Mastering SystemVerilog Enums

~hema kandregula

What is an enum in SystemVerilog, and why is it used?

Enums in SystemVerilog define a set of named values, making code more readable and maintainable. They help represent related named constants, typically states or modes in digital design.

Enums in SystemVerilog are used to:

1. **Improve readability** by giving meaningful names to values.
2. **Simplify debugging** with named values instead of raw numbers.
3. **Automate value assignment**, reducing manual errors.
4. **Define FSM states clearly**, making transitions easier to manage and understand.

NOTE:The default type for an enum is int (32-bit signed integer). You can override this by specifying a type (e.g., enum logic [1:0] {IDLE, READ, WRITE}state;)

Basic Enum Declaration and Usage

```
module enum_example;
 enum { IDLE, READ, WRITE, DONE} state ;
initial
begin
state = IDLE ;   // Assign enum value
$display("Initial state: %s", state.name());
state = READ;
$display("Current state: %s", state.name());
end   //.name() method returns the string representation of the current state
endmodule
```

Enum with Explicit Values and Type Specification

- We can explicitly assign integer values to enum members and specify a custom type.

```
module enum_with_values;
 enum logic [2:0] {IDLE = 3, READ , WRITE = 6,DONE }state;
 initial begin  // logic [2:0] specifies the enum type, restricting it to 3 bits.
  state =DONE;
  $display("Assigned State: %s  \n  value :%0d", state.name(), state);  // Assigned state : DONE  Value=7
 end
endmodule
```

Using Enums in Case Statements

- Enums improve readability in case statements by allowing you to use meaningful names.

```
module enum_case_example;
 enum {IDLE, READ, WRITE, DONE} state;
 initial begin
  state = READ;  // Case statement with enum values
  case (state)
   IDLE:  $display("State is IDLE");
   READ:  $display("State is READ");   //[O/P]: State is READ
   WRITE: $display("State is WRITE");
   DONE:  $display("State is DONE");
   default: $display("Unknown state");
  endcase
 end
endmodule
```

Checking for Valid Enum Values

- Enums don’t inherently restrict values, so you might get unexpected results if the variable is assigned an invalid value. You can use the inside to check validity.
- The inside operator is used to verify the enum values fall within the expected range

```
module enum_valid_check;
 enum {IDLE, READ, WRITE, DONE} state;
 initial begin
  state = READ;
  if (state inside {IDLE, READ, WRITE, DONE})
begin
    $display("State is valid: %s", state.name());    // [O/P] :State is valid: READ
  end
else
begin
    $error("Invalid state detected!");
  end
 end
endmodule
```

Iterating Over Enum Values using methods

Methods available for enums in SystemVerilog:

- **name()**: Returns the string name of the current enum value.
- **first()**: Sets the enum variable to the first value in the enum.
- **last()**: Sets the enum variable to the last value in the enum.
- **next()**: Sets the enum variable to the next value in sequence, or wraps around to first() if at the last value.
- **prev()**: Sets the enum variable to the previous value in sequence, or wraps around to last() if at the first value.
- **num()**: Returns the total number of values in the enum.

```
module enum_iteration_example;
 enum {IDLE, READ, WRITE, DONE} state;
 initial
 begin
  for (state = state.first(); state <= state.last(); state = state.next())
    begin
    $display("Iterating state: %s", state.name());
  end
 end
endmodule
```

Enum in Functions

- Enums can be passed as arguments to functions, ensuring type safety and readability.

```
module enum_function_example;
typedef enum {IDLE, READ, WRITE, DONE} state_enum_t;
 state_enum_t state;
 function void process_state(state_enum_t s);   // Function to process the state
   case (s)
     IDLE:  $display("Processing IDLE");
     READ:  $display("Processing READ");
     WRITE: $display("Processing WRITE");   // [O/P] :Processing WRITE
     DONE:  $display("Processing DONE");
     default: $display("Unknown state");
   endcase
 endfunction
 initial begin
   state = WRITE;
   process_state(state);
 end
endmodule
```