// DEEP COPY - TECHNICAL EXPLANATION WITH STEP-BY-STEP BREAKD
// PART 1: Understanding the address_range copy method

```
class address_range;
  bit [31:0] start_address;
  bit [31:0] end_address  ;

  function new();
    start_address = 10;
    end_address   = 50;
  endfunction
  //copy method
 -------- method 1 --------
  function address_range copy;
    copy = new(); // creating a brand object
    copy.start_address = this.start_address;
    copy.end_address   = this.end_address;
    return copy;
  endfunction
          or
-------- method 2 --------
function address_range copy;
     address_range copy ;
    copy = new(); // creating a brand object
    copy.start_address = this.start_address;
    copy.end_address   = this.end_address;
    return copy;
  endfunction
  function void display(string name);
    $display("%s: start=%0d, end=%0d", name, start_address,
end_address);
  endfunction


endclass
```

Normal function definition vs the above :

normal functions :

function int add(int a = 5);
    return a ; // returning an integer value (int data type)
endfunction

creating an class object using function ;

lets say there exist a class whose class name is parent

conventional =>

parent p1;

p1 = new(); the new() will allocate the memory and initialize it and return the pointer to p1 where p1 is a handle of parent class datatype so we can say the return datatype is of parent;

function based ;

function < return type > name ;

endfunction

function parent p1;

p1 = new();

return p1 ;

endfunction

note :The function name itself is automatically a local variable of the return type. Local variable definition parent p1 ; like int a ; this is implicit

// "address_range" = Return type (returns an address_range object)

