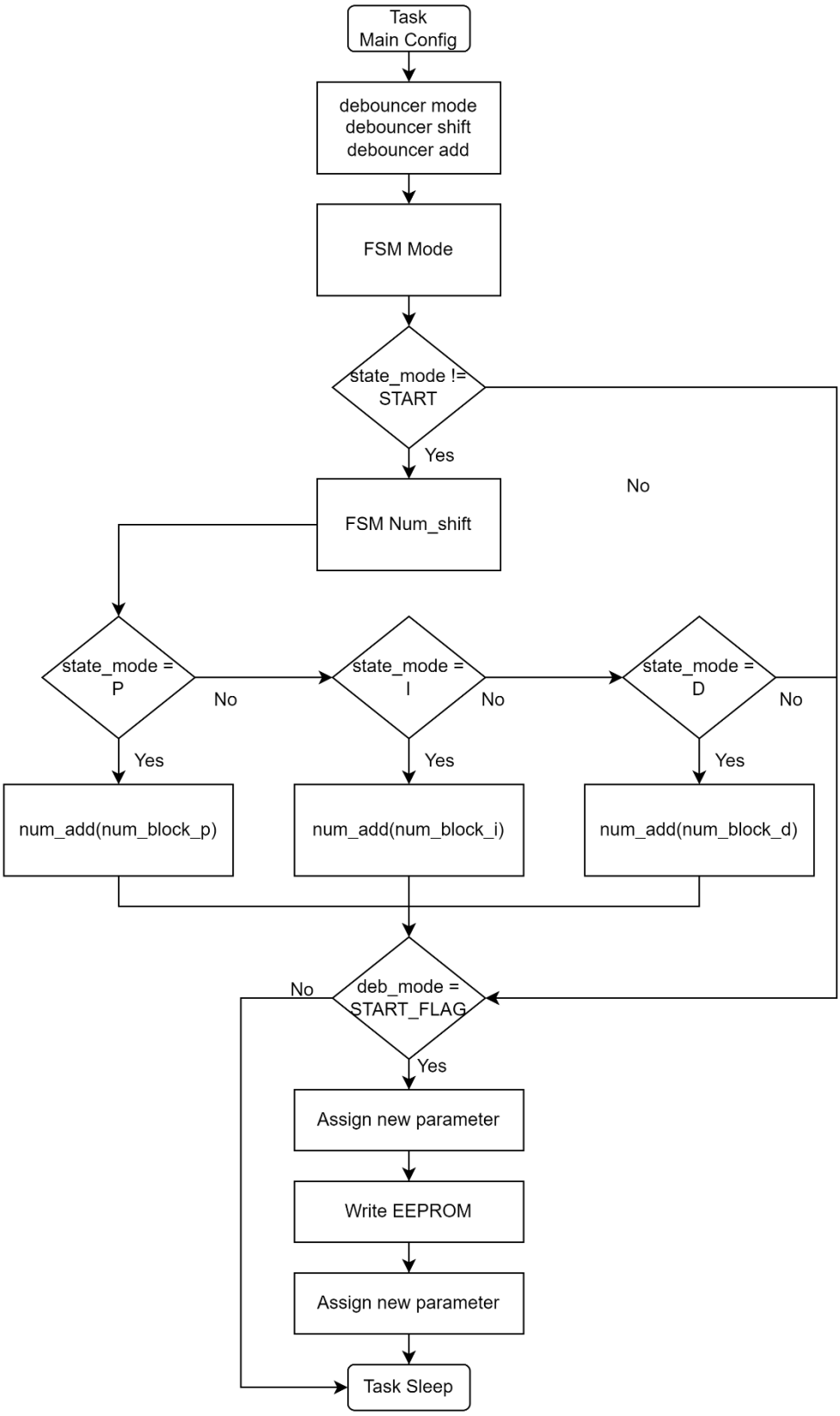


Gambar 3. Data Flow Diagram Sistem

Adapun alur kerja masing-masing diagram dapat dilihat seperti pada gambar berikut.

