These sample slides are taken from our basic Verilog training courses. They are from a variety of topics. The first two days of the 4-day course cover the same material as the 2-day course. Either the 2-day or 4-day course is a prerequisite for the 3-day *Advanced Verilog Coding Styles for Synthesis & Verification* course.

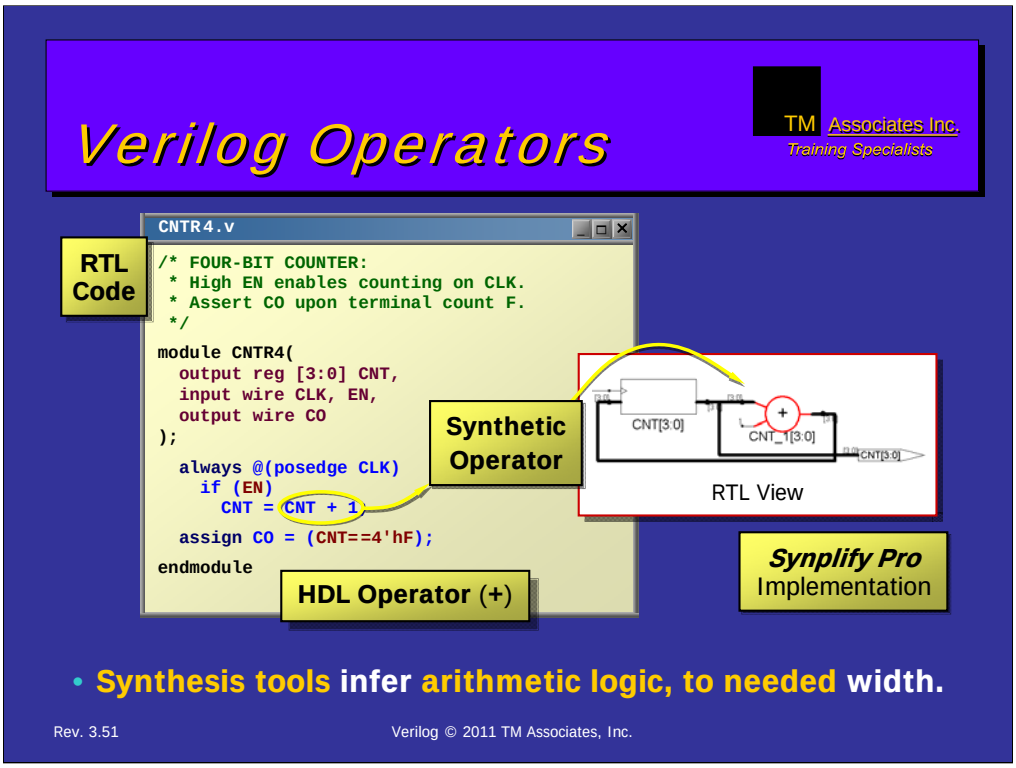
The training material was developed by a team of Verilog designers with a combined 50+ years of experience.

Comprehensive notes are included with each slide. This means the student will

- Be able to spend more time listening to the instructor instead of taking notes, and
- Be able to use the training manual as a valuable reference guide after the training.

