SystemVerilog Examples Guide

Data Types

Basic Data Types

bit - 2-state (0, 1) single bit

```
bit reset_n = 1'b1; // Active low reset
bit [7:0] counter = 8'h00; // 8-bit counter, no X/Z states
```

logic - 4-state (0, 1, X, Z) single bit, recommended for most uses

```
logic clk; // Clock signal
logic [31:0] data_bus; // 32-bit data bus
logic valid = 1'bx; // Can represent unknown state
```

reg - Legacy 4-state register type

```
reg [15:0] address; // Legacy address register
reg interrupt_flag; // Single bit register
```

wire - Net type for continuous assignments

```
wire [7:0] sum; // Output of adder
assign sum = a + b; // Continuous assignment
```

byte - 8-bit signed integer

```
byte temperature = -40; // Temperature value (-128 to 127)
byte data_array[256]; // Array of bytes
```

shortint - 16-bit signed integer

```
shortint pixel_coord = -1024; // Screen coordinate (-32768 to 32767)
```

int - 32-bit signed integer

```
int transaction_count = 0; // Counter for transactions
int error_code = -1; // Error status
```

longint - 64-bit signed integer

```
longint timestamp = $time; // 64-bit timestamp
longint memory_size = 1_000_000_000; // Large memory size (1GB)
```

integer - 32-bit 4-state signed integer

```
integer loop_var; // Loop variable that can be X
integer file_handle; // File I/O handle
```

time - 64-bit unsigned time value

```
time start_time, end_time;
start_time = $time; // Capture simulation time
```

real - Double precision floating point

```
real voltage = 3.3; // Analog voltage value
real frequency = 100.5e6; // 100.5 MHz frequency
```

shortreal - Single precision floating point

```
```

```
shortreal power_consumption = 2.5; // Power in watts
```

string - Dynamic string type

```
string test_name = "basic_test";
string message = $sformatf("Error at time %t", $time);
```

Packed Arrays

bit [7:0] - 8-bit packed array

```
bit [7:0] byte_data = 8'hAB; // Single 8-bit value
bit [31:0] instruction = 32'h12345678; // 32-bit instruction
```

logic [31:0] - 32-bit packed vector

```
logic [31:0] cpu_data; // 32-bit CPU data
logic [15:0] memory[1024]; // Memory array
```

Unpacked Arrays

int array[10] - Fixed-size unpacked array

```
int scores[10]; // Array of 10 test scores
scores[0] = 95; // Access first element
```

int queue[$] - Dynamic queue

```
int data_queue[$]; // Dynamic queue
data_queue.push_back(42); // Add element
int value = data_queue.pop_front(); // Remove element
```

int assoc[string] - Associative array

```
int register_map[string]; // Register name to address mapping
register_map["STATUS"] = 32'h100;
register_map["CONTROL"] = 32'h104;
```

User-Defined Types

typedef - Creates user-defined types

```
typedef enum {RED, GREEN, BLUE} color_t;
typedef struct {int x, y;} point_t;
color_t pixel_color = RED;
point_t screen_pos = '{10, 20};
```

enum - Enumerated types

```
typedef enum {IDLE, READ, WRITE, ERROR} state_t;
state_t current_state = IDLE;
state_t next_state;
```

struct - Structure definition

```
typedef struct {
    logic [31:0] address;
    logic [7:0] data;
    logic valid;
} bus_transaction_t;
bus_transaction_t tx = '{32'h1000, 8'hFF, 1'b1};
```

union - Union definition

```
typedef union {
    int signed_val;
    bit [31:0] unsigned_val;
} data_union_t;
data_union_t data;
data.signed_val = -1; // Same bits as unsigned_val = 32'hFFFFFFFF
```

class - Object-oriented class definition

```
class packet;
    rand int length;
    rand bit [7:0] data[];
    constraint length_c { length inside {[1:100]}; }
    function new();
        data = new[length];
    endfunction
endclass
```

Operators

Arithmetic Operators

+ - Addition

```
logic [7:0] sum = a + b; // Add two 8-bit values
int total = count + offset; // Integer addition
```

- - Subtraction

```
logic [7:0] diff = max_val - min_val; // Subtract values
int remaining = total - used; // Calculate remainder
```

****** - Multiplication

```
```

```
logic [15:0] product = a * b; // 8-bit × 8-bit = 16-bit
int area = width * height; // Calculate area
```

/ - Division

```
int average = total / count; // Integer division
real ratio = real'(num) / real'(den); // Real division
```

% - Modulus

```
int remainder = dividend % divisor; // Get remainder
logic [2:0] index = counter % 8; // Wrap around 0-7
```

****** - Exponentiation

```
int power_of_two = 2 ** exponent; // Calculate 2^exponent
real result = base ** 3.5; // Real exponentiation
```

Bitwise Operators

& - Bitwise AND

```
logic [7:0] masked = data & 8'h0F; // Mask lower 4 bits
logic parity = ^(data & mask); // Masked parity
```

| - Bitwise OR

```
logic [7:0] flags = status | error; // Combine status flags
logic interrupt = irq1 | irq2 | irq3; // Any interrupt active
```

^ - Bitwise XOR

```
```

```
logic [7:0] checksum = data1 ^ data2; // XOR checksum
logic toggle = signal ^ 1'b1; // Invert signal
```

~ - Bitwise NOT

```
logic [7:0] inverted = ~data; // Invert all bits
logic not_enable = ~enable; // Active low signal
```

~& - Bitwise NAND

```
logic nand_result = ~(a & b); // NAND operation
logic [7:0] nand_mask = ~(data & mask); // Bitwise NAND
```

~| - Bitwise NOR

```
logic nor_result = ~(a | b); // NOR operation
logic all_zero = ~|data; // True if data is all zeros
```

~^ or ^~ - Bitwise XNOR

```
logic xnor_result = ~(a ^ b); // XNOR (equivalence)
logic [7:0] compare = ~(data1 ^ data2); // Bit comparison
```

Logical Operators

&& - Logical AND

```
logic valid_transaction = valid && ready && !error;
if (clk_enable && !reset) begin
   // Clock is active and not in reset
end
```

|| - Logical OR

```
logic interrupt_pending = timer_irq || uart_irq || gpio_irq;
if (error || timeout || abort) begin
    // Handle error conditions
end
```

! - Logical NOT

```
logic not_ready = !ready;
if (!fifo_empty) begin
    data_out = fifo_read();
end
```

Comparison Operators

== - Equality (4-state)

```
if (state == IDLE) begin
    next_state = READ;
end
logic match = (data == expected_value);
```

!= - Inequality (4-state)

```
if (current_addr != previous_addr) begin
    address_changed = 1'b1;
end
```

=== - Case equality (includes X and Z)

```
if (signal === 1'bx) begin
    $display("Signal is unknown");
end
// Useful in testbenches for exact matching
```

!== - Case inequality (includes X and Z)

```
if (data !== 8'bxxxxxxxx) begin
    $display("Data is not all X");
end
```

< - Less than

```
if (counter < max_count) begin
    counter <= counter + 1;
end
```

<= - Less than or equal

```
if (temperature <= max_temp) begin
    cooling_enable = 1'b0;
end
```

> - Greater than

```
if (fifo_level > threshold) begin
    almost_full = 1'b1;
end
```

>= - Greater than or equal

```
if (voltage >= min_voltage) begin
    power_good = 1'b1;
end
```

Shift Operators

<< - Logical left shift

```
logic [7:0] shifted = data << 2; // Shift left by 2, fill with 0
logic [31:0] address = base_addr << offset;
```

>> - Logical right shift

```
logic [7:0] divided = value >> 3; // Divide by 8 (shift right 3)
logic [15:0] upper = data >> 8; // Get upper byte
```

<<< - Arithmetic left shift

```
int doubled = value <<< 1; // Multiply by 2 (signed)
```

>>> - Arithmetic right shift

```
int halved = signed_value >>> 1; // Divide by 2 (preserve sign)
```

Reduction Operators

& - Reduction AND

```
logic all_ones = &data; // True if all bits are 1
logic valid = &{ready, enable, !error}; // All conditions true
```

| - Reduction OR

```
logic any_bit = |data; // True if any bit is 1
logic active = |interrupt_vector; // Any interrupt active
```

^ - Reduction XOR

```
```

```
logic parity = ^data; // Even parity bit
logic error_detect = ^{data, parity_in}; // Parity check
```

~& - Reduction NAND

```
logic not_all_ones = ~&data; // True if not all bits are 1
```

~| - Reduction NOR

```
logic all_zeros = ~|data; // True if all bits are 0
```

~^ - Reduction XNOR

```
logic odd_parity = ~^data; // Odd parity bit
```

Assignment Operators

= - Blocking assignment

```
always_comb begin
    sum = a + b; // Blocking assignment in combo logic
    carry = (a & b) | (a & cin) | (b & cin);
end
```

<= - Non-blocking assignment

```
```

```
always_ff @(posedge clk) begin
    if (reset) begin
        counter <= 0; // Non-blocking in sequential logic
    end else begin
        counter <= counter + 1;
    end
end
```

*+=, -=, =, /= - Compound assignments

```
always_ff @(posedge clk) begin
    if (increment)
        score += points; // score = score + points
    if (decrement)
        lives -= 1; // lives = lives - 1
end
```

Concatenation and Replication

{a, b} - Concatenation

```
logic [15:0] word = {high_byte, low_byte}; // Combine bytes
logic [31:0] instruction = {opcode, rs, rt, immediate};
```

{n{pattern}} - Replication

```
logic [7:0] all_ones = {8{1'b1}}; // 8'b11111111
logic [31:0] sign_extend = {{16{data[15]}}, data}; // Sign extension
```

Control Flow Constructs

Conditional Statements

if-else - Conditional execution

```
always_comb begin
    if (select == 2'b00)
        mux_out = in0;
    else if (select == 2'b01)
        mux_out = in1;
    else if (select == 2'b10)
        mux_out = in2;
    else
        mux_out = in3;
end
```

case - Multi-way branch

```
always_comb begin
    case (opcode)
        4'b0000: result = a + b; // ADD
        4'b0001: result = a - b; // SUB
        4'b0010: result = a & b; // AND
        4'b0011: result = a | b; // OR
        default: result = 32'h0;
    endcase
end
```

casex - Case with don't care (X)

```
always_comb begin
    casex (instruction[31:26])
        6'b000xxx: alu_op = R_TYPE; // R-type instructions
        6'b001xxx: alu_op = I_TYPE; // I-type instructions
        default: alu_op = UNKNOWN;
    endcase
end
```

casez - Case with high impedance (Z)

```
always_comb       begin
   casez (priority_vector)
      4'b1zzz: highest_priority = 3;
      4'b01zz: highest_priority = 2;
      4'b001z: highest_priority = 1;
      4'b0001: highest_priority = 0;
      default: highest_priority = -1;
   endcase
end
```

unique case - Case with synthesis optimization hints

```
always_comb begin
   unique case (state) // All cases covered, no overlap
      IDLE: next_state = READ;
      READ: next_state = WRITE;
      WRITE: next_state = IDLE;
   endcase
end
```

priority case - Case with priority encoding

```
always_comb begin
   priority case (1'b1) // Priority encoder
      error: status = ERROR_STATE;
      warning: status = WARNING_STATE;
      normal: status = NORMAL_STATE;
      default: status = UNKNOWN_STATE;
   endcase
end
```

