Verilog Gotchas

Note: Selected Rules (Highligted) are important for design and verification.

Chapter 2.

1. Misspelled names
    1. Verilog variable is case-sensitive
    2. O v.s. 0
    3. enumerated WAIT v.s. wait (verilog task)
    4. **Naming guide**
        1. Variable and net names are in lowercase
        2. Constant names and enumerated labers in all uppercase
        3. class, first letter capital
        4. not conflict with keyword
5. **Implicit net declaration (Test)**
    1. undeclaraed -> implicit net (LHS), RHS erro
    2. connection to an instance -> implicit
    3. use language-aware editors
4. **Default of 1-bit internal nets (Test)** - Underclared identifier - default to a wire net type.
    1. all inernal nets and variables that are more than 1-bit wide must be explicitly delcared.
2. Single file versus multi-file compilaton of $unit declaration
3. **Local variable declarations (begin-end)**
    1. variables must be declaraed before any programming statements
    2. module/interface, or program block - variable can be declared anywhere
3. Escaped names in hierarchical paths
4. Hierarchical references to automatic variables (SV)
    1. module automatic name(...) - use automatic storage
2. Hierarchical references to variables in unnamed blocks
    1. use named begin..end, fork..join, e.g. begin: name
2. Hierarchical reference to imported package items (SV)
    1. Imported identifiers must be referenced using the scope resolution operators instead of hierarchically.

2. Importing enumerated type from package (SV)
3. Importing from multiple packages (SV)
4. **Default base of literal integers - (Test)**
    1. Literal integers in Verilog and SystemVerilog can be specified as a simple **decimal integer**(e.g. 5) or as a based integer (e.g. ' ns),
    2. <size>'s<base><value>
3. **Signedness of literal integer**
    1. Unbased literal integers default to signed. Based literal integers default to unsigned.
    2. <size>'**s**<base><value>
    3. **Size smaller than value rule**. If the bit size specified is fewer bits than the value, then the left-most bits of the value are truncated
    4. Size greater than value rule. When the bit size specified is more bits than the value, the value will be expanded to the size by left-extending.
5. **Signed literal integers zero extend to their specified size**
    1. Too small a size truncates the most-significant bits ofa value. Too large a size left-extends a value with 0, x, or z, but does not sign extend.
2. **Literal integersize mismatch in assignments**
    1. A size mismatch in an assignment might zero extend or might sign extend, depending on the types of expressiion Passing real (floating point) numbers through port on the right-hand side of the assignment.
2. Filling vectors with all ones
    1. Verilog does not have a literal integer value that fills all bits of a vector with ones.
2. Array literals versus concatenations
3. **Port connection rules**
    1. The size ofa port and the size of the net or variable connected to it can be different.
2. Back-driven ports (input but being driven)
3. Passing real (floating point) numbers through ports
    1. Verilog does not allow real numbers to be passed directly through ports. System Verilog does, but requires a special port declaration.
2. **(Chapter 3 RTL Modeling) Combinational logic sensitivity lists with function calls**

    1. If combinational logic calls a function, then the combinational sensitivity list must include signals that the function reads. @* does not infer sensitivity to values read by functions called from combinational logic.
2. **Array in sensitivity lists**
3. **Vectors in sequential logic sensivitiy list**
    1. A sequential logic sensitivity list triggers on changes to the least significant bit ofthe vector
2. **Operation in sensitivity lists (test)**
    1. Operations in sensitivity lists only trigger on changes to the operation result.-
2. **Sequential logic blocks with begin...end groups (Test)**
    1. Resettab/e sequential procedural blocks witha begin..end block can contain statements that execute asynchronous to the clock.
    2. Add begin-end even for one statement
    3. A synthesis requirement is that a resettable sequential procedural block **should only contain a single if...else statement, though each branch of the if...else might contain multiple statements**
4. **Sequential logic blocks with resets (Test)**
    1. Resettable sequential procedural blocks can incorrectly reset only some of the outputs.
2. **Asynchronous set/reset flip-flop for simulation and synthesis (Test)**
    1. always_ff @(posedge elk, negedge rst_n, negedge set_n)
    2. if ...else...if...else priority encoding style to prioritize the set and the reset
3. **Blocking assignment in sequential procedural blocks (Test)**
4. **Sequential logic that requires blocking assignments (Test)**
    1. When modeling clock dividers, the RTL synthesis design guidelines dont always apply.
2. **Nonblocking assignments in combinational logic (Test)**
    1. Nonblocking assignments in a combinational logic block can cause infinite loops that lockup simulation
2. **Combinational logic assignments in the wrong order (Test)**
    1. Synthesis might optimize away inferred storage in combinational logic.
2. **Casez/casex masks in case expression (Test)**
    1. Masked bits can be specified on either side of a casez or casex statement comparison.
2. **Incomplete decision statements (Test)**

