



**FUNDAMENTAL OF DIGITAL SYSTEM FINAL PROJECT REPORT
DEPARTMENT OF ELECTRICAL ENGINEERING
UNIVERSITAS INDONESIA**

FLIGHT CONTROLLER UNIT

GROUP IP-X/P-X/R-X

Andi Muhammad Alvin Farhansyah	2306161933
Daffa Bagus Dhiananto	2306250756
Ibnu Zaky Fauzi	2306161870
Samih Bassam	2306250623

PREFACE

Puji syukur kami panjatkan kehadirat Tuhan Yang Maha Esa karena berkat rahmat dan karunia-Nya, laporan proyek akhir ini dapat diselesaikan dengan baik. Laporan ini disusun sebagai bagian dari tugas akhir dalam mata kuliah Perancangan Sistem Digital, yang kami jalani dalam program studi ini. Proyek ini berfokus pada pengembangan FCU (Flight Unit Controller) untuk pesawat tanpa awak, yang merupakan komponen penting dalam sistem penerbangan modern. Dalam proyek ini, kami menggunakan bahasa VHDL untuk merancang dan mengimplementasikan sistem kontrol yang diperlukan.

Dalam laporan ini, kami akan menjelaskan secara rinci mengenai desain, implementasi, dan pengujian FCU yang kami kembangkan menggunakan VHDL. Kami berharap laporan ini dapat memberikan wawasan yang bermanfaat bagi pembaca, serta menjadi referensi bagi penelitian dan pengembangan lebih lanjut di bidang teknologi penerbangan dan sistem digital.

Kami menyadari bahwa penyelesaian laporan ini tidak lepas dari bantuan berbagai pihak. Oleh karena itu, kami ingin mengucapkan terima kasih kepada asisten laboratorium dari Digital Laboratorium yang telah memberikan bimbingan dan arahan, serta kepada teman-teman yang selalu mendukung dan memberikan motivasi selama proses penggerjaan proyek ini. Selain itu, kami juga berterima kasih kepada semua pihak yang telah berkontribusi dalam pengumpulan data dan informasi yang diperlukan. Semoga laporan ini dapat memberikan kontribusi positif bagi dunia pendidikan dan teknologi.

Depok, December 06, 2024

Group IP-X/P-X/R-X

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

- 1.1 Background
- 1.2 Project Description
- 1.3 Objectives
- 1.4 Roles and Responsibilities

CHAPTER 2: IMPLEMENTATION

- 2.1 Equipment
- 2.2 Basic Theory
- 2.3 Implementation

CHAPTER 3: TESTING AND ANALYSIS

- 3.1 Testing
- 3.2 Result
- 3.3 Analysis

CHAPTER 4: CONCLUSION

REFERENCES

APPENDICES

- Appendix A: Project Schematic
- Appendix B: Documentation

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

Perkembangan teknologi penerbangan tanpa awak atau yang lebih dikenal dengan istilah drone telah mengalami kemajuan yang pesat dalam beberapa tahun terakhir. Drone kini digunakan dalam berbagai bidang mulai dari pemantauan lingkungan, pengiriman barang, hingga aplikasi militer. Dengan meningkatnya permintaan akan teknologi ini, penting untuk mengembangkan sistem kontrol yang efisien agar drone dapat beroperasi dengan aman dan efektif. Salah satu komponen kunci dalam sistem ini adalah Flight Unit Controller (FCU) yang bertanggung jawab untuk mengatur dan mengendalikan semua fungsi penerbangan.

FCU berfungsi sebagai sistem saraf dari drone dengan mengintegrasikan berbagai sensor dan aktuator untuk memastikan bahwa pesawat tanpa awak dapat beroperasi sesuai dengan perintah yang diberikan. VHDL memungkinkan perancang untuk mendeskripsikan perilaku dan struktur sistem digital secara mendetail sehingga memudahkan dalam pengembangan dan pengujian sistem kontrol yang kompleks. Dengan VHDL, kami dapat merancang FCU yang tidak hanya efisien tetapi juga dapat diandalkan dalam berbagai kondisi penerbangan.

Dalam proyek ini, kami berfokus pada pengembangan FCU menggunakan VHDL untuk meningkatkan performa dan stabilitas drone. Melalui penelitian dan pengembangan yang mendalam, kami berharap dapat memberikan kontribusi yang signifikan dalam bidang teknologi penerbangan tanpa awak.

1.2 PROJECT DESCRIPTION

FCU (Flight Unit Controller) adalah komponen penting yang digunakan dalam pesawat tanpa awak kecil. FCU berfungsi sebagai 'sistem saraf' pesawat, yang menerima perintah dari komputer atau mikrokontroler untuk mengendalikan servo dan motor. Dengan cara ini, pesawat dapat bergerak ke arah yang diinginkan, mencapai ketinggian tertentu, dan mengatur kecepatan sesuai kebutuhan. Dalam proyek ini, kami akan mengembangkan FCU

yang dilengkapi dengan modul PID Controller. Modul ini akan membantu menjaga stabilitas dan navigasi pesawat dengan menghitung tiga komponen penting: Proportional, Integral, dan Derivative. Selain itu, FCU akan memiliki empat mode penerbangan, yaitu Idle, Takeoff, Hover, Cruise, dan Land, yang memungkinkan pesawat beradaptasi dengan berbagai situasi saat terbang.

FCU juga akan dilengkapi dengan unit pemrosesan sinyal yang menerima informasi dari sensor seperti LiDAR untuk mengukur ketinggian pesawat. Untuk mengendalikan setiap motor, FCU akan mengirimkan sinyal PWM (Pulse Width Modulation) yang dihasilkan dari output PID Controller. Dalam hal komunikasi data, FCU akan menggunakan protokol sederhana seperti SPI Slave atau UART untuk memastikan pertukaran informasi yang cepat dan efisien antara komponen.

1.3 OBJECTIVES

Tujuan dari proyek ini adalah sebagai berikut:

1. Meningkatkan stabilitas penerbangan.
2. Meningkatkan responsivitas sistem.
3. Mengintegrasikan sensor yang efisien
4. Membangun sistem komunikasi yang efisien

1.4 ROLES AND RESPONSIBILITIES

Roles	Responsibilities	Person
Programmer utama Source Code VHDL	Membuat kode VHDL untuk Flight Control Unit-nya. Mulai dari PID Controller sampai Flight Control Unit Utama	Andi Muhammad Alvin Farhanyah
Pembuat Laporan, Testing, dan simulasi	- Membuat dan menyusun laporan ini	Ibnu Zaky Fauzi

	<ul style="list-style-type: none"> - Melakukan Testing pada kode VHDL - Melakukan Sintesis pada kode VHDL 	
Pembuat Laporan	<ul style="list-style-type: none"> - Membuat dan menyusun laporan ini 	Samih Bassam
Pembuat Laporan	<ul style="list-style-type: none"> - Membuat dan menyusun laporan ini - Membuat PowerPoint Presentasi 	Daffa Bagus Dhiananto

Table 1. Roles and Responsibilities

CHAPTER 2

IMPLEMENTATION

2.1 EQUIPMENT

- VS Code
- ModelSim
- Intel Quartus Prime

2.2 BASIC THEORY

Proportional-Integral-Derivative (PID) controller adalah alat yang digunakan untuk mengatur variabel seperti suhu, laju aliran, kecepatan, tekanan, dan tegangan dengan menerima data dari sensor dan menghitung perbedaan antara nilai aktual dan nilai yang diinginkan. Alat ini bekerja melalui tiga mekanisme utama, yaitu kontrol

proporsional (P) yang memberikan respons sebanding dengan kesalahan saat ini, kontrol integral (I) yang menjumlahkan kesalahan dari waktu ke waktu untuk mengatasi kesalahan yang terakumulasi, dan kontrol derivative (D) yang memprediksi kesalahan di masa depan berdasarkan laju perubahan kesalahan saat ini. Dengan menggabungkan ketiga komponen ini, PID controller dapat menjaga proses tetap stabil dan efisien.

PID Controller efektif karena dapat mempertimbangkan kondisi masa lalu sistem melalui komponen integral dan juga memperkirakan perilaku sistem di masa depan lewat komponen derivative. Pada sistem loop tertutup, PID controller berfungsi untuk meningkatkan kinerja sistem dengan menyeimbangkan pengaruh dari parameter P (proporsional), I (integral), dan D (derivative) terhadap perilaku sistem. Ketiga komponen ini digabungkan menjadi rumus kontrol PID secara keseluruhan:

$$u(t) = K_p e(t) + K_i \int e(t) dt + K_d \frac{de}{dt}$$

dimana $u(t)$ adalah output controller. Transfer function PID controller ditemukan dengan mengambil transformasi Laplace dari persamaan:

$$K_p + \frac{K_i}{s} + K_d s = \frac{K_d s^2 + K_p s + K_i}{s}$$

dimana K_p = proportional gain, K_i = integral gain, dan K_d = derivative gain.

Meningkatkan proportional gain (K_p) akan meningkatkan sinyal kontrol sebanding dengan besarnya kesalahan sehingga sistem akan merespons lebih cepat namun berisiko overshoot lebih besar. Menambahkan derivative gain (K_d) memungkinkan pengendali untuk memprediksi kesalahan, mengurangi overshoot dengan memberikan sinyal kontrol lebih besar saat kesalahan mulai meningkat, tetapi tidak mempengaruhi kesalahan pada steady-state. Sementara itu, menambahkan integral gain (K_i) membantu mengurangi kesalahan pada steady-state dengan meningkatkan sinyal kontrol saat kesalahan tetap ada, namun dapat memperlambat respons sistem dan menyebabkan osilasi.

