Chapter 7: Functions and Tasks

Functions and tasks are essential constructs in SystemVerilog that allow you to create reusable blocks of code, improving modularity and maintainability. While they serve similar purposes, they have distinct characteristics and use cases.

Function Declarations and Calls

Functions in SystemVerilog are subroutines that return a value and execute in zero simulation time. They are ideal for computational operations and combinational logic.

Basic Function Syntax

```
function [return_type] function_name ([arguments]);
    // Function body
    return return_value;
endfunction
```

Simple Function Examples

```
module function_examples;

    // Function to add two integers
    function int add(int a, int b);
        return a + b;
    endfunction

    // Function to find maximum of two values
    function int max(int x, int y);
        if (x > y)
            return x;
        else
            return y;
    endfunction

    // Function with bit vector operations
    function logic [7:0] reverse_bits(logic [7:0] data);
        logic [7:0] result;
        for (int i = 0; i < 8; i++) begin
            result[i] = data[7-i];
        end
        return result;
    endfunction

    initial begin
```

```
        int result1, result2, result3;
        logic [7:0] original = 8'b10110100;
        logic [7:0] reversed;

        result1 = add(15, 25);           // Returns 40
        result2 = max(100, 50);          // Returns 100
        reversed = reverse_bits(original); // Returns 8'b00101101

        $display("Add result: %d", result1);
        $display("Max result: %d", result2);
        $display("Original: %b, Reversed: %b", original, reversed);
    end

endmodule
```

Functions with Different Return Types

```
module function_types;

    // Function returning a structure
    typedef struct {
        int quotient;
        int remainder;
    } div_result_t;

    function div_result_t divide(int dividend, int divisor);
        div_result_t result;
        result.quotient = dividend / divisor;
        result.remainder = dividend % divisor;
        return result;
    endfunction

    // Function returning an array
    function logic [3:0] [7:0] create_pattern(logic [7:0] base);
        logic [3:0] [7:0] pattern;
        for (int i = 0; i < 4; i++) begin
            pattern[i] = base << i;
        end
        return pattern;
    endfunction

    initial begin
        div_result_t div_res;
        logic [3:0] [7:0] pattern;

        div_res = divide(17, 5);
        pattern = create_pattern(8'b00001111);

        $display("17/5 = %d remainder %d", div_res.quotient, div_res.remainder);

        for (int i = 0; i < 4; i++) begin
            $display("Pattern[%d]: %b", i, pattern[i]);
        end
    end
```

```
endmodule
```

Task Declarations and Calls

Tasks are subroutines that can consume simulation time and don't return values directly. They can have input, output, and inout arguments, making them suitable for complex operations and time-consuming activities.

Basic Task Syntax

```
task task_name ([arguments]);
    // Task body
endtask
```

Task Examples

```
module task_examples;

    logic clk = 0;
    logic [7:0] data;
    logic valid;

    // Generate clock
    always #5 clk = ~clk;

    // Task to wait for a number of clock cycles
    task wait_cycles(int num_cycles);
        repeat(num_cycles) @(posedge clk);
    endtask

    // Task to send data with handshaking
    task send_data(input logic [7:0] send_data, output logic done);
        data = send_data;
        valid = 1'b1;
        @(posedge clk);
        valid = 1'b0;
        done = 1'b1;
        @(posedge clk);
        done = 1'b0;
    endtask

    // Task with multiple outputs
    task analyze_data(input logic [7:0] input_data,
                      output int ones_count,
                      output int zeros_count,
                      output logic parity);
        ones_count = 0;
        zeros_count = 0;

        for (int i = 0; i < 8; i++) begin
            if (input_data[i])
                ones_count++;
```

```
            else
                zeros_count++;
        end

        parity = ^input_data; // XOR reduction for parity
    endtask

    initial begin
        logic done;
        int ones, zeros;
        logic par;

        // Wait for some cycles
        wait_cycles(3);

        // Send some data
        send_data(8'hA5, done);
        $display("Data sent, done = %b", done);

        // Analyze data
        analyze_data(8'b11010110, ones, zeros, par);
        $display("Ones: %d, Zeros: %d, Parity: %b", ones, zeros, par);

        #100 $finish;
    end

endmodule
```

Automatic vs. Static Lifetime

The lifetime of variables in functions and tasks can be either static (default) or automatic. This affects how variables are allocated and whether they retain values between calls.

Static Lifetime (Default)

```
module static_lifetime;

    // Static function - variables retain values between calls
    function int counter();
        int count = 0;  // Initialized only once
        count++;
        return count;
    endfunction

    // Static task
    task static_task();
        static int call_count = 0;
        call_count++;
        $display("Static task called %d times", call_count);
    endtask

    initial begin
        $display("Counter: %d", counter()); // Prints 1
        $display("Counter: %d", counter()); // Prints 2
```

```
        $display("Counter: %d", counter()); // Prints 3

        static_task(); // Prints: Static task called 1 times
        static_task(); // Prints: Static task called 2 times
    end

endmodule
```

Automatic Lifetime

```
module automatic_lifetime;

    // Automatic function - fresh variables for each call
    function automatic int factorial(int n);
        if (n <= 1)
            return 1;
        else
            return n * factorial(n-1); // Recursive call possible
    endfunction

    // Automatic task
    task automatic print_sequence(int start, int count);
        for (int i = 0; i < count; i++) begin
            $display("Value: %d", start + i);
            #10; // Can consume time
        end
    endtask

    initial begin
        $display("5! = %d", factorial(5)); // Prints 120

        // Multiple concurrent task calls
        fork
            print_sequence(10, 3);
            print_sequence(20, 3);
        join
    end

endmodule
```

Pass by Reference

SystemVerilog supports passing arguments by reference using the `ref` keyword, allowing functions and tasks to modify the original variables.

Pass by Reference Examples

```
module pass_by_reference;

    // Function with reference arguments
    function automatic void swap(ref int a, ref int b);
        int temp = a;
        a = b;
```

```
        b = temp;
    endfunction

    // Task to initialize an array by reference
    task automatic init_array(ref int arr[10], input int init_value);
        for (int i = 0; i < 10; i++) begin
            arr[i] = init_value + i;
        end
    endtask

    // Function to modify a structure by reference
    typedef struct {
        int x, y;
        string name;
    } point_t;

    function automatic void move_point(ref point_t p, int dx, int dy);
        p.x += dx;
        p.y += dy;
    endfunction

    initial begin
        int x = 10, y = 20;
        int my_array[10];
        point_t my_point = '{100, 200, "PointA"};

        $display("Before swap: x=%d, y=%d", x, y);
        swap(x, y);
        $display("After swap: x=%d, y=%d", x, y);

        init_array(my_array, 50);
        $display("Array elements: %p", my_array);

        $display("Before move: %s at (%d,%d)", my_point.name, my_point.x, my_point.y);
        move_point(my_point, 15, -25);
        $display("After move: %s at (%d,%d)", my_point.name, my_point.x, my_point.y);
    end

endmodule
```

Return Statements in Functions

Functions must return a value, and the return statement determines what value is returned. You can have multiple return statements in a function.

Multiple Return Statements

```
module return_statements;

    // Function with multiple return points
    function automatic string grade_letter(int score);
        if (score >= 90)
            return "A";
        else if (score >= 80)
```

```
            return "B";
        else if (score >= 70)
            return "C";
        else if (score >= 60)
            return "D";
        else
            return "F";
    endfunction

    // Function with early return for error checking
    function automatic real safe_divide(real dividend, real divisor);
        if (divisor == 0.0) begin
            $error("Division by zero attempted");
            return 0.0; // Early return for error case
        end
        return dividend / divisor;
    endfunction

    // Function with complex logic and multiple returns
    function automatic int find_first_one(logic [31:0] data);
        for (int i = 0; i < 32; i++) begin
            if (data[i] == 1'b1)
                return i; // Return position of first '1'
        end
        return -1; // Return -1 if no '1' found
    endfunction

    initial begin
        string letter;
        real result;
        int position;

        letter = grade_letter(85);
        $display("Score 85 gets grade: %s", letter);

        result = safe_divide(10.0, 3.0);
        $display("10.0 / 3.0 = %f", result);

        result = safe_divide(5.0, 0.0); // Will show error

        position = find_first_one(32'h00008000);
        $display("First '1' found at position: %d", position);

        position = find_first_one(32'h00000000);
        $display("First '1' found at position: %d", position);
    end

endmodule
```

Void Functions

Void functions don’t return a value and are similar to tasks, but they execute in zero simulation time and cannot contain timing control statements.

Void Function Examples

```
module void_functions;

    int global_counter = 0;
    logic [7:0] memory [256];

    // Void function to increment global counter
    function automatic void increment_counter(int step);
        global_counter += step;
    endfunction

    // Void function to initialize memory
    function automatic void init_memory(logic [7:0] pattern);
        for (int i = 0; i < 256; i++) begin
            memory[i] = pattern ^ i[7:0];
        end
    endfunction

    // Void function for debug printing
    function automatic void debug_print(string msg, int value);
        $display("[DEBUG %0t] %s: %d", $time, msg, value);
    endfunction

    // Void function with reference parameter
    function automatic void reset_array(ref int arr[]);
        foreach(arr[i]) begin
            arr[i] = 0;
        end
    endfunction

    initial begin
        int test_array[5] = '{1, 2, 3, 4, 5};

        debug_print("Initial counter", global_counter);

        increment_counter(5);
        debug_print("After increment", global_counter);

        init_memory(8'hAA);
        debug_print("Memory[0]", memory[0]);
        debug_print("Memory[1]", memory[1]);
        debug_print("Memory[255]", memory[255]);

        $display("Before reset: %p", test_array);
        reset_array(test_array);
        $display("After reset: %p", test_array);
    end

endmodule
```

Best Practices and Guidelines

When to Use Functions vs. Tasks

Use Functions when: - You need to return a single value - The operation is purely combinational - No timing control is needed - The operation should complete in zero simulation time

Use Tasks when: - You need multiple outputs - Timing control is required - The operation may consume simulation time - You need to model sequential behavior

Function and Task Design Guidelines

```
module design_guidelines;

    // Good: Pure function with clear purpose
    function automatic int absolute_value(int value);
        return (value < 0) ? -value : value;
    endfunction

    // Good: Task with clear interface and timing
    task automatic wait_for_ready(ref logic ready_signal, input int timeout_cycles);
        int cycle_count = 0;
        while (!ready_signal && cycle_count < timeout_cycles) begin
            @(posedge clk);
            cycle_count++;
        end
        if (cycle_count >= timeout_cycles) begin
            $error("Timeout waiting for ready signal");
        end
    endtask

    // Good: Function with appropriate use of reference
    function automatic void normalize_vector(ref real vector[3]);
        real magnitude = $sqrt(vector[0]**2 + vector[1]**2 + vector[2]**2);
        if (magnitude != 0.0) begin
            vector[0] /= magnitude;
            vector[1] /= magnitude;
            vector[2] /= magnitude;
        end
    endfunction

endmodule
```

Summary

Functions and tasks are powerful constructs in SystemVerilog that enable code reuse and modular design:

- **Functions** return values, execute in zero time, and are ideal for combinational logic
- **Tasks** can have multiple outputs, consume time, and are suited for sequential operations
- **Automatic lifetime** creates fresh variables for each call and enables recursion
- **Static lifetime** (default) preserves variable values between calls
- **Pass by reference** allows modification of original variables
- **Void functions** provide task-like behavior without return values but in zero time

Understanding when and how to use functions and tasks effectively will greatly improve your SystemVerilog code organization and reusability.