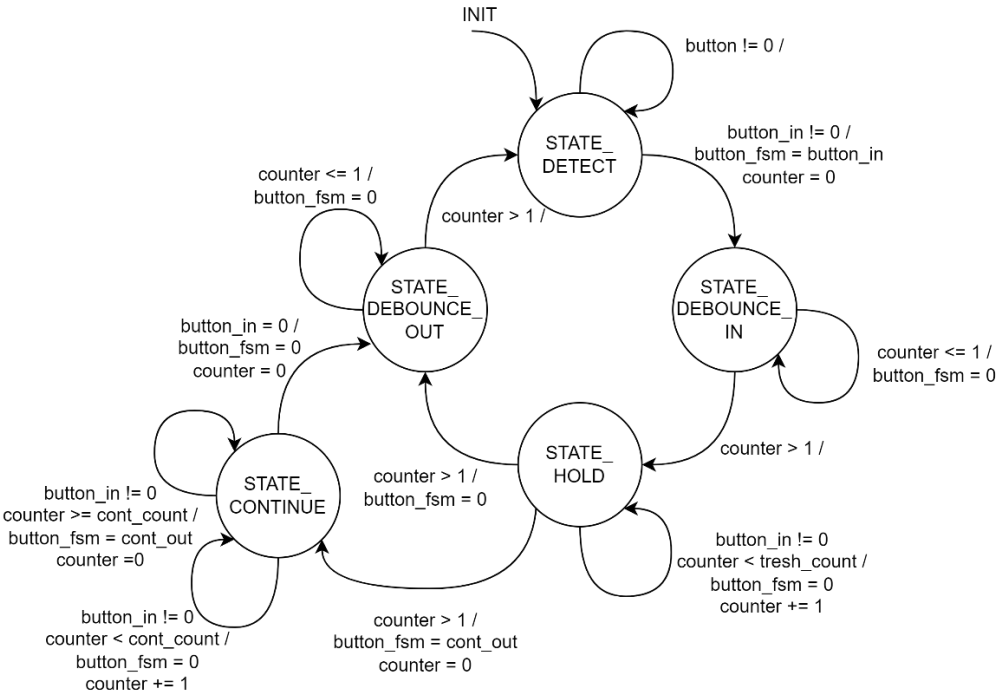


Gambar 4. Flowchart Main Control

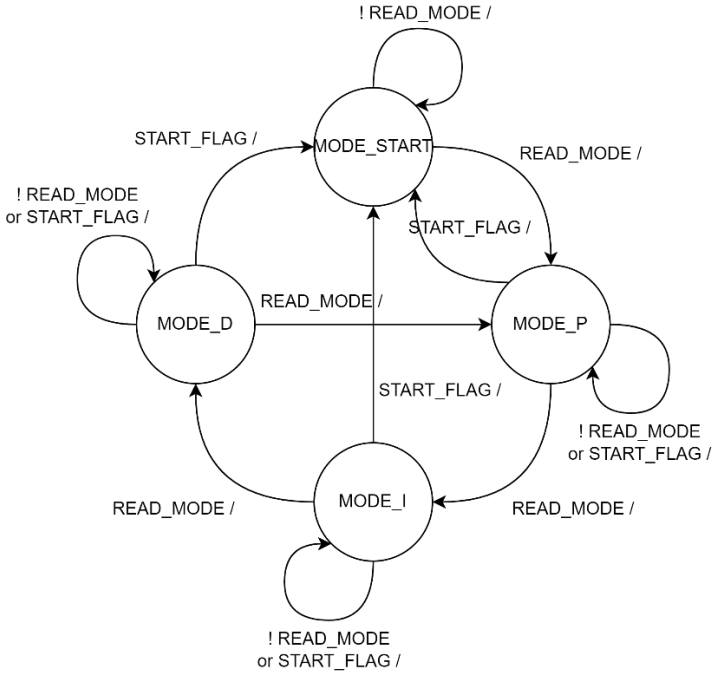


Gambar 5. Flowchart Konfigurasi

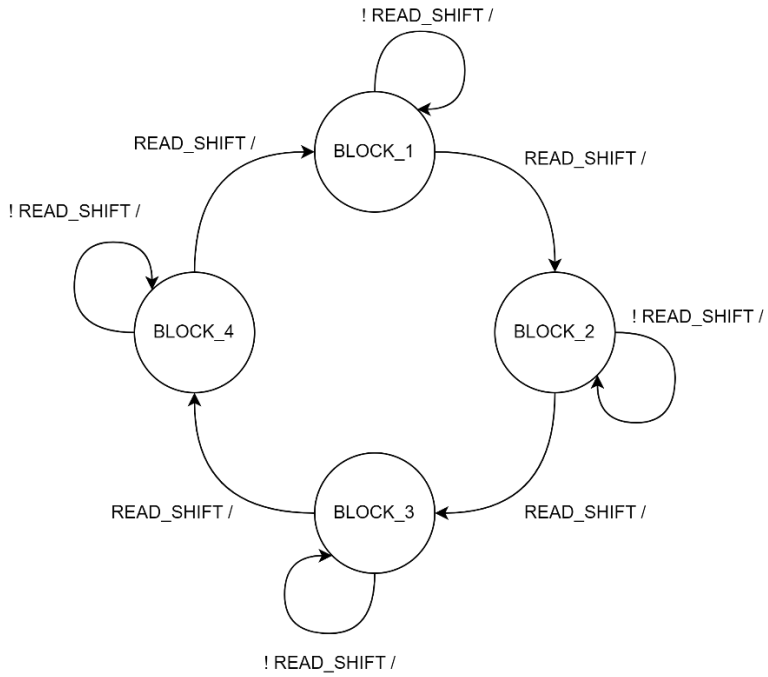
Pada tugas ini, digunakan 3 buah FSM yang berfungsi untuk *debouncing*, memilih *setting*, dan melakukan *shifting* terhadap digit parameter yang ingin diubah.



Gambar 6. Blok FSM Debouncing



Gambar 7. Blok FSM Mode



Gambar 8. Blok FSM Num_Shift

Source code yang digunakan dapat dilihat pada lampiran.

Dari ketiga FSM di atas, dilakukan simulasi terhadap *test input* yang diberikan.

3.1 Simulasi FSM Debouncing

Pada simulasi ini, digunakan *source code* sebagai berikut.

```
#include <stdio.h>
#include "./../../supervisory_control/esp32_arduino/fsm.h"


int case_button[15] = {0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1};
int counter_debounce = 0;
int debounce_out = 0;
int debounce_state = STATE_DETECT;

int main(){
    printf("SIMULASI TEST CASE MODE\n");
    for (int i = 0; i < 15; i++){
        printf("INPUT: %d\tPREV STATE: %d\t", case_button[i], debounce_state);

        debouncer_fsm(case_button[i], 1, 1, case_button[i], &counter_debounce,
&debounce_state, &debounce_out);
```

```
        printf("DEBOUNCE: %d\tCURRENT STATE: %d\n", debounce_out, debounce_state);
    }

    return 0;
}
```