CL RESPONSE	RISE TIME	OVERSHOOT	SETTLING TIME	S-S Error

Kp	Decrease	Increase	Small change	Decrease
Ki	Decrease	Increase	Increase	Decrease
Kd	Small change	Decrease	Decrease	No change

Dalam sistem kontrol, variabel proses adalah parameter yang dikendalikan, seperti suhu, tekanan, atau laju aliran yang diukur oleh sensor untuk memberikan umpan balik ke sistem. Titik setel adalah nilai target yang ingin dicapai oleh variabel tersebut. Jika terdapat perbedaan antara nilai aktual dan titik setel (kesalahan), algoritma kontrol akan menentukan berapa banyak output yang perlu diubah untuk menggerakkan aktuator. Proses ini terjadi secara berkelanjutan dalam sistem kontrol loop tertutup dimana sistem membaca data sensor, menghitung kesalahan, dan menyesuaikan output.

Dari tabel di atas, dapat dilihat bahwa pengendali proporsional (Kp) mengurangi rise time, meningkatkan overshoot, dan mengurangi kesalahan pada steady-state. Fungsi transfer loop tertutup untuk sistem umpan balik satuan dengan pengendali proporsional adalah sebagai berikut:

$$T(s) = \frac{X(s)}{R(s)} = \frac{K_p}{s^2 + 10s + (20 + K_p)}$$

Dari tabel di atas, dapat dilihat bahwa penambahan kontrol derivative (Kd) cenderung mengurangi baik overshoot maupun waktu penetapan (settling time). Fungsi transfer loop tertutup untuk sistem yang diberikan dengan PD controller adalah:

$$T(s) = \frac{X(s)}{R(s)} = \frac{K_d s + K_p}{s^2 + (10 + K_d)s + (20 + K_p)}$$

Dari tabel, dapat dilihat bahwa penambahan kontrol integral (Ki) cenderung mengurangi waktu kenaikan (rise time), meningkatkan overshoot dan waktu penetapan (settling time), serta mengurangi kesalahan pada steady-state. Untuk sistem yang diberikan, fungsi transfer loop tertutup dengan pengendali PI adalah:

$$T(s) = \frac{X(s)}{R(s)} = \frac{K_p s + K_i}{s^3 + 10s^2 + (20 + K_p)s + K_i}$$

Fungsi transfer loop tertutup untuk sistem yang diberikan dengan PID controller adalah:

$$T(s) = \frac{X(s)}{R(s)} = \frac{K_d s^2 + K_p s + K_i}{s^3 + (10 + K_d)s^2 + (20 + K_p)s + K_i}$$

2.3 IMPLEMENTATION

2.3.1 PID Controller

Sistem PID Controller terdiri dari beberapa sub-komponen, seperti Analog Converter, Integration, Error Register, Trigger, dan komponen PWM Generator. Sistem ini berpusat pada komponen utama, yaitu PID Controller yang digunakan untuk mengatur sinyal output berdasarkan tiga parameter utama yaitu: Proportional (P), Integral (I), dan Derivative (D). Kontroler membaca nilai referensi (ref_val) dan masukan ADC (adc_in) untuk menghitung error. Berdasarkan status enable (p_en, i_en, d_en), setiap komponen PID dihitung. Komponen Proportional (P) menentukan nilai berdasarkan error saat ini yang dikalikan dengan konstanta P_NUM / P_DEN. Komponen Integral (I) menambahkan akumulasi error dari modul Integration, sedangkan komponen Derivative (D) menghitung perubahan error dengan menghitung selisih antara error saat ini dan error sebelumnya. Output akhir dijumlahkan dan dikontrol dalam batas tertentu (0 hingga 4000) untuk menghasilkan sinyal PWM yang digunakan untuk mengontrol aktuator dalam sistem.

Bagian Analog Converter bertugas mengubah sinyal analog dari sensor menjadi data digital yang dapat diproses oleh logika digital. Sub-komponen ini menggunakan Analog-to-Digital Converter untuk membaca nilai dari sensor, seperti suhu, posisi, atau kecepatan, dan mengubahnya menjadi representasi digital. Data digital ini kemudian digunakan sebagai umpan balik untuk menghitung error antara nilai aktual dan setpoint. Komponen Analog Converter juga berfungsi sebagai interface untuk membaca sinyal analog dari pin masukan (8 bit) dan mengonversinya menjadi data digital menggunakan modul XADC. Proses konversi dimulai dengan sinyal ‘start_conv’ pada input clock ‘sys_clock’. Setelah selesai, sinyal ‘conv_done’ menunjukkan bahwa data digital sudah siap untuk digunakan. Berikut adalah kode VHDL untuk AnalogConverter,

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
```

```

USE IEEE.NUMERIC_STD.ALL;

ENTITY AnalogConverter IS

    PORT (
        sys_clock      : IN STD_LOGIC;
        start_conv     : IN std_logic;
        conv_done      : out std_logic;
        adc_result     : OUT std_logic_vector (15 DOWNTO 0);
        analog_pins   : IN STD_LOGIC_VECTOR (7 DOWNTO 0)
    );
END AnalogConverter;

ARCHITECTURE Behavioral OF AnalogConverter IS

    -- Signal declarations
    SIGNAL data_out : std_logic_vector(15 DOWNTO 0) := (OTHERS => '0');
    SIGNAL mux_addr : std_logic_vector(4 DOWNTO 0) := (OTHERS => '0');
    SIGNAL chan_sel : std_logic_vector(4 DOWNTO 0) := (OTHERS => '0');
    SIGNAL aux_neg : std_logic_vector(15 DOWNTO 0) := (OTHERS => '0');
    SIGNAL aux_pos : std_logic_vector(15 DOWNTO 0) := (OTHERS => '0');

    -- XADC Component Declaration
COMPONENT XADC
    GENERIC (
        INIT_40 : bit_vector := X"0000";
        INIT_41 : bit_vector := X"0000";
        INIT_42 : bit_vector := X"0800";
        INIT_48 : bit_vector := X"0000";
        INIT_49 : bit_vector := X"0000";
        INIT_4A : bit_vector := X"0000";
    
```

```

INIT_4B : bit_vector := X"0000";
INIT_4C : bit_vector := X"0000";
INIT_4D : bit_vector := X"0000";
INIT_4E : bit_vector := X"0000";
INIT_4F : bit_vector := X"0000";
INIT_50 : bit_vector := X"0000";
INIT_51 : bit_vector := X"0000";
INIT_52 : bit_vector := X"0000";
INIT_53 : bit_vector := X"0000";
INIT_54 : bit_vector := X"0000";
INIT_55 : bit_vector := X"0000";
INIT_56 : bit_vector := X"0000";
INIT_57 : bit_vector := X"0000";
INIT_58 : bit_vector := X"0000";
INIT_5C : bit_vector := X"0000";
SIM_DEVICE : string := "7SERIES";
SIM_MONITOR_FILE : string := "design.txt"
);

PORT (
    DADDR : IN STD_LOGIC_VECTOR(6 DOWNTO 0);
    DCLK : IN STD_LOGIC;
    DEN : IN STD_LOGIC;
    DI : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
    DWE : IN STD_LOGIC;
    RESET : IN STD_LOGIC;
    VAUXN : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
    VAUXP : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
    CONVST : IN STD_LOGIC;
    CONVSTCLK : IN STD_LOGIC;

```

```

VN : IN STD_LOGIC;
VP : IN STD_LOGIC;
DO : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
DRDY : OUT STD_LOGIC;
EOC : OUT STD_LOGIC;
EOS : OUT STD_LOGIC;
BUSY : OUT STD_LOGIC;
CHANNEL : OUT STD_LOGIC_VECTOR(4 DOWNTO 0);
JTAGLOCKED : OUT STD_LOGIC;
JTAGMODIFIED : OUT STD_LOGIC;
JTAGBUSY : OUT STD_LOGIC;
ALM : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
OT : OUT STD_LOGIC;
MUXADDR : OUT STD_LOGIC_VECTOR(4 DOWNTO 0)
);

END COMPONENT;

```

```

BEGIN
    adc_result <= data_out;

    -- Analog input mapping
    aux_pos(6) <= analog_pins(0);
    aux_neg(6) <= analog_pins(4);
    aux_pos(14) <= analog_pins(1);
    aux_neg(14) <= analog_pins(5);
    aux_pos(7) <= analog_pins(2);
    aux_neg(7) <= analog_pins(6);
    aux_pos(15) <= analog_pins(3);
    aux_neg(15) <= analog_pins(7);

```

```
XADC_inst : XADC
  GENERIC MAP (
    -- Same INIT values as original
    INIT_40 => X"9000",
    INIT_41 => X"2ef0",
    INIT_42 => X"0800",
    INIT_48 => X"4701",
    INIT_49 => X"00CC",
    INIT_4A => X"0000",
    INIT_4B => X"0000",
    INIT_4C => X"0000",
    INIT_4D => X"00CC",
    INIT_4E => X"0000",
    INIT_4F => X"0000",
    INIT_50 => X"b5ed",
    INIT_51 => X"5999",
    INIT_52 => X"A147",
    INIT_53 => X"dddd",
    INIT_54 => X"a93a",
    INIT_55 => X"5111",
    INIT_56 => X"91Eb",
    INIT_57 => X"ae4e",
    INIT_58 => X"5999",
    INIT_5C => X"5111",
    SIM_DEVICE => "7SERIES",
    SIM_MONITOR_FILE => "design.txt"
  )
  PORT MAP (
```

```

ALM => OPEN,
OT => OPEN,
BUSY => OPEN,
CHANNEL => chan_sel,
EOC => OPEN,
EOS => OPEN,
JTAGBUSY => OPEN,
JTAGLOCKED => OPEN,
JTAGMODIFIED => OPEN,
MUXADDR => mux_addr,
VAUXN => aux_neg,
VAUXP => aux_pos,
CONVST => start_conv,
CONVSTCLK => '0',
RESET => '0',
VN => '0',
VP => '0',
DO => data_out,
DRDY => conv_done,
DADDR => "0010110",
DCLK => sys_clock,
DEN => '1',
DI => (OTHERS => '0'),
DWE => '0'

