

FLIGHT CONTROLLER UNIT

Kelompok PA-22

Andi Muhammad Alvin Farhansyah (2306161933)

Daffa Bagus Dhiananto (2306250756)

Ibnu Zaky Fauzi (2306161870)

Samih Bassam (2306250623)

... fell in love with it.
village ...

LATAR BELAKANG

Teknologi drone telah berkembang pesat dan digunakan di berbagai bidang, seperti pemantauan lingkungan, pengiriman barang, hingga aplikasi militer. Untuk mendukung operasional yang aman dan efektif, diperlukan sistem kontrol yang efisien dengan Flight Control Unit (FCU) sebagai komponen utama. FCU berfungsi sebagai sistem saraf drone dengan mengintegrasikan sensor dan aktuator untuk memastikan fungsi penerbangan berjalan sesuai perintah. Dengan menggunakan VHDL, desain FCU dapat dikembangkan dan diuji secara detail, menghasilkan sistem kontrol yang efisien, dan dapat meningkatkan performa dan stabilitas drone. Proyek ini berfokus pada pengembangan FCU berbasis VHDL.

DESKRIPSI PROYEK

Flight Control Unit (FCU) adalah komponen utama dalam drone yang berfungsi sebagai "sistem saraf" untuk mengontrol servo dan motor berdasarkan perintah dari komputer atau mikrokontroler. FCU memastikan pesawat dapat bergerak sesuai arah, mencapai ketinggian, dan mengatur kecepatan dengan stabil. Proyek ini berfokus pada pengembangan FCU dengan komponen PID Controller untuk menjaga stabilitas dan navigasi melalui penghitungan komponen Proportional, Integral, dan Derivative. FCU akan mendukung empat mode penerbangan: Idle, Takeoff, Hover, Cruise, dan Land, yang memungkinkan adaptasi di berbagai situasi. Dilengkapi unit pemrosesan sinyal untuk membaca data sensor seperti LiDAR, FCU menghasilkan sinyal PWM untuk mengendalikan motor dan menggunakan protokol komunikasi seperti SPI Slave atau UART untuk memastikan pengiriman data cepat dan efisien.

Tujuan

- Meningkatkan stabilitas penerbangan.
- Meningkatkan responsivitas sistem.
- Mengintegrasikan sensor yang efisien
- Membangun sistem komunikasi yang efisien

EQUIPMENT

- Visual Studio Code
- ModelSim
- Intel Quartus Prime

DASAR TEORI

PID Controller adalah alat kontrol yang menjaga stabilitas dan efisiensi sistem dengan mengatur variabel seperti suhu, kecepatan, atau tekanan. Alat ini bekerja melalui tiga mekanisme: Proportional (P), yang memberikan respons sebanding dengan kesalahan saat ini; Integral (I), yang mengurangi kesalahan terakumulasi dari waktu ke waktu; dan Derivative (D), yang memprediksi kesalahan di masa depan berdasarkan perubahan saat ini. Kombinasi ketiga komponen ini memungkinkan PID Controller meningkatkan performa sistem loop tertutup, menjaga stabilitas, dan mencapai target secara efektif.

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

dimana $u(t)$ adalah output controller.

DASAR TEORI (CONT'D.)

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.

Parameter PID memiliki pengaruh yang berbeda pada kinerja sistem: Proportional Gain (**K_p**) meningkatkan kecepatan respons tetapi berisiko menyebabkan overshoot yang lebih besar; Derivative Gain (**K_d**) membantu memprediksi kesalahan dan mengurangi overshoot tanpa mempengaruhi kesalahan steady-state; sedangkan Integral Gain (**K_i**) mengurangi kesalahan steady-state dengan meningkatkan sinyal kontrol, namun dapat memperlambat respons sistem dan menyebabkan osilasi.

DASAR TEORI (contD.)

| CL RESPONSE | RISE TIME | OVERSHOOT | SETTLING TIME | S-S Error |
|----------------|--------------|-----------|---------------|-----------|
| K _p | Decrease | Increase | Small change | Decrease |
| K _i | Decrease | Increase | Increase | Decrease |
| K _d | 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. Jika ada perbedaan antara nilai aktual dan titik setel (kesalahan), algoritma kontrol akan menghitung perubahan output yang diperlukan untuk menggerakkan aktuator. Proses ini berjalan secara berkelanjutan dalam loop tertutup, di mana sistem membaca data sensor, menghitung kesalahan, dan menyesuaikan output untuk menjaga kestabilan sistem.

