**Structural Data Types**

* Wire : These nets acts like wires in a circuit (init state is z).
* Reg : They hold their values until another value is put on them (init state is x) (cannot be driven by assign).
* Both reg and wire types are declared within a module but outside initial and always block

**Behavioral Data Types**

* Real rea\_var\_name; (64-bit, e.g 14.74, 39e8)
  + real float\_v ; // a variable to store real value
* Integer int\_var\_name; (32-bit, e.g integer a[0:64] // arry of 65 int vals)
* Time time\_var\_name; (64-bit) it hold simulation time
* Parameter size = 16; // These are constants. Cannot be modified at runtime, only compilation time.
* Logic; replaces reg in modern sysveri.

| **Use Case** | **Verilog 2001** | **SystemVerilog** |
| --- | --- | --- |
| Procedural variable | reg | logic |
| Combinational/clocked logic | reg, wire (confusing) | logic |
| 2-state variable (fast sim) | not supported easily | bit (0/1 only) |
| Drive with assign | wire only | logic (if single driver) |

**Void Data Types**

* Represent non-existent data. E.g return of functions. Void’(function\_call())

**String**

* Is variable size
  + string s1 = "Hellow World";
  + string s2 = {"Hi"," ",s1};
  + bit [31:0] b = 128;
  + string s3 = b; // sets 128 to s3

bit [31:0] b = 128;

* bit [31:0] : 32-bit wide vector, indexed from 31 down to 0
* each bit can be 0 or 1 only.
* similar to logic or reg but excludes x, z
* Useful for modeling clean, deterministic logic and improving simulation speed.

**Event**

* a handle to a synchronization object that can be passed around to routines

**User Defined**

* User can define a new type using typedef
  + Interger\_v1 var1

**Enum Data Type**

* Define a set of named values, whose actual values is defaulted from zero
  + enum { red=0, green, blue=4, yellow, white=10, black } Colors;

Defining new data types using enum

* The same type can be used in may places.
  + typedef enum {GOOD, BAD} pkt\_type;
  + pkt\_type pkt\_a; // named type

enum methods

* first() returns the value of the first member of the enumeration
* last() returns the value of the last member of the enumeration
* next() returns the value of next member of the enumeration
* next(N) returns the value of next Nth member of the enumeration
* prev() returns the value of previous member of the enumeration
* prev(N) returns the value of previous Nth member of the enumeration
* num() returns the number of elements in the given enumeration
* name() returns the string representation of the given enumeration valuev
* initial begin … end procedural block tells the simulator to execute the enclosed statement once at time 0
* for() begin ---- end

1. Course: <https://verificationguide.com/systemverilog/systemverilog-tutorial/>
2. Course: <https://www.chipverify.com/systemverilog/systemverilog-data-types-logic-bit#google_vignette>
3. Playground: <https://www.edaplayground.com/x/dL7x>

* SystemVerilog: An extension on Verilog that allows engineers to perform many verification tests using complex test benches.
* It is better than Verilog because it can perform constrained random stimuli, use OOP features in TBs, functional coverage, etc.
* If the entire design flow has to be repeated, then its called a respin of the chip.

**Notes on Vivado SystemVerilog Limitations**

| **Feature** | **Synthesis** |  | **Simulation** |
| --- | --- | --- | --- |
| logic | ✅ |  | ✅ |
| always\_comb, always\_ff, always\_latch | ✅ |  | ✅ |
| interface, modport, assert | ❌ |  | ✅ |
| initial blocks | ❌ |  | ✅ |
| Dynamic constructs (foreach, classes) | ❌ |  | ✅ |

**Class Data Types**

* Class nameClass; … endclass

class packet;

// Properties

bit [31:0] address;

bit [31:0] data ;

// Method

function new();

$display("Inside new Function of packet");

endfunction

endclass : packet