Verilog HDL:
Vectors

Pravin Zode

Outline

- Introduction to vectors
- Packed and Unpacked Arrays
- Bitwise and Logical Operator
- Concatenation Operator
- Replication Operator
- Examples

Vectors

- Vectors group multiple related signals under one name
- Declared with dimensions before the name → wire [7:0] w;
- Equivalent to several separate single-bit wires
- Part-select allows access to individual bits → w[3]
- Used to simplify circuit connections with multi-bit buses
- Example:
  - `wire [99:0] my_vector;` // Declare a 100-element vector
  - `assign out = my_vector[10];` // Part-select one bit out of the vector

Declaring Vectors

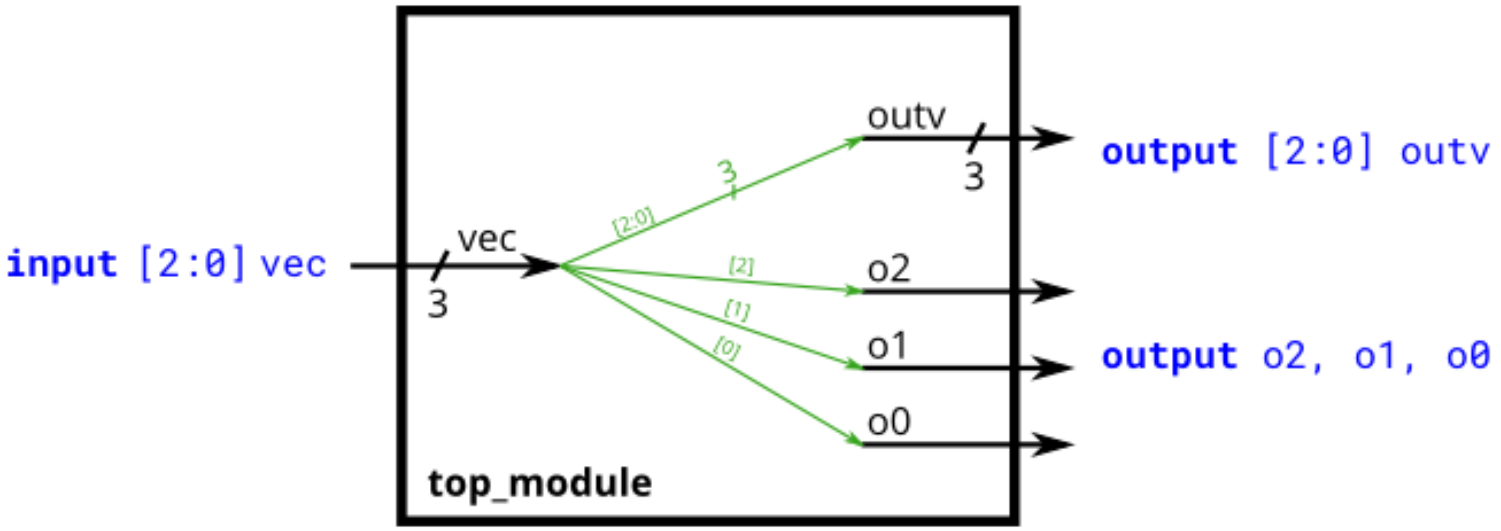
- Vectors must be declared:
  - type [upper:lower] vector_name;
- type specifies the datatype of the vector. This is usually wire or reg.
- Input or output ports can also include their type in the declaration (e.g., input wire a;, output reg y;)
- Endianness defines the bit order in a vector
  - **Little-endian**: LSB has the lower index → [3:0]
  - **Big-endian**: LSB has the higher index → [0:3]

Examples

- `wire [7:0] w;` // 8-bit wire
- `reg [4:1] x;` // 4-bit reg
- `output reg [0:0] y;` // 1-bit reg that is also an output port (this is still a vector)
- `input wire [3:-2] z;` // 6-bit wire input (negative ranges are allowed)
- `output [3:0] a;` // 4-bit output wire. Type is 'wire' unless specified otherwise.
- `wire [0:7] b;` // 8-bit wire where b[0] is the most-significant bit.