Dari program di atas, didapatkan hasil sebagai berikut.

```
SIMULASI TEST CASE DEBOUNCE
INPUT: 0        PREV STATE: 10  DEBOUNCE: 0     CURRENT STATE: 10
INPUT: 1        PREV STATE: 10  DEBOUNCE: 1     CURRENT STATE: 11
INPUT: 0        PREV STATE: 11  DEBOUNCE: 0     CURRENT STATE: 11
INPUT: 1        PREV STATE: 11  DEBOUNCE: 0     CURRENT STATE: 11
INPUT: 0        PREV STATE: 11  DEBOUNCE: 0     CURRENT STATE: 12
INPUT: 0        PREV STATE: 12  DEBOUNCE: 0     CURRENT STATE: 13
INPUT: 0        PREV STATE: 13  DEBOUNCE: 0     CURRENT STATE: 13
INPUT: 1        PREV STATE: 13  DEBOUNCE: 0     CURRENT STATE: 13
INPUT: 1        PREV STATE: 13  DEBOUNCE: 0     CURRENT STATE: 10
INPUT: 1        PREV STATE: 10  DEBOUNCE: 1     CURRENT STATE: 11
INPUT: 1        PREV STATE: 11  DEBOUNCE: 0     CURRENT STATE: 11
INPUT: 1        PREV STATE: 11  DEBOUNCE: 0     CURRENT STATE: 11
INPUT: 1        PREV STATE: 11  DEBOUNCE: 0     CURRENT STATE: 12
INPUT: 1        PREV STATE: 12  DEBOUNCE: 1     CURRENT STATE: 14
INPUT: 1        PREV STATE: 14  DEBOUNCE: 0     CURRENT STATE: 14
```

Gambar 9. Hasil simulasi FSM Debouncing

Keterangan State:

- 10 : STATE_DETECT
- 11 : STATE_DEBOUNCE_IN
- 12 : STATE_HOLD
- 13 : STATE_DEBOUNCE_OUT
- 14 : STATE_CONTINUE

Pada hasil di atas terlihat bahwa ketika tombol ditekan, maka state akan berubah dari DETECT ke DEBOUNCE_IN. Ketika tombol dilepas, maka output tidak akan membaca hingga *state* berubah menjadi DETECT kembali. Apabila tombol ditekan dan ditahan, maka setelah *state* HOLD, setelah beberapa siklus, maka akan masuk ke *state* CONTINUE.

3.2 Simulasi FSM Mode

Pada simulasi ini, digunakan *source code* sebagai berikut.

```
#include <stdio.h>
#include "./../../supervisory_control/esp32_arduino/fsm.h"

int state_mode = MODE_START;
int case_button[15] = {0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1};
```

```c
int counter_debounce = 0;
int debounce_out = 0;
int debounce_state = STATE_DETECT;

int main(){
    printf("SIMULASI TEST CASE MODE\n");
    for (int i = 0; i < 15; i++){
        printf("INPUT: %d\t", case_button[i]);
        debouncer_fsm(case_button[i], 1, 1, START_FLAG, &counter_debounce,
&debounce_state, &debounce_out);
        if(debounce_out == 1){
            debounce_out = debounce_out * READ_MODE;
        }
        fsm_mode(debounce_out, &state_mode);

        printf("DEBOUNCE: %d\tMODE: %d\n", debounce_out, state_mode);
    }
    return 0;
}
```

Program di atas apabila dijalankan, akan didapatkan hasil sebagai berikut.

```
SIMULASI TEST CASE MODE
INPUT: 0        DEBOUNCE: 0     MODE: 0
INPUT: 1        DEBOUNCE: 32    MODE: 1
INPUT: 0        DEBOUNCE: 0     MODE: 1
INPUT: 1        DEBOUNCE: 0     MODE: 1
INPUT: 0        DEBOUNCE: 0     MODE: 1
INPUT: 0        DEBOUNCE: 0     MODE: 1
INPUT: 0        DEBOUNCE: 0     MODE: 1
INPUT: 1        DEBOUNCE: 0     MODE: 1
INPUT: 1        DEBOUNCE: 0     MODE: 1
INPUT: 1        DEBOUNCE: 32    MODE: 2
INPUT: 1        DEBOUNCE: 0     MODE: 2
INPUT: 1        DEBOUNCE: 0     MODE: 2
INPUT: 1        DEBOUNCE: 0     MODE: 2
INPUT: 1        DEBOUNCE: 9     MODE: 0
INPUT: 1        DEBOUNCE: 0     MODE: 0
```

Gambar 10. Hasil simulasi FSM Mode

Pada hasil simulasi di atas, terlihat bahwa input :1 adalah kondisi pada saat tombol ditekan. Ketika tombol ditekan, maka akan menghasilkan nilai debounce :32 (READ_MODE). Ketika tombol dilepas, maka mode akan melalui state *debounce_out* terlebih dahulu sehingga selama beberapa saat, input tidak akan dibaca. Apabila tombol ditahan, maka setelah beberapa siklus *debounce* akan menghasilkan 9 (START_FLAG) yang akan mengubah *setting mode* menjadi mode : 0 (START_MODE).

3.3 Simulasi FSM Shifting

Pada simulasi ini, digunakan *source code* sebagai berikut.

```
#include <stdio.h>
#include "./../../supervisory_control/esp32_arduino/fsm.h"

int shift_state = BLOCK_1;
int case_button[15] = {1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1};
int counter_debounce = 0;
int debounce_out = 0;
int debounce_state = STATE_DETECT;

int main(){
    printf("SIMULASI TEST CASE SHIFTING\n");
    for (int i = 0; i < 15; i++){
        printf("INPUT: %d\t", case_button[i]);
        debouncer_fsm(case_button[i], 1, 1, case_button[i], &counter_debounce,
&debounce_state, &debounce_out);
        debounce_out = debounce_out * READ_SHIFT;
        fsm_num_shift(debounce_out, &shift_state);

        printf("DEBOUNCE: %d\tBLOCK: %d\n", debounce_out, shift_state);
    }

    return 0;
}
```

Pada program di atas, dilakukan simulasi menggunakan 15 buah input. Input dimasukkan ke dalam *debouncing* terlebih dahulu dan dilanjutkan ke dalam FSM *shifting*. Program di atas menghasilkan keluaran sebagai berikut.