);

END Behavioral;

```

Sub-komponen Error Register digunakan untuk menyimpan nilai error yang dihitung berdasarkan perbedaan antara nilai setpoint dan nilai feedback (hasil konversi dari Analog

Converter). Sub-komponen ini memungkinkan sistem menyimpan nilai kesalahan dari satu siklus ke siklus berikutnya yang diperlukan untuk menghitung bagian Integral dan Derivative dalam PID Controller. Error Register bekerja dengan menyimpan error sebelumnya (Error_prev) setiap kali sinyal trigger aktif sehingga memfasilitasi kalkulasi error derivative dengan membandingkan error saat ini dengan error sebelumnya. Berikut adalah kode VHDL dari sub-komponen Error Register,

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity error_register is
    port (
        Error_in : in std_logic_vector(31 downto 0);
        Error_prev : out std_logic_vector(31 downto 0);
        trig : in std_logic
    );
end error_register;

architecture Behavioral of error_register is
begin
    process(trig)
    begin
        if rising_edge(trig) then
            Error_prev <= Error_in;
        end if;
    end process;
end Behavioral;
```

Selanjutnya, bagian Integration bertugas menghitung akumulasi error untuk proses integral dalam PID Controller. Proses ini melibatkan register yang menyimpan error sebelumnya dan menghitung nilai akumulasi baru berdasarkan error saat ini. Kalkulasi dilakukan menggunakan konstanta ‘ki_numerator’, ‘ki_denominator’, dan ‘time_for_divider’. Nilai akumulasi diatur dengan batas atas dan bawah untuk memastikan stabilitas sistem dan pembaruan dilakukan secara kontinu setiap kali trigger aktif. Berikut adalah kode VHDL dari Integration,

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity Integration is
    port (
        ki_numerator : in std_logic_vector(15 downto 0);
        ki_denominator : in std_logic_vector(15 downto 0);
        time_for_divider : in std_logic_vector(15 downto 0);
        integral_on : in std_logic;
        trigger : in std_logic;
        sum : in std_logic_vector(31 downto 0);
        error : in std_logic_vector(31 downto 0);
        new_sum : out std_logic_vector(31 downto 0);
        error_difference : out std_logic_vector(31 downto 0);
        error_for_pid : out std_logic_vector(31 downto 0)
    );
end Integration;

architecture Behavioral of Integration is
    signal acc_total, curr_err, new_acc, old_err, err_delta : INTEGER
    := 0;

```

```

    signal old_err_buff : std_logic_vector(31 DOWNTO 0) := (others =>
'0');

component err_register
port (
    Error_in : in std_logic_vector(31 downto 0);
    Error_prev : out std_logic_vector(31 downto 0);
    trig : in std_logic
);
end component;

begin
ERR_FF : err_register
port map (
    Error_in => error,
    Error_prev => old_err_buff,
    trig => trigger
);

process (ki_denominator, ki_numerator, time_for_divider,
old_err_buff, trigger, sum, error, new_acc, err_delta)
begin
    -- Convert inputs to integers
    acc_total <= to_integer(signed(sum));
    curr_err <= to_integer(signed(error));
    old_err <= to_integer(signed(old_err_buff));

    -- Check limits and set new_sum

```

```

        if      new_acc      <
((3072*to_integer(unsigned(ki_denominator))*to_integer(unsigned(time_for_divider)))/to_integer(unsigned(ki_numerator))) then
            new_sum      <=
std_logic_vector(to_signed(((3072*to_integer(unsigned(ki_denominator))*to_integer(unsigned(time_for_divider)))/to_integer(unsigned(ki_numerator))), 32));
        elsif      new_acc      >
((4000*to_integer(unsigned(ki_denominator))*to_integer(unsigned(time_for_divider)))/to_integer(unsigned(ki_numerator))) then
            new_sum      <=
std_logic_vector(to_signed(((4000*to_integer(unsigned(ki_denominator))*to_integer(unsigned(time_for_divider)))/to_integer(unsigned(ki_numerator))), 32));
        else
            new_sum <= std_logic_vector(to_signed(new_acc, 32));
        end if;

error_difference <= std_logic_vector(to_signed(err_delta, 32));

if rising_edge(trigger) then
    if integral_on = '1' then
        new_acc <= acc_total + curr_err; -- Integration
    else
        new_acc <= 0;
    end if;
    err_delta <= curr_err - old_err; -- Error difference calculation
    error_for_pid <= error; -- Pass through error for PID
end if;
end process;

end Behavioral;

```

PWM Generator adalah komponen yang menghasilkan sinyal kendali untuk aktuator berdasarkan output PID Controller. Komponen ini mengubah hasil kalkulasi PID menjadi sinyal PWM dengan ukuran yang bervariasi. Sinyal PWM ini digunakan untuk mengontrol perangkat seperti motor, pemanas, atau aktuator lainnya dimana semakin besar hasil PID, semakin besar ukuran PWM yang dihasilkan. Komponen PWM Generator bekerja dengan frekuensi clock tertentu (clock_freq) dan menghasilkan sinyal sesuai duty cycle yang diatur berdasarkan input ‘duty_in’. Berikut adalah kode dari PWM Generator,

```

Library IEEE;
Use IEEE.STD_LOGIC_1164.All;
Use IEEE.NUMERIC_STD.All;

Entity PWM_Generator Is
    Generic (
        clock_freq : Integer := 100000000;
        pwm_freq   : Integer := 25000
    );
    Port (
        enable      : In std_logic;
        duty_in     : In STD_LOGIC_VECTOR (11 Downto 0);
        clock       : In STD_LOGIC;
        pwm_out     : Out STD_LOGIC
    );
End PWM_Generator;

Architecture Behavioral Of PWM_Generator Is
    Constant pwm_period      : Integer := clock_freq/pwm_freq;
    Signal count             : Integer := 0;
    Signal enable_reg        : std_logic := '0';

```

```

Signal duty_reg          : unsigned(11 Downto 0) := "000000000000";
signal idle_duty        : integer := 3072;

Begin

  Process (clock, duty_in, count, enable, idle_duty, duty_reg)
    Begin

      If clock'EVENT And clock = '1' Then

        If (count < 4000) Then      -- 400us period limit

          count <= count + 2;

        Else

          count <= 0;

        End If;

      End If;

      If count <= duty_reg Then      -- PWM generation based on duty
cycle

        pwm_out <= '1';

      Else

        pwm_out <= '0';

      End If;

      If enable = '1' Then          -- Duty cycle control

        duty_reg <= unsigned(duty_in);

      Else

        duty_reg <= to_unsigned(idle_duty, 12);

      End If;

    End Process;

  End Behavioral;

```

Terakhir, sub-komponen Trigger mengatur kapan PID Controller melakukan perhitungan atau memperbarui outputnya. Sub-komponen ini dapat dipicu oleh sinyal eksternal, seperti sensor yang memberikan indikasi data baru atau berdasarkan siklus waktu tertentu. Trigger memastikan bahwa PID Controller bekerja secara sinkron dengan perangkat keras lainnya. Sub-komponen Trigger menghasilkan sinyal periodik (adc_trigger_out) menggunakan counter internal yang bekerja dalam interval waktu tertentu sesuai dengan konstanta ‘TRIGGER_START’ dan ‘TRIGGER_END’. Dengan koordinasi ini, PID Controller dapat beroperasi secara efisien dan terintegrasi dengan perangkat lainnya dalam sistem. Berikut adalah kode VHDL dari Trigger,

```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;

ENTITY trigger IS
    PORT (
        adc_trigger_out : OUT STD_LOGIC;
        clock_in        : IN  STD_LOGIC
    );
END trigger;

ARCHITECTURE Behavioral OF trigger IS
    -- Counter signals for timing control
    SIGNAL inner_count : INTEGER := 0;
    SIGNAL outer_count : INTEGER := 0;

    -- Output buffer
    SIGNAL trigger_buffer : std_logic := '0';

    -- Constants for timing
    CONSTANT INNER_LIMIT : INTEGER := 50;

```

```

CONSTANT TRIGGER_START : INTEGER := 9800;
CONSTANT TRIGGER_END : INTEGER := 9803;

BEGIN

trigger_process : PROCESS (clock_in)
BEGIN

-- Connect buffer to output

adc_trigger_out <= trigger_buffer;

-- Check if inner counter reached limit

IF inner_count = INNER_LIMIT THEN

-- Reset inner counter and increment outer counter

inner_count <= 0;

outer_count <= outer_count + 1;

-- Trigger control logic

IF outer_count = TRIGGER_START THEN

trigger_buffer <= '1';

ELSIF outer_count = TRIGGER_END THEN

-- Reset outer counter

outer_count <= 0;

END IF;

ELSE

trigger_buffer <= '0';

END IF;

```

```

        END IF;

END PROCESS trigger_process;

END Behavioral;

```

2.3.2 Main Control Unit

MotorPIDControl adalah modul yang mengimplementasikan empat kontroler PID (Proportional-Integral-Derivative) secara paralel untuk mengendalikan empat sumbu gerakan drone (roll/guling, pitch/angguk, yaw/putar, dan height/ketinggian). Setiap kontroler menggunakan komponen PID_Controller yang menerima sinyal enable untuk parameter P, I, dan D, nilai setpoint 12-bit (SetVal), nilai aktual 8-bit dari sensor (ADC), dan menghasilkan sinyal kontrol (display_output) yang diperluas menjadi 32-bit dengan nilai nol ketika diaktifkan (saat *_en = '1') atau diatur ke nol ketika dinonaktifkan. Port 'open' menunjukkan beberapa output display dan PWM tidak digunakan dalam implementasi ini, hanya berfokus pada fungsi kontrol PID inti yang diperlukan untuk stabilisasi drone. Berikut adalah kode VHDL dari MotorPIDControl.

