FPGA FLIGHT CONTROL

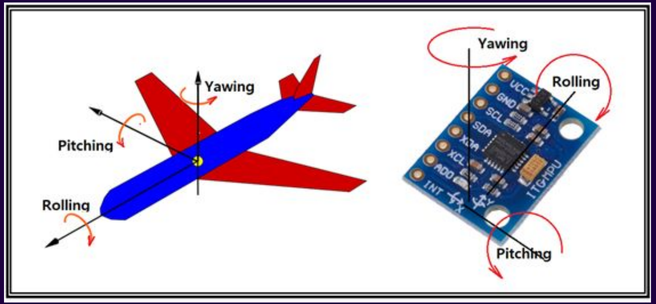
CPE 487 - FINAL PROJECT

Diego Giraldo Tabares, Aliona Heitz

TABLE OF CONTENTS

ABOUT THE PROJECT

Using the onboard accelerometer, continuously monitor the X,Y, and Z axes to simulate a flight control system. It calculates real time orientation and generates a corresponding PWM signal used to control a servo motor, mimicking how flight surfaces like ailerons or rudders respond to pitch or roll movements in an actual aircraft.



GOALS FOR THE PROJECT

1

2

3

DATA COLLECTION

Sensor interface and setting up a SPI master logic

DATA DISPLAY

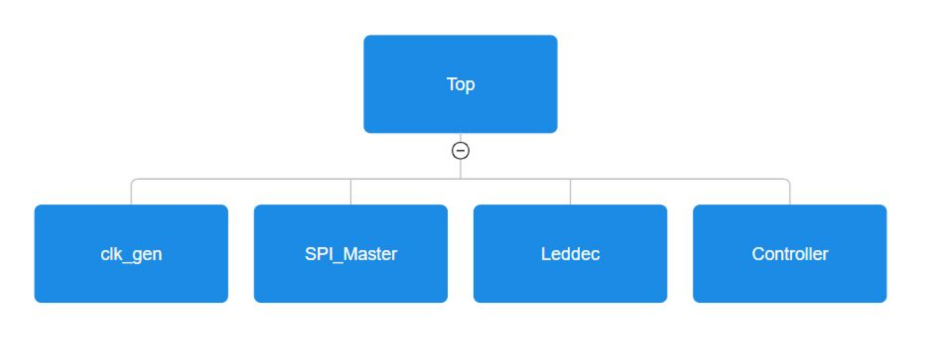
LEDs and eddec

CONTROL, OUTPUT SIGNALS

PWM signals, binary data comparison threshold, servo control

HIERARCHY OF THE CODE1

- 6 files : top, clk_gen, SPI_Master, leddec, controller, cnstr



- Top.vhd instantiates:
    - Clk_gen : generates low frequency clocks
    - SPI_Master : SPI master module that communicates with sensor
    - Leddec : drives 7-segment display
    - Controller : computes and sends servo control signal

- 1 Constraint file:
    - Assigns the 100 MHz input clock to pin E3
    - Configures SPI pins (CL_MISO, ACLD_MOSI, ACL_SCLK and AL_SS) to specific pins
    - Maps 16 LEDs
    - Connects each of the 7 segments to their pins
    - Maps anode control for all digits, enabling digit multiplexing
    - Declares pin for PWM output (servo)