SystemVerilog Functions and Tasks - Example Index

Function Declarations and Calls

Mathematical Functions

Basic arithmetic operations like factorial, power, greatest common divisor

```
// mathematical_calculator.sv
module mathematical_calculator ();               // Mathematical functions design

  // Function to calculate factorial
  function automatic integer calculate_factorial;
    input integer number_input;
    integer factorial_result;
    begin
      if (number_input <= 1)
        factorial_result = 1;
      else
        factorial_result = number_input * calculate_factorial(number_input - 1);
      calculate_factorial = factorial_result;
    end
  endfunction

  // Function to calculate power (base^exponent)
  function automatic integer calculate_power;
    input integer base_value;
    input integer exponent_value;
    integer power_result;
    integer loop_counter;
    begin
      power_result = 1;
      for (loop_counter = 0; loop_counter < exponent_value; loop_counter = loop_counter + 1)
        power_result = power_result * base_value;
      calculate_power = power_result;
    end
  endfunction

  // Function to calculate greatest common divisor using Euclidean algorithm
  function automatic integer calculate_gcd;
    input integer first_number;
    input integer second_number;
    integer temp_remainder;
    begin
      while (second_number != 0) begin
        temp_remainder = first_number % second_number;
        first_number = second_number;
        second_number = temp_remainder;
      end
      calculate_gcd = first_number;
    end
  endfunction

  initial begin
    $display();
    $display("=== Mathematical Functions Demonstration ===");
```

```
    // Test factorial function
    $display("Factorial Tests:");
    $display("  5! = %0d", calculate_factorial(5));
    $display("  0! = %0d", calculate_factorial(0));
    $display("  3! = %0d", calculate_factorial(3));

    // Test power function
    $display("Power Tests:");
    $display("  2^3 = %0d", calculate_power(2, 3));
    $display("  5^2 = %0d", calculate_power(5, 2));
    $display("  10^0 = %0d", calculate_power(10, 0));

    // Test GCD function
    $display("Greatest Common Divisor Tests:");
    $display("  GCD(48, 18) = %0d", calculate_gcd(48, 18));
    $display("  GCD(100, 25) = %0d", calculate_gcd(100, 25));
    $display("  GCD(17, 13) = %0d", calculate_gcd(17, 13));

    $display();
  end

endmodule
```

```
// mathematical_calculator_testbench.sv
module math_calculator_testbench;  // Testbench for mathematical calculator

  mathematical_calculator MATH_CALCULATOR_INSTANCE();  // Instantiate mathematical calculator

  // Test variables
  integer test_factorial_result;
  integer test_power_result;
  integer test_gcd_result;

  initial begin
    // Dump waves for simulation
    $dumpfile("math_calculator_testbench.vcd");
    $dumpvars(0, math_calculator_testbench);

    #1;  // Wait for design to initialize

    $display();
    $display("=== Testbench Verification ===");

    // Verify factorial calculations
    test_factorial_result = MATH_CALCULATOR_INSTANCE.calculate_factorial(4);
    $display("Testbench: 4! = %0d (Expected: 24)", test_factorial_result);

    // Verify power calculations
    test_power_result = MATH_CALCULATOR_INSTANCE.calculate_power(3, 4);
    $display("Testbench: 3^4 = %0d (Expected: 81)", test_power_result);

    // Verify GCD calculations
    test_gcd_result = MATH_CALCULATOR_INSTANCE.calculate_gcd(60, 48);
    $display("Testbench: GCD(60, 48) = %0d (Expected: 12)", test_gcd_result);
```

```
    $display();
    $display("Mathematical functions testbench completed successfully!");
    $display();

    #10;  // Wait before finishing
    $finish;
  end

endmodule
```

```
Verilator Simulation Output:
================================================================================

=== Mathematical Functions Demonstration ===
Factorial Tests:
  5! = 120
  0! = 1
  3! = 6
Power Tests:
  2^3 = 8
  5^2 = 25
  10^0 = 1
Greatest Common Divisor Tests:
  GCD(48, 18) = 6
  GCD(100, 25) = 25
  GCD(17, 13) = 1


=== Testbench Verification ===
Testbench: 4! = 24 (Expected: 24)
Testbench: 3^4 = 81 (Expected: 81)
Testbench: GCD(60, 48) = 12 (Expected: 12)

Mathematical functions testbench completed successfully!

================================================================================
Process finished with return code: 0
Removing Chapter_7_examples/example_1__mathematical_functions/obj_dir directory...
Chapter_7_examples/example_1__mathematical_functions/obj_dir removed successfully.

0
```

String Manipulation Functions

Functions for string processing, parsing, and formatting operations

```
// string_processor.sv
module string_processor ();

  // String manipulation functions demonstration
  function automatic string format_name(string first_name, string last_name);
    return $sformatf("%s %s", first_name, last_name);
  endfunction

  function automatic string extract_extension(string filename);
```

```
    int dot_position;
    dot_position = filename.len() - 1;

    // Find the last dot in filename
    while (dot_position >= 0 && filename[dot_position] != ".") begin
      dot_position--;
    end

    if (dot_position >= 0)
      return filename.substr(dot_position + 1, filename.len() - 1);
    else
      return "no_extension";
  endfunction

  function automatic string reverse_string(string input_text);
    string reversed_text = "";
    for (int i = input_text.len() - 1; i >= 0; i--) begin
      reversed_text = {reversed_text, input_text[i]};
    end
    return reversed_text;
  endfunction

  function automatic int count_vowels(string text);
    int vowel_count = 0;
    string lowercase_text;

    lowercase_text = text.tolower();

    for (int i = 0; i < lowercase_text.len(); i++) begin
      case (lowercase_text[i])
        "a", "e", "i", "o", "u": vowel_count++;
        default: ; // Non-vowel characters, do nothing
      endcase
    end
    return vowel_count;
  endfunction

  initial begin
    string first_name = "Alice";
    string last_name = "Johnson";
    string full_name;
    string filename = "report.txt";
    string file_extension;
    string sample_text = "SystemVerilog";
    string reversed_text;
    int vowel_count;

    $display("=== String Manipulation Functions Demo ===");
    $display();

    // Format name demonstration
    full_name = format_name(first_name, last_name);
    $display("Name formatting:");
    $display("  First: %s, Last: %s", first_name, last_name);
    $display("  Full name: %s", full_name);
```

```
    $display();

    // File extension extraction
    file_extension = extract_extension(filename);
    $display("File extension extraction:");
    $display("  Filename: %s", filename);
    $display("  Extension: %s", file_extension);
    $display();

    // String reversal
    reversed_text = reverse_string(sample_text);
    $display("String reversal:");
    $display("  Original: %s", sample_text);
    $display("  Reversed: %s", reversed_text);
    $display();

    // Vowel counting
    vowel_count = count_vowels(sample_text);
    $display("Vowel counting:");
    $display("  Text: %s", sample_text);
    $display("  Vowel count: %0d", vowel_count);
    $display();
  end

endmodule
```

```
// string_processor_testbench.sv
module string_processor_testbench;

  // Instantiate the string processor design
  string_processor STRING_PROCESSOR_INSTANCE();

  // Additional testbench verification
  task automatic verify_string_functions();
    string test_first = "Bob";
    string test_last = "Smith";
    string expected_full = "Bob Smith";
    string actual_full;

    string test_filename = "data.csv";
    string expected_ext = "csv";
    string actual_ext;

    string test_word = "hello";
    string expected_reverse = "olleh";
    string actual_reverse;

    int expected_vowels = 2; // "e" and "o" in "hello"
    int actual_vowels;

    $display("=== Testbench Verification ===");
    $display();

    // Test name formatting
```

```
    actual_full = STRING_PROCESSOR_INSTANCE.format_name(test_first, test_last);
    $display("Name Format Test:");
    $display("  Expected: %s", expected_full);
    $display("  Actual:   %s", actual_full);
    $display("  Result:   %s", (actual_full == expected_full) ? "PASS" : "FAIL");
    $display();

    // Test file extension extraction
    actual_ext = STRING_PROCESSOR_INSTANCE.extract_extension(test_filename);
    $display("Extension Test:");
    $display("  Expected: %s", expected_ext);
    $display("  Actual:   %s", actual_ext);
    $display("  Result:   %s", (actual_ext == expected_ext) ? "PASS" : "FAIL");
    $display();

    // Test string reversal
    actual_reverse = STRING_PROCESSOR_INSTANCE.reverse_string(test_word);
    $display("Reverse Test:");
    $display("  Expected: %s", expected_reverse);
    $display("  Actual:   %s", actual_reverse);
    $display("  Result:   %s", (actual_reverse == expected_reverse) ? "PASS" : "FAIL");
    $display();

    // Test vowel counting
    actual_vowels = STRING_PROCESSOR_INSTANCE.count_vowels(test_word);
    $display("Vowel Count Test:");
    $display("  Expected: %0d", expected_vowels);
    $display("  Actual:   %0d", actual_vowels);
    $display("  Result:   %s", (actual_vowels == expected_vowels) ? "PASS" : "FAIL");
    $display();

  endtask

  initial begin
    // Dump waves for analysis
    $dumpfile("string_processor_testbench.vcd");
    $dumpvars(0, string_processor_testbench);

    #1; // Wait for design to initialize

    $display("Hello from string processor testbench!");
    $display();

    // Run verification tests
    verify_string_functions();

    $display("=== Test Summary ===");
    $display("String manipulation functions tested successfully!");
    $display();

    #5; // Additional time before finish
    $finish;
  end

endmodule
```

```
Verilator Simulation Output:
================================================================================
=== String Manipulation Functions Demo ===

Name formatting:
  First: Alice, Last: Johnson
  Full name: Alice Johnson

File extension extraction:
  Filename: report.txt
  Extension: txt

String reversal:
  Original: SystemVerilog
  Reversed: golireVmetsyS

Vowel counting:
  Text: SystemVerilog
  Vowel count: 4

Hello from string processor testbench!

=== Testbench Verification ===

Name Format Test:
  Expected: Bob Smith
  Actual:   Bob Smith
  Result:   PASS

Extension Test:
  Expected: csv
  Actual:   csv
  Result:   PASS

Reverse Test:
  Expected: olleh
  Actual:   olleh
  Result:   PASS

Vowel Count Test:
  Expected: 2
  Actual:   2
  Result:   PASS

=== Test Summary ===
String manipulation functions tested successfully!

================================================================================
Process finished with return code: 0
Removing Chapter_7_examples/example_2__string_manipulation/obj_dir directory...
Chapter_7_examples/example_2__string_manipulation/obj_dir removed successfully.
0
```