```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;

ENTITY MotorPIDControl IS

    GENERIC (
        data_width : INTEGER := 32;
        internal_width : INTEGER := 16
    );
    PORT (
        clock : IN STD_LOGIC;
        reset : IN STD_LOGIC;
        -- Roll axis
        roll_kp,      roll_ki,      roll_kd      :      IN
        STD_LOGIC_VECTOR(internal_width-1 DOWNTO 0);

```

```

        roll_setpoint, roll_actual : IN STD_LOGIC_VECTOR(data_width-1
DOWNTO 0);

        roll_output : OUT STD_LOGIC_VECTOR(data_width-1 DOWNTO 0);

-- Pitch axis

        pitch_kp, pitch_ki, pitch_kd : IN STD_LOGIC_VECTOR(internal_width-1 DOWNTO 0);

        pitch_setpoint, pitch_actual : IN STD_LOGIC_VECTOR(data_width-1
DOWNTO 0);

        pitch_output : OUT STD_LOGIC_VECTOR(data_width-1 DOWNTO 0);

-- Yaw axis

        yaw_kp, yaw_ki, yaw_kd : IN STD_LOGIC_VECTOR(internal_width-1
DOWNTO 0);

        yaw_setpoint, yaw_actual : IN STD_LOGIC_VECTOR(data_width-1
DOWNTO 0);

        yaw_output : OUT STD_LOGIC_VECTOR(data_width-1 DOWNTO 0);

-- Height axis

        height_kp, height_ki, height_kd : IN STD_LOGIC_VECTOR(internal_width-1 DOWNTO 0);

        height_setpoint, height_actual : IN STD_LOGIC_VECTOR(data_width-1 DOWNTO 0);

        height_output : OUT STD_LOGIC_VECTOR(data_width-1 DOWNTO 0);

values_ready : OUT STD_LOGIC
);

END MotorPIDControl;

ARCHITECTURE Behavioral OF MotorPIDControl IS

-- Constants for scaling

```

```

CONSTANT OUTPUT_SCALE : INTEGER := 4096; -- 12-bit output scale

CONSTANT KP_SCALE : INTEGER := 100;

CONSTANT KI_SCALE : INTEGER := 100;

CONSTANT KD_SCALE : INTEGER := 100;

-- Internal signals for PID controllers

      SIGNAL roll_output_buf, pitch_output_buf, yaw_output_buf,
height_output_buf : STD_LOGIC_VECTOR(11 DOWNTO 0) := (others => '0');

      SIGNAL roll_ready, pitch_ready, yaw_ready, height_ready : STD_LOGIC
:= '0';

-- Internal enable signals

      SIGNAL roll_en, pitch_en, yaw_en, height_en : STD_LOGIC := '1'; -- Default enabled

-- Component declaration for PID_Controller

COMPONENT PID_Controller IS

  GENERIC (
    N : INTEGER := 16 -- number of bits of PWM counter
  );

  PORT (
    kp_sw : IN std_logic;
    ki_sw : IN std_logic;
    kd_sw : IN std_logic;
    SetVal : IN std_logic_vector(11 DOWNTO 0);
    PWM_PIN : OUT std_logic;
    ADC : IN std_logic_vector(7 DOWNTO 0);
    reset_button : IN std_logic;
    display_output : OUT std_logic_vector(11 DOWNTO 0);
    clk : IN std_logic;
  );

```

```

anode_activate : OUT std_logic_vector(3 DOWNTO 0);

led_out : OUT std_logic_vector(6 DOWNTO 0)

);

END COMPONENT;

BEGIN

-- Enable signal generation

roll_en <= '1' when to_integer(unsigned(roll_kp)) /= 0 else '0';
pitch_en <= '1' when to_integer(unsigned(pitch_kp)) /= 0 else '0';
yaw_en <= '1' when to_integer(unsigned(yaw_kp)) /= 0 else '0';

height_en <= '1' when to_integer(unsigned(height_kp)) /= 0 else
'0';

-- Roll axis PID controller

roll_pid : PID_Controller

PORT MAP (
    kp_sw => roll_en,                                -- Enable based on Kp value
    ki_sw => roll_en,                                -- Use same enable for all
terms

    kd_sw => roll_en,
    SetVal => roll_setpoint(11 DOWNTO 0),
    PWM_PIN => open,
    ADC => roll_actual(7 DOWNTO 0),
    reset_button => reset,
    display_output => roll_output_buf,
    clk => clock,
    anode_activate => open,
    led_out => open
);

```

```

    roll_output <= x"00000" & roll_output_buf when roll_en = '1' else
(others => '0');

-- Pitch axis PID controller

pitch_pid : PID_Controller

PORT MAP (
    kp_sw => pitch_en,
    ki_sw => pitch_en,
    kd_sw => pitch_en,
    SetVal => pitch_setpoint(11 DOWNTO 0),
    PWM_PIN => open,
    ADC => pitch_actual(7 DOWNTO 0),
    reset_button => reset,
    display_output => pitch_output_buf,
    clk => clock,
    anode_activate => open,
    led_out => open
);

pitch_output <= x"00000" & pitch_output_buf when pitch_en = '1'
else (others => '0');

-- Yaw axis PID controller

yaw_pid : PID_Controller

PORT MAP (
    kp_sw => yaw_en,
    ki_sw => yaw_en,
    kd_sw => yaw_en,
    SetVal => yaw_setpoint(11 DOWNTO 0),
    PWM_PIN => open,
    ADC => yaw_actual(7 DOWNTO 0),

```

```

    reset_button => reset,
    display_output => yaw_output_buf,
    clk => clock,
    anode_activate => open,
    led_out => open
);

    yaw_output <= x"00000" & yaw_output_buf when yaw_en = '1' else
(others => '0');

-- Height axis PID controller

height_pid : PID_Controller
PORT MAP (
    kp_sw => height_en,
    ki_sw => height_en,
    kd_sw => height_en,
    SetVal => height_setpoint(11 DOWNTO 0),
    PWM_PIN => open,
    ADC => height_actual(7 DOWNTO 0),
    reset_button => reset,
    display_output => height_output_buf,
    clk => clock,
    anode_activate => open,
    led_out => open
);

    height_output <= x"00000" & height_output_buf when height_en = '1'
else (others => '0');

-- Process to handle values_ready signal

ready_proc : PROCESS(clock)
BEGIN

```

```

        IF rising_edge(clock) THEN
            IF reset = '0' THEN
                values_ready <= '0';
            ELSE
                values_ready <= roll_en or pitch_en or yaw_en or
height_en;
            END IF;
        END IF;
    END PROCESS;

END Behavioral;

```

PWM Generator Motor merupakan komponen yang mengurus sinyal PWM (Pulse Width Modulation) untuk 4 motor di dalam sistem drone kita. Dimana setiap motor memiliki PWM generatornya tersendiri, yang mengambil duty cycle input 12 bit, clock, dan memberi kemampuan sinyal untuk dihasilkan untuk sinyal PWM. Component ini menciptakan output PWM normal dan inverted untuk setiap motor.

```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;

ENTITY PWMGeneratorMotor IS
    GENERIC (
        base_clock : INTEGER := 100000000; -- 100MHz system clock
        pwm_frequency : INTEGER := 65;      -- 65Hz PWM frequency
        resolution : INTEGER := 12;        -- 12-bit resolution
    );
    PORT (
        clock : IN STD_LOGIC;
        reset : IN STD_LOGIC;
        enable : IN STD_LOGIC;

```

```

-- Motor 1 control

motor1_duty : IN STD_LOGIC_VECTOR(resolution-1 DOWNTO 0);
motor1_pwm : OUT STD_LOGIC;
motor1_pwm_n : OUT STD_LOGIC;

-- Motor 2 control

motor2_duty : IN STD_LOGIC_VECTOR(resolution-1 DOWNTO 0);
motor2_pwm : OUT STD_LOGIC;
motor2_pwm_n : OUT STD_LOGIC;

-- Motor 3 control

motor3_duty : IN STD_LOGIC_VECTOR(resolution-1 DOWNTO 0);
motor3_pwm : OUT STD_LOGIC;
motor3_pwm_n : OUT STD_LOGIC;

-- Motor 4 control

motor4_duty : IN STD_LOGIC_VECTOR(resolution-1 DOWNTO 0);
motor4_pwm : OUT STD_LOGIC;
motor4_pwm_n : OUT STD_LOGIC
);

END PWMGeneratorMotor;

```

```

ARCHITECTURE Behavioral OF PWMGeneratorMotor IS

-- Internal signals for PWM outputs

SIGNAL pwm1_signal : STD_LOGIC := '0';
SIGNAL pwm2_signal : STD_LOGIC := '0';
SIGNAL pwm3_signal : STD_LOGIC := '0';
SIGNAL pwm4_signal : STD_LOGIC := '0';

```

```

-- Component declaration

COMPONENT PWM_Generator

PORT (
    enable : IN STD_LOGIC;
    duty_in : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
    clock : IN STD_LOGIC;
    pwm_out : OUT STD_LOGIC
);

END COMPONENT;

BEGIN

-- Motor 1 PWM instance

pwm1 : PWM_Generator
PORT MAP (
    enable => enable,
    duty_in => motor1_duty,
    clock => clock,
    pwm_out => pwm1_signal
);

-- Assign outputs using internal signal

motor1_pwm <= pwm1_signal;
motor1_pwm_n <= NOT pwm1_signal;

-- Motor 2 PWM instance

pwm2 : PWM_Generator
PORT MAP (
    enable => enable,
    duty_in => motor2_duty,

```

```

    clock => clock,
    pwm_out => pwm2_signal
);

motor2_pwm <= pwm2_signal;

motor2_pwm_n <= NOT pwm2_signal;

-- Motor 3 PWM instance

pwm3 : PWM_Generator

PORT MAP (
    enable => enable,
    duty_in => motor3_duty,
    clock => clock,
    pwm_out => pwm3_signal
);

motor3_pwm <= pwm3_signal;

motor3_pwm_n <= NOT pwm3_signal;