Exercise-01

- Build a circuit that has one 3-bit input, then outputs the same vector, and also splits it into three separate 1-bit outputs. Connect output o0 to the input vector's position 0, o1 to position 1, etc…



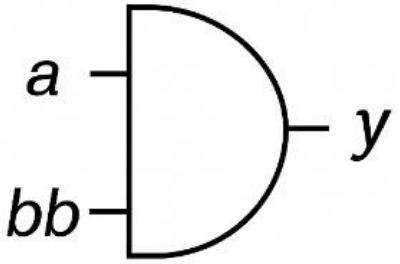
Implicit nets

- Created automatically if a signal is undeclared
- Can arise from an assign statement or undeclared module port
- Always treated as 1-bit wires
- Leads to bugs if a vector was intended
- Prevent implicit nets using: `` `default_nettype none ``

Example: Implicit nets

- Without default_nettype none (bug due to implicit net)

```
module implicit_example(input a, input b, output y);
  assign y = a & bb;   // Typo: "bb" instead of "b"
Endmodule
// Here, bb is not declared, so Verilog creates an
implicit 1-bit wire.
```



- With default_nettype none (error detected)

```
`default_nettype none
module implicit_example(input a, input b, output y);
  assign y = a & bb;   // Compiler ERROR: "bb" undeclared
endmodule
`default_nettype wire
// compiler throws an error instead of silently creating a
wrong wire.
```

Example: Implicit nets

- Without default_nettype none (bug due to implicit net)

```
module implicit_example(input a, input b, output y);
  assign y = a & bb;   // Typo: "bb" instead of "b"
Endmodule
// Here, bb is not declared, so Verilog creates an
implicit 1-bit wire.
```

- With default_nettype none (error detected)

```
`default_nettype none
module implicit_example(input a, input b, output y);
  assign y = a & bb;   // Compiler ERROR: "bb" undeclared
endmodule
`default_nettype wire
// compiler throws an error instead of silently creating a
wrong wire.
```

Example: Implicit nets

- Disabling creation of implicit nets can be done using the `` `default_nettype none `` directive.

```
wire [2:0] a, c;   // Two vectors
assign a = 3'b101;  // a = 101
assign b = a;       // b =   1  implicitly-created
wire
assign c = b;       // c = 001  <-- bug
my_module i1 (d,e); // d and e are implicitly one-bit
wide if not declared.
                    // This could be a bug if the port
was intended to be a vector.
```

Unpacked vs. Packed Arrays

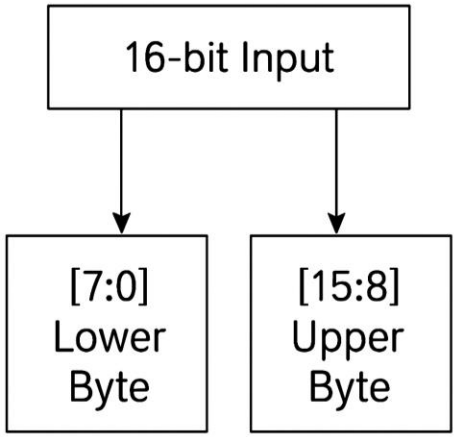
- **Packed arrays:**
  - Declared with indices before the name
  - Bits are stored together as a vector
  - Example: reg [7:0] data; // 8-bit packed vector
- **Unpacked arrays:**
  - Declared with indices after the name
  - Represent multiple elements (like memory)
  - Example:reg [7:0] mem [255:0]; // 256 elements of 8-bit vectors
  - reg mem2 [28:0]; // 29 elements of 1-bit regs
- **Usage:** Packed → represent buses / vectors

  Unpacked → represent memories / arrays

Exercise

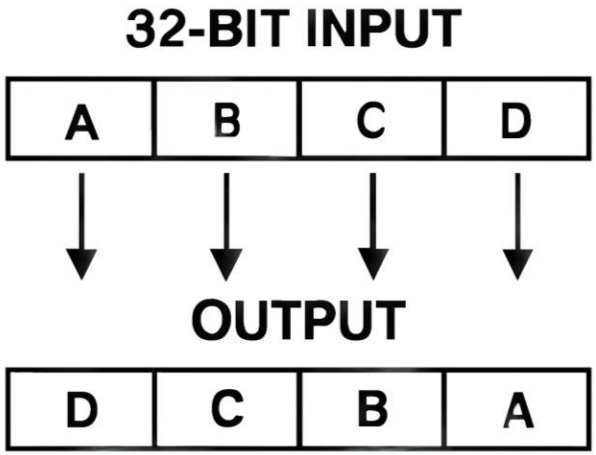
- Build a combinational circuit that splits an input half-word (16 bits, [15:0]) into lower [7:0] and upper [15:8] bytes.

```
`default_nettype none     // Disable implicit nets.
                          Reduces some types of bugs.
module top_module(
    input wire [15:0] in,
    output wire [7:0] out_hi,
    output wire [7:0] out_lo );
Write your code here
endmodule
```