Bit Manipulation Functions

Functions for bit operations, masking, shifting, and bit counting

```
// bit_manipulation_functions.sv
module bit_manipulation_functions ();

  // Function to count number of set bits (population count)
  function automatic int count_ones(input logic [31:0] data_word);
    int bit_count = 0;
    for (int bit_index = 0; bit_index < 32; bit_index++) begin
      if (data_word[bit_index]) bit_count++;
    end
    return bit_count;
  endfunction

  // Function to reverse bits in a byte
  function automatic logic [7:0] reverse_byte_bits(input logic [7:0] byte_in);
    logic [7:0] reversed_byte;
    for (int bit_pos = 0; bit_pos < 8; bit_pos++) begin
      reversed_byte[7-bit_pos] = byte_in[bit_pos];
    end
    return reversed_byte;
  endfunction

  // Function to create bit mask with specified width and position
  function automatic logic [31:0] create_bit_mask(input int mask_width,
                                                  input int start_position);
    logic [31:0] generated_mask = 0;
    for (int mask_bit = 0; mask_bit < mask_width; mask_bit++) begin
      generated_mask[start_position + mask_bit] = 1'b1;
    end
    return generated_mask;
  endfunction

  // Function to extract bit field from word
  function automatic logic [31:0] extract_bit_field(input logic [31:0] source_word,
                                                    input int field_width,
                                                    input int field_position);
    logic [31:0] extraction_mask;
    extraction_mask = create_bit_mask(field_width, field_position);
    return (source_word & extraction_mask) >> field_position;
  endfunction

  // Function to perform circular left shift
  function automatic logic [15:0] circular_left_shift(input logic [15:0] shift_data,
                                                      input int shift_amount);
    int effective_shift = shift_amount % 16;
    return (shift_data << effective_shift) | (shift_data >> (16 - effective_shift));
  endfunction

  initial begin
    logic [31:0] test_word = 32'hA5C3_F0E1;
    logic [7:0] test_byte = 8'b1010_0011;
    logic [15:0] rotation_data = 16'hBEEF;

    $display("=== Bit Manipulation Functions Demo ===");
    $display();
```

```
    // Test count_ones function
    $display("Count ones in 0x%08h: %0d bits", test_word, count_ones(test_word));

    // Test reverse_byte_bits function
    $display("Reverse bits in 0x%02h: 0x%02h", test_byte,
             reverse_byte_bits(test_byte));

    // Test create_bit_mask function
    $display("Create 4-bit mask at position 8: 0x%08h",
             create_bit_mask(4, 8));

    // Test extract_bit_field function
    $display("Extract 8 bits from position 16 of 0x%08h: 0x%02h",
             test_word, extract_bit_field(test_word, 8, 16));

    // Test circular_left_shift function
    $display("Circular left shift 0x%04h by 4 positions: 0x%04h",
             rotation_data, circular_left_shift(rotation_data, 4));

    $display();
  end

endmodule
```

```
// bit_manipulation_functions_testbench.sv
module bit_manipulation_functions_testbench;

  // Instantiate design under test
  bit_manipulation_functions BIT_MANIP_DUT();

  // Test variables
  logic [31:0] test_data_word;
  logic [7:0] test_input_byte;
  logic [15:0] shift_test_data;
  int expected_ones_count;
  logic [7:0] expected_reversed_byte;
  logic [31:0] expected_mask_value;
  logic [31:0] expected_extracted_field;
  logic [15:0] expected_shifted_result;

  initial begin
    // Setup waveform dump
    $dumpfile("bit_manipulation_functions_testbench.vcd");
    $dumpvars(0, bit_manipulation_functions_testbench);

    $display("=== Comprehensive Bit Manipulation Testing ===");
    $display();

    // Test Case 1: Count ones function verification
    test_data_word = 32'hFFFF_0000;  // 16 ones, 16 zeros
    expected_ones_count = 16;
    #1;
    if (BIT_MANIP_DUT.count_ones(test_data_word) == expected_ones_count) begin
      $display("PASS: count_ones(0x%08h) = %0d",
```

```
            test_data_word, BIT_MANIP_DUT.count_ones(test_data_word));
    end else begin
      $display("FAIL: count_ones(0x%08h) = %0d, expected %0d",
            test_data_word, BIT_MANIP_DUT.count_ones(test_data_word),
            expected_ones_count);
    end

    // Test Case 2: Byte bit reversal verification
    test_input_byte = 8'b1100_0011;  // 0xC3
    expected_reversed_byte = 8'b1100_0011;  // Same when palindromic
    #1;
    if (BIT_MANIP_DUT.reverse_byte_bits(test_input_byte) == expected_reversed_byte) begin
      $display("PASS: reverse_byte_bits(0x%02h) = 0x%02h",
            test_input_byte, BIT_MANIP_DUT.reverse_byte_bits(test_input_byte));
    end else begin
      $display("FAIL: reverse_byte_bits(0x%02h) = 0x%02h, expected 0x%02h",
            test_input_byte, BIT_MANIP_DUT.reverse_byte_bits(test_input_byte),
            expected_reversed_byte);
    end

    // Test Case 3: Bit mask creation verification
    expected_mask_value = 32'h0000_0F00;  // 4 bits at position 8
    #1;
    if (BIT_MANIP_DUT.create_bit_mask(4, 8) == expected_mask_value) begin
      $display("PASS: create_bit_mask(4, 8) = 0x%08h",
            BIT_MANIP_DUT.create_bit_mask(4, 8));
    end else begin
      $display("FAIL: create_bit_mask(4, 8) = 0x%08h, expected 0x%08h",
            BIT_MANIP_DUT.create_bit_mask(4, 8), expected_mask_value);
    end

    // Test Case 4: Bit field extraction verification
    test_data_word = 32'hABCD_EF12;
    expected_extracted_field = 32'h0000_00CD;  // Bits [15:8] = 0xCD
    #1;
    if (BIT_MANIP_DUT.extract_bit_field(test_data_word, 8, 8) ==
        expected_extracted_field) begin
      $display("PASS: extract_bit_field(0x%08h, 8, 8) = 0x%08h",
            test_data_word, BIT_MANIP_DUT.extract_bit_field(test_data_word, 8, 8));
    end else begin
      $display("FAIL: extract_bit_field(0x%08h, 8, 8) = 0x%08h, expected 0x%08h",
            test_data_word, BIT_MANIP_DUT.extract_bit_field(test_data_word, 8, 8),
            expected_extracted_field);
    end

    // Test Case 5: Circular shift verification
    shift_test_data = 16'hF000;  // 1111_0000_0000_0000
    expected_shifted_result = 16'h000F;  // After 4-bit left shift
    #1;
    if (BIT_MANIP_DUT.circular_left_shift(shift_test_data, 4) ==
        expected_shifted_result) begin
      $display("PASS: circular_left_shift(0x%04h, 4) = 0x%04h",
            shift_test_data,
            BIT_MANIP_DUT.circular_left_shift(shift_test_data, 4));
    end else begin
```

```
      $display("FAIL: circular_left_shift(0x%04h, 4) = 0x%04h, expected 0x%04h",
               shift_test_data,
               BIT_MANIP_DUT.circular_left_shift(shift_test_data, 4),
               expected_shifted_result);
    end

    $display();
    $display("=== Bit Manipulation Testing Complete ===");

    #1;
    $finish;
  end

endmodule
```

```
Verilator Simulation Output:
================================================================================
=== Bit Manipulation Functions Demo ===

Count ones in 0xa5c3f0e1: 16 bits
Reverse bits in 0xa3: 0xc5
Create 4-bit mask at position 8: 0x00000f00
Extract 8 bits from position 16 of 0xa5c3f0e1: 0xc3
Circular left shift 0xbeef by 4 positions: 0xeefb

=== Comprehensive Bit Manipulation Testing ===

PASS: count_ones(0xffff0000) = 16
PASS: reverse_byte_bits(0xc3) = 0xc3
PASS: create_bit_mask(4, 8) = 0x00000f00
FAIL: extract_bit_field(0xabcdef12, 8, 8) = 0x000000ef, expected 0x000000cd
PASS: circular_left_shift(0xf000, 4) = 0x000f

=== Bit Manipulation Testing Complete ===
================================================================================
Process finished with return code: 0
Removing Chapter_7_examples/example_3__bit_manipulation_functions/obj_dir directory...
Chapter_7_examples/example_3__bit_manipulation_functions/obj_dir removed successfully.

0
```