Loops

for - Traditional for loop

```
```

```
always_comb begin
    parity = 1'b0;
    for (int i = 0; i < 8; i++) begin
        parity = parity ^ data[i]; // Calculate parity
    end
end
```

while - While loop

```
task find_first_one;
    input [31:0] data;
    output [4:0] position;
    begin
        position = 0;
        while (position < 32 && data[position] == 1'b0) begin
            position = position + 1;
        end
    end
endtask
```

do-while - Do-while loop

```
task wait_for_ready;
    begin
        do begin
            @(posedge clk);
        end while (!ready);
    end
endtask
```

foreach - Iterate over arrays

```
int memory[256];
initial begin
    foreach (memory[i]) begin
        memory[i] = i * 2; // Initialize array
    end
end
```

repeat - Repeat a fixed number of times

```
always_ff @(posedge clk) begin
    if (shift_enable) begin
        repeat (8) begin
            shift_reg <= {shift_reg[6:0], serial_in};
        end
    end
end
```

forever - Infinite loop

```
initial begin
    forever begin
        @(posedge clk);
        if (monitor_enable) begin
            $display("Time: %t, Signal: %h", $time, signal);
        end
    end
end
```

Loop Control

break - Exit loop

```
for (int i = 0; i < 100; i++) begin
    if (data[i] == target) begin
        found_index = i;
        break; // Exit loop when found
    end
end
```

continue - Skip to next iteration

```
for (int i = 0; i < 10; i++) begin
    if (skip_array[i]) begin
        continue; // Skip this iteration
    end
    process_data(data[i]);
end
```

return - Return from function/task

```
function int find_max(int array[]);
    int max_val = array[0];
    foreach (array[i]) begin
        if (array[i] > max_val) begin
            max_val = array[i];
        end
    end
    return max_val; // Return maximum value
endfunction
```

Module and Interface Constructs

Module Definition

module - Basic design unit

```
module counter #(
    parameter WIDTH = 8
)(
    input logic clk,
    input logic reset,
    input logic enable,
    output logic [WIDTH-1:0] count
);
    // Module implementation
    always_ff @(posedge clk) begin
        if (reset)
            count <= 0;
        else if (enable)
            count <= count + 1;
    end
endmodule
```

endmodule - End module definition

```
module simple_mux(
    input logic [1:0] select,
    input logic [7:0] in0, in1, in2, in3,
    output logic [7:0] out
);
    always_comb begin
        case (select)
            2'b00: out = in0;
            2'b01: out = in1;
            2'b10: out = in2;
            2'b11: out = in3;
        endcase
    end
endmodule // End of module definition
```

parameter - Compile-time constant

```
module fifo #(
    parameter DEPTH = 16,
    parameter WIDTH = 8,
    parameter ADDR_WIDTH = $clog2(DEPTH)
)(
    input logic clk,
    input logic [WIDTH-1:0] data_in,
    output logic [WIDTH-1:0] data_out
);
    logic [WIDTH-1:0] memory[DEPTH];
    // FIFO implementation using parameters
endmodule
```

localparam - Local parameter

```
module state_machine(
    input logic clk, reset,
    input logic start,
    output logic done
);
    localparam IDLE = 2'b00; // Local to this module
    localparam ACTIVE = 2'b01;
    localparam FINISH = 2'b10;

    logic [1:0] state, next_state;
    // State machine implementation
endmodule
```

generate - Generate blocks for replication

```
module parallel_adder #(parameter N = 4)(
    input logic [N-1:0] a, b,
    input logic cin,
    output logic [N-1:0] sum,
    output logic cout
);
    logic [N:0] carry;
    assign carry[0] = cin;

    generate
        for (genvar i = 0; i < N; i++) begin : adder_stage
            full_adder fa(
                .a(a[i]),
                .b(b[i]),
                .cin(carry[i]),
                .sum(sum[i]),
                .cout(carry[i+1])
            );
        end
    endgenerate

    assign cout = carry[N];
endmodule
```

Port Declarations

input - Input port

```
module decoder(
    input logic [2:0] address, // 3-bit input address
    input logic enable, // Enable input
    output logic [7:0] decode_out // 8-bit output
);
    // Decoder implementation
endmodule
```

output - Output port

```
module encoder(
    input logic [7:0] data_in,
    output logic [2:0] encoded, // 3-bit encoded output
    output logic valid // Valid output signal
);
    // Priority encoder implementation
endmodule
```

inout - Bidirectional port

```
module memory_controller(
    input logic clk,
    input logic [15:0] address,
    inout logic [7:0] data_bus, // Bidirectional data bus
    input logic read_enable,
    input logic write_enable
);
    // Tristate logic for bidirectional bus
    assign data_bus = write_enable ? write_data : 8'bz;
endmodule
```

ref - Reference port (for interfaces)

```
interface cpu_bus;
    logic [31:0] address, data;
    logic read, write, ready;
endinterface

module cpu(
    input logic clk,
    ref cpu_bus bus // Reference to interface
);
    // CPU can modify interface signals directly
    always_ff @(posedge clk) begin
        bus.address <= next_address;
        bus.read <= read_request;
    end
endmodule
```

Interface Constructs

interface - Interface definition

```
interface axi_if #(parameter DATA_WIDTH = 32);
    logic [DATA_WIDTH-1:0] data;
    logic valid;
    logic ready;
    logic [1:0] resp;

    // Interface methods
    task write_transaction(input [DATA_WIDTH-1:0] write_data);
        data = write_data;
        valid = 1'b1;
        @(posedge ready);
        valid = 1'b0;
    endtask
endinterface
```

endinterface - End interface definition

```
interface simple_bus;
    logic [7:0] addr;
    logic [31:0] data;
    logic valid;

    modport master (
        output addr, data, valid
    );

    modport slave (
        input addr, data, valid
    );
endinterface // End of interface definition
```

modport - Module port within interface

```
interface memory_if;
    logic [15:0] address;
    logic [7:0] data;
    logic read, write, ready;

    modport cpu ( // CPU's view of interface
        output address, data, read, write,
        input ready
    );

    modport memory ( // Memory's view of interface
        input address, data, read, write,
        output ready
    );
endinterface

module cpu_core(
    input logic clk,
    memory_if.cpu mem_bus // Use CPU modport
);
    // CPU implementation
endmodule
```

clocking - Clocking block definition

```
interface test_if(input logic clk);
    logic [7:0] data;
    logic valid, ready;

    clocking cb @(posedge clk); // Clocking block
        default input #1step output #1ns;
        input ready;
        output data, valid;
    endclocking

    modport testbench (clocking cb);
endinterface
```

Procedural Blocks

Always Blocks

always - General always block

```
always @(posedge clk or negedge reset_n) begin
    if (!reset_n) begin
        counter <= 8'b0;
    end else begin
        counter <= counter + 1;
    end
end
```

always_ff - Flip-flop (sequential) logic

```
always_ff @(posedge clk) begin
    if (reset) begin
        state <= IDLE;
        data_reg <= 32'h0;
    end else begin
        state <= next_state;
        if (load_enable) begin
            data_reg <= input_data;
        end
    end
end
```

always_comb - Combinational logic

```
always_comb begin
    case (opcode)
        3'b000: alu_result = operand_a + operand_b;
        3'b001: alu_result = operand_a - operand_b;
        3'b010: alu_result = operand_a & operand_b;
        3'b011: alu_result = operand_a | operand_b;
        default: alu_result = 32'h0;
    endcase
    zero_flag = (alu_result == 32'h0);
    negative_flag = alu_result[31];
end
```

always_latch - Latch inference

```
always_latch begin
    if (enable) begin
        latched_data = input_data; // Creates a latch
    end
    // No else clause - maintains previous value when !enable
end
```

Initial and Final

initial - Execute once at time zero

```
initial begin
    // Initialize signals at start of simulation
    clk = 1'b0;
    reset = 1'b1;
    data = 8'h00;

    // Release reset after 100ns
    #100 reset = 1'b0;

    // Generate clock
    forever #5 clk = ~clk; // 10ns period clock
end
```

final - Execute at end of simulation

```
int transaction_count = 0;

always @(posedge clk) begin
    if (valid_transaction) begin
        transaction_count++;
    end
end

final begin
    $display("Simulation completed");
    $display("Total transactions: %d", transaction_count);
    $display("Final time: %t", $time);
end
```

Functions and Tasks

Function Definition

function - Pure function definition

```
function int calculate_parity(input [7:0] data);
    int parity = 0;
    for (int i = 0; i < 8; i++) begin
        parity = parity ^ data[i];
    end
    return parity;
endfunction

// Usage
logic parity_bit = calculate_parity(data_byte);
```

endfunction - End function definition

```
function [31:0] gray_to_binary(input [31:0] gray_code);
    gray_to_binary[31] = gray_code[31];
    for (int i = 30; i >= 0; i--) begin
        gray_to_binary[i] = gray_to_binary[i+1] ^ gray_code[i];
    end
endfunction // End of function definition
```

automatic - Automatic (recursive) function

```
function automatic int factorial(input int n);
    if (n <= 1) begin
        return 1;
    end else begin
        return n * factorial(n - 1); // Recursive call
    end
endfunction

// Usage
int result = factorial(5); // Returns 120
```

static - Static function

```
class math_utils;
    static function int gcd(int a, int b);
        while (b != 0) begin
            int temp = b;
            b = a % b;
            a = temp;
        end
        return a;
    endfunction
endclass

// Usage - no object instance needed
int result = math_utils::gcd(48, 18); // Returns 6
```

Task Definition

task - Task definition (can have timing)

```
task send_packet(
    input [7:0] data[],
    input int length
);
    for (int i = 0; i < length; i++) begin
        @(posedge clk);
        tx_data <= data[i];
        tx_valid <= 1'b1;
        @(posedge clk);
        tx_valid <= 1'b0;
    end
endtask

// Usage
logic [7:0] packet_data[10] = '{8'h01, 8'h02, 8'h03, /*...*/};
send_packet(packet_data, 10);
```

endtask - End task definition

```
task wait_for_interrupt(input int timeout_cycles);
    int cycle_count = 0;
    while (!interrupt && cycle_count < timeout_cycles) begin
        @(posedge clk);
        cycle_count++;
    end
    if (cycle_count >= timeout_cycles) begin
        $display("Timeout waiting for interrupt");
    end
endtask // End of task definition
```

Subroutine Arguments

input - Input argument

```
```

```
task check_memory(
    input [15:0] start_addr, // Input arguments
    input [15:0] end_addr,
    input [7:0] expected_pattern
);
    for (int addr = start_addr; addr <= end_addr; addr++) begin
        if (memory[addr] !== expected_pattern) begin
            $display("Memory error at address %h", addr);
        end
    end
endtask
```

output - Output argument

```
task divide_and_remainder(
    input int dividend,
    input int divisor,
    output int quotient, // Output arguments
    output int remainder
);
    quotient = dividend / divisor;
    remainder = dividend % divisor;
endtask

// Usage
int q, r;
divide_and_remainder(17, 5, q, r); // q=3, r=2
```

inout - Bidirectional argument

```
task byte_swap(
    inout [31:0] data // Bidirectional argument
);
    logic [7:0] temp;
    temp = data[31:24];
    data[31:24] = data[7:0];
    data[7:0] = temp;
    temp = data[23:16];
    data[23:16] = data[15:8];
    data[15:8] = temp;
endtask

// Usage
logic [31:0] word = 32'h12345678;
byte_swap(word); // word becomes 32'h78563412
```

ref - Reference argument

```
task sort_array(ref int array[]); // Reference argument
    int n = array.size();
    for (int i = 0; i < n-1; i++) begin
        for (int j = 0; j < n-i-1; j++) begin
            if (array[j] > array[j+1]) begin
                int temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
            end
        end
    end
endtask

// Usage - array is modified directly
int numbers[] = {64, 34, 25, 12, 22, 11, 90};
sort_array(numbers);
```