-- Motor 4 PWM instance

pwm4 : PWM_Generator

PORT MAP (
    enable => enable,
    duty_in => motor4_duty,
    clock => clock,
    pwm_out => pwm4_signal
);

motor4_pwm <= pwm4_signal;

motor4_pwm_n <= NOT pwm4_signal;

END Behavioral;

```

Semua dari komponen diatas disusun menjadi suatu komponen utama dalam bentuk FlightControlUnit dengan kode VHDL seperti berikut,

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;

ENTITY FlightControlUnit IS
    GENERIC (
        data_width : INTEGER := 32;
        internal_width : INTEGER := 16;
        pwm_resolution : INTEGER := 12
    );
    PORT (
        -- System signals
        clock : IN STD_LOGIC;
        reset : IN STD_LOGIC;

        -- Control inputs
        roll_setpoint : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
        pitch_setpoint : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
        yaw_setpoint : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
        height_setpoint : IN STD_LOGIC_VECTOR(11 DOWNTO 0);

        -- Sensor inputs
        roll_sensor : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        pitch_sensor : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        yaw_sensor : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        height_sensor : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
    );

```

```

    -- Motor outputs

    motor1_pwm, motor1_pwm_n : OUT STD_LOGIC;
    motor2_pwm, motor2_pwm_n : OUT STD_LOGIC;
    motor3_pwm, motor3_pwm_n : OUT STD_LOGIC;
    motor4_pwm, motor4_pwm_n : OUT STD_LOGIC;

    -- Status output
    system_ready : OUT STD_LOGIC
);

END FlightControlUnit;

ARCHITECTURE Behavioral OF FlightControlUnit IS

    -- Internal signals for extended setpoints and actual values
    SIGNAL roll_setpoint_ext : STD_LOGIC_VECTOR(31 DOWNTO 0);
    SIGNAL roll_actual_ext : STD_LOGIC_VECTOR(31 DOWNTO 0);
    SIGNAL pitch_setpoint_ext : STD_LOGIC_VECTOR(31 DOWNTO 0);
    SIGNAL pitch_actual_ext : STD_LOGIC_VECTOR(31 DOWNTO 0);
    SIGNAL yaw_setpoint_ext : STD_LOGIC_VECTOR(31 DOWNTO 0);
    SIGNAL yaw_actual_ext : STD_LOGIC_VECTOR(31 DOWNTO 0);
    SIGNAL height_setpoint_ext : STD_LOGIC_VECTOR(31 DOWNTO 0);
    SIGNAL height_actual_ext : STD_LOGIC_VECTOR(31 DOWNTO 0);

    SIGNAL roll_out, pitch_out, yaw_out, height_out :
STD_LOGIC_VECTOR(data_width-1 DOWNTO 0);
    SIGNAL motor1_duty, motor2_duty, motor3_duty, motor4_duty :
STD_LOGIC_VECTOR(pwm_resolution-1 DOWNTO 0);
    SIGNAL pid_ready : STD_LOGIC;

    -- Constants for PID gains

```

```

CONSTANT KP : STD_LOGIC_VECTOR(internal_width-1 DOWNTO 0) :=
x"0014"; -- 20

CONSTANT KI : STD_LOGIC_VECTOR(internal_width-1 DOWNTO 0) :=
x"0019"; -- 25

CONSTANT KD : STD_LOGIC_VECTOR(internal_width-1 DOWNTO 0) :=
x"0001"; -- 1

-- Component declarations

COMPONENT MotorPIDControl IS

  GENERIC (
    data_width : INTEGER := 32;
    internal_width : INTEGER := 16
  );

  PORT (
    clock : IN STD_LOGIC;
    reset : IN STD_LOGIC;

    -- Roll axis
    roll_kp, roll_ki, roll_kd : IN
STD_LOGIC_VECTOR(internal_width-1 DOWNTO 0);
    roll_setpoint, roll_actual : IN
STD_LOGIC_VECTOR(data_width-1 DOWNTO 0);
    roll_output : OUT STD_LOGIC_VECTOR(data_width-1 DOWNTO 0);

    -- Pitch axis
    pitch_kp, pitch_ki, pitch_kd : IN
STD_LOGIC_VECTOR(internal_width-1 DOWNTO 0);
    pitch_setpoint, pitch_actual : IN
STD_LOGIC_VECTOR(data_width-1 DOWNTO 0);
    pitch_output : OUT STD_LOGIC_VECTOR(data_width-1 DOWNTO 0);
  );

```

```

-- Yaw axis

yaw_kp, yaw_ki, yaw_kd : IN
STD_LOGIC_VECTOR(internal_width-1 DOWNTO 0);

yaw_setpoint, yaw_actual : IN STD_LOGIC_VECTOR(data_width-1
DOWNTO 0);

yaw_output : OUT STD_LOGIC_VECTOR(data_width-1 DOWNTO 0);

-- Height axis

height_kp, height_ki, height_kd : IN
STD_LOGIC_VECTOR(internal_width-1 DOWNTO 0);

height_setpoint, height_actual : IN
STD_LOGIC_VECTOR(data_width-1 DOWNTO 0);

height_output : OUT STD_LOGIC_VECTOR(data_width-1 DOWNTO
0);

values_ready : OUT STD_LOGIC
);

END COMPONENT;

COMPONENT PWMGeneratorMotor IS

GENERIC (
    base_clock : INTEGER := 100000000; -- 100MHz system clock
    pwm_frequency : INTEGER := 65; -- 65Hz PWM frequency
    resolution : INTEGER := 12 -- 12-bit resolution
);

PORT (
    clock : IN STD_LOGIC;
    reset : IN STD_LOGIC;
    enable : IN STD_LOGIC;

    -- Motor 1 control

```

```

motor1_duty : IN STD_LOGIC_VECTOR(resolution-1 DOWNTO 0);
motor1_pwm : OUT STD_LOGIC;
motor1_pwm_n : OUT STD_LOGIC;

-- Motor 2 control

motor2_duty : IN STD_LOGIC_VECTOR(resolution-1 DOWNTO 0);
motor2_pwm : OUT STD_LOGIC;
motor2_pwm_n : OUT STD_LOGIC;

-- Motor 3 control

motor3_duty : IN STD_LOGIC_VECTOR(resolution-1 DOWNTO 0);
motor3_pwm : OUT STD_LOGIC;
motor3_pwm_n : OUT STD_LOGIC;

-- Motor 4 control

motor4_duty : IN STD_LOGIC_VECTOR(resolution-1 DOWNTO 0);
motor4_pwm : OUT STD_LOGIC;
motor4_pwm_n : OUT STD_LOGIC
);

END COMPONENT;

BEGIN

-- Signal extension process

PROCESS(roll_setpoint, roll_sensor, pitch_setpoint, pitch_sensor,
        yaw_setpoint, yaw_sensor, height_setpoint, height_sensor)
BEGIN

    roll_setpoint_ext <= x"00000" & roll_setpoint;
    roll_actual_ext <= x"000000" & roll_sensor;
    pitch_setpoint_ext <= x"00000" & pitch_setpoint;

```

```

pitch_actual_ext <= x"000000" & pitch_sensor;
yaw_setpoint_ext <= x"00000" & yaw_setpoint;
yaw_actual_ext <= x"000000" & yaw_sensor;
height_setpoint_ext <= x"00000" & height_setpoint;
height_actual_ext <= x"000000" & height_sensor;

END PROCESS;

-- PID controller instantiation

pid_control : MotorPIDControl
  GENERIC MAP (
    data_width => data_width,
    internal_width => internal_width
  )
  PORT MAP (
    clock => clock,
    reset => reset,
    -- Roll
    roll_kp => KP, roll_ki => KI, roll_kd => KD,
    roll_setpoint => roll_setpoint_ext,
    roll_actual => roll_actual_ext,
    roll_output => roll_out,
    -- Pitch
    pitch_kp => KP, pitch_ki => KI, pitch_kd => KD,
    pitch_setpoint => pitch_setpoint_ext,
    pitch_actual => pitch_actual_ext,
    pitch_output => pitch_out,
    -- Yaw
    yaw_kp => KP, yaw_ki => KI, yaw_kd => KD,
    yaw_setpoint => yaw_setpoint_ext,

```

```

yaw_actual => yaw_actual_ext,
yaw_output => yaw_out,
-- Height

height_kp => KP, height_ki => KI, height_kd => KD,
height_setpoint => height_setpoint_ext,
height_actual => height_actual_ext,
height_output => height_out,
values_ready => pid_ready

);

-- PWM generator instantiation

pwm_gen : PWMGeneratorMotor
GENERIC MAP (
    base_clock => 100000000,
    pwm_frequency => 65,
    resolution => pwm_resolution
)

PORT MAP (
    clock => clock,
    reset => reset,
    enable => pid_ready,
    motor1_duty => motor1_duty,
    motor1_pwm => motor1_pwm,
    motor1_pwm_n => motor1_pwm_n,
    motor2_duty => motor2_duty,
    motor2_pwm => motor2_pwm,
    motor2_pwm_n => motor2_pwm_n,
    motor3_duty => motor3_duty,
    motor3_pwm => motor3_pwm,

```

```

motor3_pwm_n => motor3_pwm_n,
motor4_duty => motor4_duty,
motor4_pwm => motor4_pwm,
motor4_pwm_n => motor4_pwm_n
);

-- Mix PID outputs to motor commands

PROCESS(roll_out, pitch_out, yaw_out, height_out)
VARIABLE roll, pitch, yaw, height : INTEGER;
BEGIN

    roll := to_integer(unsigned(roll_out(11 DOWNTO 0)));
    pitch := to_integer(unsigned(pitch_out(11 DOWNTO 0)));
    yaw := to_integer(unsigned(yaw_out(11 DOWNTO 0)));
    height := to_integer(unsigned(height_out(11 DOWNTO 0)));

    -- Motor mixing algorithm

    motor1_duty <= std_logic_vector(to_unsigned(height + pitch +
roll - yaw, 12));
    motor2_duty <= std_logic_vector(to_unsigned(height + pitch -
roll + yaw, 12));
    motor3_duty <= std_logic_vector(to_unsigned(height - pitch -
roll - yaw, 12));
    motor4_duty <= std_logic_vector(to_unsigned(height - pitch +
roll + yaw, 12));

    END PROCESS;

-- System ready signal

system_ready <= pid_ready;

END Behavioral;