DASAR TEORI (CONT'D.)

Fungsi transfer loop untuk PID Controller adalah sebagai berikut:

- Proportional controller

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

- PD controller

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

- PI controller

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

Maka diperoleh untuk fungsi transfer loop tertutup untuk sistem

yang diberikan dengan PID controller adalah

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

IMPLEMENTASI

PID CONTROLLER

Sistem PID Controller berfungsi untuk mengatur sinyal output berdasarkan tiga parameter utama: Proportional (P), Integral (I), dan Derivative (D). Komponen Proportional (P) menghitung nilai berdasarkan error saat ini dengan mengalikan error tersebut dengan konstanta tertentu. Integral (I) menambahkan akumulasi error dari waktu ke waktu untuk mengatasi kesalahan jangka panjang, sedangkan Derivative (D) menghitung perubahan error dengan membandingkan error saat ini dengan error sebelumnya. Hasil akhir dari perhitungan ini digunakan untuk menghasilkan sinyal PWM yang mengontrol aktuator dalam sistem.

Beberapa sub-komponen yang mendukung proses ini termasuk Analog Converter, yang mengubah sinyal analog dari sensor menjadi data digital untuk perhitungan error, dan Error Register, yang menyimpan nilai error untuk perhitungan lebih lanjut. Komponen Integration bertugas menghitung akumulasi error yang digunakan dalam proses integral, sedangkan Trigger mengontrol kapan PID Controller memperbarui perhitungannya, dengan memicu proses berdasarkan sinyal eksternal atau siklus waktu tertentu. PWM Generator mengubah hasil PID menjadi sinyal PWM yang digunakan untuk mengontrol motor atau aktuator.

MAIN CONTROL UNIT

MotorPIDControl adalah modul yang mengimplementasikan empat kontroler PID paralel untuk mengendalikan sumbu gerakan drone: roll, pitch, yaw, dan height. Setiap kontroler PID menerima sinyal enable untuk parameter P, I, dan D, setpoint 12-bit (**SetVal**), serta nilai aktual 8-bit dari sensor (**ADC**), dan menghasilkan sinyal kontrol 32-bit (**display_output**). Modul ini berfungsi untuk stabilisasi drone dengan output diatur nol saat dinonaktifkan.

PWM Generator Motor bertanggung jawab menghasilkan sinyal PWM untuk empat motor, masing-masing memiliki generator PWM sendiri. Komponen ini menerima input duty cycle 12-bit, clock, dan menghasilkan output PWM normal serta inverted untuk menggerakkan motor.

PENGUJIAN 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.

```
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;
```

PENGUJIAN PID CONTROLLER

```

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;
END PROCEDURE;

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

-- 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(4095, 12));
    ADC <= std_logic_vector(to_unsigned(0, 8));
    wait for CLK_PERIOD*10;
    print_test_case("Full PID", 4095, 0, 153);

    report "==== Test Complete ===" severity note;
    wait for CLK_PERIOD;
    std.env.stop;
END PROCESS;

END Behavioral;

```

PENGUJIAN 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.TEXTIO.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 & " ====";
    PORT MAP (
        m1, m2, m3, m4
    );
    END PROCEDURE;
    -- Procedure to inject periodic disturbances
    PROCEDURE inject_disturbance(
        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;
    -- Component Instantiation
    UUT : FlightControlUnit
    GENERIC MAP (
        data_width => 32,
        internal_width => 16,
        pwm_resolution => 12
    )
    BEGIN
        -- System Ready check
        IF system_ready = '1' THEN
            REPORT "System Ready: " & STD_LOGIC'image(system_ready);
        END IF;
        -- Motor PWM values
        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;
END ARCHITECTURE;
```

PENGUJIAN FLIGHT CONTROL UNIT

```

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,
motor1_pwm => motor1_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;

```

```

    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;