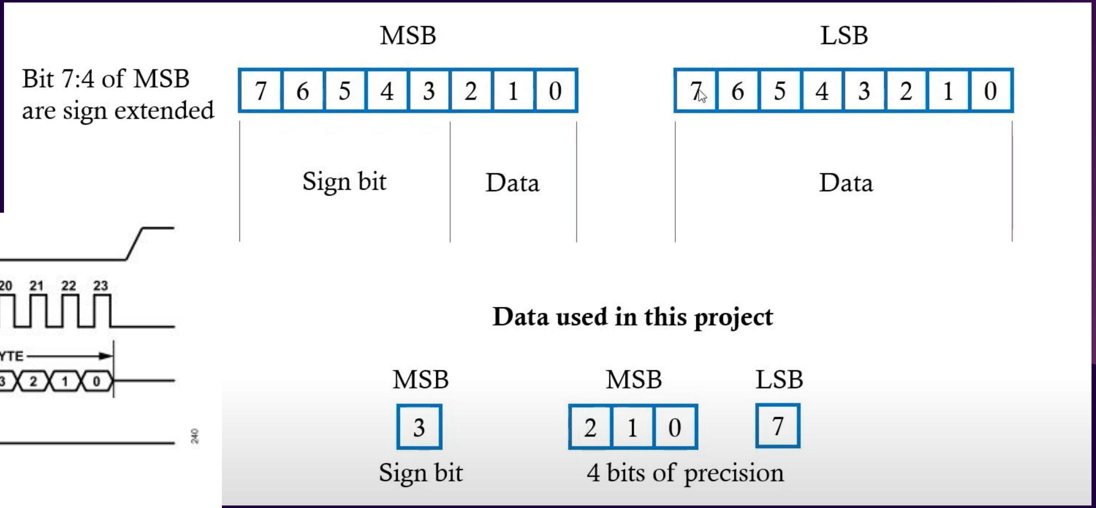
HIERARCHY cont. - Clk_gen

- The counter goes from 0 to 24, completing one full cycle every 25 clock ticks.
- At count = 12, it toggles clk_reg (halfway point), creating a rising or falling edge.
- At count = 24, it toggles clk_reg again and resets the counter.

```
---- keeps track of 25 system clock cycles to create the 4MHz signal ~52% duty cycle
process(clk_100MHz)
begin
    if rising_edge(clk_100MHz) then
        if counter = 12 then
            clk_reg <= not clk_reg;
            counter <= counter + 1;
        elsif counter = 24 then
            clk_reg <= not clk_reg;
            counter <= 0;
        else
            counter <= counter + 1;
        end if;
    end if;
end process;
```

1

DATA COLLECTION (1/3)



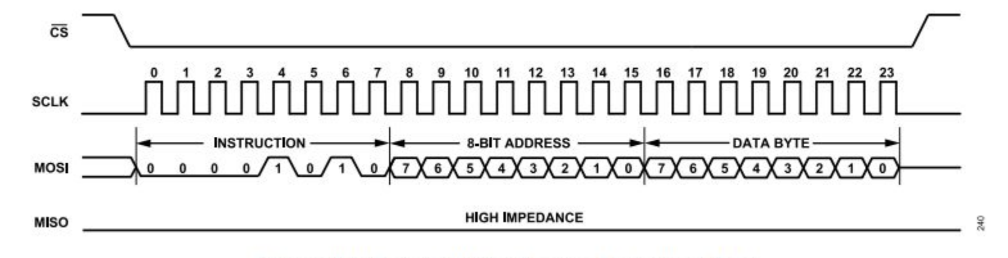


Figure 37. Register Write (Receive Instruction Only)