```

CHAPTER 3

TESTING AND ANALYSIS

3.1 TESTING

3.1.1 PID Controller

Dengan Testbench seperti berikut untuk PID Controller, dimana dia akan memverifikasi fungsionalitas PID Controller dengan melakukan test dalam berbagai konfigurasi dan skenario. Testbench ini melakukan lima kasus uji utama: kontrol P saja (proporsional), kontrol PI (proporsional-integral), kontrol PID penuh, uji reset sistem, dan uji integral windup. Setiap kasus uji menggunakan sinyal clock dengan periode 10ns dan mengatur nilai setpoint (SetVal) serta input ADC yang berbeda.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;

ENTITY PID_Controller_tb IS
END PID_Controller_tb;

ARCHITECTURE Behavioral OF PID_Controller_tb IS
    -- Component Declaration
    COMPONENT PID_Controller IS
        GENERIC (
            N : INTEGER := 16    -- number of bits of PWM counter
        );
        PORT (
            kp_sw : IN std_logic;
```

```

        ki_sw : IN std_logic;
        kd_sw : IN std_logic;
        SetVal : IN std_logic_vector(11 DOWNTO 0);
        PWM_PIN : OUT std_logic;
        ADC : IN std_logic_vector(7 DOWNTO 0);
        reset_button : IN std_logic;
        display_output : OUT std_logic_vector(11 DOWNTO 0);
        clk : IN std_logic;
        anode_activate : OUT std_logic_vector(3 DOWNTO 0);
        led_out : OUT std_logic_vector(6 DOWNTO 0)
    );
END COMPONENT;

-- Test Signals
SIGNAL clk : std_logic := '0';
SIGNAL reset_button : std_logic := '0';
SIGNAL kp_sw : std_logic := '0';
SIGNAL ki_sw : std_logic := '0';
SIGNAL kd_sw : std_logic := '0';
SIGNAL SetVal : std_logic_vector(11 downto 0) := (others => '0');
SIGNAL ADC : std_logic_vector(7 downto 0) := (others => '0');
SIGNAL PWM_PIN : std_logic;
SIGNAL display_output : std_logic_vector(11 downto 0);
SIGNAL anode_activate : std_logic_vector(3 downto 0);
SIGNAL led_out : std_logic_vector(6 downto 0);

CONSTANT CLK_PERIOD : time := 10 ns;

-- Helper procedure

```

```

PROCEDURE print_test_case(
    test_name : in string;
    ref_value : in integer;
    adc_value : in integer;
    expected_pwm : in integer) IS

BEGIN

    report "==== Test Case: " & test_name & " ===" severity note;

    report "Reference value: " & integer'image(ref_value) severity
note;

    report "ADC input value: " & integer'image(adc_value) severity
note;

    report "Expected PWM value: " & integer'image(expected_pwm)
severity note;

    report "Actual output value: " &
integer'image(to_integer(unsigned(display_output))) severity note;

    report "Controls: P=" & std_logic'image(kp_sw) &
    " I=" & std_logic'image(ki_sw) &
    " D=" & std_logic'image(kd_sw) severity note;

END PROCEDURE;

BEGIN

-- DUT instantiation

UUT: PID_Controller

PORT MAP (
    kp_sw => kp_sw,
    ki_sw => ki_sw,
    kd_sw => kd_sw,
    SetVal => SetVal,
    PWM_PIN => PWM_PIN,
    ADC => ADC,

```

```

reset_button => reset_button,
display_output => display_output,
clk => clk,
anode_activate => anode_activate,
led_out => led_out
);

-- Clock process
clk_process: PROCESS
BEGIN
    clk <= '0';
    wait for CLK_PERIOD/2;
    clk <= '1';
    wait for CLK_PERIOD/2;
END PROCESS;

-- Stimulus process
stim_proc: PROCESS
BEGIN
    wait for 100 ns;
    reset_button <= '1';
    wait for CLK_PERIOD*2;

    -- Test Case 1: P control only
    kp_sw <= '1';
    ki_sw <= '0';
    kd_sw <= '0';
    SetVal <= std_logic_vector(to_unsigned(2048, 12));
    ADC <= std_logic_vector(to_unsigned(128, 8));

```

```

wait for CLK_PERIOD*10;

print_test_case("P Control Only", 2048, 128, 409);


-- Test Case 2: PI control

kp_sw <= '1';

ki_sw <= '1';

kd_sw <= '0';

SetVal <= std_logic_vector(to_unsigned(3072, 12));

ADC <= std_logic_vector(to_unsigned(64, 8));

wait for CLK_PERIOD*10;

print_test_case("PI Control", 3072, 64, 657);


-- Test Case 3: Full PID

kp_sw <= '1';

ki_sw <= '1';

kd_sw <= '1';

SetVal <= std_logic_vector(to_unsigned(4000, 12));

ADC <= std_logic_vector(to_unsigned(255, 8));

wait for CLK_PERIOD*10;

print_test_case("Full PID", 4000, 255, 931);


-- Test Case 4: System Reset

reset_button <= '0';

wait for CLK_PERIOD*5;

print_test_case("System Reset", 4000, 255, 0);


-- Test Case 5: Integral Windup Test

reset_button <= '1';

kp_sw <= '0';

```

```

ki_sw <= '1';

kd_sw <= '0';

SetVal <= std_logic_vector(to_unsigned(4095, 12));

ADC <= std_logic_vector(to_unsigned(0, 8));

wait for CLK_PERIOD*20;

print_test_case("Integral Windup Test", 4095, 0, 153);

report "==== Test Complete ===" severity note;

wait for CLK_PERIOD;

std.env.stop;

END PROCESS;

END Behavioral;

```

3.1.2 Flight Control Unit

Selanjutnya, kita akan melakukan testing pada Flight Control Unit-nya itu sendiri. Dimana akan ada banyak output yang bisa kita analisis. Berikut adalah Testbench dari Flight Control Unit

```

LIBRARY IEEE;

USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
USE STD.ENV.ALL; -- Add this for stop procedure

ENTITY FlightControlUnit_tb IS
END FlightControlUnit_tb;

ARCHITECTURE Behavioral OF FlightControlUnit_tb IS
-- Component Declaration
COMPONENT FlightControlUnit IS

```

```

  GENERIC (
    data_width : INTEGER := 32;
    internal_width : INTEGER := 16;
    pwm_resolution : INTEGER := 12
  );
  PORT (
    clock : IN STD_LOGIC;
    reset : IN STD_LOGIC;

    -- Control inputs
    roll_setpoint : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
    pitch_setpoint : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
    yaw_setpoint : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
    height_setpoint : IN STD_LOGIC_VECTOR(11 DOWNTO 0);

    -- Sensor inputs
    roll_sensor : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
    pitch_sensor : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
    yaw_sensor : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
    height_sensor : IN STD_LOGIC_VECTOR(7 DOWNTO 0);

    -- Motor outputs
    motor1_pwm, motor1_pwm_n : OUT STD_LOGIC;
    motor2_pwm, motor2_pwm_n : OUT STD_LOGIC;
    motor3_pwm, motor3_pwm_n : OUT STD_LOGIC;
    motor4_pwm, motor4_pwm_n : OUT STD_LOGIC;

    -- Status output
    system_ready : OUT STD_LOGIC
  );

```

```

) ;

END COMPONENT;

-- Test Signals

SIGNAL clock : STD_LOGIC := '0';

SIGNAL reset : STD_LOGIC := '0';

-- Control setpoints

SIGNAL roll_setpoint : STD_LOGIC_VECTOR(11 DOWNTO 0) := (OTHERS =>
'0');

SIGNAL pitch_setpoint : STD_LOGIC_VECTOR(11 DOWNTO 0) := (OTHERS =>
'0');

SIGNAL yaw_setpoint : STD_LOGIC_VECTOR(11 DOWNTO 0) := (OTHERS =>
'0');

SIGNAL height_setpoint : STD_LOGIC_VECTOR(11 DOWNTO 0) := (OTHERS =>
'0');

-- Sensor inputs

SIGNAL roll_sensor : STD_LOGIC_VECTOR(7 DOWNTO 0) := (OTHERS =>
'0');

SIGNAL pitch_sensor : STD_LOGIC_VECTOR(7 DOWNTO 0) := (OTHERS =>
'0');

SIGNAL yaw_sensor : STD_LOGIC_VECTOR(7 DOWNTO 0) := (OTHERS =>
'0');

SIGNAL height_sensor : STD_LOGIC_VECTOR(7 DOWNTO 0) := (OTHERS =>
'0');

-- Motor outputs

SIGNAL motor1_pwm, motor1_pwm_n : STD_LOGIC;
SIGNAL motor2_pwm, motor2_pwm_n : STD_LOGIC;
SIGNAL motor3_pwm, motor3_pwm_n : STD_LOGIC;
SIGNAL motor4_pwm, motor4_pwm_n : STD_LOGIC;

```

```

-- Status

SIGNAL system_ready : STD_LOGIC;

-- Clock period and timing constants

CONSTANT CLK_PERIOD : TIME := 10 ns;

CONSTANT STABILIZATION_TIME : TIME := 10 us;           -- Shorter
stabilization

CONSTANT TEST_TIME : TIME := 50 us;                     -- Longer test time

CONSTANT EMERGENCY_TIME : TIME := 5 us;                -- Emergency response
time