When a function has a return type of a class (like `address_range`), it returns the **Handle** (the memory address/pointer) to that object.In SystemVerilog, you never actually "pass" the whole physical object (the data boxes); you only ever pass the **Handle** (the remote control).

```
// "copy"              = Function name
// In SystemVerilog, when you don't declare a return variable ie : address_range copy;
// the function automatically creates one with the SAME NAME as the function!
// So there's an implicit variable: address_range copy;

// LINE 1: copy = new();

// Result:  copy now points to a brand new address_range object
// Memory at this point:
//   copy → [NEW address_range: start=10, end=50] ← From constructor

// LINE 2: copy.start_address = this.start_address;

// "copy.start_address" = The NEW object's start_address
// "this.start_address" = The CURRENT object's start_address (being copied)
//  Result: Copy the value from current object to new object
// "this" = The object that called this method
// Example: If ar1.copy() is called, "this" = ar1
copy.start_address = this.start_address;

// LINE 3: copy.end_address = this.end_address;
// Same as above - copy the end_address value
copy.end_address = this.end_address;

// LINE 4: return copy;
// Return the newly created object
// This object is now a COMPLETE COPY of the original (returns the handle )
```

```
    return copy;
  endfunction

  // Display method for demonstration
  function void display(string name);
    $display("%s: start=%0d, end=%0d", name, start_address, end_address);
  endfunction
endclass

//
================================================================================
=====
// DEMONSTRATION: How address_range.copy() works
//
================================================================================
=====

module address_range_copy_demo;
  initial begin
    address_range  ar_original, ar_copied;

    $display("\n========== address_range.copy() EXPLAINED ==========\n");

    // STEP 1: Create original object
    ar_original = new();
    ar_original.start_address = 100;
    ar_original.end_address = 200;
    $display("STEP 1: Created original");
    ar_original.display("  ar_original");

    // STEP 2: Call copy method
    $display("\nSTEP 2: Calling ar_copied = ar_original.copy()");
    $display("  Inside copy():");
    $display("    - 'this' refers to ar_original");
    $display("    - 'copy = new()' creates NEW object");
    $display("    - 'copy.start = this.start' copies value 100");
    $display("    - 'copy.end = this.end' copies value 200");
    $display("    - 'return copy' returns the new object");

    ar_copied = ar_original.copy();

    // STEP 3: Verify they're independent
    $display("\nSTEP 3: Both objects now exist");
    ar_original.display("  ar_original");
    ar_copied.display("  ar_copied");

    $display("\nSTEP 4: Modify ar_copied");
    ar_copied.start_address = 999;
    ar_original.display("  ar_original");
    ar_copied.display("  ar_copied");
    $display("  ✅ They are INDEPENDENT!");
  end
```

```
endmodule

//
================================================================================
// PART 2: Understanding the packet copy method (DEEP COPY)
//
================================================================================

class packet;
  bit [31:0] addr;
  bit [31:0] data;
  address_range ar;  // This is an OBJECT handle!

  function new();
    addr = 32'h10;
    data = 32'hFF;
    ar = new();  // Create address_range object
  endfunction

  function void display(string name);
    $display("%s:", name);
    $display("  addr = 0x%0h", addr);
    $display("  data = 0x%0h", data);
    $display("  ar.start_address = %0d", ar.start_address);
    $display("  ar.end_address = %0d", ar.end_address);
  endfunction

  // DEEP COPY METHOD - Technical breakdown
  function packet copy();
    // SYNTAX: function <return_type> <function_name>;
    // Implicit variable created: packet copy;

    // LINE 1: copy = new();
    //
--------------------------------------------------------------------------------
    // Creates a NEW packet object
    // This new object has its OWN ar handle created by constructor
    //
    // Memory at this point:
    //   Current object (this):
    //     this.addr = 0x10
    //     this.data = 0xFF
    //     this.ar ——→ [address_range A: start=10, end=50]
    //
    //   New object (copy):
    //     copy.addr = 0x10 (from constructor)
    //     copy.data = 0xFF (from constructor)
    //     copy.ar ——→ [address_range B: start=10, end=50] ← NEW object!
    //
```

```
    copy = new();

    // LINE 2: copy.addr = this.addr;
    //
-------------------------------------------------------------------------------
-------

    // Copy primitive value
    // "this.addr" = Current packet's addr
    // "copy.addr" = New packet's addr
    copy.addr = this.addr;

    // LINE 3: copy.data = this.data;
    //
-------------------------------------------------------------------------------
-------

    // Copy primitive value
    copy.data = this.data;

    // LINE 4: copy.ar = ar.copy();  ← THIS IS THE KEY LINE!
    //
-------------------------------------------------------------------------------
-------

    // TECHNICAL BREAKDOWN:
    //
    // "ar"        = this.ar (implied) - Current packet's address_range object
    // "ar.copy()" = Call the copy() method of the address_range class
    //               This creates a NEW address_range object
    // "copy.ar"   = The new packet's address_range handle
    // "="         = Assign the NEW address_range to the new packet
    //
    // WHAT HAPPENS:
    // 1. ar.copy() executes:
    //    - Inside address_range, creates new address_range object
    //    - Copies start_address and end_address values
    //    - Returns the new address_range object
    // 2. copy.ar = <returned object>
    //    - New packet's ar handle now points to this NEW object
    //
    // RESULT: copy.ar and this.ar point to DIFFERENT objects!
    //
    copy.ar = ar.copy();  // This line makes it DEEP copy!

    // LINE 5: return copy;
    //
-------------------------------------------------------------------------------
-------

    // Return the complete deep copy
    return copy;
  endfunction
endclass
```

```
//
================================================================================
=====
// DEMONSTRATION: Deep Copy in Action
//
================================================================================
=====

module deep_copy_demo;
  initial begin
    packet pkt1, pkt2;

    $display("\n========== DEEP COPY DEMONSTRATION ==========\n");

    // STEP 1: Create original packet
    pkt1 = new();
    pkt1.addr = 32'h100;
    pkt1.data = 32'hAAA;
    pkt1.ar.start_address = 1000;
    pkt1.ar.end_address = 2000;

    $display("STEP 1: Created original packet");
    pkt1.display("  pkt1");

    // STEP 2: Deep copy
    $display("\nSTEP 2: Creating deep copy - pkt2 = pkt1.copy()");
    $display("\nWhat happens inside copy():");
    $display("  1. copy = new()           → Create new packet object");
    $display("  2. copy.addr = this.addr  → Copy addr value (0x100)");
    $display("  3. copy.data = this.data  → Copy data value (0xAAA)");
    $display("  4. copy.ar = ar.copy()    → Create NEW address_range!");
    $display("     - Calls address_range.copy()");
    $display("     - Creates separate address_range object");
    $display("     - Copies start=1000, end=2000 to new object");
    $display("  5. return copy            → Return the deep copy");

    pkt2 = pkt1.copy();

    $display("\nSTEP 3: Both packets now exist");
    pkt1.display("  pkt1");
    pkt2.display("  pkt2");

    // STEP 4: Modify pkt2's address_range
    $display("\nSTEP 4: Modifying pkt2.ar.start_address to 9999");
    pkt2.ar.start_address = 9999;

    pkt1.display("  pkt1");
    pkt2.display("  pkt2");

    $display("\n✅ RESULT: pkt1.ar unchanged! Deep copy successful!");
  end
endmodule
```

```
//
================================================================================
=====
// COMPARISON: Shallow vs Deep Copy Side by Side
//
================================================================================
=====

class packet_shallow;
  bit [31:0] addr;
  address_range ar;

  function new();
    addr = 32'h10;
    ar = new();
  endfunction

  // SHALLOW COPY - Just copies handle
  function packet_shallow copy();
    copy = new();
    copy.addr = this.addr;
    copy.ar = this.ar;  // ⚠ SHALLOW: Just copies handle!
    return copy;
  endfunction
endclass

class packet_deep;
  bit [31:0] addr;
  address_range ar;

  function new();
    addr = 32'h10;
    ar = new();
  endfunction

  // DEEP COPY - Creates new object
  function packet_deep copy();
    copy = new();
    copy.addr = this.addr;
    copy.ar = ar.copy();  // ✅ DEEP: Creates new address_range!
    return copy;
  endfunction
endclass

module comparison;
  initial begin
    packet_shallow s1, s2;
    packet_deep d1, d2;

    $display("\n========== SHALLOW vs DEEP COMPARISON ==========\n");
```

```
    // Shallow copy test
    s1 = new();
    s1.ar.start_address = 100;
    s2 = s1.copy();

    $display("SHALLOW COPY:");
    $display("  Before: s1.ar.start = %0d, s2.ar.start = %0d",
            s1.ar.start_address, s2.ar.start_address);

    s2.ar.start_address = 999;

    $display("  After changing s2.ar.start to 999:");
    $display("  s1.ar.start = %0d ❌ (ALSO CHANGED!)", s1.ar.start_address);
    $display("  s2.ar.start = %0d", s2.ar.start_address);
    $display("  → Both share SAME address_range object\n");

    // Deep copy test
    d1 = new();
    d1.ar.start_address = 100;
    d2 = d1.copy();

    $display("DEEP COPY:");
    $display("  Before: d1.ar.start = %0d, d2.ar.start = %0d",
            d1.ar.start_address, d2.ar.start_address);

    d2.ar.start_address = 999;

    $display("  After changing d2.ar.start to 999:");
    $display("  d1.ar.start = %0d ✅ (UNCHANGED!)", d1.ar.start_address);
    $display("  d2.ar.start = %0d", d2.ar.start_address);
    $display("  → Each has its OWN address_range object\n");
  end
endmodule

//
================================================================================
=====
// KEY CONCEPTS SUMMARY
//
================================================================================
=====

/*
================================================================================
================================
KEY SYNTAX EXPLANATIONS:
================================================================================
================================

1. function address_range copy;
   ──────────────────────────────
   - "address_range" = Return type
```