Object-Oriented Programming

Class Definition

class - Class definition

```
class transaction;
    rand bit [31:0] address;
    rand bit [7:0] data;
    rand bit [1:0] transaction_type;

    constraint addr_range { address inside {[32'h1000:32'h1FFF]}; }
    constraint valid_type { transaction_type inside {0, 1, 2}; }

    function void print();
        $display("Address: %h, Data: %h, Type: %d",
                 address, data, transaction_type);
    endfunction
endclass
```

endclass - End class definition

```
class fifo #(parameter WIDTH = 8, DEPTH = 16);
    local bit [WIDTH-1:0] memory[DEPTH];
    local int write_ptr, read_ptr;
    local int count;

    function new();
        write_ptr = 0;
        read_ptr = 0;
        count = 0;
    endfunction

    function bit push(bit [WIDTH-1:0] data);
        if (count < DEPTH) begin
            memory[write_ptr] = data;
            write_ptr = (write_ptr + 1) % DEPTH;
            count++;
            return 1;
        end
        return 0; // FIFO full
    endfunction
endclass // End of class definition
```

extends - Class inheritance

```
class base_packet;
    rand bit [7:0] header;
    rand bit [15:0] length;

    virtual function void print();
        $display("Base packet: header=%h, length=%d", header, length);
    endfunction
endclass

class ethernet_packet extends base_packet;
    rand bit [47:0] dest_mac;
    rand bit [47:0] src_mac;

    function void print();
        super.print(); // Call parent method
        $display("Ethernet: dest=%h, src=%h", dest_mac, src_mac);
    endfunction
endclass
```

virtual - Virtual class/method

```
virtual class base_driver; // Cannot be instantiated
    pure virtual task drive_transaction();
    pure virtual function bit is_ready();

    // Common functionality
    task wait_cycles(int n);
        repeat(n) @(posedge clk);
    endtask
endclass

class uart_driver extends base_driver;
    task drive_transaction(); // Must implement pure virtual
        // UART-specific driving logic
    endtask

    function bit is_ready();
        return uart_ready;
    endfunction
endclass
```

pure virtual - Pure virtual method

```
virtual class protocol_checker;
    pure virtual function bit check_packet(packet p);
    pure virtual function void report_error(string msg);

    // Template method using pure virtual functions
    function bit validate(packet p);
        if (!check_packet(p)) begin
            report_error("Packet validation failed");
            return 0;
        end
        return 1;
    endfunction
endclass
```

Class Members

static - Static class member

```
class counter;
    static int global_count = 0; // Shared across all instances
    int instance_id;

    function new();
        global_count++;
        instance_id = global_count;
        $display("Created instance %d", instance_id);
    endfunction

    static function int get_total_count();
        return global_count; // Access without instance
    endfunction
endclass

// Usage
counter c1 = new(); // "Created instance 1"
counter c2 = new(); // "Created instance 2"
int total = counter::get_total_count(); // Returns 2
```

local - Local (private) member

```
class secure_data;
    local bit [31:0] secret_key; // Private member
    bit [7:0] public_data;

    function new(bit [31:0] key);
        secret_key = key; // Only accessible within class
    endfunction

    function bit [7:0] encrypt(bit [7:0] data);
        return data ^ secret_key[7:0]; // Use private key
    endfunction
endclass
```

protected - Protected member

```
class base_component;
    protected string name; // Accessible to derived classes
    local int id; // Private to this class

    function new(string comp_name);
        name = comp_name;
        id = $urandom();
    endfunction
endclass

class derived_component extends base_component;
    function void show_info();
        $display("Component name: %s", name); // OK - protected member
        // $display("ID: %d", id); // Error - private member
    endfunction
endclass
```

const - Constant member

```
class configuration;
    const int MAX_TRANSACTIONS = 1000; // Constant member
    const string VERSION = "2.1";

    function new();
        // MAX_TRANSACTIONS = 2000; // Error - cannot modify const
    endfunction
endclass
```

rand - Random variable

```
class test_data;
    rand bit [7:0] data; // Random variable
    rand bit [3:0] address;
    rand bit read_write;

    constraint data_range { data inside {[10:250]}; }
    constraint addr_align { address[1:0] == 2'b00; } // Word aligned
endclass

// Usage
test_data td = new();
if (td.randomize()) begin
    $display("Generated: addr=%h, data=%h, rw=%b",
             td.address, td.data, td.read_write);
end
```

randc - Random cyclic variable

```
class test_sequence;
    randc bit [2:0] test_id; // Cycles through all values
    rand bit [7:0] data;

    // test_id will generate 0,1,2,3,4,5,6,7 in random order
    // then repeat the cycle
endclass

// Usage
test_sequence seq = new();
for (int i = 0; i < 10; i++) begin
    seq.randomize();
    $display("Test %d: ID=%d", i, seq.test_id);
end
```

Class Methods

new() - Constructor method

```
class memory_model;
    bit [7:0] mem_array[];
    int size;

    function new(int mem_size = 1024); // Constructor with default
        size = mem_size;
        mem_array = new[size];
        // Initialize memory
        foreach (mem_array[i]) begin
            mem_array[i] = 8'h00;
        end
        $display("Memory created with size %d", size);
    endfunction
endclass

// Usage
memory_model mem1 = new(); // Use default size
memory_model mem2 = new(2048); // Specify size
```

this - Reference to current object

```
class linked_node;
    int data;
    linked_node next;

    function new(int value);
        this.data = value; // Explicit reference to this object
        this.next = null;
    endfunction

    function linked_node add_node(int value);
        linked_node new_node = new(value);
        new_node.next = this; // Link to current object
        return new_node;
    endfunction
endclass
```

super - Reference to parent class

```
class vehicle;
    string make, model;

    function new(string m, string md);
        make = m;
        model = md;
    endfunction

    virtual function void start();
        $display("%s %s starting", make, model);
    endfunction
endclass

class car extends vehicle;
    int doors;

    function new(string m, string md, int d);
        super.new(m, md); // Call parent constructor
        doors = d;
    endfunction

    function void start();
        super.start(); // Call parent method
        $display("Car with %d doors ready", doors);
    endfunction
endclass
```

Assertions and Coverage

Immediate Assertions

assert - Immediate assertion

```
always_comb begin
    // Check that address is word-aligned
    assert (address[1:0] == 2'b00)
        else $error("Address %h is not word-aligned", address);

    // Check FIFO bounds
    assert (fifo_count <= FIFO_DEPTH)
        else $fatal("FIFO overflow: count=%d, depth=%d",
                    fifo_count, FIFO_DEPTH);
end

// In procedural block
always_ff @(posedge clk) begin
    if (write_enable) begin
        assert (!fifo_full)
            else $warning("Writing to full FIFO");
    end
end
```

assume - Assumption for formal verification

```
// Constrain inputs for formal verification
always_comb begin
    assume (reset -> !enable); // Reset implies not enabled
    assume (valid -> ready); // Valid implies ready
    assume (address < MAX_ADDRESS); // Address within bounds
end

// Environment assumptions
initial begin
    assume property (@(posedge clk) $rose(reset) |-> ##[1:5] $fell(reset));
end
```

cover - Coverage point

```
always_ff @(posedge clk) begin
    // Cover interesting scenarios
    cover (state == IDLE && interrupt);
    cover (fifo_empty && read_request);
    cover (cache_hit && cache_miss); // Simultaneous hit/miss

    // Cover sequence of events
    cover ($past(error_detected) && recovery_complete);
end
```

Concurrent Assertions

property - Property definition

```
property valid_handshake;
    @(posedge clk) disable iff (reset)
    valid |-> ##[1:5] ready; // Valid followed by ready in 1-5 cycles
endproperty

property no_back_to_back_writes;
    @(posedge clk) disable iff (reset)
    write_enable |-> ##1 !write_enable;
endproperty

// Use properties in assertions
assert property (valid_handshake)
    else $error("Handshake protocol violated");
cover property (no_back_to_back_writes);
```

endproperty - End property definition

```
property fifo_operation;
    @(posedge clk) disable iff (reset)
    (push && !pop && !full) |-> ##1 (count == $past(count) + 1);
endproperty // End of property definition

assert property (fifo_operation);
```

sequence - Sequence definition

```
sequence request_sequence;
   @(posedge clk) request ##1 grant ##[1:3] done;
endsequence

sequence reset_sequence;
   @(posedge clk) $rose(reset) ##[1:10] $fell(reset);
endsequence

// Use sequences in properties
property request_completion;
   @(posedge clk) request_sequence |-> ##1 status_ok;
endproperty
```

endsequence - End sequence definition

```
sequence bus_cycle;
   @(posedge clk)
   address_valid ##1 data_valid ##1 ready ##1 complete;
endsequence // End of sequence definition

assert property (@(posedge clk) bus_cycle |-> ##1 ack);
```

Temporal Operators

- Delay operator

```
property clock_delay;
   @(posedge clk) start |-> ##5 finish; // Exactly 5 cycles later
endproperty

property range_delay;
   @(posedge clk) req |-> ##[2:8] ack; // Between 2 and 8 cycles
endproperty

property variable_delay;
   @(posedge clk) cmd |-> ##delay_count response; // Variable delay
endproperty
```

|-> - Implication

```
property write_implies_ready;
   @(posedge clk) write_request |-> ready; // If write_request then ready
endproperty

property error_handling;
   @(posedge clk) error_detected |-> ##[1:3] error_cleared;
endproperty

assert property (write_implies_ready);
```

|=> - Overlapping implication

```
property overlapping_check;
   @(posedge clk) valid |=> data_stable; // valid and data_stable overlap
endproperty

// Equivalent to: valid |-> ##1 data_stable
property equivalent_check;
   @(posedge clk) valid |-> ##1 data_stable;
endproperty
```

throughout - Throughout operator

```
property data_stable_during_valid;
   @(posedge clk)
   (data_stable throughout (valid ##1 ready ##1 done));
endproperty

// Data must remain stable from valid until done
assert property (data_stable_during_valid);
```

within - Within operator

```
```

```
property response_within_request;
    @(posedge clk)
    (ack within (req ##[1:$] grant)); // ack must occur between req and grant
endproperty

assert property (response_within_request);
```

System Tasks and Functions

Display Functions

$display - Print formatted text

```
initial begin
    $display("Simulation started at time %t", $time);
    $display("Counter value: %d (hex: %h)", counter, counter);
    $display("Status: %s", error ? "ERROR" : "OK");

    // Formatted output
    $display("Address: %4h, Data: %2h, Valid: %b",
             address, data, valid);
end
```