    1. Incomplete case statements or if..else decision statements can result in hard-to-detect design errors.
    2. The synthesis full_case pragma instructs synthesis to ignore any unspecified decision selection values. The parallel_case pragma instructs synthesis to ignore the possibility of redundant selection values.
3. **Overlapped decision statements (Test)**
    1. **Linting tool**
2. Inappropriate use of unique case statements
3. Resetting 2-state models
4. Locked state machines modeled with enumerated types
5. **Hidden design problems with 4-state logic (Test)**
6. Hidden design problems using 2-state types
7. Hidden problems with out-or-bounds array access
    1. Out-of-bounds errors mightnotpropagate through 2-state logic.
    2. Using 4-state array, return X
3. Out-or-bounds assignments to enumerated types
    1. Enumerated types are strongly typed, but can still have values other than those in their enumerated list.
    2. Verilog is a loosely typed language
3. **Undetected shared variables in modules (Test)**
    1. Variables written to by multiple processes create shared resource conflicts.
2. Undetected shared variables in interfaces and packages
3. (Chapter 4: Operator Gotchas) Assignments in experssions
4. **Self-determined versus context-determined operators**
5. **Operation size and sign extension in assignment statements**
    1. In an assignment statement, sign extension context is only dependent on the right-hand side of the assignment.-
2. Signed arithmetic rules
    1. The entire right-hand side context of an assignment must be signed, in order to have signed arithmetic operations
2. Bit-select and part-select operations
    1. The result ofa part-select operation is always unsigned, even when the entire vector is selected-
2. **Increment, decrement and assignment operators (Test)**
    1. Increment, decrement, and assignment operations are blocking assignments.

2. Pre-increment versus post-increment operations (SV)
    1. Pre-increment versus post-increment can affect the result of some expressions.
2. Modifying a variable multiple times in one statement (SV)
    1. The evaluation order is undefined when a compound expression modifies the same variable multiple times on the right-hand side of an assignment statement.
2. **Operator evaluation short circuiting (Test)**
    1. Simulation might not evaluate all operands in some circumstances.
2. **The not operator (!) versus the invert operator (~) (Test)**
    1. The logical NOT operator and the bitwise invert operator perform different operations and can be used incorrectly.
    2. The bitwise invert operator should never be used to negate logical true/false tests.
3. Array method operations (SV)
4. Array method operations on an array subset (SV)
5. (Chapter 5: General Programming Gotchas) **Verifying asynchronous and synchronous reset at time zero (Test)**
    1. Initial procedural blocks can activate in any order relative to always procedural blocks.
    2. Avoid clocking and resetting a design at time zero
3. Nested if ...else blocks
    1. An else construct pairs with the nearest if statement that does not havean else; begin...endcan override this default pairing
2. **Evaluation of equality with 4-state values (Test)**
    1. The equality operators have three answers, true, false and unknown, but if..else decision statements only have two branches.
    2. in order for a 4-state equivalence test to be true or false, no bits in the expression can havea Z or X value.
    3. corrected by using special verification
    4. operators in Verilog, the identity operators (=== and ! ==), instead of the usual programming equality operators (= and !=).
5. **Event triger race conditions (SV)**
    1. **->, @**
    2. **System verilog ->>**
3. Using semaphores for synchronization (SV)
    1. Verilog event used to synchronize procedural blocks
2. Using mailboxes for synchronization

3. Trigger on clocking blocks (SV)
4. Misplaced semicolons after decision statements
5. Misplaced semicolons in for loops
    1. misplaced semicolon has the effect of making the loop appear to execute only one time.
2. **Infinite for loops**
    1. Declaring too small afor loop control variable can result in loops that never exits.
2. Locked simulation due to concurrent for loops
    1. Parallel for loops that use the same control variable can interfere with each other.
    2. use different variabls, declaraing a local variable
3. **Referencing for loop control variables**
    1. declare control variable locally
2. Default function return size
    1. default return type of 1-bit logic
2. Task/function arguments with default values
3. **Continuous assignments with delays cancel glitches**
    1. Continuous assignments with delays will cancel input glitches
    2. inertial delay mechanism
    3. transport delay : <= #delay (nonblocking intra-assignment