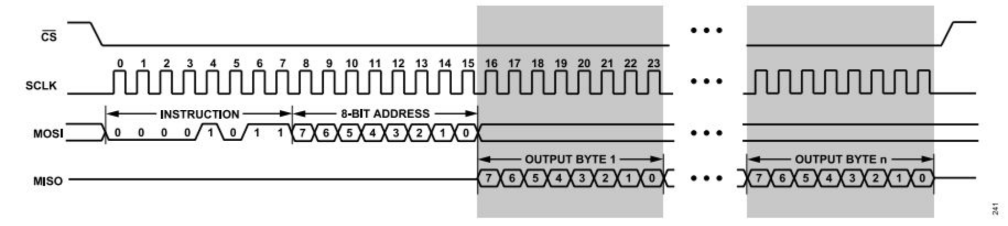


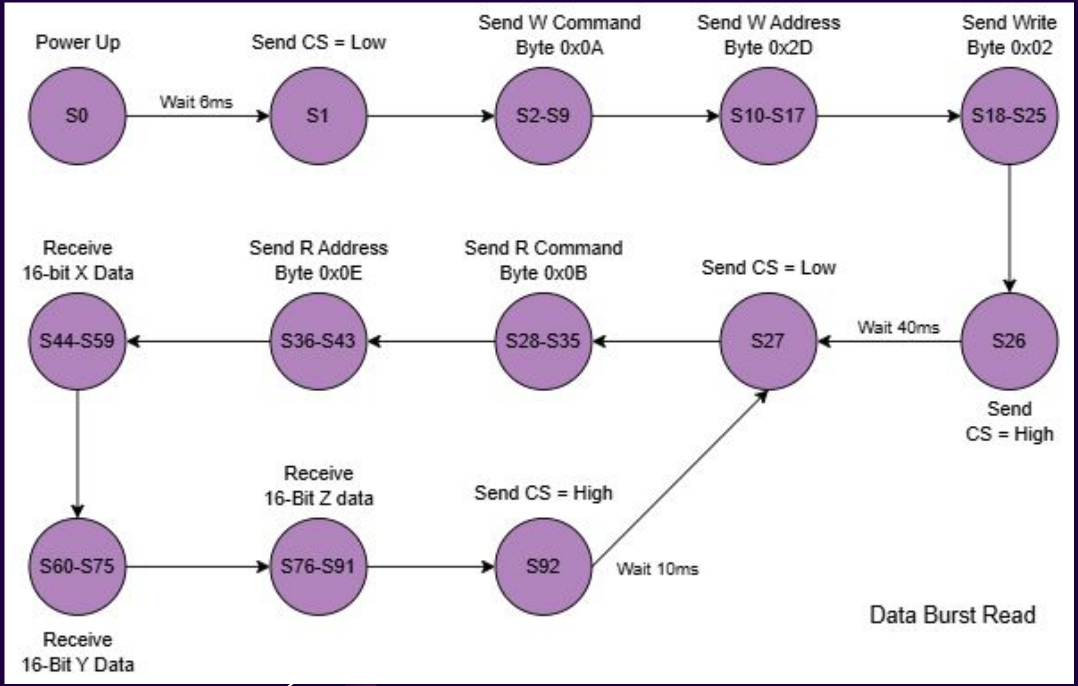
Figure 38. Burst Read

- The ADXL362 communicates with the FPGA using SPI mode 0
- Three different ways to read data {8 bits, 16 bits, FIFO}
- Output data rate is 100Hz with a range of ±2g

1

DATA COLLECTION (2/3)

State Transition Diagram



IMPLEMENTATION

1

DATA COLLECTION (3/3)

- Generates 1MHz from 4Mhz
- Detects the rising and falling edge of the 1MHz (act as flags), Manages FSM transition, has the timing for the delays
- FSM logic depending on current_state, sclk_rising, sclk_falling, count, MISO

```
process(clk_4MHz) -- creates the 1MHz from 4MHz
begin
    if rising_edge(clk_4MHz) then
        if sclk_counter = 1 then
            sclk_counter <= 0;
            sclk_reg <= not sclk_reg;
        else
            sclk_counter <= sclk_counter + 1;
        end if;
    end if;
end process;
```

```
process(clk_4MHz) -- detects rising edge of 1MHz for sclk
begin
    if rising_edge(clk_4MHz) then
        sclk_prev <= sclk_curr;
        sclk_curr <= sclk_reg;

        if (sclk_prev = '0' and sclk_curr = '1') then
            sclk_rising <= '1';  -- Rising edge detected
        else
            sclk_rising <= '0';
        end if;

        if (sclk_prev = '1' and sclk_curr = '0') then
            sclk_falling <= '1'; -- Falling edge detected
        else
            sclk_falling <= '0';
        end if;
```

```
case current_state is

    when S0 =>
        if count < 24000 then
            count <= count + 1;
        end if;

    when S26 =>
        if count < 160000 then
            count <= count + 1;
        end if;

    when S92 =>
        temp_acl_ALL <= (temp_x(11 downto 7) & temp_y(11 downto 7) & temp_z(11 downto 7));
        temp_acl_X <= temp_x;
        temp_acl_Y <= temp_y;
        temp_acl_Z <= temp_z;

        if count < 40000 then
            count <= count + 1;
        end if;
```

```
process(current_state, sclk_rising, sclk_falling, count, MISO)
begin
    ss_reg <= '1';
    mosi_reg <= '0';

    case current_state is
        ---- Power up waits 6ms
        when S0 =>
            if count < 24000 then -- 6ms
                next_state <= S0;
            else
                next_state <= S1;
            end if;
        ---- Sends CS = Low
        when S1 =>
            ss_reg <= '0';
            next_state <= S2;
        ---- Sends Write command Byte 0x0A
        when S2 =>  -- Bit 7 (0)
            ss_reg <= '0';
            mosi_reg <= write_instr(7);
            if sclk_falling = '1' then
                next_state <= S3;
            else
                next_state <= S2;
            end if;
```

```
when S59 =>
    ss_reg <= '0';
    temp_x(8) <= MISO;
    if sclk_rising = '1' then
        next_state <= S60;
    else
        next_state <= S59;
    end if;
---- Y-data LSB
```

We use the 7-segment display on the FPGA to show the acceleration data from the sensor in real time.