$write - Print without newline

```
initial begin
    $write("Loading memory: ");
    for (int i = 0; i < 10; i++) begin
        memory[i] = i;
        $write("%d ", i); // Print progress on same line
    end
    $display("Done!"); // Add newline at end
end
```

$monitor - Monitor signal changes

```
```

```
initial begin
    $monitor("Time: %t, Clock: %b, Reset: %b, State: %s",
             $time, clk, reset, state.name());
    // Monitor will print whenever any monitored signal changes
    #1000 $monitoroff; // Stop monitoring
    #500 $monitoron; // Resume monitoring
end
```

$strobe - Display at end of time step

```
always @(posedge clk) begin
    data <= new_data;
    valid <= 1'b1;
    $strobe("End of cycle: data=%h, valid=%b", data, valid);
    // $strobe shows the final values after all updates
end
```

File I/O

$fopen - Open file

```
int file_handle;

initial begin
    file_handle = $fopen("output.txt", "w");
    if (file_handle == 0) begin
        $display("Error: Could not open file");
        $finish;
    end
    $fwrite(file_handle, "Simulation Log\n");
    $fwrite(file_handle, "Started at %t\n", $time);
end
```

$fclose - Close file

```
final begin
    if (file_handle != 0) begin
        $fwrite(file_handle, "Simulation ended at %t\n", $time);
        $fclose(file_handle);
        $display("Log file closed");
    end
end
```

$fwrite - Write to file

```
always @(posedge clk) begin
    if (transaction_valid) begin
        $fwrite(file_handle, "%t: ADDR=%h, DATA=%h, TYPE=%s\n",
                $time, address, data,
                write_enable ? "WRITE" : "READ");
    end
end
```

$readmemh - Read hex memory file

```
logic [7:0] rom[256];

initial begin
    $readmemh("program.hex", rom); // Load hex file into ROM
    $display("Loaded %d bytes from program.hex", $size(rom));

    // Display first few bytes
    for (int i = 0; i < 8; i++) begin
        $display("ROM[%d] = %h", i, rom[i]);
    end
end
```

$readmemb - Read binary memory file

```
logic [15:0] pattern_memory[64];

initial begin
    $readmemb("test_patterns.bin",  pattern_memory);
    $display("Loaded binary test patterns");
end
```

Simulation Control

$finish - End simulation

```
initial begin
    #10000; // Run for 10000 time units
    $display("Test completed successfully");
    $finish; // Normal exit
end

always @(posedge clk) begin
    if (fatal_error) begin
        $display("FATAL ERROR: %s", error_message);
        $finish(2); // Exit with error code
    end
end
```

$stop - Suspend simulation

```
always @(posedge clk) begin
    if (breakpoint_hit) begin
        $display("Breakpoint hit at PC=%h", program_counter);
        $stop; // Suspend for debugging
    end
end

// Interactive debugging
initial begin
    #1000;
    $stop; // Pause for inspection
    // Simulation can be resumed from simulator
end
```

$time - Current simulation time

```
always @(posedge clk) begin
    if (enable) begin
        start_time = $time;
        // Perform operation
        end_time = $time;
        duration = end_time - start_time;
        $display("Operation took %t time units", duration);
    end
end
```

$realtime - Real-valued simulation time

```
real timestamp;

always @(posedge clk) begin
    timestamp = $realtime;
    $display("Real time: %f ns", timestamp);

    // Calculate frequency
    if (last_timestamp > 0) begin
        real period = timestamp - last_timestamp;
        real frequency = 1.0 / period;
        $display("Clock frequency: %f MHz", frequency * 1e6);
    end
    last_timestamp = timestamp;
end
```

Random Functions

$random - Pseudo-random number

```
```

```
initial begin
    int seed = 12345;
    int random_val;

    random_val = $random(seed); // Initialize seed
    for (int i = 0; i < 10; i++) begin
        random_val = $random(); // Generate random numbers
        $display("Random[%d] = %d", i, random_val);
    end
end
```

$urandom - Uniform random number

```
task generate_test_data();
    for (int i = 0; i < 100; i++) begin
        test_data[i] = $urandom(); // 32-bit uniform random
        test_addr[i] = $urandom() % 1024; // Random address 0-1023
    end
endtask
```

$urandom_range - Random in range

```
initial begin
    int delay, priority, size;

    delay = $urandom_range(1, 10); // Random delay 1-10 cycles
    priority = $urandom_range(0, 7); // Random priority 0-7
    size = $urandom_range(64, 1024); // Random size 64-1024 bytes

    $display("Generated: delay=%d, priority=%d, size=%d",
             delay, priority, size);
end
```

String Functions

$sformatf - Format string

```
```

```
string message, filename, status_msg;

always_comb begin
    message = $sformatf("Error at address %h, data %h",
                        error_addr, error_data);
    filename = $sformatf("test_run_%0d.log", test_number);
    status_msg = $sformatf("Progress: %0d/%0d (%0.1f%%)",
                           completed, total,
                           100.0 * completed / total);
end
```

$sscanf - Parse string

```
string input_line = "ADDR:1234 DATA:ABCD TYPE:READ";
string address_str, data_str, type_str;
int addr_val, data_val;

initial begin
    $sscanf(input_line, "ADDR:%s DATA:%s TYPE:%s",
            address_str, data_str, type_str);
    addr_val = address_str.atohex();
    data_val = data_str.atohex();
    $display("Parsed: addr=%h, data=%h, type=%s",
             addr_val, data_val, type_str);
end
```

$strlen - String length

```
function void print_status(string msg);
    int len = msg.len(); // Or $strlen(msg)
    string border = {len+4{"*"}};
    $display("%s", border);
    $display("* %s *", msg);
    $display("%s", border);
endfunction

// Usage
print_status("Test Complete");
```

$substr - Substring

```
string full_path = "/home/user/project/testbench.sv";
string filename, extension;

initial begin
    int last_slash = 0;
    int dot_pos = 0;

    // Find last slash and dot
    for (int i = 0; i < full_path.len(); i++) begin
        if (full_path[i] == "/") last_slash = i;
        if (full_path[i] == ".") dot_pos = i;
    end

    filename = full_path.substr(last_slash + 1, dot_pos - 1);
    extension = full_path.substr(dot_pos + 1, full_path.len() - 1);
    $display("File: %s, Extension: %s", filename, extension);
end
```

Constraints (for Random Verification)

Constraint Blocks

constraint - Constraint definition

```
class packet;
   rand bit [7:0] length;
   rand bit [7:0] data[];
   rand bit [1:0] priority;

   constraint length_c {
      length inside {[10:100]}; // Length between 10 and 100
      length % 4 == 0; // Must be multiple of 4
   }

   constraint data_c {
      data.size() == length; // Array size matches length
      foreach (data[i]) {
         data[i] inside {[1:254]}; // No 0 or 255 values
      }
   }

   constraint priority_c {
      priority dist {0 := 10, 1 := 30, 2 := 40, 3 := 20};
   }
endclass
```

solve...before - Constraint ordering

```
class memory_transaction;
    rand bit [31:0] address;
    rand bit [7:0] data[];
    rand bit [3:0] burst_length;

    constraint solve_order {
        solve burst_length before data; // Solve burst_length first
        solve address before burst_length;
    }

    constraint burst_c {
        burst_length inside {[1:16]};
    }

    constraint data_c {
        data.size() == burst_length; // Depends on burst_length
    }

    constraint addr_c {
        address[1:0] == 2'b00; // Word aligned
        if (burst_length > 8) {
            address[3:0] == 4'b0000; // Cache line aligned for long bursts
        }
    }
endclass
```

if...else - Conditional constraints

```
class bus_transaction;
    rand bit read_write; // 0=read, 1=write
    rand bit [7:0] data;
    rand bit [15:0] address;
    rand bit error_inject;

    constraint conditional_c {
        if (read_write == 1) { // Write transaction
            data inside {[1:255]}; // Valid data for writes
            error_inject == 0; // No errors on writes
        } else { // Read transaction
            data == 0; // Data unused for reads
            if (address < 16'h1000) {
                error_inject == 0; // No errors in system area
            }
        }
    }
endclass
```

foreach - Array constraints

```
class test_pattern;
    rand bit [7:0] pattern[16];
    rand bit ascending;

    constraint pattern_c {
        if (ascending) {
            foreach (pattern[i]) {
                if (i > 0) {
                    pattern[i] > pattern[i-1]; // Ascending pattern
                }
            }
        }
        foreach (pattern[i]) {
            pattern[i] inside {[10:245]}; // Valid range
            if (i % 2 == 0) {
                pattern[i][0] == 1'b1; // Even indices have LSB set
            }
        }
    }
endclass
```

Constraint Operators

inside - Membership operator

```
class address_generator;
    rand bit [31:0] address;
    rand bit [2:0] size;

    constraint valid_addr {
        address inside {[32'h1000:32'h1FFF], // Range 1
                        [32'h4000:32'h4FFF], // Range 2
                        32'h8000, 32'h8004, // Specific addresses
                        32'h8008, 32'h800C};
        size inside {0, 1, 2, 4}; // Valid sizes only
    }
endclass
```

dist - Distribution constraint

```
class weighted_generator;
    rand bit [3:0] opcode;
    rand bit [7:0] data;

    constraint opcode_dist {
        opcode dist {
            4'b0000 := 40, // ADD: 40% weight
            4'b0001 := 30, // SUB: 30% weight
            4'b0010 := 20, // AND: 20% weight
            4'b0011 := 10, // OR: 10% weight
            [4'b0100:4'b1111] :/ 20 // Others: 20% total (distributed)
        };
    }

    constraint data_dist {
        data dist {
            [0:63] := 50, // Low values: 50% weight
            [64:191] := 30, // Mid values: 30% weight
            [192:255] := 20 // High values: 20% weight
        };
    }
endclass
```

-> - Constraint implication

```
class conditional_packet;
    rand bit [1:0] packet_type;
    rand bit [7:0] length;
    rand bit [15:0] checksum;
    rand bit priority;

    constraint implication_c {
        (packet_type == 2'b11) -> (length >= 64); // Long packets are big
        (length > 128) -> (priority == 1'b1); // Big packets are priority
        (priority == 1'b1) -> (checksum != 16'h0000); // Priority has checksum

        // Multiple implications
        (packet_type == 2'b00) -> {length inside {[8:32]}; checksum == 0;};
    }
endclass
```

Compiler Directives

Preprocessing

`define - Macro definition

```
`define CLK_PERIOD 10 // Simple macro
`define MAX_ADDR 32'hFFFF
`define DEBUG_MSG(msg) $display("[DEBUG] %s at %t", msg, $time)

// Parameterized macro
`define ASSERT_CLK(signal, message) \
    assert property (@(posedge clk) signal) \
        else $error(message)

module test;
    logic clk, reset, valid;

    initial begin
        clk = 0;
        forever #(`CLK_PERIOD/2) clk = ~clk; // Use macro
    end

    always @(posedge clk) begin
        `DEBUG_MSG("Clock edge detected"); // Use debug macro
    end

    `ASSERT_CLK(reset || valid, "Valid must be high when not in reset")
endmodule
```