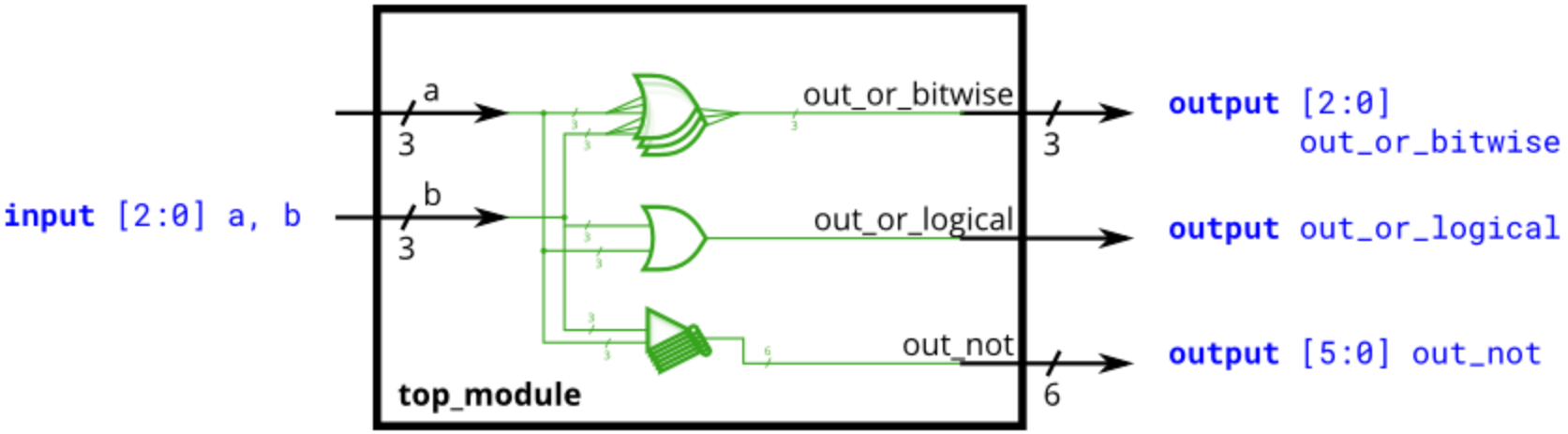
Exercise

- A 32-bit vector can be viewed as containing 4 bytes (bits [31:24], [23:16], etc.). Build a circuit that will reverse the byte ordering of the 4-byte word.
- AAAAAAAABBBBBBBBCCCCCCCCDDDDDDDD => DDDDDDDDCCCCCCCCBBBBBBBBAAAAAAAA



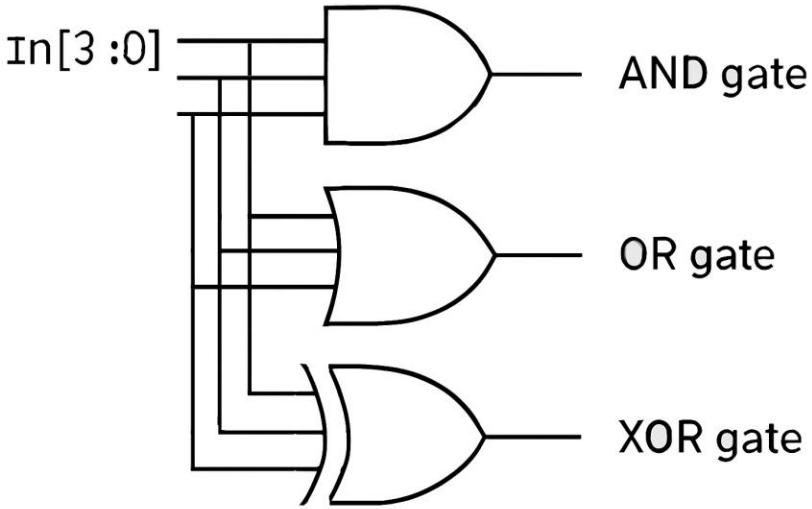
Exercise : Bitwise vs. Logical Operators

- Build a circuit that has two 3-bit inputs that computes the bitwise-OR of the two vectors, the logical-OR of the two vectors, and the inverse (NOT) of both vectors. Place the inverse of b in the upper half of out_not (i.e., bits [5:3]), and the inverse of a in the lower half.