-- Helper procedure for printing test results

PROCEDURE print_test_case(
    test_name : IN STRING;
    m1, m2, m3, m4 : IN STD_LOGIC
) IS
BEGIN

    REPORT "==== Test Case: " & test_name & " ===";
    REPORT "Motor 1 PWM: " & STD_LOGIC'image(m1);
    REPORT "Motor 2 PWM: " & STD_LOGIC'image(m2);
    REPORT "Motor 3 PWM: " & STD_LOGIC'image(m3);
    REPORT "Motor 4 PWM: " & STD_LOGIC'image(m4);
    REPORT "System Ready: " & STD_LOGIC'image(system_ready);

END PROCEDURE;

-- Procedure to inject periodic disturbances

PROCEDURE inject_disturbance(
    SIGNAL sensor : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
    base_value : IN INTEGER;

```

```

        amplitude : IN INTEGER) IS
      BEGIN
        sensor <= std_logic_vector(to_unsigned(base_value + amplitude,
8));
        WAIT FOR TEST_TIME/4;
        sensor <= std_logic_vector(to_unsigned(base_value - amplitude,
8));
        WAIT FOR TEST_TIME/4;
      END PROCEDURE;

BEGIN
  -- Component Instantiation
  UUT : FlightControlUnit
  GENERIC MAP (
    data_width => 32,
    internal_width => 16,
    pwm_resolution => 12
  )
  PORT MAP (
    clock => clock,
    reset => reset,
    roll_setpoint => roll_setpoint,
    pitch_setpoint => pitch_setpoint,
    yaw_setpoint => yaw_setpoint,
    height_setpoint => height_setpoint,
    roll_sensor => roll_sensor,
    pitch_sensor => pitch_sensor,
    yaw_sensor => yaw_sensor,
    height_sensor => height_sensor,
    motorl_pwm => motorl_pwm,
  );

```

```

motor1_pwm_n => motor1_pwm_n,
motor2_pwm => motor2_pwm,
motor2_pwm_n => motor2_pwm_n,
motor3_pwm => motor3_pwm,
motor3_pwm_n => motor3_pwm_n,
motor4_pwm => motor4_pwm,
motor4_pwm_n => motor4_pwm_n,
system_ready => system_ready
);

-- Clock generation process

clock_process : PROCESS
BEGIN
    clock <= '0';
    WAIT FOR CLK_PERIOD/2;
    clock <= '1';
    WAIT FOR CLK_PERIOD/2;
END PROCESS;

-- Modified stimulus process

stim_proc : PROCESS
BEGIN
    -- Initial reset
    reset <= '0';
    WAIT FOR CLK_PERIOD * 10;
    reset <= '1';
    WAIT FOR STABILIZATION_TIME;

    -- Test Case 1: Hover condition with disturbance

```

```

roll_setpoint <= x"400"; -- 1024 (mid-point)

pitch_setpoint <= x"400";

yaw_setpoint <= x"400";

height_setpoint <= x"400";

-- Initial sensor values

roll_sensor <= x"40"; -- 64 (mid-point for 8-bit)

pitch_sensor <= x"40";

yaw_sensor <= x"40";

height_sensor <= x"40";

WAIT FOR TEST_TIME;

print_test_case("Initial Hover", motor1_pwm, motor2_pwm,
motor3_pwm, motor4_pwm);

-- Test Case 2: Forward pitch with disturbance

pitch_setpoint <= x"600"; -- Increase pitch

FOR i IN 1 TO 4 LOOP

    inject_disturbance(pitch_sensor, 64, 16); -- Add
oscillation

END LOOP;

print_test_case("Forward Pitch", motor1_pwm, motor2_pwm,
motor3_pwm, motor4_pwm);

-- Test Case 3: Right roll with disturbance

roll_setpoint <= x"600"; -- Roll right

FOR i IN 1 TO 4 LOOP

    inject_disturbance(roll_sensor, 64, 16);

END LOOP;

```

```

        print_test_case("Right  Roll",  motor1_pwm,  motor2_pwm,
motor3_pwm,  motor4_pwm);

-- Test Case 4: Yaw right with disturbance

yaw_setpoint <= x"600";      -- Yaw right

FOR i IN 1 TO 4 LOOP

    inject_disturbance(yaw_sensor, 64, 16);

END LOOP;

print_test_case("Yaw  Right",  motor1_pwm,  motor2_pwm,
motor3_pwm,  motor4_pwm);

-- Test Case 5: Increase height with disturbance

height_setpoint <= x"600"; -- Increase altitude

FOR i IN 1 TO 4 LOOP

    inject_disturbance(height_sensor, 64, 16);

END LOOP;

print_test_case("Ascending",  motor1_pwm,  motor2_pwm,
motor3_pwm,  motor4_pwm);

-- Test Case 6: Emergency stop

reset <= '0';

WAIT FOR EMERGENCY_TIME;

print_test_case("Emergency Stop",  motor1_pwm,  motor2_pwm,
motor3_pwm,  motor4_pwm);

-- Test Case 7: Recovery with active control

reset <= '1';

roll_setpoint <= x"400";      -- Return to neutral

pitch_setpoint <= x"400";

yaw_setpoint <= x"400";

```

```

height_setpoint <= x"400";

-- Simulate recovery disturbance

FOR i IN 1 TO 8 LOOP

    roll_sensor <= std_logic_vector(to_unsigned(48 + i*4, 8));
    pitch_sensor <= std_logic_vector(to_unsigned(48 + i*4, 8));
    WAIT FOR TEST_TIME/8;

END LOOP;

WAIT FOR TEST_TIME;

print_test_case("Recovery", motor1_pwm, motor2_pwm, motor3_pwm,
motor4_pwm);

-- End simulation properly

REPORT "==== Test Complete ====";
WAIT FOR CLK_PERIOD * 10; -- Allow for final prints
std.env.stop; -- Stop the simulation
WAIT;
END PROCESS;