`undef - Undefine macro

```
`define TEMP_VALUE 42

module temp_user;
    int value = `TEMP_VALUE;
endmodule

`undef TEMP_VALUE // Remove macro definition

module another_module;
    // int value = `TEMP_VALUE; // Error - undefined macro
endmodule
```

`ifdef - Conditional compilation

```
`define SYNTHESIS // Define for synthesis

module design_top;
   logic clk, reset, data_valid;
   logic [31:0] data;

`ifdef SYNTHESIS
   // Synthesis-specific code
   always_ff @(posedge clk) begin
      if (reset) begin
         data <= 32'h0;
      end else if (data_valid) begin
         data <= input_data;
      end
   end
`else
   // Simulation-specific code with assertions
   always_ff @(posedge clk) begin
      if (reset) begin
         data <= 32'h0;
      end else if (data_valid) begin
         data <= input_data;
         assert(input_data !== 32'hxxxxxxxx)
            else $warning("Unknown data input!");
      end
   end

   // Additional simulation checks
   property valid_data;
      @(posedge clk) data_valid |-> !$isunknown(input_data);
   endproperty
   assert property (valid_data);
`endif
endmodule
```

`ifndef - Negative conditional compilation

```
`ifndef CLK_FREQ
    `define CLK_FREQ 100 // Default clock frequency
`endif

module clock_gen;
    logic clk = 0;

`ifndef FAST_SIM
    // Normal simulation timing
    initial forever #(500/`CLK_FREQ) clk = ~clk;
`else
    // Fast simulation - higher frequency
    initial forever #1 clk = ~clk;
`endif
endmodule
```

`else - Else in conditional compilation

```
`ifdef FPGA_TARGET
    // FPGA-specific implementation
    module memory_block(
        input clk,
        input [9:0] addr,
        inout [31:0] data
    );
        // Use FPGA block RAM
        (* ram_style = "block" *) logic [31:0] mem[1024];
        // FPGA implementation
    endmodule
`else
    // ASIC or simulation implementation
    module memory_block(
        input clk,
        input [9:0] addr,
        inout [31:0] data
    );
        logic [31:0] mem[1024];
        // Standard implementation
    endmodule
`endif
```

`endif - End conditional compilation

```
`ifdef DEBUG_LEVEL_1
   `define DBG1(msg) $display("[DBG1] %s", msg)
   `ifdef DEBUG_LEVEL_2
      `define DBG2(msg) $display("[DBG2] %s", msg)
   `else
      `define DBG2(msg) // Empty for level 1
   `endif
`else
   `define DBG1(msg) // Empty
   `define DBG2(msg) // Empty
`endif // End of debug level conditionals

module debug_example;
   initial begin
      `DBG1("Starting test");
      `DBG2("Detailed debug info");
   end
endmodule
```

`include - Include file

```
// common_defines.sv
`define BUS_WIDTH 32
`define ADDR_WIDTH 16
`define MAX_TRANSACTIONS 1000

// interfaces.sv
interface cpu_if;
    logic [`ADDR_WIDTH-1:0] address;
    logic [`BUS_WIDTH-1:0] data;
    logic read, write, ready;
endinterface

// main_design.sv
`include "common_defines.sv"
`include "interfaces.sv"

module cpu_top(
    input logic clk,
    input logic reset,
    cpu_if.master bus
);
    logic [`BUS_WIDTH-1:0] internal_data;
    // Use included definitions and interfaces
endmodule
```

Simulation Directives

`timescale - Time unit and precision

```
`timescale 1ns/1ps // 1ns time unit, 1ps precision

module timing_example;
    logic clk = 0;

    initial begin
        forever #5 clk = ~clk; // 10ns period (100MHz)
        // Precise timing with picosecond precision
        #10.5 $display("Time: %t", $time); // 10.5ns
        #0.1 $display("Time: %t", $time); // 10.6ns
    end
endmodule

// Different timescale for another module
`timescale 1us/1ns
module slow_module;
    logic slow_clk = 0;
    initial forever #1 slow_clk = ~slow_clk; // 2us period
endmodule
```

`default_nettype - Default net type

```
`default_nettype none // Disable implicit nets

module safe_design(
    input wire clk,
    input wire reset,
    output wire [7:0] result
);
    logic [7:0] counter;

    // typo_signal would cause error (not implicitly declared as wire)
    // assign typo_signal = counter[0]; // Error caught!

    always_ff @(posedge clk) begin
        if (reset)
            counter <= 8'h00;
        else
            counter <= counter + 1;
    end

    assign result = counter;
endmodule

`default_nettype wire // Restore default
```

`celldefine - Cell definition start

```
`celldefine // Mark as library cell
module and_gate(
    input a, b,
    output y
);
    assign y = a & b;

    // Timing specifications for library cell
    specify
        (a => y) = (1.2, 1.5);
        (b => y) = (1.1, 1.4);
    endspecify
endmodule
`endcelldefine // End cell definition
```

`endcelldefine - Cell definition end

```
`celldefine
module flip_flop(
    input d, clk, reset,
    output reg q
);
    always @(posedge clk or posedge reset) begin
        if (reset)
            q <= 1'b0;
        else
            q <= d;
    end

    specify
        $setup(d, posedge clk, 0.5);
        $hold(posedge clk, d, 0.3);
        (posedge clk => (q +: d)) = (0.8, 1.0);
    endspecify
endmodule
`endcelldefine // Library cell definition complete
```

Packages and Imports

Package Definition

package - Package definition

```
package cpu_pkg;
    // Type definitions
    typedef enum {ADD, SUB, AND, OR, XOR, SHL, SHR, JMP} opcode_t;
    typedef enum {IDLE, FETCH, DECODE, EXECUTE, WRITEBACK} cpu_state_t;

    // Constants
    parameter int REGISTER_COUNT = 32;
    parameter int INSTRUCTION_WIDTH = 32;
    parameter int DATA_WIDTH = 32;

    // Function definitions
    function automatic bit [4:0] count_ones(bit [31:0] data);
        count_ones = 0;
        for (int i = 0; i < 32; i++) begin
            count_ones += data[i];
        end
    endfunction

    // Class definitions
    class instruction;
        opcode_t opcode;
        bit [4:0] rs, rt, rd;
        bit [15:0] immediate;

        function new(bit [31:0] inst_word);
            {opcode, rs, rt, rd, immediate} = inst_word;
        endfunction
    endclass
endpackage
```

endpackage - End package definition

```
package math_pkg;
    // Mathematical constants
    parameter real PI = 3.14159265359;
    parameter real E = 2.71828182846;

    // Utility functions
    function automatic real sin_approx(real x);
        // Taylor series approximation
        return x - (x**3)/6.0 + (x**5)/120.0;
    endfunction

    function automatic int gcd(int a, int b);
        while (b != 0) begin
            int temp = b;
            b = a % b;
            a = temp;
        end
        return a;
    endfunction
endpackage // End of math package definition
```

export - Export from package

```
package base_pkg;
    typedef enum {RED, GREEN, BLUE} color_t;
    parameter int MAX_SIZE = 1024;

    function int square(int x);
        return x * x;
    endfunction

    export *::*; // Export everything
endpackage

package extended_pkg;
    import base_pkg::*;
    typedef enum {CYAN, MAGENTA, YELLOW} extended_color_t;

    export base_pkg::color_t; // Re-export specific items
    export base_pkg::MAX_SIZE;
    export extended_color_t; // Export new items
endpackage
```

Import Statements

import - Import from package

```
import cpu_pkg::*; // Import everything from package

module cpu_core(
    input logic clk, reset,
    input logic [INSTRUCTION_WIDTH-1:0] instruction,
    output cpu_state_t current_state
);
    opcode_t current_opcode;
    instruction current_inst;

    always_ff @(posedge clk) begin
        if (reset) begin
            current_state <= IDLE;
        end else begin
            case (current_state)
                IDLE: current_state <= FETCH;
                FETCH: current_state <= DECODE;
                DECODE: begin
                    current_inst = new(instruction);
                    current_opcode = current_inst.opcode;
                    current_state <= EXECUTE;
                end
                EXECUTE: current_state <= WRITEBACK;
                WRITEBACK: current_state <= FETCH;
            endcase
        end
    end
endmodule

// Selective import
module alu(
    input logic clk,
    input cpu_pkg::opcode_t operation, // Specific import
    input logic [31:0] operand_a, operand_b,
    output logic [31:0] result
);
    import cpu_pkg::count_ones; // Import specific function

    always_comb begin
        case (operation)
            cpu_pkg::ADD: result = operand_a + operand_b;
            cpu_pkg::SUB: result = operand_a - operand_b;
            cpu_pkg::AND: result = operand_a & operand_b;
```

```
        cpu_pkg::OR: result = operand_a | operand_b;
        default: result = 32'h0;
      endcase
    end
endmodule
```

:: - Scope resolution operator

```
package pkg_a;
    parameter int SIZE = 64;
    typedef enum {STATE_A, STATE_B} state_t;
endpackage

package pkg_b;
    parameter int SIZE = 128; // Same name, different value
    typedef enum {STATE_X, STATE_Y} state_t;
endpackage

module scope_example;
    // Explicit scope resolution to avoid conflicts
    logic [pkg_a::SIZE-1:0] buffer_a; // 64-bit buffer
    logic [pkg_b::SIZE-1:0] buffer_b; // 128-bit buffer

    pkg_a::state_t state_machine_a;
    pkg_b::state_t state_machine_b;

    initial begin
        state_machine_a = pkg_a::STATE_A;
        state_machine_b = pkg_b::STATE_X;
        $display("Package A size: %d", pkg_a::SIZE);
        $display("Package B size: %d", pkg_b::SIZE);
    end
endmodule
```

Special Keywords

Simulation and Synthesis

$root - Top-level scope

```
```

```
module top;
    int global_counter = 0;

    always @(posedge clk) begin
        global_counter++;
    end
endmodule

module test_module;
    initial begin
        // Access top-level scope from anywhere
        $display("Global counter: %d", $root.top.global_counter);

        // Force signal in top module
        force $root.top.global_counter = 100;
        #10;
        release $root.top.global_counter;
    end
endmodule
```