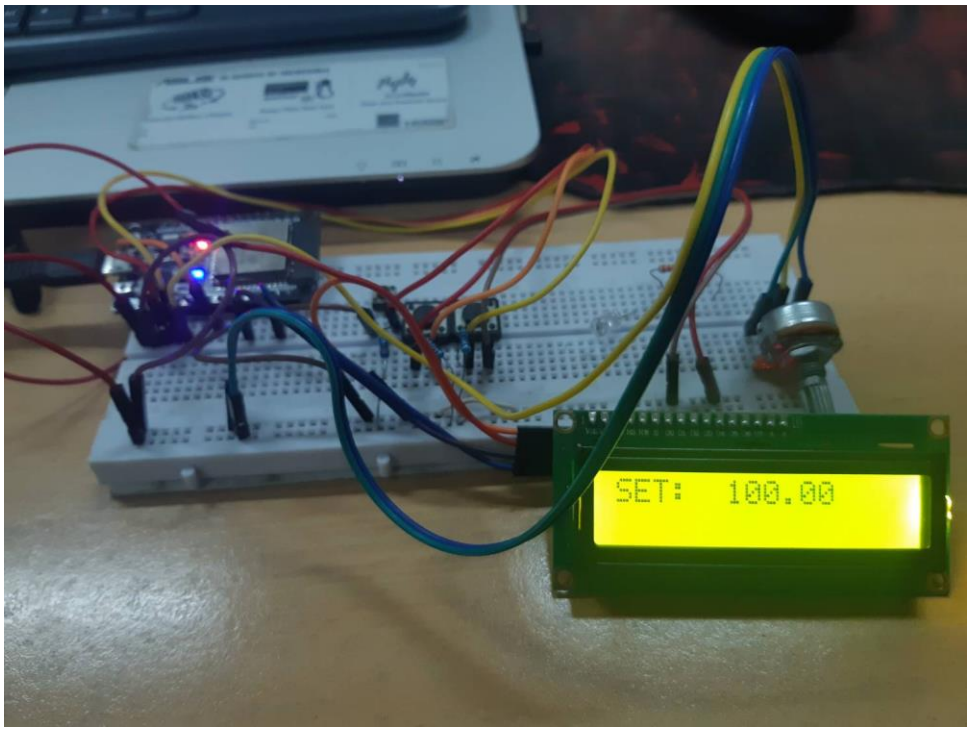
```
SIMULASI TEST CASE SHIFTING
INPUT: 1        DEBOUNCE: 30    BLOCK: 1
INPUT: 0        DEBOUNCE: 0     BLOCK: 1
INPUT: 0        DEBOUNCE: 0     BLOCK: 1
INPUT: 0        DEBOUNCE: 0     BLOCK: 1
INPUT: 1        DEBOUNCE: 30    BLOCK: 2
INPUT: 1        DEBOUNCE: 0     BLOCK: 2
INPUT: 1        DEBOUNCE: 30    BLOCK: 3
INPUT: 1        DEBOUNCE: 0     BLOCK: 3
INPUT: 1        DEBOUNCE: 30    BLOCK: 0
INPUT: 0        DEBOUNCE: 0     BLOCK: 0
INPUT: 0        DEBOUNCE: 0     BLOCK: 0
INPUT: 0        DEBOUNCE: 0     BLOCK: 0
INPUT: 0        DEBOUNCE: 0     BLOCK: 0
INPUT: 1        DEBOUNCE: 30    BLOCK: 1
INPUT: 1        DEBOUNCE: 0     BLOCK: 1
```

Gambar 11. Hasil simulasi FSM Shifting

Pada hasil di atas terlihat bahwa, ketika tombol ditekan, maka akan menghasilkan *debounce :30* (READ_SHIFT). Selama beberapa saat kemudian, input tidak terbaca karena melalui *debouncing*. Namun, apabila tombol ditekan, setelah beberapa siklus, maka terdapat fitur *repeat* yang secara berkala akan menghasilkan output *debounce* yang sama (READ_SHIFT) yang akan mengganti *state* BLOCK.

