



The SystemVerilog Reference Guide

enum

Declares a user-defined enumerated type having a set of explicitly named values.

Syntax

```
enum data_type {item1, item2...} var
```

Where `data_type` is optional and defaults to `int`. `var` is a variable name.

Rules and Examples

```
enum {idle, start, pause, done} mstate;
```

Declares a variable `mstate` which can only take values `idle`, `start`, `pause` or `done`. By default, `mstate` is of type `int`, and the first item in the enumerated list, `idle`, is represented as 0, `start` as 1, `pause` as 2 and `done` as 3.

An enum type declaration is only visible in the scope in which it is declared, and the enumeration values must be unique in that scope.

A range notation, using integer constants, can be used for name sequences in the enumerated list.

```
enum {S[2]} seq;           // = S0, S1, S2  
enum {RST, P[4:5]} rng; // = RST, P4, P5
```

Control of Encoding

Explicit values can be defined in the enumerated list to allow one-hot, gray encoding etc.:-

```
enum {idle = 1, start = 2,  
      pause = 4, done = 8} mstate;
```

Explicit and default encoding can be mixed. Names without an explicit encoding increment from the previous name:-

```
enum {S0 = 2, S1, S2, S3 = 8, S4} states;
```

Here `S1` has the encoding 3, `S2 = 4`, `S4 = 9`. It is an *error* if explicit and default encodings overlap, for example if `S3 = 3`, `S2` and `S3` would have the same encoding, and this would be a compilation error.

Explicit Data Types

An explicit data type can also be defined. Here enum variable `stbit` is a bit vector of length 3:-

```
enum bit [2:0] {S[2]} stbit;
```

With explicit data types, any explicit encodings *must* match the length of the data type:-

```
enum bit [2:0] {S0 = 3'b001,  
               S1 = 3'b010, S2 = 3'b100} stbit;
```

Typing in Assignment

SystemVerilog enumerated types are strongly typed. An enum variable can only be directly assigned:-

- A value from the its enumerated list
- A variable of the same enumerated type

```
mstate = idle;  
mstate = next_mstate;  
mstate = 2;           // error
```

Type casting must be used to assign other types to the enumerated variable.

A typedef declaration can be used to associate a name with the enumerated type. The type name can then be used for type casting:-

```
typedef enum {idle, start,  
             pause, done} state_t;  
state_t mstate, next_mstate;  
mstate = state_t'(2); // type cast
```

Typing in Expression

When used in an expression, an enumerated variable becomes an object of its data type.

```
int aint;  
// mstate is int in expression  
aint = mstate + 1;
```

Enum Methods

Enumerate values can be accessed via methods:-

Method	Description
<code>first()</code>	Returns first value of enumeration
<code>last()</code>	Returns last value of enumeration
<code>next(N)</code>	Returns Nth next value, wrapping to beginning if need be. N defaults to 1.
<code>prev(N)</code>	Returns Nth previous value, wrapping to end if need be. N defaults to 1.
<code>num()</code>	Returns number of values in the enumeration
<code>name()</code>	Returns string representation of given enumeration value

Methods can be cascaded.

```
typedef enum {idle, start, done} state_t;
state_t st = st.last();
initial
  forever begin
    $display ("%s = %d", st.name(), st);
    if (st == st.first())
      break
    st = st.prev();
  end
```

This example produces the following output:-

```
done = 2
start = 1
idle = 0
```

```
typedef enum {idle, start, done} state_t;
state_t st = start;
initial
  st = st.next(2); // start + 2 = idle
```

`next` and `prev` methods wrap around the enumerate type declaration order.

Warning: use of `first`, `last`, `prev` and `next` methods creates code which is dependent on the declaration order of the enumerated type values.

See Also

`int`, `typedef`