$unit - Compilation unit scope

```
// File: common_declarations.sv
int $unit::debug_level = 2; // Compilation unit variable
typedef enum {PASS, FAIL, ERROR} $unit::test_result_t;

function automatic void $unit::print_debug(string msg);
    if (debug_level > 0) begin
        $display("[DEBUG] %s", msg);
    end
endfunction

// File: test1.sv
module test1;
   initial begin
        $unit::print_debug("Test 1 starting");
        $unit::test_result_t result = $unit::PASS;
    end
endmodule

// File: test2.sv
module test2;
   initial begin
        $unit::debug_level = 3; // Modify unit scope variable
        $unit::print_debug("Test 2 with higher debug level");
    end
endmodule
```

bind - Bind assertion modules

```
// Assertion module
module fifo_assertions(
    input logic clk, reset,
    input logic push, pop,
    input logic full, empty,
    input logic [3:0] count
);
    // FIFO property checks
    property no_push_when_full;
        @(posedge clk) disable iff (reset)
        full |-> !push;
    endproperty

    property no_pop_when_empty;
        @(posedge clk) disable iff (reset)
        empty |-> !pop;
    endproperty

    property count_consistency;
        @(posedge clk) disable iff (reset)
        (push && !pop && !full) |-> ##1 (count == $past(count) + 1);
    endproperty

    assert property (no_push_when_full);
    assert property (no_pop_when_empty);
    assert property (count_consistency);
endmodule

// FIFO design
module fifo(
    input logic clk, reset,
    input logic push, pop,
    input logic [7:0] data_in,
    output logic [7:0] data_out,
    output logic full, empty,
    output logic [3:0] count
);
    // FIFO implementation
    logic [7:0] memory[16];
    logic [3:0] write_ptr, read_ptr;
    // Implementation details...
endmodule
```

```
// Bind assertions to design
bind fifo fifo_assertions fifo_check(
    .clk(clk), .reset(reset),
    .push(push), .pop(pop),
    .full(full), .empty(empty),
    .count(count)
);
```

alias - Create signal alias

```
module bus_interface(
    inout wire [31:0] cpu_data,
    inout wire [31:0] mem_data,
    input logic bridge_enable
);
    // Create aliases for cleaner code
    alias data_bus = bridge_enable ? cpu_data : mem_data;
    alias addr_high = cpu_data[31:16];
    alias addr_low = cpu_data[15:0];

    // Use aliases in logic
    always_comb begin
        if (bridge_enable) begin
            // data_bus refers to cpu_data
            mem_data = data_bus;
        end else begin
            // data_bus refers to mem_data
            cpu_data = data_bus;
        end
    end
endmodule
```

Miscellaneous

void - Void data type

```
class test_driver;
    task drive_transaction();
        // Task implementation
    endtask

    function void print_status(); // Function returns nothing
        $display("Driver status: ready");
    endfunction

    function void randomize_and_send();
        void'(this.randomize()); // Ignore randomize return value
        drive_transaction();
    endfunction
endclass

// Void pointer equivalent
class base_item;
    int data;
endclass

class container;
    base_item items[$];

    function void add_item(base_item item);
        items.push_back(item);
    endfunction
endclass
```

null - Null pointer

```
class linked_list;
   int data;
   linked_list next;

   function new(int value);
      data = value;
      next = null; // Initialize to null
   endfunction

   function void append(int value);
      linked_list current = this;
      // Find end of list
      while (current.next != null) begin
         current = current.next;
      end
      // Add new node
      current.next = new(value);
   endfunction

   function int find(int value);
      linked_list current = this;
      int position = 0;
      while (current != null) begin
         if (current.data == value) begin
            return position;
         end
         current = current.next;
         position++;
      end
      return -1; // Not found
   endfunction
endclass
```

default - Default case/clocking

```
```

```
// Default in case statements
always_comb begin
    case (opcode)
        4'b0000: result = a + b;
        4'b0001: result = a - b;
        4'b0010: result = a & b;
        default: result = 32'hDEADBEEF; // Default case
    endcase
end

// Default clocking
interface test_if(input logic clk);
    logic [7:0] data;
    logic valid, ready;

    default clocking cb @(posedge clk); // Default clocking block
        default input #1step output #1ns;
        input ready;
        output data, valid;
    endclocking

    // Can reference default clocking without name
    task send_data(logic [7:0] d);
        cb.data <= d; // Uses default clocking
        cb.valid <= 1'b1;
        wait(cb.ready);
        cb.valid <= 1'b0;
    endtask
endinterface
```

global - Global clocking

```
```

```
// Global clocking domain
global clocking @(posedge system_clk);
   default input #1step output #1ns;
endclocking

program test_program;
   initial begin
      // Use global clocking
      ##5; // Wait 5 clock cycles
      $display("Five cycles elapsed");
      ##[1:10]; // Wait 1 to 10 cycles
      $display("Random wait completed");
   end
endprogram
```

disable - Disable named blocks

```
module timeout_example;
  event timeout, transaction_done;

  initial begin
    fork : main_block
      // Transaction thread
      begin : transaction_thread
        // Simulate transaction
        #1000;
        ->transaction_done;
      end

      // Timeout thread
      begin : timeout_thread
        #500; // 500 time unit timeout
        ->timeout;
        disable main_block.transaction_thread;
      end
    join

    if (timeout.triggered) begin
      $display("Transaction timed out");
    end else begin
      $display("Transaction completed");
    end
  end

  // Disable from external event
  always @(reset) begin
    if (reset) begin
      disable main_block; // Disable entire block
    end
  end
endmodule
```

fork...join - Parallel execution

```
initial begin
    $display("Starting parallel execution");
    fork
        // Thread 1
        begin
            #10;
            $display("Thread 1 completed at %t", $time);
        end

        // Thread 2
        begin
            #20;
            $display("Thread 2 completed at %t", $time);
        end

        // Thread 3
        begin
            #15;
            $display("Thread 3 completed at %t", $time);
        end
    join // Wait for ALL threads to complete

    $display("All threads finished at %t", $time);
end
```

fork...join_any - Join when any completes

```
task wait_for_any_interrupt();
    fork
        // Wait for timer interrupt
        begin
            wait(timer_interrupt);
            $display("Timer interrupt occurred");
        end

        // Wait for UART interrupt
        begin
            wait(uart_interrupt);
            $display("UART interrupt occurred");
        end

        // Wait for GPIO interrupt
        begin
            wait(gpio_interrupt);
            $display("GPIO interrupt occurred");
        end
    join_any // Return when ANY thread completes

    disable fork; // Kill remaining threads
    $display("Interrupt handling complete");
endtask
```

fork...join_none - Non-blocking fork

```
initial begin
    $display("Starting background tasks");
    fork
        // Background monitoring
        begin
            forever begin
                @(posedge clk);
                if (error_flag) begin
                    $display("Error detected at %t", $time);
                end
            end
        end

        // Background logging
        begin
            forever begin
                @(posedge transaction_complete);
                $display("Transaction logged at %t", $time);
            end
        end
    join_none // Don't wait - continue immediately

    $display("Background tasks started, continuing main flow");
    // Main test continues here while background tasks run
    #1000;
    $display("Main test completed");
    $finish; // Background tasks killed at finish
end
```

Array Methods

Dynamic Arrays

.size() - Get array size

```
int dynamic_array[];
int queue_data[$];

initial begin
    dynamic_array = new[10];
    queue_data = {1, 2, 3, 4, 5};

    $display("Dynamic array size: %d", dynamic_array.size());
    $display("Queue size: %d", queue_data.size());

    // Resize array
    dynamic_array = new[20](dynamic_array); // Copy old data
    $display("New size: %d", dynamic_array.size());
end
```

.delete() - Delete array elements

```
int assoc_array[string];
int queue_data[$] = {10, 20, 30, 40, 50};

initial begin
    // Populate associative array
    assoc_array["apple"] = 1;
    assoc_array["banana"] = 2;
    assoc_array["cherry"] = 3;

    $display("Array size before: %d", assoc_array.size());

    // Delete specific element
    assoc_array.delete("banana");
    $display("Array size after deleting banana: %d", assoc_array.size());

    // Delete all elements
    assoc_array.delete();
    $display("Array size after delete(): %d", assoc_array.size());

    // Delete queue elements
    queue_data.delete(2); // Delete index 2 (value 30)
    $display("Queue after delete(2): %p", queue_data);
end
```

.exists(index) - Check if index exists

```
int sparse_array[int];

initial begin
    sparse_array[5] = 50;
    sparse_array[10] = 100;
    sparse_array[15] = 150;

    for (int i = 0; i < 20; i++) begin
        if (sparse_array.exists(i)) begin
            $display("Index %d exists with value %d", i, sparse_array[i]);
        end else begin
            $display("Index %d does not exist", i);
        end
    end

    // Check before accessing to avoid warnings
    if (sparse_array.exists(10)) begin
        int value = sparse_array[10];
        $display("Safe access: value at index 10 is %d", value);
    end
end
```

Queues

.size() - Queue size

```
int data_queue[$] = {1, 2, 3, 4, 5};

initial begin
    $display("Initial queue size: %d", data_queue.size());

    // Add elements
    data_queue.push_back(6);
    data_queue.push_front(0);

    $display("Queue after additions: size=%d, data=%p",
            data_queue.size(), data_queue);
end
```

.insert(index, item) - Insert at index

```
string word_queue[$] = {"hello", "world"};

initial begin
    $display("Before insert: %p", word_queue);

    // Insert at specific positions
    word_queue.insert(1, "beautiful"); // Insert between hello and world
    $display("After insert(1): %p", word_queue);

    word_queue.insert(0, "say"); // Insert at beginning
    $display("After insert(0): %p", word_queue);

    word_queue.insert(word_queue.size(), "today"); // Insert at end
    $display("Final queue: %p", word_queue);
end
```

.delete(index) - Delete at index

```
int number_queue[$] = {10, 20, 30, 40, 50};

initial begin
    $display("Before delete: %p", number_queue);

    // Delete specific indices
    number_queue.delete(2); // Delete index 2 (value 30)
    $display("After delete(2): %p", number_queue);

    number_queue.delete(0); // Delete first element
    $display("After delete(0): %p", number_queue);
end
```

.push_front(item) - Add to front

```
int fifo_queue[$];

task add_high_priority(int data);
    fifo_queue.push_front(data); // High priority goes to front
    $display("Added high priority %d, queue: %p", data, fifo_queue);
endtask

initial begin
    // Add normal items
    fifo_queue = {1, 2, 3, 4};
    $display("Initial queue: %p", fifo_queue);

    // Add high priority items
    add_high_priority(99);
    add_high_priority(88);
end
```

.push_back(item) - Add to back

```
string log_queue[$];

task log_message(string msg);
    log_queue.push_back(msg); // Add to end of log
    $display("Logged: %s", msg);

    // Keep only last 10 messages
    if (log_queue.size() > 10) begin
        string old_msg = log_queue.pop_front();
    end
endtask

initial begin
    log_message("System startup");
    log_message("Initialization complete");
    log_message("Ready for operation");
    $display("Full log: %p", log_queue);
end
```

.pop_front() - Remove from front

```
int command_queue[$] = {1, 2, 3, 4, 5};

task process_commands();
    while (command_queue.size() > 0) begin
        int cmd = command_queue.pop_front();
        $display("Processing command %d", cmd);

        // Simulate command processing
        #10;
        $display("Command %d complete, remaining: %p", cmd, command_queue);
    end
    $display("All commands processed");
endtask

initial begin
    process_commands();
end
```

.pop_back() - Remove from back

```
int stack_queue[$];

task push_stack(int data);
    stack_queue.push_back(data); // Add to back (top of stack)
    $display("Pushed %d, stack: %p", data, stack_queue);
endtask

function int pop_stack();
    if (stack_queue.size() > 0) begin
        int data = stack_queue.pop_back(); // Remove from back (top)
        $display("Popped %d, stack: %p", data, stack_queue);
        return data;
    end else begin
        $display("Stack underflow!");
        return -1;
    end
endfunction

initial begin
    // Stack operations
    push_stack(10);
    push_stack(20);
    push_stack(30);
    int value1 = pop_stack(); // Should get 30
    int value2 = pop_stack(); // Should get 20
    int value3 = pop_stack(); // Should get 10
    int value4 = pop_stack(); // Should underflow
end
```

