SystemVerilog

This is an extension of Verilog. It is a combination of hardware description language and hardware verification language. The feature-set of SystemVerilog can be divided into two distinct roles:

* SystemVerilog for [RTL](https://en.wikipedia.org/wiki/Register-transfer_level) design is an extension of [Verilog-2005](https://en.wikipedia.org/wiki/Verilog); all features of that language are available in SystemVerilog.
* SystemVerilog for verification uses extensive [object-oriented programming](https://en.wikipedia.org/wiki/Object-oriented_programming) techniques and is more closely related to [Java](https://en.wikipedia.org/wiki/C_(programming_language)) than Verilog.

The things I have studied under System Verilog are:

* A brief introduction of Verilog
  + Introduction to the various steps in Verilog
    - Specification
    - High Level Design
    - Low Level Design
    - RTL (Register-Transfer Level) coding
    - Verification
    - Synthesis
  + Definition of modules, ports(input, output, bi-directional), drivers, data types, operators
  + Control statements (if else, case, while, for loop, repeat)
  + Variable assignment for combinational elements (assign, always) sequential elements (always statement) and testbenches (initial statement)
  + Definition of always and assign statements
  + Tasks and functions with their minor differences
    - Tasks can have delays, but functions cannot
    - Functions can return a value, but tasks cannot
  + Testbenches
* Literals – Integer and Logic, Real, Time, String, Arrays, Structures
* Data Types –Integer data types (2 state and 4 state), null, strings, event data type, user defined data type, enumerated data types, type-casting (static and dynamic)
* Arrays – packed and unpacked arrays, multi-dimensional arrays, dynamic arrays and methods, associative arrays and methods, array methods , queues and methods,
* Structures and Unions – packed, unpacked, tagged
* Operators – assignment operators, logical and bit-wise operators, wild equality and inequality, operators’ precedence and associativity, concatenation, streaming operators and set membership operators.
* Procedural statement and control flow – selection statements, loop statements, jump statements, final blocks, named blocks, disable block, event control, level-sensitive sequence control
* Processes – always assignments (always\_comb, always\_latch, always\_ff), continuous assignments, fork join (join all, join any, join none), fork control (wait fork and disable fork)
* Subroutines – Tasks and Functions, discarding function-return values, Argument Passing (pass by value, pass by reference, pass by name, default argument values, optional argument lists)
* OOPs (Object Oriented Programming) –
  + Introduction,
  + Objects (member and methods),
  + Constructors(for initialization),
  + Classes (Introduction)
  + Assignment of classes,
  + Inheritance
    - Subclasses and Superclasses
    - Overriding (Members)
  + Data hiding and encapsulation with access specifiers (private, protected and public)
  + Polymorphism
  + Virtual classes (abstract classes)
  + Parameterized classes (type, value, generic, extending)
  + Nested Classes
  + Constants ( global constant, constant class, instance constant)
  + Statics - static classes, static methods, static lifetime method
  + Casting - as tasks as well as functions
  + Copy – shallow copy, deep copy, cloning
  + Scope resolution operator, null, external declarations
  + Differences between classes and structures
  + Typedef classes – forward reference, and circular dependency
  + Multiple inheritance and method overloading
* Randomization
  + Random Variables (declaration, rand and randc modifiers)
  + Randomization methods (pre built-in methods, pre randomize, post randomize,
  + Constraint blocks
  + Inline constraints - adding constraints to already existing constraints
  + Global constraints – adding constraints between variables of different classes
  + Static constraints
  + Constraint modes
  + Randomization Controllability – additional controllability by giving maximum and minimum values
  + Randomizing objects –
  + CRV (Constrained Random Verification)
  + Verilog CRV
  + System Verilog CRV
* Coverage - It is defined as the extent or degree to which something is observed, analyzed, and reported. In typical methodologies you would
  + - Write the test plan.
    - Write the tests
    - Perform functional coverage to cover everything specified in the test plan.
    - Perform code coverage to identify any redundant or overlooked items.
    - Update the test plan to include any new/untouched scenarios or remove any redundancy.
    - Iterate back through the functional coverage.

There are two types of coverage: functional coverage and code coverage. Each type can be broken down into smaller categories, each with its own interpretation as to how well the design has been exercised.

* + Functional Coverage - Functional coverage brings the specification, the test plan, and the actual tests together by providing information about which functions have or have not been exercised in the design. This can be further categorized into:
    - Application Coverage -
    - Interface Coverage
    - Structural Coverage
  + Code Coverage – This coverage tells us about the degree to which the code that has been exercised. This information is very valuable but should be appropriately utilized. This can be further categorized into:
    - Statement/Line Coverage –This coverage tells us whether or not a code has been exercised or not.
    - Expression Coverage – This generally deals with codes involving expression, especially Boolean expression. This coverage tells whether the expression has been exercised in every way possible.
    - State Machine Coverage -This coverage provides information on the visited transitions, arcs, or states in a finite state machine. It can also provide the actual path taken through the state machine.