Exercise : Multiple Input Gates

- Build a combinational circuit with four inputs, in[3:0].
- There are 3 outputs:
  - **out_and**: output of a 4-input AND gate.
  - **out_or**: output of a 4-input OR gate.
  - **out_xor:** output of a 4-input XOR gate.



Part Selection and Concatenation

- **Part Selection** : Used to select portions of a vector.
- **Concatenation Operator { } :** Creates larger vectors by joining smaller ones
- Examples:
  - {3'b111, 3'b000} → 6'b111000
  - {1'b1, 1'b0, 3'b101} → 5'b10101
  - {4'ha, 4'd10} → 8'b10101010
  - Rule: All components must have defined width.{1, 2, 3} ❌

    Illegal → Error: unsized constants not allowed

Example : Concatenation

```
input  [15:0] in;
output [23:0] out;

assign {out[7:0], out[15:8]} = in;        // Swap two bytes
assign out[15:0] = {in[7:0], in[15:8]};// Same as above
assign out = {in[7:0], in[15:8]};        // Extended to 24-
                                          bit, out[23:16] = 0
```

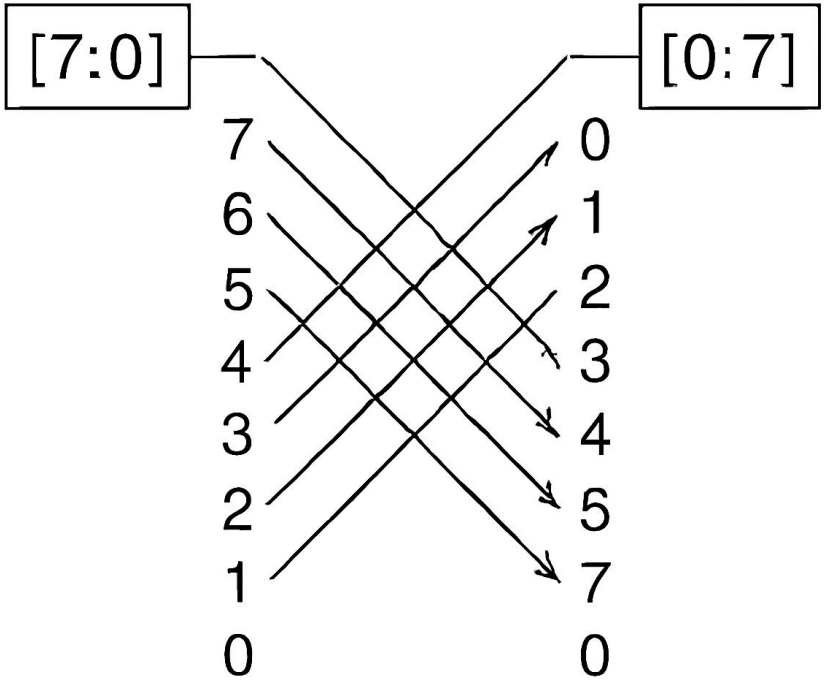
Exercise :

- Inputs: six 5-bit vectors → a, b, c, d, e, f → 30 bits
- Outputs: four 8-bit vectors → w, x, y, z → 32 bits
- Requirement: Concatenate all inputs + two 1 bits

```
assign {w, x, y, z} = {a, b, c, d, e, f, 2'b11};
```