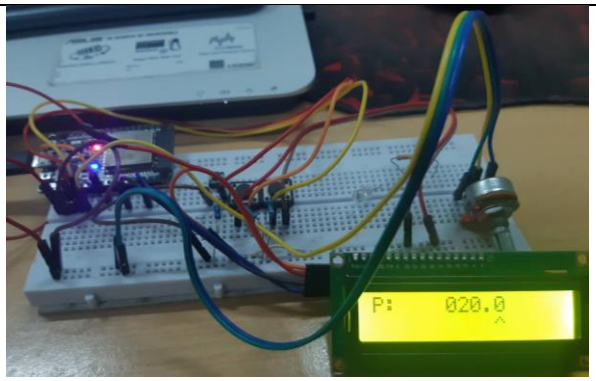
4 Implementasi Sistem

Berikut adalah rangkaian yang telah dibuat oleh penulis pada *breadboard :*

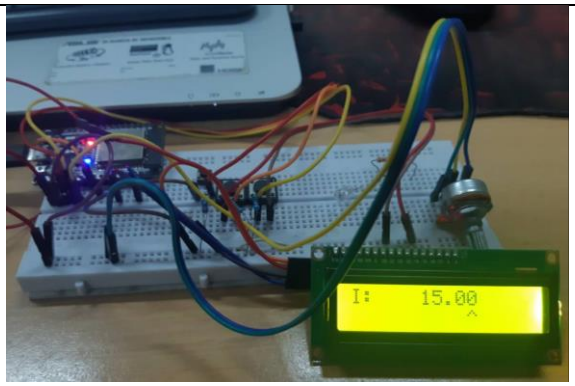


Gambar 12. Percobaan Sistem - Tampilan Awal

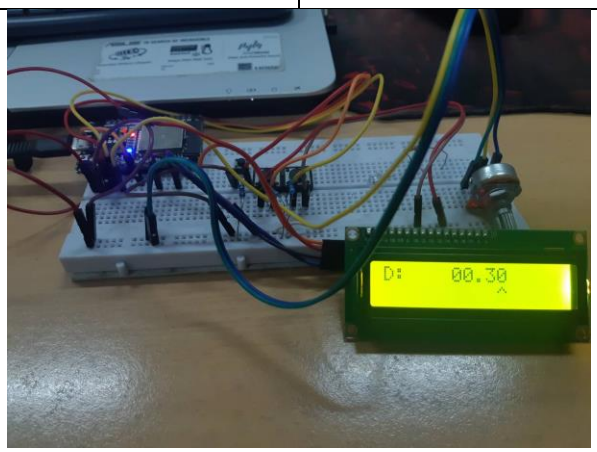
Terdapat 3 buah tombol pada *breadboard*, yaitu tombol MODE, tombol ADD, dan tombol SHIFT. Ketika tombol MODE ditekan, maka tampilan akan berubah dengan urutan P-I-D dengan tampilan seperti berikut.



Gambar 13. Tampilan Setting P



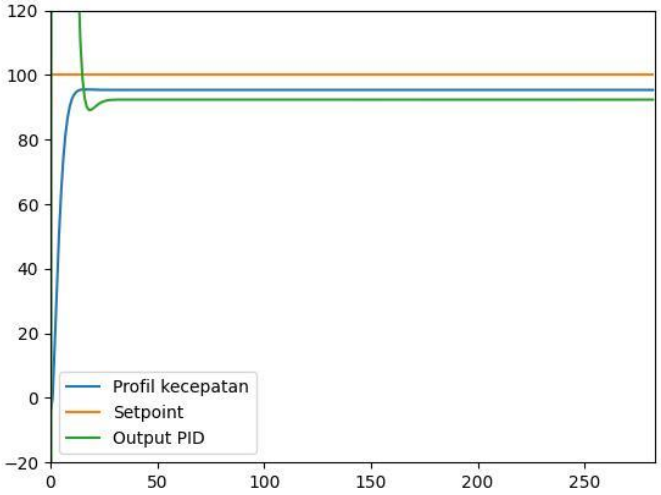
Gambar 14. Tampilan Setting I



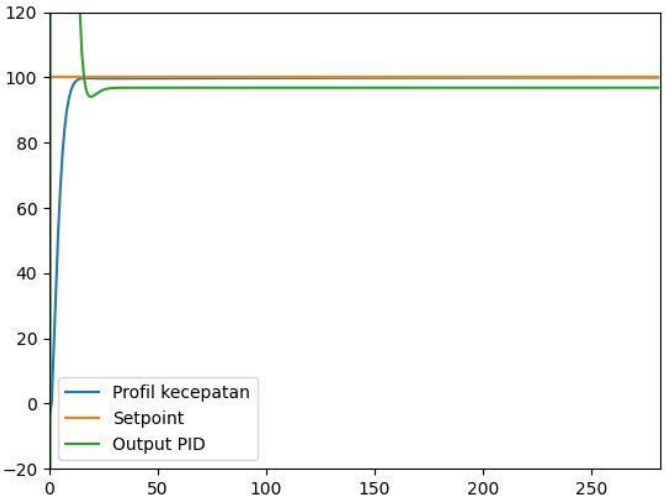
Gambar 15. Tampilan Setting D

Tanda (^) pada *setting* menunjukkan blok *shift* yang akan mengubah angka pada blok tersebut apabila tombol ADD ditekan. Ketika parameter telah ditentukan, maka pengguna hanya perlu menekan tombol MODE selama beberapa detik, dan tampilan akan kembali ke tampilan SET (awal) dan program motor pada PC akan berjalan kembali.

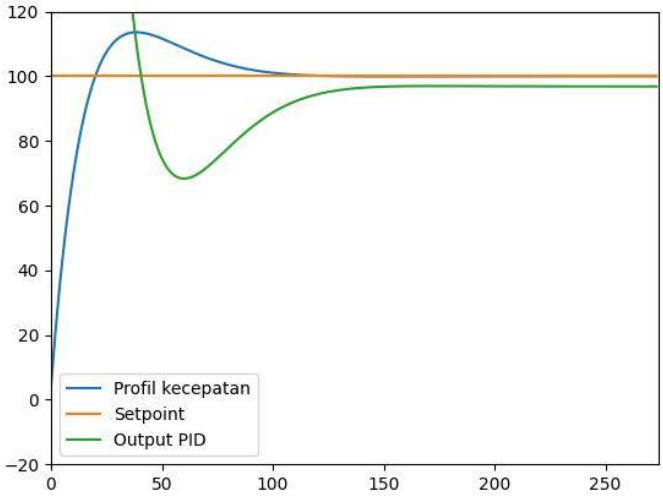
Berikut adalah tampilan pada PC :



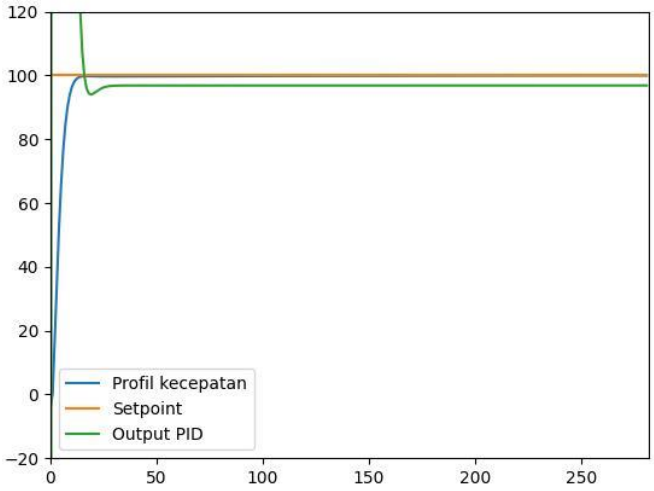
Gambar 16. Data motor PID (20, 0, 0)



Gambar 17. Data motor PID (20, 15, 0)



Gambar 18. Data motor PID (10, 35, 0)



Gambar 19. Data motor PID (20, 15, 0.3)

5 Kesimpulan

Dari percobaan pada tugas ini, didapatkan beberapa kesimpulan sebagai berikut.