CRC Calculator Function

Function to calculate cyclic redundancy check for data integrity

```
// crc_calculator_design.sv
module crc_calculator_design_module (
  input  logic        clock_signal,
  input  logic        reset_active_low,
  input  logic        data_valid_input,
  input  logic [7:0]  data_byte_input,
  output logic [7:0]  crc_checksum_output,
  output logic        crc_calculation_complete
);

  // CRC-8 polynomial: x^8 + x^2 + x^1 + 1 (0x07)
```

```
  parameter logic [7:0] CRC_POLYNOMIAL_CONSTANT = 8'h07;

  // Internal registers for CRC calculation
  logic [7:0] crc_register_current;
  logic [7:0] crc_register_next;
  logic       calculation_in_progress;

  // CRC calculation logic
  always_comb begin
    crc_register_next = crc_register_current;

    if (data_valid_input) begin
      // XOR input data with current CRC
      crc_register_next = crc_register_current ^ data_byte_input;

      // Process each bit through the polynomial
      for (int bit_position = 0; bit_position < 8; bit_position++) begin
        if (crc_register_next[7]) begin
          crc_register_next = (crc_register_next << 1) ^ CRC_POLYNOMIAL_CONSTANT;
        end else begin
          crc_register_next = crc_register_next << 1;
        end
      end
    end
  end

  // Sequential logic for state updates
  always_ff @(posedge clock_signal or negedge reset_active_low) begin
    if (!reset_active_low) begin
      crc_register_current     <= 8'h00;
      calculation_in_progress  <= 1'b0;
      crc_calculation_complete <= 1'b0;
    end else begin
      crc_register_current <= crc_register_next;

      if (data_valid_input) begin
        calculation_in_progress  <= 1'b1;
        crc_calculation_complete <= 1'b0;
      end else if (calculation_in_progress) begin
        calculation_in_progress  <= 1'b0;
        crc_calculation_complete <= 1'b1;
      end else begin
        crc_calculation_complete <= 1'b0;
      end
    end
  end

  // Output assignment
  assign crc_checksum_output = crc_register_current;

  // Display messages for debugging
  initial $display("CRC Calculator Design Module Initialized");

  always @(posedge crc_calculation_complete) begin
    $display("CRC calculation complete! Checksum: 0x%02h",
```

```
            crc_checksum_output);
  end

endmodule
```

```
// crc_calculator_design_testbench.sv
module crc_calculator_testbench_module;

  // Testbench signals with descriptive names
  logic        clock_signal_tb;
  logic        reset_active_low_tb;
  logic        data_valid_input_tb;
  logic [7:0]  data_byte_input_tb;
  logic [7:0]  crc_checksum_output_tb;
  logic        crc_calculation_complete_tb;

  // Test data array for CRC calculation
  logic [7:0] test_data_array [4] = '{8'hAB, 8'hCD, 8'hEF, 8'h12};
  integer     data_index_counter;

  // Instantiate the CRC calculator design under test
  crc_calculator_design_module CRC_CALCULATOR_INSTANCE (
    .clock_signal              (clock_signal_tb),
    .reset_active_low          (reset_active_low_tb),
    .data_valid_input          (data_valid_input_tb),
    .data_byte_input           (data_byte_input_tb),
    .crc_checksum_output       (crc_checksum_output_tb),
    .crc_calculation_complete  (crc_calculation_complete_tb)
  );

  // Clock generation - 10ns period
  always #5 clock_signal_tb = ~clock_signal_tb;

  // Main test sequence
  initial begin
    // Initialize waveform dump
    $dumpfile("crc_calculator_testbench_module.vcd");
    $dumpvars(0, crc_calculator_testbench_module);

    // Display test start message
    $display();
    $display("=== CRC Calculator Testbench Starting ===");
    $display();

    // Initialize all signals
    clock_signal_tb      = 1'b0;
    reset_active_low_tb  = 1'b1;
    data_valid_input_tb  = 1'b0;
    data_byte_input_tb   = 8'h00;
    data_index_counter   = 0;

    // Apply reset sequence
    $display("Applying reset sequence...");
    #10 reset_active_low_tb = 1'b0;  // Assert reset
```

```
    #20 reset_active_low_tb = 1'b1;  // Release reset
    #10;

    // Test CRC calculation with sample data
    $display("Starting CRC calculation with test data:");
    for (data_index_counter = 0; data_index_counter < 4;
         data_index_counter++) begin

      $display("Processing byte %0d: 0x%02h",
               data_index_counter, test_data_array[data_index_counter]);

      // Present data to CRC calculator
      data_byte_input_tb  = test_data_array[data_index_counter];
      data_valid_input_tb = 1'b1;
      #10;

      // Remove data valid signal
      data_valid_input_tb = 1'b0;
      #10;

      // Wait for calculation completion
      wait (crc_calculation_complete_tb);
      #5;
    end

    // Final CRC result display
    $display();
    $display("Final CRC checksum result: 0x%02h", crc_checksum_output_tb);
    $display();

    // Test reset functionality
    $display("Testing reset functionality...");
    #10 reset_active_low_tb = 1'b0;
    #10 reset_active_low_tb = 1'b1;
    #10;

    $display("CRC after reset: 0x%02h", crc_checksum_output_tb);
    $display();

    // Complete testbench execution
    $display("=== CRC Calculator Testbench Complete ===");
    $display();
    #50;
    $finish;
  end

  // Monitor for important signal changes
  initial begin
    forever begin
      @(posedge crc_calculation_complete_tb);
      $display("Time %0t: CRC calculation completed, result = 0x%02h",
               $time, crc_checksum_output_tb);
    end
  end
```

```
endmodule
```

```
Verilator Simulation Output:
================================================================================
CRC Calculator Design Module Initialized

=== CRC Calculator Testbench Starting ===

Applying reset sequence...
Starting CRC calculation with test data:
Processing byte 0: 0xab
Time 55: CRC calculation completed, result = 0x58
CRC calculation complete! Checksum: 0x58
Processing byte 1: 0xcd
Time 75: CRC calculation completed, result = 0xe2
CRC calculation complete! Checksum: 0xe2
Processing byte 2: 0xef
Time 105: CRC calculation completed, result = 0x23
CRC calculation complete! Checksum: 0x23
Processing byte 3: 0x12
Time 125: CRC calculation completed, result = 0x97
CRC calculation complete! Checksum: 0x97

Final CRC checksum result: 0x97

Testing reset functionality...
CRC after reset: 0x00

=== CRC Calculator Testbench Complete ===

================================================================================
Process finished with return code: 0
Removing Chapter_7_examples/example_4__crc_calculator/obj_dir directory...
Chapter_7_examples/example_4__crc_calculator/obj_dir removed successfully.

0
```

Lookup Table Function

Function implementing ROM or lookup table functionality

```
// lookup_table_rom.sv
module lookup_table_rom (
    input  logic [2:0] address_input,           // 3-bit address for 8 entries
    output logic [7:0] data_output              // 8-bit data output
);

    // ROM lookup table with 8 entries of 8-bit data
    logic [7:0] rom_memory [0:7];

    // Initialize ROM with predefined values
    initial begin
        rom_memory[0] = 8'h10;  // Address 0: 0x10
        rom_memory[1] = 8'h25;  // Address 1: 0x25
        rom_memory[2] = 8'h3A;  // Address 2: 0x3A
        rom_memory[3] = 8'h47;  // Address 3: 0x47
```

```
        rom_memory[4] = 8'h5C;  // Address 4: 0x5C
        rom_memory[5] = 8'h69;  // Address 5: 0x69
        rom_memory[6] = 8'h7E;  // Address 6: 0x7E
        rom_memory[7] = 8'h83;  // Address 7: 0x83
    end

    // Combinational lookup - output changes immediately with address
    assign data_output = rom_memory[address_input];

endmodule
```

```
// lookup_table_rom_testbench.sv
module lookup_table_rom_testbench;

    // Testbench signals
    logic [2:0] test_address;                   // Address input signal
    logic [7:0] expected_data;                  // Expected data output
    logic [7:0] actual_data;                    // Actual data from ROM

    // Instantiate the design under test
    lookup_table_rom ROM_INSTANCE (
        .address_input(test_address),
        .data_output(actual_data)
    );

    initial begin
        // Setup waveform dumping
        $dumpfile("lookup_table_rom_testbench.vcd");
        $dumpvars(0, lookup_table_rom_testbench);

        $display();
        $display("=== ROM Lookup Table Test Started ===");
        $display();

        // Test all ROM addresses
        for (int addr = 0; addr < 8; addr++) begin
            test_address = addr[2:0];           // Set address
            #1;                                 // Wait for combinational delay

            // Display results
            $display("Address: %0d (0x%h) -> Data: 0x%h",
                     test_address, test_address, actual_data);
        end

        $display();
        $display("=== Verification Test ===");

        // Verify specific values
        test_address = 3'b000; #1;
        if (actual_data !== 8'h10)
            $display(
                "ERROR: Address 0 expected 0x10, got 0x%h", actual_data);

        test_address = 3'b011; #1;
```

```
        if (actual_data !== 8'h47)
            $display(
                "ERROR: Address 3 expected 0x47, got 0x%h", actual_data);

        test_address = 3'b111; #1;
        if (actual_data !== 8'h83)
            $display(
                "ERROR: Address 7 expected 0x83, got 0x%h", actual_data);

        $display();
        $display("=== ROM Lookup Table Test Completed ===");
        $display();

        $finish;
    end

endmodule
```

```
Verilator Simulation Output:
================================================================================

=== ROM Lookup Table Test Started ===

Address: 0 (0x0) -> Data: 0x10
Address: 1 (0x1) -> Data: 0x25
Address: 2 (0x2) -> Data: 0x3a
Address: 3 (0x3) -> Data: 0x47
Address: 4 (0x4) -> Data: 0x5c
Address: 5 (0x5) -> Data: 0x69
Address: 6 (0x6) -> Data: 0x7e
Address: 7 (0x7) -> Data: 0x83

=== Verification Test ===

=== ROM Lookup Table Test Completed ===

================================================================================
Process finished with return code: 0
Removing Chapter_7_examples/example_5__lookup_table_function/obj_dir directory...
Chapter_7_examples/example_5__lookup_table_function/obj_dir removed successfully.

0
```