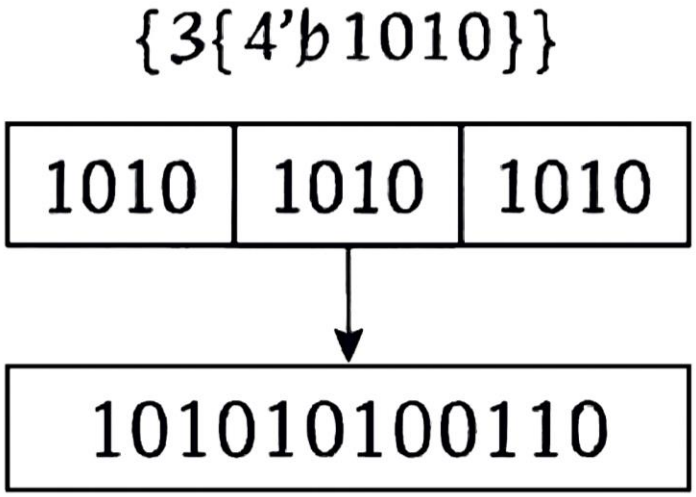
Exercise

- Given an 8-bit input vector [7:0], reverse its bit ordering



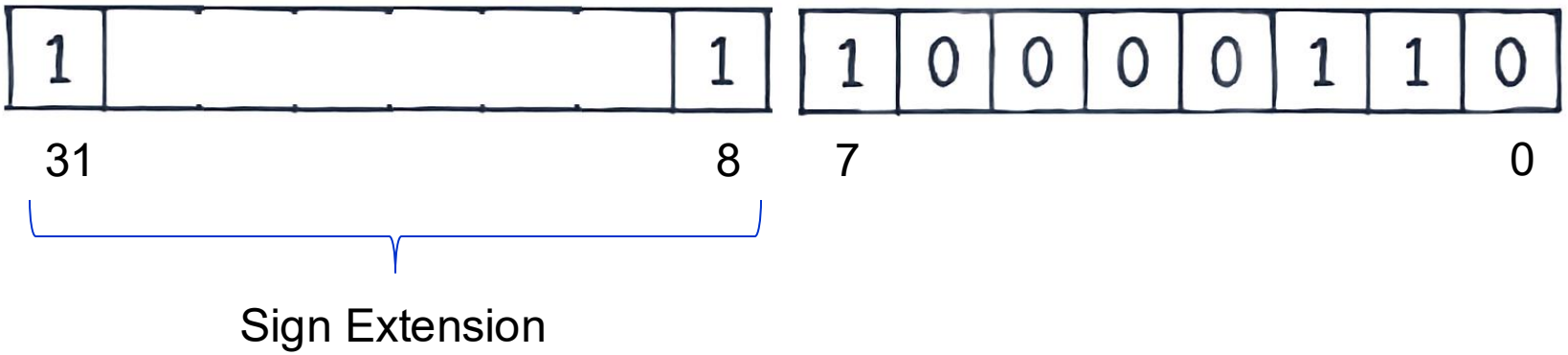
Replication Operator

- Used to repeat a vector multiple times
- Shorthand to avoid tedious concatenation
- Requires two braces: `{ num { vector } }`
- `{3{4'b1010}}  // This results in 12 bits: 1010_1010_1010`



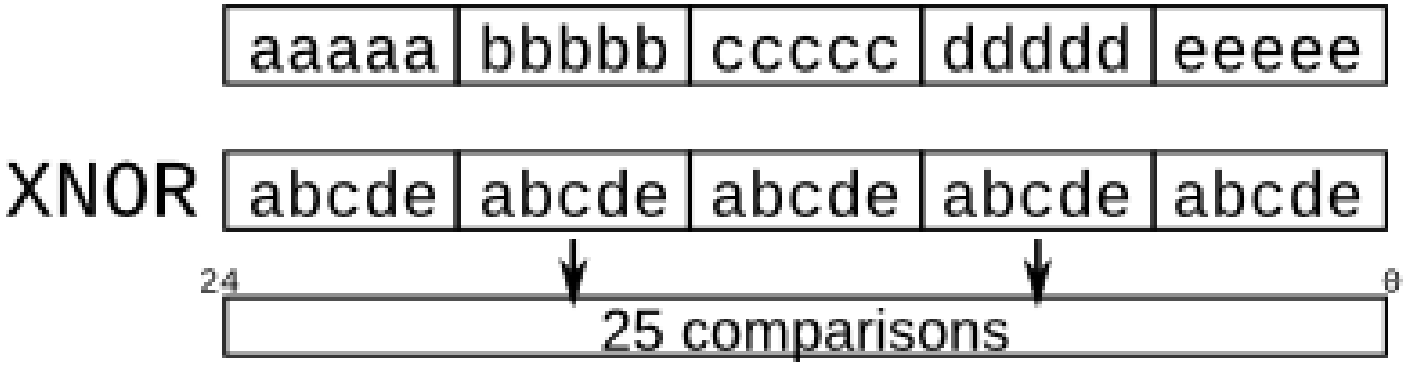
Exercise-1: Replication Operator

- Build a circuit that sign-extends an 8-bit number to 32 bits. This requires a concatenation of 24 copies of the sign bit (i.e., replicate bit[7] 24 times) followed by the 8-bit number itself.