2

DISPLAY (1/3)

Using Leddec, LEDS, and switches

- Data Segmentation

The accelerometer sends a 15 bit signal through the SPI to the FPGA. It has three 5-bit unsigned values

- Bits 14–10 represent the X-axis
- Bits 9–5 represent the Y-axis
- Bits 4–0 represent the Z-axis

```
--     --Split 15-bit word into three 5-bit unsigned values
    X_bin <= UNSIGNED(acl_dataALL(13 DOWNTO 10));
    Y_bin <= UNSIGNED(acl_dataALL( 8 DOWNTO  5));
    Z_bin <= UNSIGNED(acl_dataALL( 3 DOWNTO  0));

    -- Converting binary
    X_tens  <= TO_UNSIGNED( TO_INTEGER(X_bin) / 10, 4 );
    X_ones  <= TO_UNSIGNED( TO_INTEGER(X_bin) MOD 10, 4 );
    Y_tens  <= TO_UNSIGNED( TO_INTEGER(Y_bin) / 10, 4 );
    Y_ones  <= TO_UNSIGNED( TO_INTEGER(Y_bin) MOD 10, 4 );
    Z_tens  <= TO_UNSIGNED( TO_INTEGER(Z_bin) / 10, 4 );
    Z_ones  <= TO_UNSIGNED( TO_INTEGER(Z_bin) MOD 10, 4 );
```

- Data Conversion

Since data comes in as binary, change it to decimal through division and modulo operations. Each 5 bit value is split into two BCD digits (27 becomes 2 and 7).

2

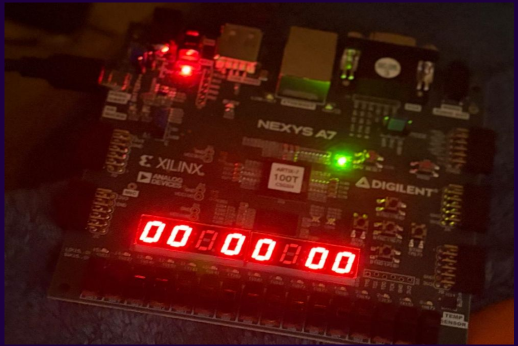
DISPLAY (2/3)

Using Leddec, LEDS, and switches

We use the 7-segment display on the FPGA to show the acceleration data from the sensor in real time.

- Display on Leddec with multiplexing

All digits are then packed into a 32 bit signal, and data display gets refreshed at a 125hz frequency) Which looks like this




```
--packing eight nibbles: X_tens, X_one
bcd32 <= STD_LOGIC_VECTOR(X_tens)  &
         STD_LOGIC_VECTOR(X_ones)  &
         "1111" &
         STD_LOGIC_VECTOR(Y_tens)  &
         STD_LOGIC_VECTOR(Y_ones)  &
         "1111" &
         STD_LOGIC_VECTOR(Z_tens)  &
         STD_LOGIC_VECTOR(Z_ones);
```

2

DISPLAY (3/3)

Using Leddec, LEDS, and switches

We use the LEDs on the board to show the raw data coming from the accelerometer.

- Display on LEDs
  - Each LED turns on or off depending on whether its corresponding bit is a 1 or a 0.
  - X, y and z axis coming from the SPI is shown across the LED bar.

- Alternative display
  - X, y, or z displaying on the bottom of led depending on which switch is on

3

CONTROL/OUTPUT SIGNALS

```
-- 50Hz signal with variable duty cycle
process(CLK_100MHZ)
begin
    if rising_edge(CLK_100MHZ) then
        if pwm_cnt < pwm_duty then
            pwm_out <= '1';
        else
            pwm_out <= '0';
        end if;

        if pwm_cnt = 1999999 then
            pwm_cnt <= 0;
        else
            pwm_cnt <= pwm_cnt + 1;
        end if;
    end if;
end process;
```

```
process(CLK_100MHZ)
begin
    if rising_edge(CLK_100MHZ) then
        if x_raw > threshold then
            target_duty <= 200000; -- 2ms, right turn
        elsif x_raw < -threshold then
            target_duty <= 100000; -- 1ms, left turn
        else
            target_duty <= 150000; -- 1.5ms, middle
        end if;

        -- Slowly move pwm_duty toward target_duty
        if pwm_duty < target_duty then
            pwm_duty <= pwm_duty + 70;  -- Tune this step size for speed
        elsif pwm_duty > target_duty then
            pwm_duty <= pwm_duty - 70;
        end if;
    end if;
end process;
```