Data Conversion Functions

Functions for converting between different data types and formats

```
// data_converter_unit.sv
module data_converter_unit();                    // Data conversion functions

  // Function to convert binary to BCD (Binary Coded Decimal)
  function automatic [7:0] binary_to_bcd(input [7:0] binary_value);
    reg [7:0] bcd_result;
    reg [7:0] temp_value;
    integer digit_index;
    begin
```

```
    bcd_result = 8'b0;
    temp_value = binary_value;

    // Convert using double dabble algorithm (simplified for 8-bit)
    for (digit_index = 0; digit_index < 8; digit_index++) begin
      // Add 3 to BCD digits >= 5 before shifting
      if (bcd_result[3:0] >= 5) bcd_result[3:0] = bcd_result[3:0] + 3;
      if (bcd_result[7:4] >= 5) bcd_result[7:4] = bcd_result[7:4] + 3;

      // Shift left and bring in next binary bit
      bcd_result = {bcd_result[6:0], temp_value[7]};
      temp_value = temp_value << 1;
    end

    binary_to_bcd = bcd_result;
  end
endfunction

// Function to convert signed to unsigned with overflow detection
function automatic [8:0] signed_to_unsigned_safe(input signed [7:0]
                                                 signed_input);
  reg overflow_flag;
  reg [7:0] unsigned_result;
  begin
    if (signed_input < 0) begin
      overflow_flag = 1'b1;
      unsigned_result = 8'b0;  // Clamp to zero for negative values
    end else begin
      overflow_flag = 1'b0;
      unsigned_result = signed_input;
    end

    // Return {overflow_flag, unsigned_result}
    signed_to_unsigned_safe = {overflow_flag, unsigned_result};
  end
endfunction

// Function to convert temperature from Celsius to Fahrenheit
function automatic [15:0] celsius_to_fahrenheit(input signed [7:0]
                                                celsius_temp);
  reg signed [15:0] fahrenheit_result;
  begin
    // F = (C * 9/5) + 32, using fixed point arithmetic
    fahrenheit_result = (celsius_temp * 9) / 5 + 32;
    celsius_to_fahrenheit = fahrenheit_result;
  end
endfunction

// Function to pack RGB values into single word
function automatic [23:0] pack_rgb_color(input [7:0] red_channel,
                                         input [7:0] green_channel,
                                         input [7:0] blue_channel);
  begin
    pack_rgb_color = {red_channel, green_channel, blue_channel};
  end
```

```
  endfunction

  // Function to extract RGB components from packed color
  function automatic [23:0] unpack_rgb_color(input [23:0] packed_color,
                                             input [1:0] channel_select);
    begin
      case (channel_select)
        2'b00: unpack_rgb_color = {16'b0, packed_color[23:16]}; // Red
        2'b01: unpack_rgb_color = {16'b0, packed_color[15:8]};  // Green
        2'b10: unpack_rgb_color = {16'b0, packed_color[7:0]};   // Blue
        2'b11: unpack_rgb_color = packed_color;                 // All
      endcase
    end
  endfunction

  initial begin
    $display();
    $display("Data Conversion Functions Demonstration");
    $display("=======================================");
    $display();
  end

endmodule
```

```
// data_converter_unit_testbench.sv
module data_conversion_testbench;              // Data conversion testbench

  // Instantiate the data converter unit
  data_converter_unit CONVERTER_INSTANCE();

  // Test variables
  reg [7:0] test_binary_value;
  reg signed [7:0] test_signed_value;
  reg signed [7:0] test_celsius_temp;
  reg [7:0] test_red, test_green, test_blue;
  reg [23:0] test_packed_color;
  integer test_channel_select;

  // Result variables
  reg [7:0] bcd_result;
  reg [8:0] unsigned_result;
  reg [15:0] fahrenheit_result;
  reg [23:0] packed_result;
  reg [23:0] unpacked_result;

  initial begin
    // Setup waveform dumping
    $dumpfile("data_conversion_testbench.vcd");
    $dumpvars(0, data_conversion_testbench);

    #1;  // Wait for initialization

    $display("Testing Binary to BCD Conversion:");
    $display("================================");
```

```
    // Test binary to BCD conversion
    test_binary_value = 8'd42;
    bcd_result = CONVERTER_INSTANCE.binary_to_bcd(test_binary_value);
    $display(
        "Binary %d converts to BCD 0x%h", test_binary_value, bcd_result);

    test_binary_value = 8'd99;
    bcd_result = CONVERTER_INSTANCE.binary_to_bcd(test_binary_value);
    $display(
        "Binary %d converts to BCD 0x%h", test_binary_value, bcd_result);

    $display();
    $display("Testing Signed to Unsigned Conversion:");
    $display("======================================");

    // Test signed to unsigned conversion
    test_signed_value = -25;
    unsigned_result = CONVERTER_INSTANCE.signed_to_unsigned_safe(
                      test_signed_value);
    $display(
        "Signed %d converts to unsigned %d (overflow: %b)",
        test_signed_value, unsigned_result[7:0], unsigned_result[8]);

    test_signed_value = 100;
    unsigned_result = CONVERTER_INSTANCE.signed_to_unsigned_safe(
                      test_signed_value);
    $display(
        "Signed %d converts to unsigned %d (overflow: %b)",
        test_signed_value, unsigned_result[7:0], unsigned_result[8]);

    $display();
    $display("Testing Temperature Conversion:");
    $display("===============================");

    // Test temperature conversion
    test_celsius_temp = 0;
    fahrenheit_result = CONVERTER_INSTANCE.celsius_to_fahrenheit(
                        test_celsius_temp);
    $display(
        "Temperature %d Celsius converts to %d Fahrenheit",
        test_celsius_temp, fahrenheit_result);

    test_celsius_temp = 25;
    fahrenheit_result = CONVERTER_INSTANCE.celsius_to_fahrenheit(
                        test_celsius_temp);
    $display(
        "Temperature %d Celsius converts to %d Fahrenheit",
        test_celsius_temp, fahrenheit_result);

    $display();
    $display("Testing RGB Color Packing/Unpacking:");
    $display("====================================");

    // Test RGB color packing
    test_red = 8'hFF;    // Maximum red
```

```
    test_green = 8'h80;  // Medium green
    test_blue = 8'h40;   // Low blue

    packed_result = CONVERTER_INSTANCE.pack_rgb_color(test_red, test_green,
                                                      test_blue);
    $display("RGB(%h, %h, %h) packs to 0x%h", test_red, test_green,
             test_blue, packed_result);

    // Test RGB color unpacking
    test_packed_color = 24'hFF8040;

    for (test_channel_select = 0; test_channel_select < 4;
         test_channel_select = test_channel_select + 1) begin
      unpacked_result = CONVERTER_INSTANCE.unpack_rgb_color(
                        test_packed_color, test_channel_select[1:0]);
      case (test_channel_select[1:0])
        2'b00: $display("Red channel: 0x%h", unpacked_result[7:0]);
        2'b01: $display("Green channel: 0x%h", unpacked_result[7:0]);
        2'b10: $display("Blue channel: 0x%h", unpacked_result[7:0]);
        2'b11: $display("All channels: 0x%h", unpacked_result);
      endcase
    end

    $display();
    $display("Data conversion function testing completed successfully!");
    $display();

  end

endmodule
```

```
Verilator Simulation Output:
================================================================================

Data Conversion Functions Demonstration
========================================

Testing Binary to BCD Conversion:
================================
Binary  42 converts to BCD 0x42
Binary  99 converts to BCD 0x99

Testing Signed to Unsigned Conversion:
====================================
Signed  -25 converts to unsigned   0 (overflow: 1)
Signed  100 converts to unsigned 100 (overflow: 0)

Testing Temperature Conversion:
==============================
Temperature    0 Celsius converts to    32 Fahrenheit
Temperature   25 Celsius converts to    77 Fahrenheit

Testing RGB Color Packing/Unpacking:
==================================
RGB(ff, 80, 40) packs to 0xff8040
Red channel: 0xff
```

```
Green channel: 0x80
Blue channel: 0x40
All channels: 0xff8040

Data conversion function testing completed successfully!
================================================================================
Process finished with return code: 0
Removing Chapter_7_examples/example_6__data_conversion_functions/obj_dir directory...
Chapter_7_examples/example_6__data_conversion_functions/obj_dir removed successfully.

0
```

Functions with Different Return Types

Complex Number Operations

Functions returning structured data for complex arithmetic operations

```
// complex_arithmetic_processor.sv
module complex_arithmetic_processor ();

  // Define complex number structure
  typedef struct packed {
    logic signed [15:0] real_component;
    logic signed [15:0] imaginary_component;
  } complex_number_type;

  // Function to add two complex numbers and return structured result
  function complex_number_type add_complex_numbers(
    complex_number_type first_complex_operand,
    complex_number_type second_complex_operand
  );
    complex_number_type addition_result;
    addition_result.real_component = first_complex_operand.real_component +
                                     second_complex_operand.real_component;
    addition_result.imaginary_component =
      first_complex_operand.imaginary_component +
      second_complex_operand.imaginary_component;
    return addition_result;
  endfunction

  // Function to multiply two complex numbers and return structured result
  function complex_number_type multiply_complex_numbers(
    complex_number_type first_complex_operand,
    complex_number_type second_complex_operand
  );
    complex_number_type multiplication_result;
    // (a + bi) * (c + di) = (ac - bd) + (ad + bc)i
    multiplication_result.real_component =
      (first_complex_operand.real_component *
       second_complex_operand.real_component) -
      (first_complex_operand.imaginary_component *
       second_complex_operand.imaginary_component);

    multiplication_result.imaginary_component =
      (first_complex_operand.real_component *
```

```
      second_complex_operand.imaginary_component) +
     (first_complex_operand.imaginary_component *
      second_complex_operand.real_component);

  return multiplication_result;
endfunction

// Function to calculate magnitude squared (avoids square root)
function logic [31:0] calculate_magnitude_squared(
  complex_number_type input_complex_number
);
  return (input_complex_number.real_component *
          input_complex_number.real_component) +
         (input_complex_number.imaginary_component *
          input_complex_number.imaginary_component);
endfunction

initial begin
  complex_number_type first_number = '{real_component: 16'd3,
                                       imaginary_component: 16'd4};
  complex_number_type second_number = '{real_component: 16'd1,
                                        imaginary_component: 16'd2};
  complex_number_type addition_result;
  complex_number_type multiplication_result;
  logic [31:0] magnitude_squared_result;

  $display();
  $display("Complex Number Arithmetic Operations Demo");
  $display("=========================================");

  // Display input numbers
  $display("First Number:  %0d + %0di",
           first_number.real_component,
           first_number.imaginary_component);
  $display("Second Number: %0d + %0di",
           second_number.real_component,
           second_number.imaginary_component);
  $display();

  // Perform addition
  addition_result = add_complex_numbers(first_number, second_number);
  $display("Addition Result: %0d + %0di",
           addition_result.real_component,
           addition_result.imaginary_component);

  // Perform multiplication
  multiplication_result = multiply_complex_numbers(first_number,
                                                   second_number);
  $display("Multiplication Result: %0d + %0di",
           multiplication_result.real_component,
           multiplication_result.imaginary_component);

  // Calculate magnitude squared of first number
  magnitude_squared_result = calculate_magnitude_squared(first_number);
  $display("Magnitude Squared of First Number: %0d",
```

```
              magnitude_squared_result);
    $display();
  end

endmodule
```

```
// complex_arithmetic_processor_testbench.sv
module complex_arithmetic_testbench;

  // Instantiate the design under test
  complex_arithmetic_processor complex_processor_instance();

  initial begin
    // Configure wave dumping for analysis
    $dumpfile("complex_arithmetic_testbench.vcd");
    $dumpvars(0, complex_arithmetic_testbench);

    #1;  // Allow design to execute

    $display("Testbench: Complex arithmetic operations completed");
    $display("Testbench: Check VCD file for signal analysis");
    $display();

    $finish;  // End simulation
  end

endmodule
```

```
Verilator Simulation Output:
================================================================================

Complex Number Arithmetic Operations Demo
==========================================
First Number:  3 + 4i
Second Number: 1 + 2i

Addition Result: 4 + 6i
Multiplication Result: -5 + 10i
Magnitude Squared of First Number: 25

Testbench: Complex arithmetic operations completed
Testbench: Check VCD file for signal analysis

================================================================================
Process finished with return code: 0
Removing Chapter_7_examples/example_7__complex_number_operations/obj_dir directory...
Chapter_7_examples/example_7__complex_number_operations/obj_dir removed successfully.

0
```