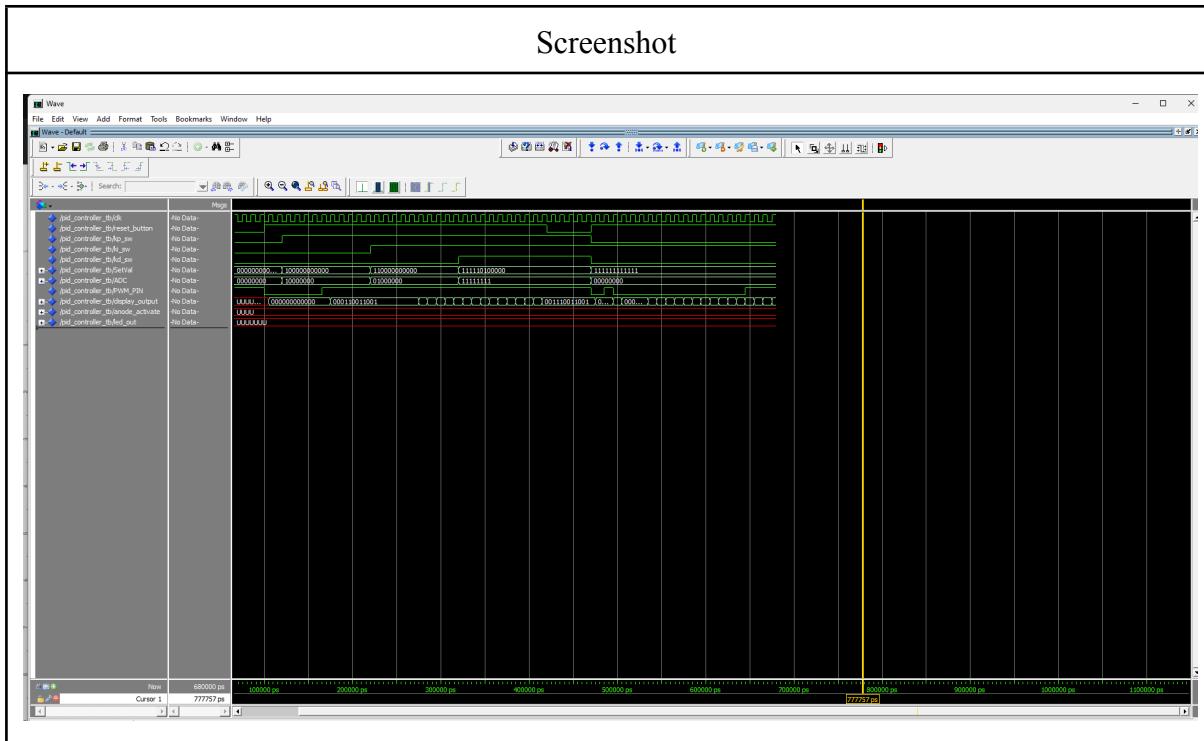
-- Add monitoring process for detailed debugging
monitor_proc : PROCESS
BEGIN
    WAIT FOR CLK_PERIOD;
    IF reset = '1' THEN
        REPORT "Current values:" &
                " Roll SP: " &
integer'image(to_integer(unsigned(roll_setpoint))) &
                " Roll Act: " &
integer'image(to_integer(unsigned(roll_sensor))) &

```

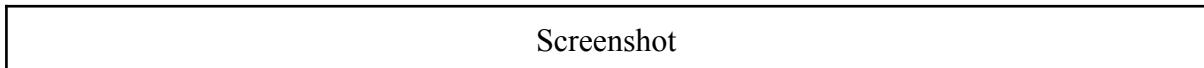
```
        " M1:" & std_logic'image(motor1_pwm) &
        " M2:" & std_logic'image(motor2_pwm) &
        " M3:" & std_logic'image(motor3_pwm) &
        " M4:" & std_logic'image(motor4_pwm);
END IF;
END PROCESS;
END Behavioral;
```

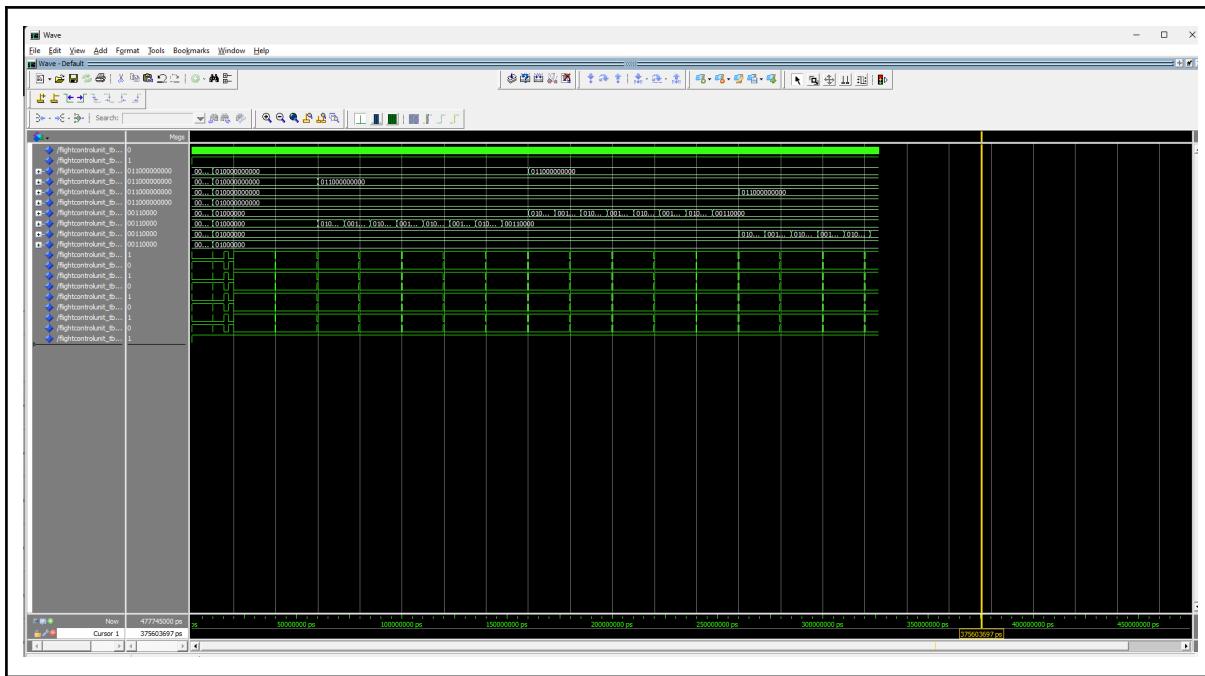
3.2 RESULT

3.2.1 PID Controller



3.2.2 Flight Control Unit





3.3 ANALYSIS

3.3.1 PID Controller

Dapat dilihat di dalam hasil dari PID Controller, pada masing-masing testcase, menunjukkan osilasi dalam perbaikan PID values yang kita masukkan. Dimana dapat dilihat, PID values yang dikeluarkan oleh PID controller akan mencoba mendekati values yang ditentukan. Tetapi karena perhitungan PID tidak sempurna, dia akan terus berosilasi mendekati nilai yang diinginkan, dengan semakin lama semakin akurat. Untuk nilai merah pada simulasi, dia dikarenakan komponen yang terlibat belum disambungkan ke led, dimana di desain kami mengimplementasikan output untuk LED.

3.3.2 Flight Control Unit

Testbench dalam implementasi ini melakukan serangkaian pengujian untuk mensimulasikan berbagai kondisi penerbangan drone, seperti hover, pitch ke depan, roll ke samping, rotasi yaw, dan perubahan ketinggian. Setiap kasus uji memberikan nilai set point yang berbeda dan mensimulasikan pembacaan sensor yang sesuai, yang memungkinkan kita untuk mengamati bagaimana PID controller menyesuaikan output PWM untuk keempat motor. Selama pengujian ini, output PWM berosilasi karena PID controller terus menyesuaikan nilai untuk mencapai dan mempertahankan posisi atau orientasi yang diinginkan. Untuk hover, sistem berusaha menjaga drone pada ketinggian stabil, sementara

untuk pitch, roll, dan yaw, sistem menyesuaikan orientasi drone agar mencapai sudut yang ditentukan.

Osilasi yang terlihat pada output PWM mencerminkan upaya PID controller untuk menstabilkan drone pada nilai set point, dengan osilasi ini secara bertahap berkurang seiring dengan stabilisasi sistem. Testbench ini menangkap respons drone terhadap gangguan kecil maupun besar, menguji ketangguhan controller dalam mengatasi perubahan kondisi. Output yang dihasilkan selama pengujian ini memungkinkan evaluasi kinerja PID controller dalam skenario dunia nyata, memastikan bahwa drone dapat mengontrol posisi, orientasi, dan ketinggiannya dengan akurat, bahkan dalam kondisi dinamis.

CHAPTER 4

CONCLUSION

Proyek akhir ini bertujuan untuk merancang dan mengimplementasikan Flight Controller Unit (FCU) menggunakan bahasa VHDL yang merupakan bagian integral dari sistem pesawat tanpa awak. Proyek ini mencakup pengembangan komponen utama, seperti PID Controller dan PWM Generator untuk mengatur stabilitas penerbangan, seperti pengendalian roll, pitch, yaw, dan ketinggian. Dengan memanfaatkan teori PID, sistem ini dirancang agar responsif terhadap perubahan kondisi, stabil dalam mengatasi gangguan, dan efisien dalam komunikasi dengan komponen seperti sensor LiDAR. Hasil pengujian menunjukkan bahwa FCU mampu mengontrol dan menstabilkan drone secara efektif dalam berbagai mode penerbangan termasuk hover, takeoff, dan landing dengan osilasi yang berkurang seiring waktu.

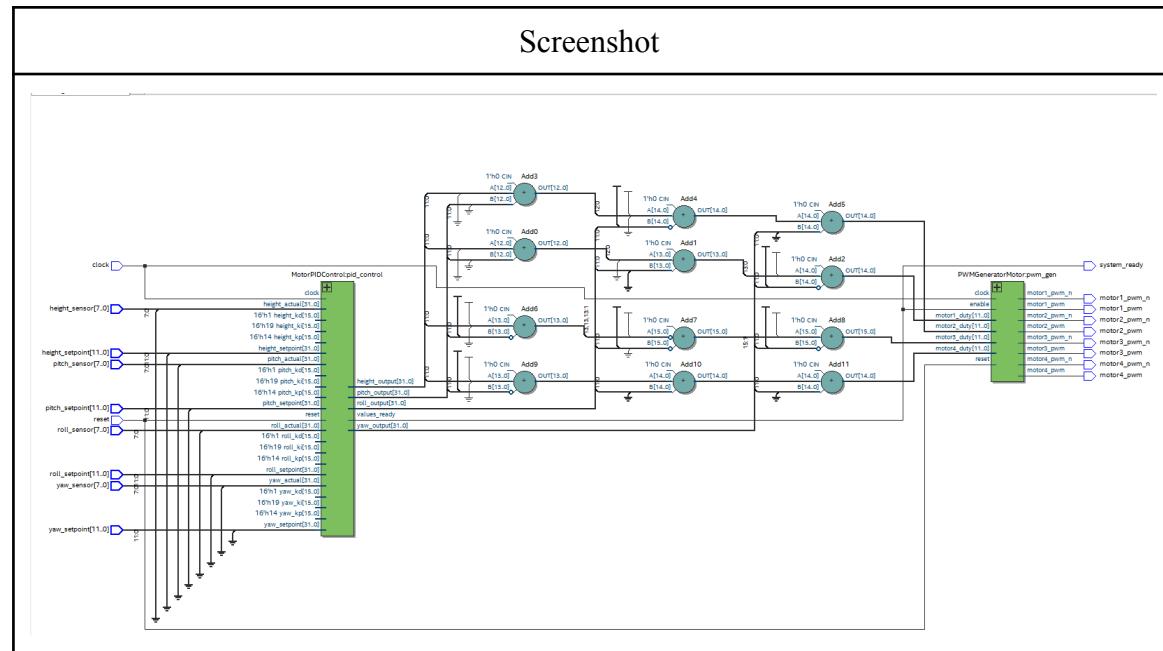
Selain implementasi teknis, proyek ini juga menekankan analisis performa sistem melalui testbench yang mengintegrasikan berbagai gaya pemrograman seperti dataflow, behavioral, dan structural untuk membangun arsitektur modular FCU. Komponen utama yang digunakan, seperti PID Controller dan PWM Generator dikembangkan dengan menggunakan looping dan function untuk memproses data, termasuk penghitungan kesalahan dan output kontrol secara real-time. Sistem ini juga memanfaatkan konsep microprogramming untuk mengatur alur kerja pengendali utama dan subsistem pendukung secara efisien guna memastikan stabilitas dan responsivitas.

REFERENCES

- [1] PX4 Outopilot, “CubePilot Cube Orange Flight Controller | PX4 User Guide,” docs.px4.io. https://docs.px4.io/main/en/flight_controller/cubepilot_cube_orange.html (accessed Dec. 06, 2024).
- [2] Ni, “The PID Controller & Theory Explained,” [www.ni.com](https://www.ni.com/en/shop/labview/pid-theory-explained.html#section--2054068860), Sep. 09, 2024. <https://www.ni.com/en/shop/labview/pid-theory-explained.html#section--2054068860> (accessed Dec. 06, 2024).
- [3] Control Tutorial for Matlab & Simulink, “Control Tutorials for MATLAB and Simulink - Introduction: PID Controller Design,” [ctms.engin.umich.edu](https://ctms.engin.umich.edu/CTMS/index.php?example=Introduction§ion=ControlPID#7). <https://ctms.engin.umich.edu/CTMS/index.php?example=Introduction§ion=ControlPID#7> (accessed Dec. 06, 2024).

APPENDICES

Appendix A: Project Schematic



Appendix B: Documentation

