```
function [complex_data] = process_labsat_ls3w(ls3w_filename, ini_filename)
    % Function to process LabSat .ls3w and .ini files and extract I/Q data

    %% Step 1: Parse the .ini file
    ini_data = read_ini_file(ini_filename);

    % Extract important parameters from the .ini file
    sample_rate = str2double(ini_data.config.SMP);  % Sample rate
    quantization = str2double(ini_data.config.QUA); % Quantization bits per sample
    num_channels = str2double(ini_data.config.CHN); % Number of RF channels

    % Display information about the file
    fprintf('Sample Rate: %d Sps\n', sample_rate);
    fprintf('Quantization: %d bits per sample\n', quantization);
    fprintf('Number of Channels: %d\n', num_channels);

    %% Step 2: Open the .ls3w file and read binary data
    fid = fopen(ls3w_filename, 'rb');
    if fid == -1
        error('Could not open the .ls3w file.');
    end

    % Read the entire file into a uint64 array (assuming 64-bit registers as per LS3W ↙
format)
    raw_data = fread(fid, 'uint64=>uint64');
    fclose(fid);

    %% Step 3: Process the binary data
    num_samples_per_register = get_samples_per_register(quantization, num_channels);
    complex_data = decode_ls3w_registers(raw_data, quantization, num_channels, ↙
num_samples_per_register);

    % Display how many complex samples were decoded
    fprintf('Total Complex Samples: %d\n', numel(complex_data));
end

%% Helper function to read the .ini configuration file
function ini_data = read_ini_file(filename)
    ini_data = struct();

    fid = fopen(filename, 'r');
    if fid == -1
        error('Could not open the .ini file.');
    end

    % Read and parse the .ini file line by line
    while ~feof(fid)
        line = fgetl(fid);
        if startsWith(line, '[config]')
```

```
            section = 'config';
        elseif startsWith(line, '[channel A]')
            section = 'channel_A';
        elseif startsWith(line, '[channel B]')
            section = 'channel_B';
        elseif startsWith(line, '[channel C]')
            section = 'channel_C';
        else
            % Extract key-value pairs
            tokens = regexp(line, '(.*)=(.*)', 'tokens');
            if ~isempty(tokens)
                key = strtrim(tokens{1}{1});
                value = strtrim(tokens{1}{2});
                ini_data.(section).(key) = value;
            end
        end
    end

    fclose(fid);
end

%% Helper function to calculate the number of samples per register
function num_samples = get_samples_per_register(quantization, num_channels)
    % Based on quantization and number of channels, determine how many I/Q samples
    % fit into a 64-bit register
    switch quantization
        case 1
            num_samples = floor(64 / (quantization * 2 * num_channels));
        case 2
            num_samples = floor(64 / (quantization * 2 * num_channels));
        case 3
            num_samples = floor(64 / (quantization * 2 * num_channels));
        otherwise
            error('Unsupported quantization level: %d', quantization);
    end
end

function complex_data = decode_ls3w_registers(raw_data, quantization, num_channels, ↙
num_samples_per_register)
    % Initialize arrays to store the I and Q components
    total_samples = numel(raw_data) * num_samples_per_register;
    complex_data = zeros(total_samples, 1);

    % Spare bits and shift bits are based on the .ini configuration and quantization
    d_ls3w_spare_bits = 64 - num_samples_per_register * quantization * 2;  % Adjust↙
based on quantization
    d_ls3w_SFT = quantization * 2;   % Shift per sample

    % Channel offset logic (adjust based on channels selected)
```

```
    d_ls3w_selected_channel_offset = (1:num_channels) * quantization * 2;

    sample_counter = 1;

    % Process each 64-bit register
    for reg_idx = 1:numel(raw_data)
        % Convert the 64-bit register into individual bits (bitset)
        bs = bitget(raw_data(reg_idx), 1:64); % Extract bits from least significant to most significant

        % Iterate over each sample in the register
        for sample_idx = 0:(num_samples_per_register-1)
            for channel_idx = 1:num_channels
                % Compute the bit positions for I and Q in this sample
                bit_offset = d_ls3w_spare_bits + sample_idx * d_ls3w_SFT + d_ls3w_selected_channel_offset(channel_idx);

                % Initialize I and Q sample values
                sampleI = 0.0;
                sampleQ = 0.0;

                switch quantization
                    case 1
                        % 1 bit per I and Q
                        sampleI = bs(bit_offset) * 2 - 1; % Convert {0, 1} to {-1, 1}
                        sampleQ = bs(bit_offset + 1) * 2 - 1;
                    case 2
                        % 2 bits per I and Q (4 states: 00, 01, 10, 11)
                        sampleI = decode_2bit_to_float(bs(bit_offset:bit_offset+1));
                        sampleQ = decode_2bit_to_float(bs(bit_offset+2: bit_offset+3));
                    case 3
                        % 3 bits per I and Q (8 states: 000, 001, 010, ... 111)
                        sampleI = decode_3bit_to_float(bs(bit_offset:bit_offset+2));
                        sampleQ = decode_3bit_to_float(bs(bit_offset+3: bit_offset+5));
                    otherwise
                        error('Unsupported quantization level: %d', quantization);
                end

                % Store the complex sample (I + jQ)
                complex_data(sample_counter) = complex(sampleI, sampleQ);
                sample_counter = sample_counter + 1;
            end
        end
    end
end

% Helper function to decode 2-bit signed values into float
```

```
function value = decode_2bit_to_float(bits)
    if bits(1) == 1
        if bits(2) == 1
            value = -0.5;  % 11 -> -0.5
        else
            value = -1.0;  % 10 -> -1.0
        end
    else
        if bits(2) == 1
            value = 1.0;   % 01 -> 1.0
        else
            value = 0.5;   % 00 -> 0.5
        end
    end
end

% Helper function to decode 3-bit signed values into float
function value = decode_3bit_to_float(bits)
    if bits(1) == 1
        if bits(2) == 1
            if bits(3) == 1
                value = -0.25; % 111 -> -0.25
            else
                value = -0.5;  % 110 -> -0.5
            end
        else
            if bits(3) == 1
                value = -0.75; % 101 -> -0.75
            else
                value = -1.0;  % 100 -> -1.0
            end
        end
    else
        if bits(2) == 1
            if bits(3) == 1
                value = 1.0;   % 011 -> 1.0
            else
                value = 0.75;  % 010 -> 0.75
            end
        else
            if bits(3) == 1
                value = 0.5;   % 001 -> 0.5
            else
                value = 0.25;  % 000 -> 0.25
            end
        end
    end
end
```