Statistical Analysis Functions

Functions returning multiple statistical measures (mean, variance, etc.)

```
// statistical_analyzer.sv
module statistical_data_processor ();        // Statistical analysis module

  // Function returning multiple statistical measures
  function automatic void calculate_statistics(
    input  real data_samples[5],              // Input data array
    output real mean_value,                   // Mean output
    output real variance_value,               // Variance output
    output real std_deviation,                // Standard deviation output
    output real min_value,                    // Minimum value output
    output real max_value                     // Maximum value output
  );
    real sum_of_values;
    real sum_of_squares;
    int  sample_count;

    // Initialize values
    sum_of_values = 0.0;
    sum_of_squares = 0.0;
    sample_count = 5;
    min_value = data_samples[0];
    max_value = data_samples[0];

    // Calculate sum and find min/max
    foreach (data_samples[i]) begin
      sum_of_values += data_samples[i];
      sum_of_squares += (data_samples[i] * data_samples[i]);
      if (data_samples[i] < min_value) min_value = data_samples[i];
      if (data_samples[i] > max_value) max_value = data_samples[i];
    end

    // Calculate mean
    mean_value = sum_of_values / sample_count;

    // Calculate variance
    variance_value = (sum_of_squares / sample_count) -
                     (mean_value * mean_value);

    // Calculate standard deviation
    std_deviation = $sqrt(variance_value);

  endfunction

  // Function to analyze data quality
  function automatic void quality_metrics(
    input  real mean_val,
    input  real std_dev,
    output string quality_rating,
    output bit   is_consistent_data
  );
    real coefficient_of_variation;

    coefficient_of_variation = (std_dev / mean_val) * 100.0;

    if (coefficient_of_variation < 10.0) begin
```

```
      quality_rating = "EXCELLENT";
      is_consistent_data = 1'b1;
    end else if (coefficient_of_variation < 25.0) begin
      quality_rating = "GOOD";
      is_consistent_data = 1'b1;
    end else begin
      quality_rating = "POOR";
      is_consistent_data = 1'b0;
    end

  endfunction

  initial begin
    $display();
    $display("Statistical Data Processor Ready");
  end

endmodule
```

```
// statistical_analyzer_testbench.sv
module statistical_analyzer_testbench;       // Statistical analyzer testbench

  // Instantiate the statistical data processor
  statistical_data_processor STATS_PROCESSOR_INSTANCE();

  // Test data and result variables
  real sample_data_set[5];                    // Input data samples
  real calculated_mean;                       // Mean result
  real calculated_variance;                   // Variance result
  real calculated_std_dev;                    // Standard deviation result
  real minimum_sample;                        // Minimum value result
  real maximum_sample;                        // Maximum value result
  string data_quality;                        // Quality assessment
  bit data_consistency_flag;                  // Consistency indicator

  initial begin
    // Configure wave dumping
    $dumpfile("statistical_analyzer_testbench.vcd");
    $dumpvars(0, statistical_analyzer_testbench);

    $display();
    $display("=== Statistical Analysis Functions Test ===");
    $display();

    // Test Case 1: Consistent data set
    sample_data_set = '{10.5, 10.2, 10.8, 10.1, 10.4};

    $display("Test Case 1: Analyzing consistent data samples");
    $display("Input samples: %0.1f, %0.1f, %0.1f, %0.1f, %0.1f",
             sample_data_set[0], sample_data_set[1], sample_data_set[2],
             sample_data_set[3], sample_data_set[4]);

    // Call statistical analysis function
    STATS_PROCESSOR_INSTANCE.calculate_statistics(
```

```
    sample_data_set,
    calculated_mean,
    calculated_variance,
    calculated_std_dev,
    minimum_sample,
    maximum_sample
  );

  // Display statistical results
  $display("  Mean value:        %0.3f", calculated_mean);
  $display("  Variance:          %0.6f", calculated_variance);
  $display("  Standard deviation: %0.6f", calculated_std_dev);
  $display("  Minimum sample:    %0.1f", minimum_sample);
  $display("  Maximum sample:    %0.1f", maximum_sample);

  // Assess data quality
  STATS_PROCESSOR_INSTANCE.quality_metrics(
    calculated_mean,
    calculated_std_dev,
    data_quality,
    data_consistency_flag
  );

  $display("  Quality rating:    %s", data_quality);
  $display("  Data consistent:   %s",
           data_consistency_flag ? "YES" : "NO");
  $display();

  #5;  // Wait between test cases

  // Test Case 2: Variable data set
  sample_data_set = '{5.0, 15.0, 8.0, 20.0, 12.0};

  $display("Test Case 2: Analyzing variable data samples");
  $display("Input samples: %0.1f, %0.1f, %0.1f, %0.1f, %0.1f",
           sample_data_set[0], sample_data_set[1], sample_data_set[2],
           sample_data_set[3], sample_data_set[4]);

  // Rerun analysis with new data
  STATS_PROCESSOR_INSTANCE.calculate_statistics(
    sample_data_set,
    calculated_mean,
    calculated_variance,
    calculated_std_dev,
    minimum_sample,
    maximum_sample
  );

  $display("  Mean value:        %0.3f", calculated_mean);
  $display("  Variance:          %0.6f", calculated_variance);
  $display("  Standard deviation: %0.6f", calculated_std_dev);
  $display("  Minimum sample:    %0.1f", minimum_sample);
  $display("  Maximum sample:    %0.1f", maximum_sample);

  STATS_PROCESSOR_INSTANCE.quality_metrics(
```

```
      calculated_mean,
      calculated_std_dev,
      data_quality,
      data_consistency_flag
    );

    $display("  Quality rating:    %s", data_quality);
    $display("  Data consistent:   %s",
             data_consistency_flag ? "YES" : "NO");
    $display();

    $display("Statistical analysis testing completed!");
    $display();

  end

endmodule
```

```
Verilator Simulation Output:
================================================================================

Statistical Data Processor Ready

=== Statistical Analysis Functions Test ===

Test Case 1: Analyzing consistent data samples
Input samples: 10.5, 10.2, 10.8, 10.1, 10.4
  Mean value:        10.400
  Variance:          0.060000
  Standard deviation: 0.244949
  Minimum sample:    10.1
  Maximum sample:    10.8
  Quality rating:    EXCELLENT
  Data consistent:   YES

Test Case 2: Analyzing variable data samples
Input samples: 5.0, 15.0, 8.0, 20.0, 12.0
  Mean value:        12.000
  Variance:          27.600000
  Standard deviation: 5.253570
  Minimum sample:    5.0
  Maximum sample:    20.0
  Quality rating:    POOR
  Data consistent:    NO

Statistical analysis testing completed!
================================================================================
Process finished with return code: 0
Removing Chapter_7_examples/example_8__statistical_analysis_functions/obj_dir directory...
Chapter_7_examples/example_8__statistical_analysis_functions/obj_dir removed successfully.

0
```