- "copy" = Function name
- SystemVerilog creates implicit variable: address_range copy;
- This variable has the SAME name as the function!

2. copy = new();
   ————————————

- "copy" = The implicit return variable
- "new()" = Create a NEW object (calls constructor)
- Result: copy points to a brand new object in memory

3. copy.start_address = this.start_address;
   ——

- "this" = Current object (the one that called this method)
- "this.start_address" = Value from current object
- "copy.start_address" = Value in new object
- Action: COPY the value from old to new

4. copy.ar = ar.copy();
   ———————————————————

- "ar" = this.ar (implicit) - current packet's address_range
- "ar.copy()" = Call the copy() method of address_range class
- This creates a NEW address_range object
- "copy.ar" = New packet's address_range handle
- Result: New packet gets its OWN address_range object

5. return copy;
   ———————————

- Return the newly created object
- This object is now available to the caller

═══

MEMORY VISUALIZATION:
═══

SHALLOW COPY:
————————————

pkt1.ar ———┐
           ├——→ [Same address_range object]
pkt2.ar ———┘

Code: copy.ar = this.ar; ← Just copies the handle

DEEP COPY:
—————————

pkt1.ar ———→ [address_range object 1]

pkt2.ar ———→ [address_range object 2] ← NEW object!

Code: copy.ar = ar.copy(); ← Creates new object

THE CRITICAL DIFFERENCE:

SHALLOW: copy.ar = this.ar; ← Copies HANDLE (both point to same)
DEEP: copy.ar = ar.copy(); ← Creates NEW OBJECT (independent)

The ".copy()" method call is what makes it DEEP!
It recursively copies nested objects.