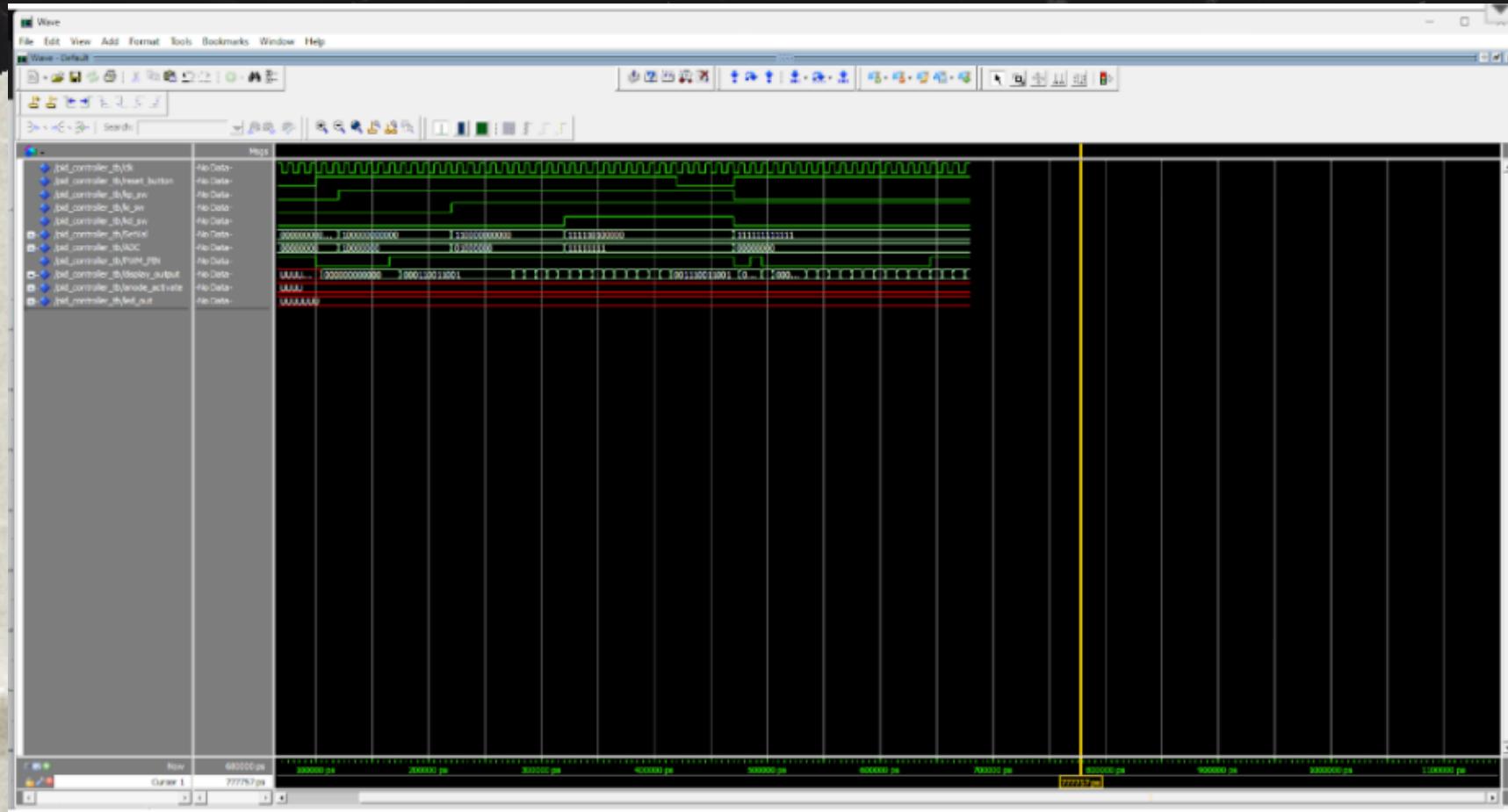
-- Add monitoring process for detailed debugging
monitor_proc : PROCESS
BEGIN
    WAIT FOR CLK_PERIOD;
    IF reset = '1' THEN
        REPORT "Current values:" &
               " Roll SP: " &
               integer'image(to_integer(unsigned(roll_setpoint))) &
               " Roll Act: " &
         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

```

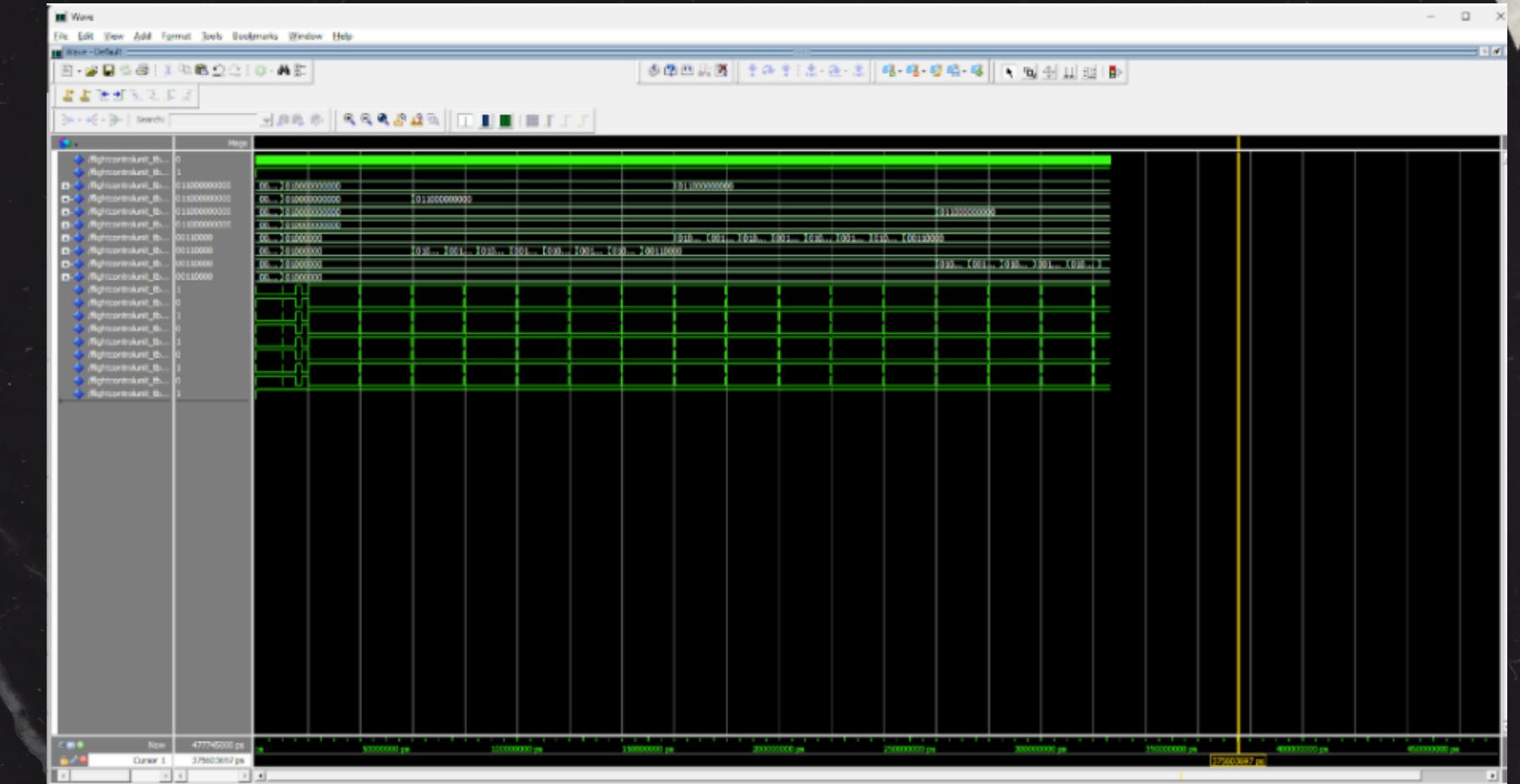
END Behavioral;

HASIC

PID Controller



Flight Control Unit



ANALISIS

- **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.

- **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.

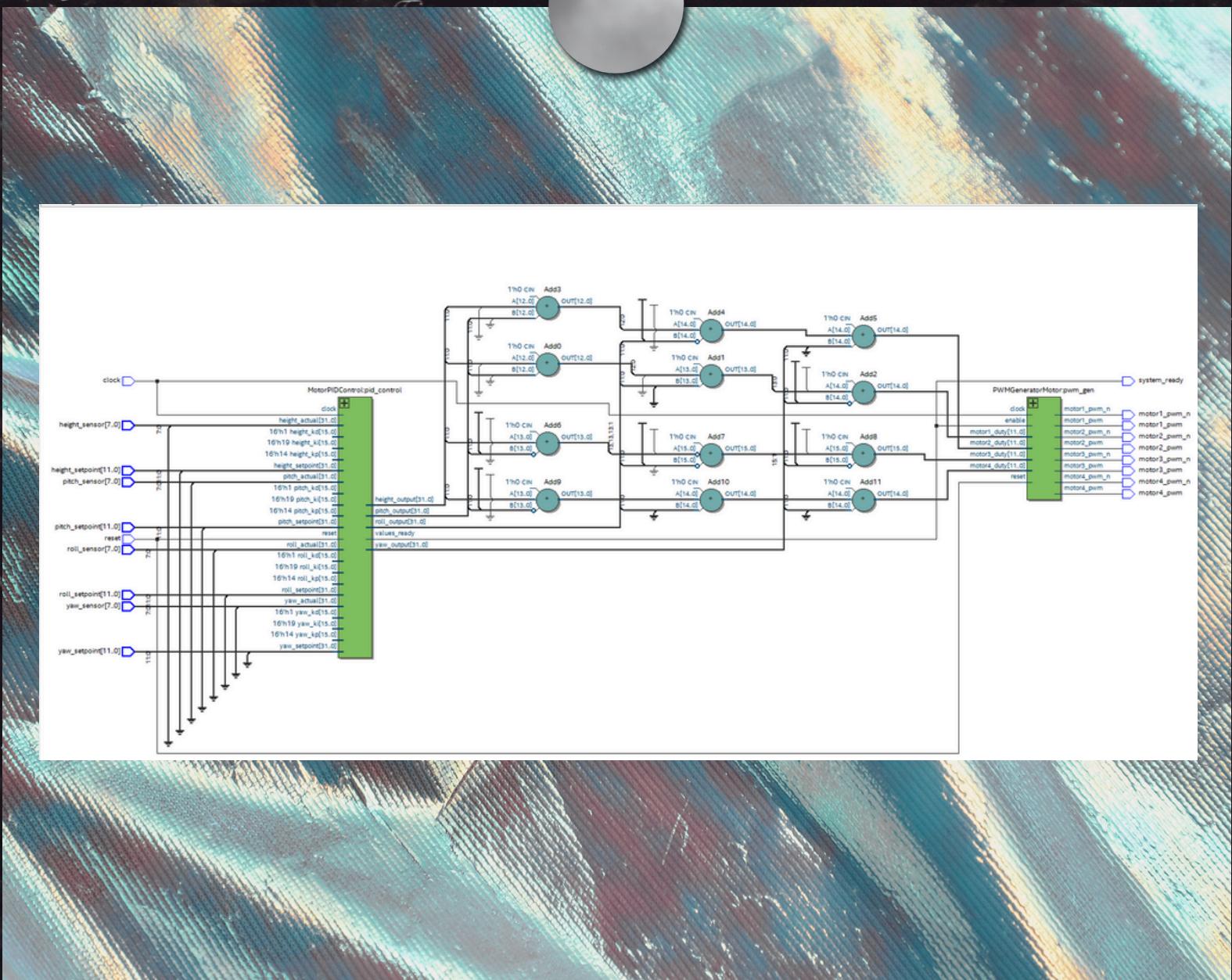