Vector Operations

Functions for vector mathematics returning arrays or packed structures

```
// vector_math_operations.sv
module vector_math_calculator();

  // Define packed structure for 3D vector
  typedef struct packed {
    logic signed [15:0] x_component;
    logic signed [15:0] y_component;
    logic signed [15:0] z_component;
  } vector_3d_packed_t;

  // Define array type for vector operations
  typedef logic signed [15:0] vector_array_t [2:0];

  // Function: Add two 3D vectors returning packed structure
  function vector_3d_packed_t add_vectors_packed(
    input vector_3d_packed_t first_vector,
    input vector_3d_packed_t second_vector
  );
    vector_3d_packed_t result_vector;
    result_vector.x_component = first_vector.x_component +
                                second_vector.x_component;
    result_vector.y_component = first_vector.y_component +
                                second_vector.y_component;
    result_vector.z_component = first_vector.z_component +
                                second_vector.z_component;
    return result_vector;
  endfunction

  // Function: Multiply vector by scalar returning array
  function vector_array_t scale_vector_array(
    input vector_array_t input_vector,
    input logic signed [7:0] scale_factor
  );
    vector_array_t scaled_result;
    scaled_result[0] = input_vector[0] * scale_factor;
    scaled_result[1] = input_vector[1] * scale_factor;
    scaled_result[2] = input_vector[2] * scale_factor;
    return scaled_result;
  endfunction

  // Function: Calculate dot product returning single value
  function logic signed [31:0] compute_dot_product(
    input vector_array_t vector_a,
    input vector_array_t vector_b
  );
    logic signed [31:0] dot_product_result;
    dot_product_result = (vector_a[0] * vector_b[0]) +
                         (vector_a[1] * vector_b[1]) +
                         (vector_a[2] * vector_b[2]);
    return dot_product_result;
  endfunction

  // Function: Convert packed to array format
  function vector_array_t packed_to_array_converter(
    input vector_3d_packed_t packed_vector
```

```
  );
    vector_array_t array_result;
    array_result[0] = packed_vector.x_component;
    array_result[1] = packed_vector.y_component;
    array_result[2] = packed_vector.z_component;
    return array_result;
  endfunction

  initial begin
    $display("Vector Operations Functions Module Loaded");
    $display("Supports: Addition, Scaling, Dot Product, Conversion");
  end

endmodule
```

```
// vector_math_operations_testbench.sv
module vector_math_testbench;

  // Instantiate design under test
  vector_math_calculator VECTOR_CALC_INSTANCE();

  // Import types from design module
  typedef struct packed {
    logic signed [15:0] x_component;
    logic signed [15:0] y_component;
    logic signed [15:0] z_component;
  } vector_3d_packed_t;

  typedef logic signed [15:0] vector_array_t [2:0];

  // Test variables
  vector_3d_packed_t first_test_vector, second_test_vector;
  vector_3d_packed_t addition_result;
  vector_array_t array_vector_a, array_vector_b;
  vector_array_t scaling_result;
  logic signed [31:0] dot_product_value;
  logic signed [7:0] multiplication_factor;

  initial begin
    // Setup VCD dumping
    $dumpfile("vector_math_testbench.vcd");
    $dumpvars(0, vector_math_testbench);

    $display("=== Vector Mathematics Operations Testbench ===");
    $display();

    // Test 1: Vector addition with packed structures
    $display("Test 1: Adding packed vectors");
    first_test_vector.x_component = 16'd10;
    first_test_vector.y_component = 16'd20;
    first_test_vector.z_component = 16'd30;

    second_test_vector.x_component = 16'd5;
    second_test_vector.y_component = 16'd15;
```

```
    second_test_vector.z_component = 16'd25;

    addition_result = VECTOR_CALC_INSTANCE.add_vectors_packed(
      first_test_vector, second_test_vector);

    $display("Vector A: (%0d, %0d, %0d)",
             first_test_vector.x_component,
             first_test_vector.y_component,
             first_test_vector.z_component);
    $display("Vector B: (%0d, %0d, %0d)",
             second_test_vector.x_component,
             second_test_vector.y_component,
             second_test_vector.z_component);
    $display("Sum Result: (%0d, %0d, %0d)",
             addition_result.x_component,
             addition_result.y_component,
             addition_result.z_component);
    $display();

    // Test 2: Vector scaling with arrays
    $display("Test 2: Scaling vector using arrays");
    array_vector_a[0] = 16'd4;
    array_vector_a[1] = 16'd6;
    array_vector_a[2] = 16'd8;
    multiplication_factor = 8'd3;

    scaling_result = VECTOR_CALC_INSTANCE.scale_vector_array(
      array_vector_a, multiplication_factor);

    $display("Original Vector: [%0d, %0d, %0d]",
             array_vector_a[0], array_vector_a[1], array_vector_a[2]);
    $display("Scale Factor: %0d", multiplication_factor);
    $display("Scaled Result: [%0d, %0d, %0d]",
             scaling_result[0], scaling_result[1], scaling_result[2]);
    $display();

    // Test 3: Dot product calculation
    $display("Test 3: Computing dot product");
    array_vector_b[0] = 16'd2;
    array_vector_b[1] = 16'd3;
    array_vector_b[2] = 16'd4;

    dot_product_value = VECTOR_CALC_INSTANCE.compute_dot_product(
      array_vector_a, array_vector_b);

    $display("Vector A: [%0d, %0d, %0d]",
             array_vector_a[0], array_vector_a[1], array_vector_a[2]);
    $display("Vector B: [%0d, %0d, %0d]",
             array_vector_b[0], array_vector_b[1], array_vector_b[2]);
    $display("Dot Product: %0d", dot_product_value);
    $display();

    // Test 4: Format conversion
    $display("Test 4: Converting packed to array format");
    array_vector_a = VECTOR_CALC_INSTANCE.packed_to_array_converter(
```

```
      first_test_vector);

    $display("Packed Vector: (%0d, %0d, %0d)",
             first_test_vector.x_component,
             first_test_vector.y_component,
             first_test_vector.z_component);
    $display("Converted Array: [%0d, %0d, %0d]",
             array_vector_a[0], array_vector_a[1], array_vector_a[2]);

    $display();
    $display("=== All Vector Operations Tests Completed ===");

    #10 $finish;
  end

endmodule
```

```
Verilator Simulation Output:
================================================================================
Vector Operations Functions Module Loaded
Supports: Addition, Scaling, Dot Product, Conversion
=== Vector Mathematics Operations Testbench ===

Test 1: Adding packed vectors
Vector A: (10, 20, 30)
Vector B: (5, 15, 25)
Sum Result: (15, 35, 55)

Test 2: Scaling vector using arrays
Original Vector: [4, 6, 8]
Scale Factor: 3
Scaled Result: [12, 18, 24]

Test 3: Computing dot product
Vector A: [4, 6, 8]
Vector B: [2, 3, 4]
Dot Product: 58

Test 4: Converting packed to array format
Packed Vector: (10, 20, 30)
Converted Array: [10, 20, 30]

=== All Vector Operations Tests Completed ===
================================================================================
Process finished with return code: 0
Removing Chapter_7_examples/example_9__vector_operations/obj_dir directory...
Chapter_7_examples/example_9__vector_operations/obj_dir removed successfully.
0
```

Packet Parser Function

Function parsing network packets and returning structured information

```
// packet_parser_module.sv
module packet_parser_module ();

  // Packet header structure
  typedef struct packed {
    logic [7:0]  protocol_type;
    logic [15:0] source_port;
    logic [15:0] destination_port;
    logic [15:0] packet_length;
    logic [7:0]  checksum;
  } packet_header_t;

  // Function to parse network packet and extract header information
  function packet_header_t parse_network_packet(input [63:0] raw_packet_data);
    packet_header_t parsed_header;

    // Extract fields from raw packet data
    parsed_header.protocol_type    = raw_packet_data[63:56];
    parsed_header.source_port      = raw_packet_data[55:40];
    parsed_header.destination_port = raw_packet_data[39:24];
    parsed_header.packet_length    = raw_packet_data[23:8];
    parsed_header.checksum         = raw_packet_data[7:0];

    return parsed_header;
  endfunction

  initial begin
    logic [63:0] incoming_packet;
    packet_header_t extracted_header;

    $display("=== Network Packet Parser Function Demo ===");
    $display();

    // Simulate incoming network packet
    incoming_packet = 64'h06_1F90_0050_0400_A5;

    // Parse the packet using the function
    extracted_header = parse_network_packet(incoming_packet);

    // Display parsed results
    $display("Raw packet data:     0x%016h", incoming_packet);
    $display("Protocol type:       0x%02h", extracted_header.protocol_type);
    $display("Source port:         %0d", extracted_header.source_port);
    $display("Destination port:    %0d", extracted_header.destination_port);
    $display("Packet length:       %0d bytes", extracted_header.packet_length);
    $display("Checksum:            0x%02h", extracted_header.checksum);
    $display();
  end

endmodule
```

```
// packet_parser_module_testbench.sv
module packet_parser_testbench;
```

```
    // Instantiate the packet parser module
    packet_parser_module PACKET_PARSER_INSTANCE();

    // Additional testbench variables
    logic [63:0] test_packets [];

    initial begin
      // Setup VCD file for waveform viewing
      $dumpfile("packet_parser_testbench.vcd");
      $dumpvars(0, packet_parser_testbench);

      $display("=== Packet Parser Function Testbench ===");
      $display();

      // Initialize test packet array
      test_packets = new[3];
      test_packets[0] = 64'h11_1F40_0050_0200_3C;  // UDP packet
      test_packets[1] = 64'h06_0050_1F90_0800_7F;  // TCP packet
      test_packets[2] = 64'h01_0000_0000_001C_FF;  // ICMP packet

      // Test multiple packets
      for (int packet_index = 0; packet_index < 3; packet_index++) begin
        $display("--- Testing Packet %0d ---", packet_index + 1);
        test_packet_parsing(test_packets[packet_index]);
        $display();
        #10;  // Wait between tests
      end

      $display("Packet parser function testing completed!");
      $display();
      $finish;
    end

    // Task to test packet parsing with different inputs
    task test_packet_parsing(input [63:0] test_packet_data);
      // Call the parser function from the design module
      $display("Testing packet: 0x%016h", test_packet_data);

      // Note: In a real testbench, we would instantiate the design
      // and call its functions. For this simple example, we demonstrate
      // the concept of testing the packet parsing functionality.
      $display("Packet successfully processed by parser function");
    endtask

  endmodule
```

```
Verilator Simulation Output:
================================================================================
=== Network Packet Parser Function Demo ===

Raw packet data:     0x061f9000500400a5
Protocol type:       0x06
Source port:         8080
Destination port:    80
Packet length:       1024 bytes
Checksum:            0xa5
```

```
=== Packet Parser Function Testbench ===

--- Testing Packet 1 ---
Testing packet: 0x111f40005002003c
Packet successfully processed by parser function

--- Testing Packet 2 ---
Testing packet: 0x0600501f9008007f
Packet successfully processed by parser function

--- Testing Packet 3 ---
Testing packet: 0x0100000000001cff
Packet successfully processed by parser function

Packet parser function testing completed!

================================================================================
Process finished with return code: 0
Removing Chapter_7_examples/example_10__packet_parser_function/obj_dir directory...
Chapter_7_examples/example_10__packet_parser_function/obj_dir removed successfully.

0
```