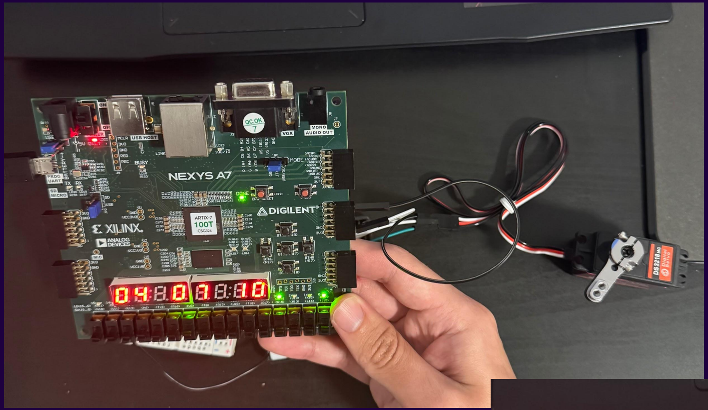
- Compares the x values against a positive and negative threshold
- Based on tilt direction it determines if it should go left or right
- Smooths out servo motion to prevent stuttering or jitter
- Generates a standard 50Hz PWM signal for servo motor

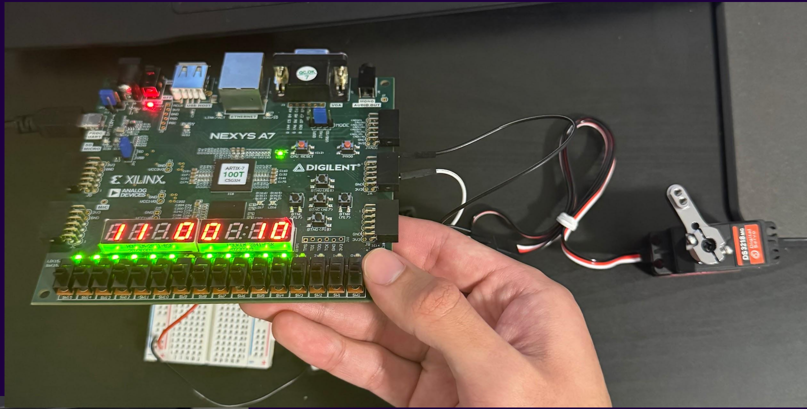
BUGS AND FIXES

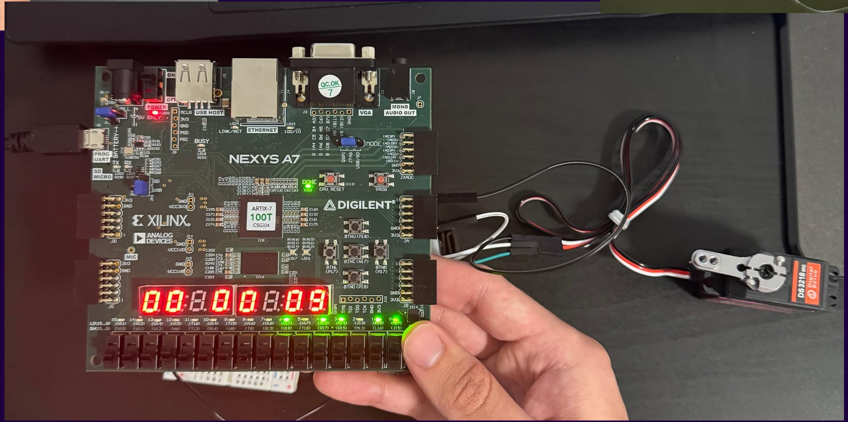
CLK

DATA / DISPLAY

DEMONSTRATION







NEXT STEPS

1

Alternative display using switches for wider range of features for user

2

Cleaning up and figuring out why the last 4 bits of data are unstable for cleaner and more accurate data

3

Implementing Y axis for the PWM in addition to X

4

Angle conversion for real-life application

5

PID algorithm for accurate stabilizing controls

THANK YOU.
Questions?

https://github.com/alionaheitz/CPE487Project/tree/main