- *Supervisory Control* dengan Fitur Parameter yang bisa diubah berhasil dibuat dengan ESP32 sebagai *controller* serta PC sebagai model motor.
- *User Interface* sistem berhasil dibuat dengan menggunakan *push button* sebagai input parameter dan *setting* dan potensiometer sebagai pengatur *setpoint.*
- Parameter yang diubah berhasil disimpan pada EEPROM dan dapat di*update* oleh pengguna.

- Model motor pada PC berhasil dijalankan dengan lancar dengan menampilkan 3 buah nilai, yaitu profil kecepatan motor, *setpoint*, dan output PID.
- Fitur pengubahan parameter disertai dengan *shifting* dan *repeat* yang telah sesuai seperti fitur yang ada pada REX-C100.

6 Referensi

[1] RKC Instrument Inc., "REX-C100/C400/C410/C700/C900 Instruction Manual," Tokyo, 2013.

[2] Arduino.cc, "Arduino Reference," Arduino, [Online]. Available: https://www.arduino.cc/reference/en/libraries/. [Accessed 26 November 2022].

[3] F. d. Brabander, "Arduino-LiquidCrystal-I2C-library," 9 March 2017. [Online]. Available: https://github.com/fdebrabander/Arduino-LiquidCrystal-I2C-library. [Accessed 27 November 2022].

7 Lampiran

7.1 Motor_pc.py

```
import serial
import time
import numpy as np
import matplotlib.pyplot as plt

def motor(input, output):
    output[1] = output[0]
    output[0] = 0.00413 * input[0] + 0.00413 * input[1] + 0.992 * output[1]
    return output

xmin = 0
ser = serial.Serial('COM5', 115200, bytesize=serial.EIGHTBITS,
        parity=serial.PARITY_NONE,
        stopbits=serial.STOPBITS_ONE,
        timeout=None)
plt.axis([0, 1000, -20, 120])
xdata = []
ydata = []
piddata = []
setdata = []

x = [0.0,0.0]
y = [0.0,0.0]
```

```
i = 0

# posisi = 0.0
if ser.is_open:
    # Write a dummy data to start program in Arduino
    ser.write(str("0.0;").encode("utf-8"))

    while (True):
        try:
            size = ser.inWaiting()
            if size:
                x[1] = x[0]
                [setpoint, x[0]] = [float(v) for v in (ser.readline().decode("utf-
8").split(";"))]
                y = motor(x, y)

                ser.write(("%s;" % (str(y[0]))).encode("utf-8"))

                ydata.append(y[0])
                xdata.append(i)
                setdata.append(setpoint)
                piddata.append(x[0])

                i += 1
                print(str(y[0]) + str("    ") + str(x[0]))
                if (i % 10 == 0):
                    plt.clf()
                    plt.axis([0, i, -20, 120])

                    # Plotting velocity, setpoint, adn PID output
                    plt.plot(xdata, ydata, label="Profil kecepatan")
                    plt.plot(xdata, setdata, label="Setpoint")
                    plt.plot(xdata, piddata, label="Output PID")
                    plt.draw()
                    plt.pause(0.03)

        except Exception as e:
            plt.clf()
            plt.axis([0, i, -20, 120])
            plt.plot(xdata, ydata, label="Profil kecepatan")
            plt.plot(xdata, setdata, label="Setpoint")
            plt.plot(xdata, piddata, label="Output PID")
            plt.legend()
            plt.savefig("./hasil.jpg") # Saving picture
            print(e)
```

```
            break
```

7.2 Esp32_arduino.ino

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <EEPROM.h>

#include "fsm.h"

#define PIN_BUTTON_MODE   15
#define PIN_BUTTON_SHIFT  2
#define PIN_BUTTON_ADD    5
#define PIN_POTENTIO      4

LiquidCrystal_I2C lcd(0x27,20,4);  // set the LCD address to 0x27 for a 16 chars
and 2 line display

// BUTTON INPUT SIGNAL
int button_mode = 0, button_shift = 0, button_add = 0;

// DEBOUNCER SIGNAL
int count_deb_shift = 0, count_deb_add = 0, count_deb_mode = 0;
int state_deb_shift = STATE_DETECT, state_deb_add = STATE_DETECT, state_deb_mode =
STATE_DETECT;
int deb_shift = 0, deb_add = 0, deb_mode = 0;

// FSM_MODE SIGNAL
int state_mode = MODE_START;
int counter_mode = 0;

// FSM_NUM_SHIFT SIGNAL
int state_num_shift = BLOCK_1;

// PARAMETER
float Kp, Ki, Kd;

// NUM_ADD SIGNAL
int num_block_p[4] = {0, 0, 0, 0};
int num_block_i[4] = {0, 0, 0, 0};
int num_block_d[4] = {0, 0, 0, 0};

// PID VAR
```

```
float integral = 0;
float last_err = 0;

// SETPOINT (POTENTIO)
float setpoint = 0.0;

// COMMUNICATION WITH MOTOR
int next_valid = 0;
float motor_out = 0.0, pid_out = 0.0;
String buff;
int available = 0;

void pid(float input, float *output, float setpoint, float kp, float ki, float kd,
float time){

    float error = setpoint - input;
    integral += error * time;
    float deriv = (last_err - error)/time;
    last_err = error;
    *output = kp * error + ki * integral + kd * deriv;