Array Reduction Methods

.sum() - Sum all elements

```
int numbers[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
real scores[] = {85.5, 92.0, 78.5, 96.0, 88.5};

initial begin
    int total = numbers.sum();
    $display("Sum of numbers 1-10: %d", total); // Should be 55

    real average = scores.sum() / scores.size();
    $display("Average score: %f", average);

    // Sum with condition using with clause
    int even_sum = numbers.sum() with (item % 2 == 0 ? item : 0);
    $display("Sum of even numbers: %d", even_sum);
end
```

.product() - Product of elements

```
int factors[] = {2, 3, 4, 5};
real multipliers[] = {1.5, 2.0, 0.5};

initial begin
    int result = factors.product();
    $display("Product of factors: %d", result); // 2*3*4*5 = 120

    real scale_factor = multipliers.product();
    $display("Combined scale factor: %f", scale_factor); // 1.5*2.0*0.5 = 1.5

    // Product with condition
    int odd_product = factors.product() with (item % 2 == 1 ? item : 1);
    $display("Product of odd numbers: %d", odd_product); // 3*5 = 15
end
```

.and() - AND reduction

```
```

```
bit [7:0] status_flags[] = {8'hFF, 8'hF0, 8'h0F, 8'hAA};
bit control_bits[] = {1'b1, 1'b1, 1'b0, 1'b1};

initial begin
    bit [7:0] and_result = status_flags.and();
    $display("AND of all flags: %h", and_result); // Should be 8'h00

    bit all_ready = control_bits.and();
    $display("All systems ready: %b", all_ready); // Should be 0 (one is 0)

    // AND with condition
    bit [7:0] high_bits = status_flags.and() with (item[7:4]);
    $display("AND of high nibbles: %h", high_bits);
end
```

.or() - OR reduction

```
bit [7:0] error_flags[] = {8'h00, 8'h01, 8'h00, 8'h04};
bit interrupt_sources[] = {1'b0, 1'b0, 1'b1, 1'b0};

initial begin
    bit [7:0] any_error = error_flags.or();
    $display("Any error flags: %h", any_error); // Should be 8'h05

    bit interrupt_pending = interrupt_sources.or();
    $display("Interrupt pending: %b", interrupt_pending); // Should be 1

    // Check if any value exceeds threshold
    int values[] = {10, 25, 30, 15};
    bit over_threshold = values.or() with (item > 20 ? 1'b1 : 1'b0);
    $display("Any over threshold: %b", over_threshold);
end
```

.xor() - XOR reduction

```
bit [7:0] data_bytes[] = {8'hA5, 8'h5A, 8'hFF, 8'h00};
bit parity_bits[] = {1'b1, 1'b0, 1'b1, 1'b1};

initial begin
    bit [7:0] checksum = data_bytes.xor();
    $display("XOR checksum: %h", checksum);

    bit overall_parity = parity_bits.xor();
    $display("Overall parity: %b", overall_parity); // Should be 1

    // Calculate parity of specific bits
    bit lsb_parity = data_bytes.xor() with (item[0]);
    $display("LSB parity: %b", lsb_parity);
end
```

Array Locator Methods

.find() - Find elements matching condition

```
int test_scores[] = {85, 92, 78, 96, 88, 74, 91};
string names[] = {"Alice", "Bob", "Charlie", "David", "Eve"};

initial begin
    // Find all scores above 90
    int high_scores[] = test_scores.find(x) with (x > 90);
    $display("High scores: %p", high_scores); // {92, 96, 91}

    // Find names with more than 4 characters
    string long_names[] = names.find(x) with (x.len() > 4);
    $display("Long names: %p", long_names); // {"Alice", "Charlie", "David"}

    // Find even numbers
    int even_scores[] = test_scores.find(x) with (x % 2 == 0);
    $display("Even scores: %p", even_scores); // {92, 96, 88, 74}
end
```

.find_index() - Find indices of matching elements

```
int data[] = {10, 25, 30, 15, 40, 35};
bit [7:0] status[] = {8'h00, 8'hFF, 8'h80, 8'h7F, 8'hAA};

initial begin
    // Find indices of values greater than 25
    int high_indices[] = data.find_index(x) with (x > 25);
    $display("Indices of high values: %p", high_indices); // {2, 4, 5}

    // Find indices where MSB is set
    int msb_indices[] = status.find_index(x) with (x[7] == 1'b1);
    $display("MSB set indices: %p", msb_indices); // {1, 2, 4}

    // Use indices to access original array
    foreach (high_indices[i]) begin
        int idx = high_indices[i];
        $display("High value at index %d: %d", idx, data[idx]);
    end
end
```

.find_first() - Find first matching element

```
string commands[] = {"read", "write", "erase", "verify", "read", "write"};
int temperatures[] = {20, 25, 30, 35, 40, 45};

initial begin
    // Find first read command
    string first_reads[] = commands.find_first(x) with (x == "read");
    $display("First read command: %p", first_reads); // {"read"}

    // Find first temperature above 30
    int hot_temps[] = temperatures.find_first(x) with (x > 30);
    $display("First hot temperature: %p", hot_temps); // {35}

    // Check if found
    if (first_reads.size() > 0) begin
        $display("Found first read command: %s", first_reads[0]);
    end else begin
        $display("No read commands found");
    end
end
```

.find_last() - Find last matching element

```
int transaction_ids[] = {100, 200, 150, 300, 250, 400};
string log_levels[] = {"INFO", "ERROR", "DEBUG", "ERROR", "INFO", "WARN"};

initial begin
    // Find last transaction ID > 200
    int last_high[] = transaction_ids.find_last(x) with (x > 200);
    $display("Last high transaction: %p", last_high); // {400}

    // Find last error message
    string last_errors[] = log_levels.find_last(x) with (x == "ERROR");
    $display("Last error: %p", last_errors); // {"ERROR"}

    // Get the actual last error index
    int error_indices[] = log_levels.find_index(x) with (x == "ERROR");
    if (error_indices.size() > 0) begin
        int last_error_idx = error_indices[error_indices.size()-1];
        $display("Last error at index: %d", last_error_idx);
    end
end
```

.min() - Find minimum element

```
int response_times[] = {150, 200, 75, 300, 125, 400};
real voltages[] = {3.3, 2.5, 1.8, 5.0, 1.2};

initial begin
    // Find minimum response time
    int min_times[] = response_times.min();
    $display("Minimum response time: %p", min_times); // {75}

    // Find minimum voltage
    real min_volts[] = voltages.min();
    $display("Minimum voltage: %p", min_volts); // {1.2}

    // Find minimum of specific field
    typedef struct {
        string name;
        int age;
        real salary;
    } person_t;

    person_t people[] = '{
        '{"Alice", 25, 50000.0},
        '{"Bob", 30, 45000.0},
        '{"Charlie", 22, 55000.0}
    };

    int min_ages[] = people.min() with (item.age);
    $display("Minimum age: %p", min_ages); // {22}
end
```

.max() - Find maximum element

```
int cpu_usage[] = {45, 78, 92, 23, 67, 89};
string versions[] = {"v1.0", "v2.1", "v1.5", "v3.0", "v2.0"};

initial begin
    // Find maximum CPU usage
    int max_cpu[] = cpu_usage.max();
    $display("Maximum CPU usage: %p", max_cpu); // {92}

    // For strings, max is lexicographically last
    string max_version[] = versions.max();
    $display("Maximum version: %p", max_version); // {"v3.0"}

    // Find maximum with custom comparison
    typedef struct {
        string product;
        int price;
        int rating; // 1-5 stars
    } item_t;

    item_t products[] = '{
        '{"Laptop", 1200, 4},
        '{"Phone", 800, 5},
        '{"Tablet", 600, 3}
    };

    int max_ratings[] = products.max() with (item.rating);
    $display("Maximum rating: %p", max_ratings); // {5}
end
```

.unique() - Find unique elements

```
int numbers[] = {1, 2, 3, 2, 4, 1, 5, 3, 6};
string colors[] = {"red", "blue", "red", "green", "blue", "yellow"};

initial begin
    // Get unique numbers
    int unique_nums[] = numbers.unique();
    $display("Unique numbers: %p", unique_nums); // {1, 2, 3, 4, 5, 6}

    // Get unique colors
    string unique_colors[] = colors.unique();
    $display("Unique colors: %p", unique_colors); // {"red", "blue", "green", "yellow"}

    // Unique with transformation
    int values[] = {10, 11, 20, 21, 30, 31};
    int unique_tens[] = values.unique() with (item / 10);
    $display("Unique tens digits: %p", unique_tens); // {1, 2, 3}
end
```

Array Ordering Methods

.reverse() - Reverse array order

```
int sequence[] = {1, 2, 3, 4, 5};
string words[] = {"hello", "world", "SystemVerilog"};

initial begin
    $display("Original sequence: %p", sequence);
    sequence.reverse();
    $display("Reversed sequence: %p", sequence); // {5, 4, 3, 2, 1}

    $display("Original words: %p", words);
    words.reverse();
    $display("Reversed words: %p", words); // {"SystemVerilog", "world", "hello"}

    // Reverse twice to restore original
    sequence.reverse();
    $display("Double reversed: %p", sequence); // {1, 2, 3, 4, 5}
end
```

.sort() - Sort array

```
int unsorted[] = {64, 34, 25, 12, 22, 11, 90};
string names[] = {"Charlie", "Alice", "Bob", "David"};
real prices[] = {29.99, 15.50, 42.00, 8.25};

initial begin
    $display("Before sort: %p", unsorted);
    unsorted.sort();
    $display("After sort: %p", unsorted); // {11, 12, 22, 25, 34, 64, 90}

    $display("Names before sort: %p", names);
    names.sort();
    $display("Names after sort: %p", names); // {"Alice", "Bob", "Charlie", "David"}

    $display("Prices before sort: %p", prices);
    prices.sort();
    $display("Prices after sort: %p", prices); // {8.25, 15.50, 29.99, 42.00}
end
```

.rsort() - Reverse sort array

```
int scores[] = {85, 92, 78, 96, 88};
string priority[] = {"low", "high", "medium", "critical"};

initial begin
    $display("Original scores: %p", scores);
    scores.rsort();
    $display("Reverse sorted scores: %p", scores); // {96, 92, 88, 85, 78}

    $display("Original priority: %p", priority);
    priority.rsort();
    $display("Reverse sorted priority: %p", priority); // {"medium", "low", "high", "critical"}

    // Custom reverse sort using with clause
    typedef struct {
        string name;
        int score;
    } student_t;

    student_t students[] = '{
        '{"Alice", 85},
        '{"Bob", 92},
        '{"Charlie", 78}
    };

    students.rsort() with (item.score);
    $display("Students by score (high to low):");
    foreach (students[i]) begin
        $display("  %s: %d", students[i].name, students[i].score);
    end
end
```

.shuffle() - Randomly shuffle array

```
int deck[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
string cards[] = {"A", "K", "Q", "J", "10", "9", "8", "7"};

initial begin
    $display("Original deck: %p", deck);

    // Shuffle multiple times to show randomness
    for (int i = 0; i < 3; i++) begin
        deck.shuffle();
        $display("Shuffle %d: %p", i+1, deck);
    end

    $display("Original cards: %p", cards);
    cards.shuffle();
    $display("Shuffled cards: %p", cards);

    // Shuffle with seed for reproducible results
    int seeded_array[] = {100, 200, 300, 400, 500};
    $display("Before seeded shuffle: %p", seeded_array);

    // Set random seed for reproducible shuffle
    void'($urandom(42)); // Set seed
    seeded_array.shuffle();
    $display("Seeded shuffle result: %p", seeded_array);
end
```