Color Space Converter

Function converting between RGB, HSV, and other color representations

```
// color_space_converter.sv
module rgb_to_hsv_converter (
    input  logic        clk,
    input  logic        reset_n,
    input  logic        convert_enable,
    input  logic [7:0]  red_channel,
    input  logic [7:0]  green_channel,
    input  logic [7:0]  blue_channel,
    output logic [8:0]  hue_output,        // 0-359 degrees
    output logic [7:0]  saturation_output, // 0-255 (0-100%)
    output logic [7:0]  value_output,      // 0-255 (0-100%)
    output logic        conversion_ready
);

    // Internal registers for RGB values
    logic [7:0] rgb_max, rgb_min, rgb_delta;
    logic [7:0] red_reg, green_reg, blue_reg;

    // State machine for conversion process
    typedef enum logic [1:0] {
        IDLE = 2'b00,
        CALCULATE = 2'b01,
        READY = 2'b10
    } converter_state_t;

    converter_state_t current_state, next_state;

    // Find maximum and minimum RGB values
```

```
    always_comb begin
        rgb_max = (red_reg >= green_reg) ?
                  ((red_reg >= blue_reg) ? red_reg : blue_reg) :
                  ((green_reg >= blue_reg) ? green_reg : blue_reg);

        rgb_min = (red_reg <= green_reg) ?
                  ((red_reg <= blue_reg) ? red_reg : blue_reg) :
                  ((green_reg <= blue_reg) ? green_reg : blue_reg);

        rgb_delta = rgb_max - rgb_min;
    end

    // State machine sequential logic
    always_ff @(posedge clk or negedge reset_n) begin
        if (!reset_n) begin
            current_state <= IDLE;
            red_reg <= 8'h0;
            green_reg <= 8'h0;
            blue_reg <= 8'h0;
        end else begin
            current_state <= next_state;
            if (convert_enable && current_state == IDLE) begin
                red_reg <= red_channel;
                green_reg <= green_channel;
                blue_reg <= blue_channel;
            end
        end
    end

    // State machine combinational logic
    always_comb begin
        next_state = current_state;
        case (current_state)
            IDLE: begin
                if (convert_enable)
                    next_state = CALCULATE;
            end
            CALCULATE: begin
                next_state = READY;
            end
            READY: begin
                if (!convert_enable)
                    next_state = IDLE;
            end
            default: begin
                next_state = IDLE;
            end
        endcase
    end

    // HSV calculation (simplified for demonstration)
    always_ff @(posedge clk or negedge reset_n) begin
        if (!reset_n) begin
            hue_output <= 9'h0;
            saturation_output <= 8'h0;
```

```
            value_output <= 8'h0;
            conversion_ready <= 1'b0;
        end else begin
            case (current_state)
                CALCULATE: begin
                    // Value is simply the maximum RGB component
                    value_output <= rgb_max;

                    // Saturation calculation
                    if (rgb_max == 0)
                        saturation_output <= 8'h0;
                    else
                        saturation_output <= (rgb_delta * 8'd255) / rgb_max;

                    // Simplified hue calculation (demonstration only)
                    if (rgb_delta == 0) begin
                        hue_output <= 9'h0;  // Undefined, set to 0
                    end else if (rgb_max == red_reg) begin
                        hue_output <= 9'd60;  // Red dominant - simplified
                    end else if (rgb_max == green_reg) begin
                        hue_output <= 9'd120; // Green dominant - simplified
                    end else begin
                        hue_output <= 9'd240; // Blue dominant - simplified
                    end

                    conversion_ready <= 1'b1;
                end
                IDLE: begin
                    conversion_ready <= 1'b0;
                end
                READY: begin
                    // Maintain current values
                end
                default: begin
                    hue_output <= 9'h0;
                    saturation_output <= 8'h0;
                    value_output <= 8'h0;
                    conversion_ready <= 1'b0;
                end
            endcase
        end
    end

endmodule
```

```
// color_space_converter_testbench.sv
module color_converter_testbench;

    // Clock and reset signals
    logic        system_clock;
    logic        reset_signal;

    // Input signals to design under test
    logic        conversion_trigger;
```

```
    logic [7:0]  input_red_value;
    logic [7:0]  input_green_value;
    logic [7:0]  input_blue_value;

    // Output signals from design under test
    logic [8:0]  output_hue_degrees;
    logic [7:0]  output_saturation_percent;
    logic [7:0]  output_value_brightness;
    logic        output_conversion_complete;

    // Instantiate the RGB to HSV converter
    rgb_to_hsv_converter COLOR_CONVERTER_INSTANCE (
        .clk(system_clock),
        .reset_n(reset_signal),
        .convert_enable(conversion_trigger),
        .red_channel(input_red_value),
        .green_channel(input_green_value),
        .blue_channel(input_blue_value),
        .hue_output(output_hue_degrees),
        .saturation_output(output_saturation_percent),
        .value_output(output_value_brightness),
        .conversion_ready(output_conversion_complete)
    );

    // Clock generation
    initial begin
        system_clock = 1'b0;
        forever #5 system_clock = ~system_clock; // 100MHz clock
    end

    // Test stimulus
    initial begin
        // Initialize VCD dump
        $dumpfile("color_converter_testbench.vcd");
        $dumpvars(0, color_converter_testbench);

        // Initialize signals
        reset_signal = 1'b0;
        conversion_trigger = 1'b0;
        input_red_value = 8'h0;
        input_green_value = 8'h0;
        input_blue_value = 8'h0;

        // Display test header
        $display();
        $display("=== RGB to HSV Color Space Converter Test ===");
        $display();

        // Reset sequence
        #10 reset_signal = 1'b1;
        #20;

        // Test Case 1: Pure Red (255, 0, 0)
        $display("Test 1: Converting Pure Red RGB(255, 0, 0)");
        input_red_value = 8'd255;
```

```
        input_green_value = 8'd0;
        input_blue_value = 8'd0;
        conversion_trigger = 1'b1;
        #10;

        // Wait for conversion to complete
        wait(output_conversion_complete);
        #10;
        $display("  Result: H=%d, S=%d, V=%d",
                output_hue_degrees, output_saturation_percent,
                output_value_brightness);

        conversion_trigger = 1'b0;
        #20;

        // Test Case 2: Pure Green (0, 255, 0)
        $display("Test 2: Converting Pure Green RGB(0, 255, 0)");
        input_red_value = 8'd0;
        input_green_value = 8'd255;
        input_blue_value = 8'd0;
        conversion_trigger = 1'b1;
        #10;

        wait(output_conversion_complete);
        #10;
        $display("  Result: H=%d, S=%d, V=%d",
                output_hue_degrees, output_saturation_percent,
                output_value_brightness);

        conversion_trigger = 1'b0;
        #20;

        // Test Case 3: Pure Blue (0, 0, 255)
        $display("Test 3: Converting Pure Blue RGB(0, 0, 255)");
        input_red_value = 8'd0;
        input_green_value = 8'd0;
        input_blue_value = 8'd255;
        conversion_trigger = 1'b1;
        #10;

        wait(output_conversion_complete);
        #10;
        $display("  Result: H=%d, S=%d, V=%d",
                output_hue_degrees, output_saturation_percent,
                output_value_brightness);

        conversion_trigger = 1'b0;
        #20;

        // Test Case 4: White (255, 255, 255)
        $display("Test 4: Converting White RGB(255, 255, 255)");
        input_red_value = 8'd255;
        input_green_value = 8'd255;
        input_blue_value = 8'd255;
        conversion_trigger = 1'b1;
```

```
        #10;

        wait(output_conversion_complete);
        #10;
        $display("  Result: H=%d, S=%d, V=%d",
                output_hue_degrees, output_saturation_percent,
                output_value_brightness);

        conversion_trigger = 1'b0;
        #20;

        // Test Case 5: Gray (128, 128, 128)
        $display("Test 5: Converting Gray RGB(128, 128, 128)");
        input_red_value = 8'd128;
        input_green_value = 8'd128;
        input_blue_value = 8'd128;
        conversion_trigger = 1'b1;
        #10;

        wait(output_conversion_complete);
        #10;
        $display("  Result: H=%d, S=%d, V=%d",
                output_hue_degrees, output_saturation_percent,
                output_value_brightness);

        conversion_trigger = 1'b0;
        #20;

        $display();
        $display("=== Color Space Conversion Tests Complete ===");
        $display();

        $finish;
    end

    // Monitor for debugging
    initial begin
        $monitor("Time=%0t: RGB(%d,%d,%d) -> HSV(%d,%d,%d) Ready=%b",
                $time, input_red_value, input_green_value, input_blue_value,
                output_hue_degrees, output_saturation_percent,
                output_value_brightness, output_conversion_complete);
    end

endmodule
```

```
Verilator Simulation Output:
================================================================================

=== RGB to HSV Color Space Converter Test ===

Time=0: RGB(  0,  0,  0) -> HSV(  0,  0,  0) Ready=0
Test 1: Converting Pure Red RGB(255, 0, 0)
Time=30: RGB(255,  0,  0) -> HSV(  0,  0,  0) Ready=0
Time=45: RGB(255,  0,  0) -> HSV( 60,  0,255) Ready=1
  Result: H= 60, S=  0, V=255
Time=65: RGB(255,  0,  0) -> HSV( 60,  0,255) Ready=0
```

```
Test 2: Converting Pure Green RGB(0, 255, 0)
Time=75: RGB(  0,255,  0) -> HSV( 60,  0,255) Ready=0
Time=85: RGB(  0,255,  0) -> HSV(120,  0,255) Ready=1
  Result: H=120, S=  0, V=255
Time=105: RGB(  0,255,  0) -> HSV(120,  0,255) Ready=0
Test 3: Converting Pure Blue RGB(0, 0, 255)
Time=115: RGB(  0,  0,255) -> HSV(120,  0,255) Ready=0
Time=125: RGB(  0,  0,255) -> HSV(240,  0,255) Ready=1
  Result: H=240, S=  0, V=255
Time=145: RGB(  0,  0,255) -> HSV(240,  0,255) Ready=0
Test 4: Converting White RGB(255, 255, 255)
Time=155: RGB(255,255,255) -> HSV(240,  0,255) Ready=0
Time=165: RGB(255,255,255) -> HSV(  0,  0,255) Ready=1
  Result: H=  0, S=  0, V=255
Time=185: RGB(255,255,255) -> HSV(  0,  0,255) Ready=0
Test 5: Converting Gray RGB(128, 128, 128)
Time=195: RGB(128,128,128) -> HSV(  0,  0,255) Ready=0
Time=205: RGB(128,128,128) -> HSV(  0,  0,128) Ready=1
  Result: H=  0, S=  0, V=128
Time=225: RGB(128,128,128) -> HSV(  0,  0,128) Ready=0

=== Color Space Conversion Tests Complete ===

================================================================================
Process finished with return code: 0
Removing Chapter_7_examples/example_11__color_space_converter/obj_dir directory...
Chapter_7_examples/example_11__color_space_converter/obj_dir removed successfully.
0
```

Task Declarations and Calls

Bus Transaction Tasks

Tasks for performing read/write operations on system buses

Protocol Handler Tasks

Tasks implementing communication protocol sequences

Test Pattern Generator Task

Task generating various test patterns for verification

Memory Test Tasks

Tasks for comprehensive memory testing with different patterns

Waveform Generator Task

Task producing different types of waveforms for testing

File I/O Tasks

Tasks for reading from and writing to files during simulation

Automatic vs. Static Lifetime

Recursive Algorithm Examples

Automatic functions implementing recursive algorithms like tree traversal

Call Counter Examples

Static functions maintaining call statistics and usage counters

Stack Implementation

Automatic functions for stack operations with proper variable scoping

Random Number Generator

Static function maintaining seed state between calls

Nested Function Calls

Examples showing variable lifetime in nested function scenarios

Pass by Reference

Array Sorting Tasks

Tasks that sort arrays in-place using pass by reference

Matrix Operations

Functions performing matrix operations on large data structures

Configuration Update Tasks

Tasks modifying system configuration structures by reference

Buffer Management Functions

Functions managing circular buffers and memory pools

State Machine Tasks

Tasks updating complex state machine structures

Return Statements in Functions

Input Validation Functions

Functions with multiple return points for different validation cases

Search Functions

Functions with early returns for efficient searching algorithms

Error Handling Functions

Functions demonstrating proper error detection and return patterns

Conditional Processing Functions

Functions with complex conditional logic and multiple exit points

Void Functions

Debug and Logging Functions

Void functions for system debugging and message logging

Hardware Initialization Functions

Void functions for initializing hardware components and registers

Data Structure Maintenance

Void functions for maintaining internal data structure consistency

Performance Monitoring Functions

Void functions for collecting and updating performance metrics

Assertion Functions

Void functions implementing custom assertion and checking logic