}

void read_eeprom(){
  Kp = EEPROM.read(0);
  Ki = EEPROM.read(1);
  Kd = EEPROM.read(2);

  for(int z=0; z<4; z++){
    num_block_p[z] = EEPROM.read(3+z);
  }
  for(int z=0; z<4; z++){
    num_block_i[z] = EEPROM.read(7+z);
  }
  for(int z=0; z<4; z++){
    num_block_d[z] = EEPROM.read(11+z);
  }
}

void write_eeprom(){
  EEPROM.writeFloat(0, Kp);
  EEPROM.writeFloat(1, Ki);
  EEPROM.writeFloat(2, Kd);

  for(int z=0; z<4; z++){
```

```
    EEPROM.write(3+z, num_block_p[z]);
  }

  for(int z=0; z<4; z++){
    EEPROM.write(7+z, num_block_i[z]);
  }

  for(int z=0; z<4; z++){
    EEPROM.write(11+z, num_block_d[z]);
  }
  EEPROM.commit();
}

void setup() {
  EEPROM.begin(64);
  read_eeprom();
  param_assign(num_block_p, num_block_i, num_block_d, &Kp, &Ki, &Kd);
  lcd.init();
  lcd.backlight();
  Serial.begin(115200);
  pinMode(PIN_BUTTON_MODE, INPUT_PULLUP);
  pinMode(PIN_BUTTON_SHIFT, INPUT_PULLUP);
  pinMode(PIN_BUTTON_ADD, INPUT_PULLUP);
  pinMode(PIN_POTENTIO, INPUT);

  // Task untuk button dan display pada core 0
  xTaskCreatePinnedToCore(button_task, "Button Task", 2048, NULL, 2, NULL, 0);
  xTaskCreatePinnedToCore(display_task, "Display Task", 2048, NULL, 1, NULL, 0);
    // Task untuk kontrol PID pada core 1
  xTaskCreatePinnedToCore(main_control, "Main Task", 2048, NULL, 1, NULL, 1);
  xTaskCreatePinnedToCore(main_config, "Main Config Task", 2048, NULL, 2, NULL, 0);
}

void animate_shift(int mode, int shift_block, int num_block[4]){
    int y = 3;
    int dot_pos;
    switch(mode){
        case MODE_P:
            dot_pos = 3;
            break;

        case MODE_I:
            dot_pos = 2;
            break;
```

```
        case MODE_D:
            dot_pos = 2;
            break;

        default:
            break;
    }
    for(int x = 0; x < 5; x++){
        lcd.setCursor(6 + x, 0);
        if(x == dot_pos){
            lcd.print(".");
        }
        else{
            lcd.print(num_block[y]);
            if(y == shift_block){
                lcd.setCursor(6 + x, 1);
                lcd.print("^");
            }
            y--;
        }
    }
}

void display_task(void *pvParam){
  while(1){
    TickType_t xLastWakeTime = xTaskGetTickCount();

    switch(state_mode){
      case MODE_START:
        lcd.clear();
        lcd.setCursor(0,0);
        lcd.print("SET: ");
        lcd.setCursor(6,0);
        lcd.print(setpoint);
        break;

      case MODE_P:
        lcd.clear();
        lcd.setCursor(0,0);
        lcd.print("P: ");
        animate_shift(MODE_P, state_num_shift, num_block_p);
        break;

      case MODE_I:
        lcd.clear();
```

```
        lcd.setCursor(0,0);
        lcd.print("I: ");
        animate_shift(MODE_I, state_num_shift, num_block_i);
        break;

      case MODE_D:
        lcd.clear();
        lcd.setCursor(0,0);
        lcd.print("D: ");
        animate_shift(MODE_D, state_num_shift, num_block_d);
        break;

      default:
        break;
    }

    vTaskDelay(100/portTICK_PERIOD_MS);
  }
}

void button_task(void *pvParam){
  while(1){
    TickType_t xLastWakeTime = xTaskGetTickCount();
        if(digitalRead(PIN_BUTTON_SHIFT)==0){
            button_shift = READ_SHIFT;
        }
        else{
            button_shift = 0;
        }

        if(digitalRead(PIN_BUTTON_ADD) == 0){
            button_add = READ_ADD;
        }
        else{
            button_add = 0;
        }

        if(digitalRead(PIN_BUTTON_MODE) == 0){
            button_mode = READ_MODE;
        }
        else{
            button_mode = 0;
        }

        setpoint = map(analogRead(PIN_POTENTIO), 200, 4095, 0, 100);
```

```
    vTaskDelay(50/portTICK_PERIOD_MS);
  }
}

void main_control(void *pvParam){
  while(1){
    TickType_t xLastWakeTime1 = xTaskGetTickCount();

      while(Serial.available()>0){
        buff = Serial.readStringUntil(';');

        available = 1;
        Serial.flush();
      }
      if (available == 1){
        motor_out = buff.toFloat();
        // Dijalankan setelah pembacaan berhasil
        pid(motor_out, &pid_out, setpoint, Kp, Ki, Kd, 0.01);
        if(state_mode != MODE_START){
          setpoint = 0;
          pid_out = 0;
        }
        // Mengirimkan output_pid ke desktop
        Serial.print(setpoint);
        Serial.print(";");
        Serial.println(pid_out);

        available = 0;
      }

       vTaskDelay(50/portTICK_PERIOD_MS);
  }
}

void main_config(void *pvParam){
  while(1){
    TickType_t xLastWakeTime1 = xTaskGetTickCount();

    debouncer_fsm(button_mode, 20, 20, START_FLAG, &count_deb_mode,
&state_deb_mode, &deb_mode);
    debouncer_fsm(button_shift, 10, 2, READ_SHIFT, &count_deb_shift,
&state_deb_shift, &deb_shift);
    debouncer_fsm(button_add, 10, 2, READ_ADD, &count_deb_add, &state_deb_add,
&deb_add);
```