Coverage Constructs

Covergroup

covergroup - Coverage group definition

```
class transaction;
  rand bit [7:0] address;
  rand bit [31:0] data;
  rand bit [1:0] cmd_type; // 0=read, 1=write, 2=rmw

  covergroup cg @(posedge clk);
    // Basic coverage points
    address_cp: coverpoint address {
      bins low = {[0:63]};
      bins mid = {[64:191]};
      bins high = {[192:255]};
    }

    data_cp: coverpoint data {
      bins zero = {32'h0};
      bins small = {[1:1000]};
      bins large = {[1001:$]};
      bins all_ones = {32'hFFFFFFFF};
    }

    cmd_cp: coverpoint cmd_type {
      bins read = {0};
      bins write = {1};
      bins rmw = {2};
      illegal_bins reserved = {3};
    }

    // Cross coverage
    addr_cmd_cross: cross address_cp, cmd_cp {
      ignore_bins no_rmw_low = binsof(cmd_cp.rmw) && binsof(address_cp.low);
    }
  endgroup

  function new();
    cg = new();
  endfunction

  function void sample();
    cg.sample();
  endfunction
endclass

// Usage
```

```
transaction tx = new();
initial begin
  repeat(1000) begin
    tx.randomize();
    tx.sample();
    @(posedge clk);
  end
  $display("Coverage: %f%%", tx.cg.get_coverage());
end
```

endgroup - End coverage group

```
interface bus_if(input logic clk);
    logic [15:0] addr;
    logic [31:0] data;
    logic read, write;
    logic [1:0] burst_len;

    covergroup bus_coverage @(posedge clk iff (read || write));
        addr_range: coverpoint addr {
            bins low_mem = {[16'h0000:16'h3FFF]};
            bins mid_mem = {[16'h4000:16'hBFFF]};
            bins high_mem = {[16'hC000:16'hFFFF]};
        }

        operation: coverpoint {read, write} {
            bins read_op = {2'b10};
            bins write_op = {2'b01};
            illegal_bins both = {2'b11};
            illegal_bins neither = {2'b00};
        }

        burst: coverpoint burst_len {
            bins single = {0};
            bins burst2 = {1};
            bins burst4 = {2};
            bins burst8 = {3};
        }

        // Complex cross coverage
        addr_op_burst: cross addr_range, operation, burst;
    endgroup // End of coverage group definition

    // Instantiate coverage
    bus_coverage cov_inst = new();
endinterface
```

coverpoint - Coverage point

```
module cpu_coverage;
    logic [3:0] opcode;
    logic [4:0] reg_addr;
    logic [31:0] immediate;
    logic branch_taken;

    covergroup instruction_cov @(posedge clk);
        // Basic coverpoint with automatic bins
        opcode_cp: coverpoint opcode;

        // Coverpoint with explicit bins
        register_cp: coverpoint reg_addr {
            bins general_regs = {[0:15]};
            bins special_regs = {[16:31]};
            bins reg_0 = {0}; // Special attention to register 0
        }

        // Coverpoint with conditional sampling
        immediate_cp: coverpoint immediate iff (opcode inside {4'b0100, 4'b0101}) {
            bins small_imm = {[0:255]};
            bins medium_imm = {[256:65535]};
            bins large_imm = {[65536:$]};
        }

        // Expression coverpoint
        branch_cp: coverpoint branch_taken iff (opcode == 4'b1000) {
            bins taken = {1};
            bins not_taken = {0};
        }

        // Transition coverage
        opcode_trans: coverpoint opcode {
            bins arith_to_logic = (4'b0001 => 4'b0010);
            bins logic_to_branch = (4'b0010 => 4'b1000);
            bins any_to_nop = ([0:15] => 4'b0000);
        }
    endgroup

    instruction_cov cov = new();
endmodule
```

cross - Cross coverage

```
class memory_controller_cov;
    logic [1:0] cmd; // 0=read, 1=write, 2=refresh
    logic [2:0] bank; // 0-7
    logic cache_hit;
    logic [31:0] address;

    covergroup mem_cov @(sample_event);
        command: coverpoint cmd {
            bins read = {0};
            bins write = {1};
            bins refresh = {2};
        }

        bank_select: coverpoint bank;

        hit_miss: coverpoint cache_hit {
            bins hit = {1};
            bins miss = {0};
        }

        addr_range: coverpoint address {
            bins low = {[0:32'h0FFF_FFFF]};
            bins high = {[32'h1000_0000:32'hFFFF_FFFF]};
        }

        // Two-way cross coverage
        cmd_bank: cross command, bank_select {
            // Ignore refresh operations to bank 0 (not allowed)
            ignore_bins no_refresh_bank0 = binsof(command.refresh) &&
                                           binsof(bank_select) intersect {0};
        }

        // Three-way cross coverage
        cmd_hit_addr: cross command, hit_miss, addr_range {
            // Only care about read/write operations
            ignore_bins refresh_ops = binsof(command.refresh);

            // Writes to high memory should always miss
            illegal_bins write_high_hit = binsof(command.write) &&
                                          binsof(hit_miss.hit) &&
                                          binsof(addr_range.high);
        }
```

```
    // Four-way cross with selective coverage
    full_cross: cross command, bank_select, hit_miss, addr_range {
      // Too many combinations - only cover specific scenarios
      bins read_scenarios = binsof(command.read);
      bins write_scenarios = binsof(command.write) &&
                             binsof(bank_select) intersect {[0:3]};
    }
  endgroup

  function new();
    mem_cov = new();
  endfunction

  event sample_event;
  function void sample_coverage();
    ->sample_event;
  endfunction
endclass
```

bins - Coverage bins

```
```

```
module coverage_bins_example;
    logic [7:0] data;
    logic [3:0] state;
    logic error;

    covergroup detailed_bins @(posedge clk);
        // Explicit value bins
        data_values: coverpoint data {
            bins zero = {0};
            bins low_vals = {[1:50]};
            bins mid_vals = {[51:200]};
            bins high_vals = {[201:254]};
            bins max_val = {255};
        }

        // Range bins with specific values
        state_coverage: coverpoint state {
            bins idle_reset = {0, 1};
            bins active_states = {[2:10]};
            bins error_states = {[11:14]};
            bins invalid = {15};
        }

        // Wildcard bins
        pattern_bins: coverpoint data {
            wildcard bins pattern_0x = {8'b0xxx_xxxx}; // MSB = 0
            wildcard bins pattern_x0 = {8'bxxxx_xxx0}; // LSB = 0
            wildcard bins pattern_AA = {8'b1010_1010}; // Specific pattern
        }

        // Array bins for multiple ranges
        multi_range: coverpoint data {
            bins special[] = {0, 64, 128, 192, 255}; // Individual bins for each
            bins ranges[] = {[1:63], [65:127], [129:191], [193:254]};
        }

        // Transition bins
        state_transitions: coverpoint state {
            bins reset_to_idle = (0 => 1);
            bins idle_to_active = (1 => [2:10]);
            bins active_loop = ([2:10] => [2:10]);
            bins error_recovery = ([11:14] => 0);
            bins multi_step = (1 => 2 => 3 => 4); // Specific sequence
```

```
    }

    // Conditional bins
    error_data: coverpoint data iff (error) {
      bins error_codes = {[128:255]};
    }

    // Default bin for uncovered values
    data_default: coverpoint data {
      bins covered_vals = {[0:100]};
      bins default_bin = default; // Catches everything else
    }
  endgroup

  detailed_bins cov = new();
endmodule
```

ignore_bins - Bins to ignore

```
class protocol_coverage;
    typedef enum {IDLE, REQ, ACK, DATA, ERROR} state_t;
    state_t current_state, next_state;
    logic [2:0] error_code;
    logic timeout;

    covergroup protocol_cov @(posedge clk);
        current_st: coverpoint current_state;
        next_st: coverpoint next_state;

        // State transition coverage
        state_trans: cross current_st, next_st {
            // Ignore impossible transitions
            ignore_bins no_idle_to_data = binsof(current_st) intersect {IDLE} &&
                                          binsof(next_st) intersect {DATA};
            ignore_bins no_ack_to_req = binsof(current_st) intersect {ACK} &&
                                        binsof(next_st) intersect {REQ};
            ignore_bins no_data_to_idle = binsof(current_st) intersect {DATA} &&
                                          binsof(next_st) intersect {IDLE};
        }

        error_coverage: coverpoint error_code iff (current_state == ERROR) {
            bins timeout_err = {1};
            bins protocol_err = {2};
            bins data_err = {3};
            // Ignore reserved error codes
            ignore_bins reserved = {[4:7]};
        }

        // Combined coverage with ignores
        state_error: cross current_st, error_coverage {
            // Only cover error codes when in ERROR state
            ignore_bins non_error_states = !binsof(current_st) intersect {ERROR};
        }
    endgroup

    function new();
        protocol_cov = new();
    endfunction
endclass
```

illegal_bins - Illegal value bins

```
module safety_coverage;
  logic [2:0] safety_level; // 0-4 valid, 5-7 illegal
  logic [1:0] operation; // 0=read, 1=write, 2=erase, 3=illegal
  logic emergency_stop;
  logic [7:0] temperature; // Safe range: 0-85°C

  covergroup safety_cov @(posedge clk);
    safety_cp: coverpoint safety_level {
      bins safe_levels = {[0:4]};
      illegal_bins dangerous = {[5:7]}; // These should never occur
    }

    operation_cp: coverpoint operation {
      bins normal_ops = {[0:2]};
      illegal_bins invalid_op = {3}; // Illegal operation
    }

    temp_cp: coverpoint temperature {
      bins normal_temp = {[0:85]};
      bins warning_temp = {[86:100]};
      illegal_bins dangerous_temp = {[101:255]}; // Overtemperature
    }

    // Cross coverage with illegal combinations
    safety_operation: cross safety_cp, operation_cp {
      // Level 0 should not allow erase operations
      illegal_bins no_erase_at_level0 = binsof(safety_cp) intersect {0} &&
                                        binsof(operation_cp) intersect {2};
      // Level 1-2 should not allow write operations
      illegal_bins no_write_low_level = binsof(safety_cp) intersect {[1:2]} &&
                                        binsof(operation_cp) intersect {1};
    }

    // Emergency conditions
    emergency_temp: cross emergency_stop, temp_cp {
      // Emergency stop should activate for dangerous temperatures
      illegal_bins no_stop_hot = binsof(emergency_stop) intersect {0} &&
                                 binsof(temp_cp.dangerous_temp);
    }
  endgroup

  safety_cov cov = new();
```

```
    // Check for illegal conditions
    always @(posedge clk) begin
      if (safety_level > 4) begin
          $error("ILLEGAL: Safety level %d detected!", safety_level);
      end
      if (operation == 3) begin
          $error("ILLEGAL: Invalid operation code %d!", operation);
      end
      if (temperature > 100) begin
          $error("ILLEGAL: Dangerous temperature %d°C!", temperature);
      end
    end
endmodule
```