KESIMPULAN

Proyek ini bertujuan untuk merancang dan mengimplementasikan Flight Controller Unit (FCU) menggunakan VHDL yang menjadi bagian utama sistem pesawat tanpa awak. FCU dirancang dengan komponen utama seperti PID Controller dan PWM Generator yang berfungsi mengendalikan stabilitas penerbangan pada roll, pitch, yaw, dan ketinggian. Sistem ini menggunakan teori PID untuk menghasilkan respons yang cepat, stabilitas tinggi, serta efisiensi komunikasi dengan sensor seperti LiDAR. Hasil pengujian menunjukkan bahwa FCU mampu menstabilkan drone dalam mode penerbangan seperti hover, takeoff, dan landing dengan osilasi yang semakin berkurang. Proyek ini juga menggunakan testbench untuk menganalisis performa sistem dengan pendekatan dataflow, behavioral, dan structural, serta memanfaatkan looping, function, dan microprogramming untuk memastikan arsitektur modular yang stabil, responsif, dan efisien.

REFERENSI

- 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).
- NI, “The PID Controller & Theory Explained,” www.ni.com, Sep. 09, 2024.
<https://www.ni.com/en/shop/labview/pid-theory-explained.html#section-2054068860> (accessed Dec. 06, 2024).
- 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> (accessed Dec. 06, 2024).

APPEnDICES



Project Schematic



Proses Penggeraan

TERIMA KASIH