```
    fsm_mode(deb_mode, &state_mode);
    if(state_mode != MODE_START){
      fsm_num_shift(deb_shift, &state_num_shift);
      switch(state_mode){
        case MODE_P:
          num_add(deb_add, state_num_shift, num_block_p);
          break;

        case MODE_I:
          num_add(deb_add, state_num_shift, num_block_i);
          break;

        case MODE_D:
          num_add(deb_add, state_num_shift, num_block_d);
          break;

        default:
          break;
      }
    }

    if(deb_mode == START_FLAG){
      param_assign(num_block_p, num_block_i, num_block_d, &Kp, &Ki, &Kd);
      write_eeprom();

    }

    vTaskDelay(100/portTICK_PERIOD_MS);
  }
}


void loop() {
  // put your main code here, to run repeatedly:

}
```

7.3 Fsm.h

```
#ifndef FSM_H
#define FSM_H

#include <stdio.h>
```

```c
#include <stdlib.h>
#include <time.h>

#define MODE_START   0
#define MODE_P      1
#define MODE_I      2
#define MODE_D      3

#define STATE_DETECT        10
#define STATE_DEBOUNCE_IN   11
#define STATE_HOLD          12
#define STATE_DEBOUNCE_OUT  13
#define STATE_CONTINUE      14

#define STATE_UP        20
#define STATE_UP_CONT   21
#define STATE_IDLE      22

#define READ_SHIFT      30
#define READ_ADD        31
#define READ_MODE       32

#define START_FLAG      9

#define BLOCK_1         0
#define BLOCK_2         1
#define BLOCK_3         2
#define BLOCK_4         3



void fsm_num_shift(int input, int *state){
    switch(*state){
        case BLOCK_1:
            if(input == READ_SHIFT){
                *state = BLOCK_2;
            }
            break;

        case BLOCK_2:
            if(input == READ_SHIFT){
                *state = BLOCK_3;
            }
            break;
```

```
        case BLOCK_3:
            if(input == READ_SHIFT){
                *state = BLOCK_4;
            }
            break;

        case BLOCK_4:
            if(input == READ_SHIFT){
                *state = BLOCK_1;
            }
            break;

        default:
            *state = BLOCK_1;
            break;
    }
}

void num_add(int button_add, int block_in, int num_out[4]){
    if(button_add == READ_ADD){
        if(num_out[block_in] < 9){
            num_out[block_in] += 1;
        }
        else{
            num_out[block_in] = 0;
        }
    }

}

void fsm_mode(int input, int *state){
    switch(*state){
        case MODE_START:
            if(input == READ_MODE){
                *state = MODE_P;
            }
            break;

        case MODE_P:
            if(input == READ_MODE){
                *state = MODE_I;

            }
            else if(input == START_FLAG){
                *state = MODE_START;
```

```
            }
            break;

        case MODE_I:
            if(input == READ_MODE){
                *state = MODE_D;
            }
            else if(input == START_FLAG){
                *state = MODE_START;
            }
            break;

        case MODE_D:
            if(input == READ_MODE){
                *state = MODE_P;
            }
            else if(input == START_FLAG){
                *state = MODE_START;
            }
            break;

        default:
            *state = MODE_START;
            break;
    }
}


void param_assign(int num_p[4], int num_i[4], int num_d[4], float *p, float *i,
float *d){
    *p = 100 * num_p[3] + 10 * num_p[2] + num_p[1] + 0.1 * num_p[0];
    *i = 10 * num_i[3] + num_i[2] + 0.1 * num_i[1] + 0.01 * num_i[0];
    *d = 10 * num_d[3] + num_d[2] + 0.1 * num_d[1] + 0.01 * num_d[0];
}


void debouncer_fsm(int button_in, int tresh_count, int cont_count, int cont_out,
int *counter, int *state_debounce, int *button_fsm){
    switch(*state_debounce){
        case STATE_DETECT:
            if(button_in != 0){
                *state_debounce = STATE_DEBOUNCE_IN;
                *button_fsm = button_in;
                *counter = 0;
            }
```

```
            break;

        case STATE_DEBOUNCE_IN:
            if(*counter > 1){
                *state_debounce = STATE_HOLD;
            }
            else{
                *button_fsm = 0;
                *counter += 1;
            }
            break;

        case STATE_HOLD:
            if(button_in == 0){
                *state_debounce = STATE_DEBOUNCE_OUT;
                *button_fsm = 0;
                *counter = 0;
            }
            else if(button_in != 0 && *counter < tresh_count){
                *button_fsm = 0;
                *counter += 1;
            }
            else if(button_in != 0 && *counter >= tresh_count){
                *state_debounce = STATE_CONTINUE;
                *button_fsm = cont_out;
                *counter = 0;
            }
            break;

        case STATE_CONTINUE:
            if(button_in == 0){
                *state_debounce = STATE_DEBOUNCE_OUT;
                *button_fsm = 0;
                *counter = 0;
            }
            else if(button_in != 0 && *counter < cont_count){
                *button_fsm = 0;
                *counter += 1;
            }
            else if(button_in != 0 && *counter >= cont_count){
                *button_fsm = cont_out;
                *counter = 0;
            }
            break;
```

```
        case STATE_DEBOUNCE_OUT:
            if(*counter > 1){
                *state_debounce = STATE_DETECT;
            }
            else{
                *button_fsm = 0;
                *counter += 1;
            }
            break;

        default:
            *button_fsm = 0;
            *state_debounce = STATE_DETECT;
            break;
    }
}

#endif
```