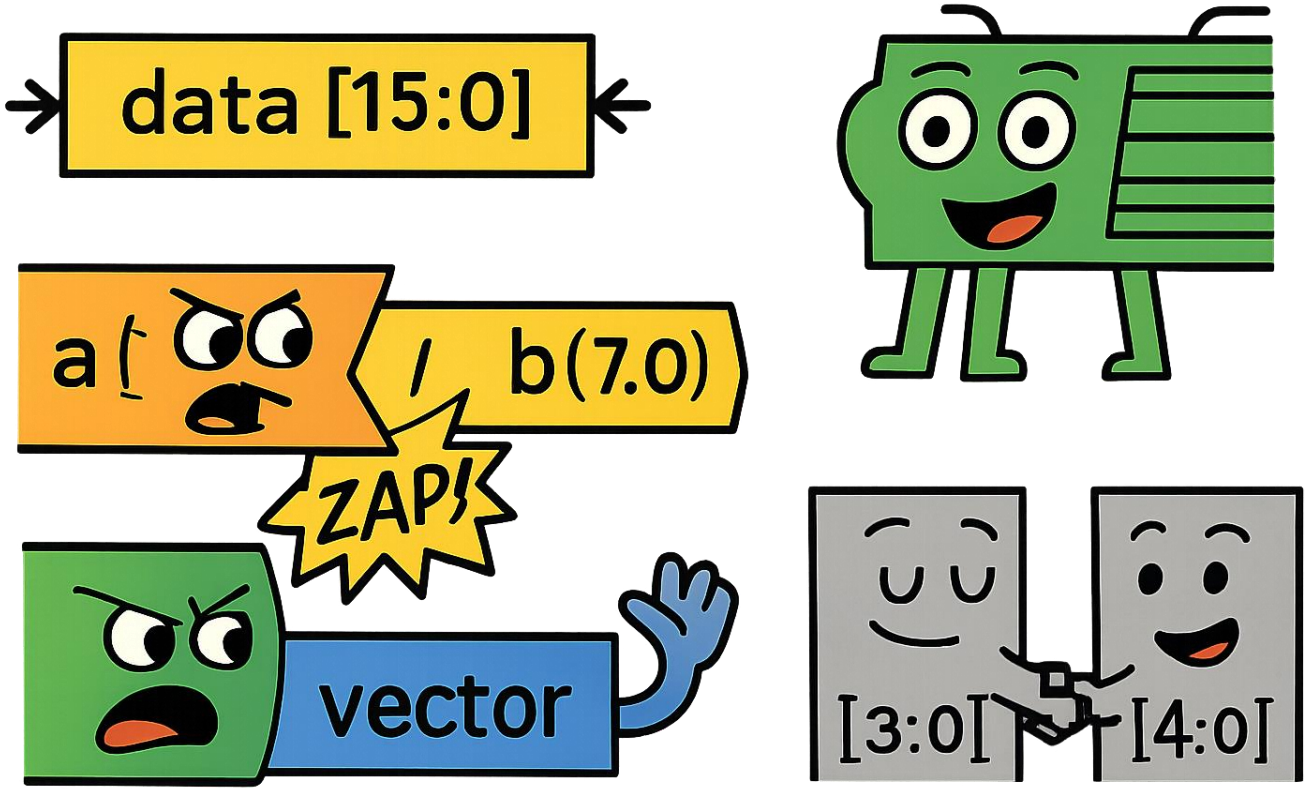
Exercise-2 : Replication Operator

- Given five 1-bit signals (a, b, c, d, and e), compute all 25 pairwise one-bit comparisons in the 25-bit output vector. The output should be 1 if the two bits being compared are equal



Conclusion

- **Vectors:** Represent multi-bit signals efficiently
- **Packed & Unpacked Arrays:** Organize data in structured formats
- **Concatenation Operator {}:** Joins smaller vectors into larger ones
- **Replication Operator {num{vector}}:** Repeats vectors compactly
- **Bitwise Operators (& | ^ ~):** Operate on signals bit by bit
- **Logical Operators (&& || !):** Evaluate overall true/false conditions.



Thank you !

Happy Learning