For complete information on all of our courses visit our website

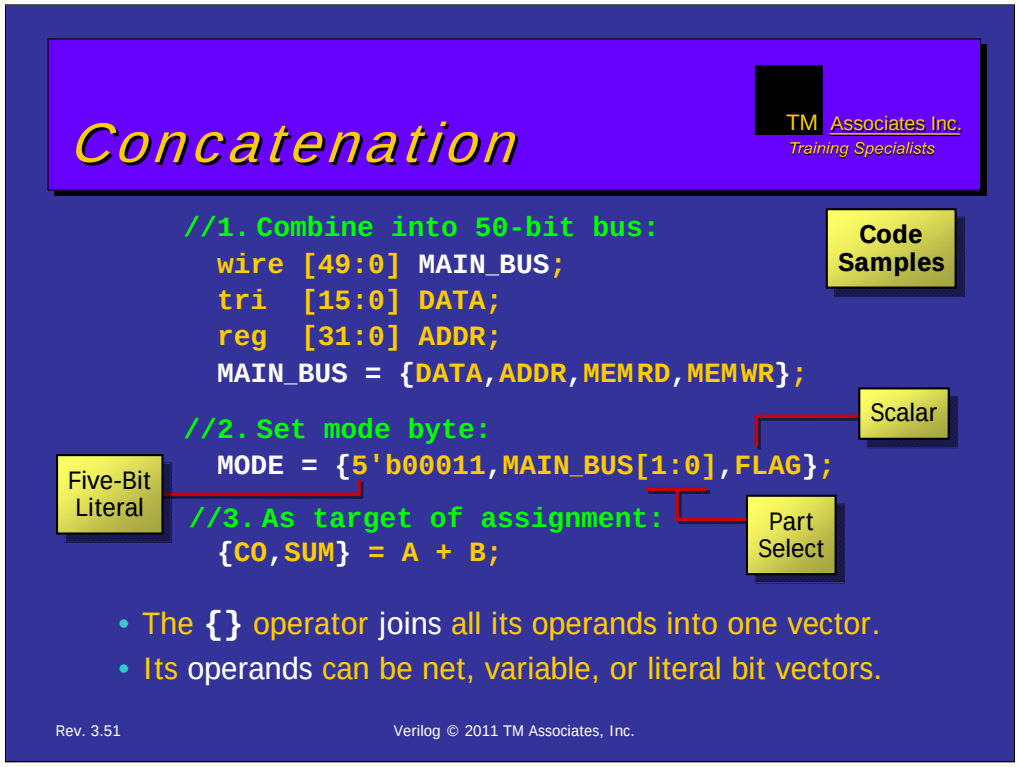
www.tm-associates.com



- Verilog is an *operator-rich* language. Using its arithmetic, relational, and logical operators, you can describe complex functions with concise code.
- Almost all of Verilog's operators are *synthesizable*—within restrictions. These restrictions can *vary* from tool to tool, and even with tool versions.
- For example, some tools synthesize **/** only for fixed divisors **2**, **4**, **8**, **16**… Higher-end tools may synthesize a combinational-logic *divider* IP block.
- Only two operators are inherently *non-synthesizable*—the four-valued-logic *identity* operators, **===** and **!==**. As we'll see, they're for verification only.
- This slide gives insight into how HDL operators are synthesized. The tool *infers* needed logic (e.g., increment) from the operator (**+**) and its context.

Operator Inference:

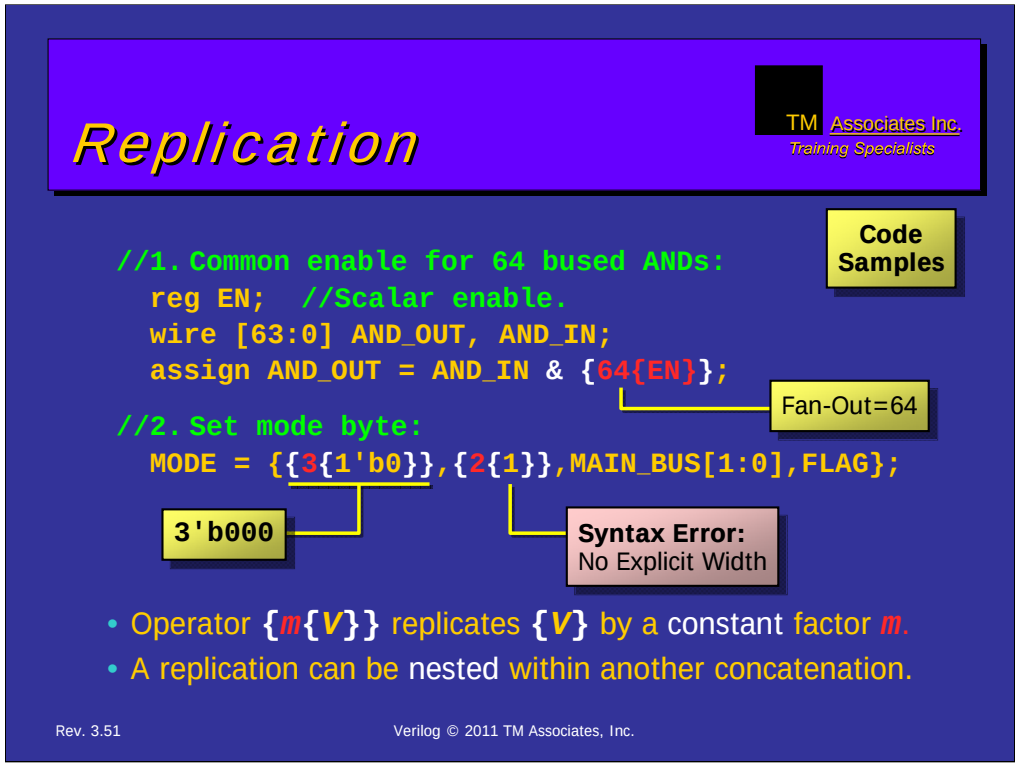
- Simpler operators, like the bitwise AND (**&**) and OR (**|**), or the two-valued *equality* operator (e.g. **CNT==4'hF**), are synthesized out of random logic.
- More complex operators—for example, signed or unsigned multiply (*****)—are implemented from tool-specific IP blocks or custom module compilers.
- These implementations are *parameterized* to fit the *bit width* and *signing* implied in the RTL code. Thus, **CNT + 1** yields four-bit *incrementer* logic.
- Thus, when the tool identifies an operator (like **+**), it *infers* the appropriate parameterized logic, optimizing it to the target ASIC or FPGA technology.



- The first Verilog operator we'll investigate in full detail is **concatenation**.
- Its formal definition is: *concatenation* ::= **{** *expression* {**,** *expression*} **}**
- The *outer* curly braces (in bold) represent the literal concatenation operator, while the *inner* braces (light) are Backus-Naur form for *zero or more* items.
- A concatenation is thus a curly-braced, comma-separated list of items to be joined into a single bit vector, as wide as the *sum* of the individual widths.
- Operands can be either scalars, or vectors of identifiable width—including data type **`integer`**—but not *real* or *unsized* numbers (e.g., **`1`** or **`'b0`**).

Code Samples:

- These code samples are intended to present Verilog syntax in a concise yet realistic context, and do not always include all supporting declarations, etc.
- In sample 1, four vector or scalar signals of various data types are joined (concatenated) into a single bus, whose width is thus 16 + 32 + 2 or 50 bits.
- The assignment to **`MAIN_BUS`** is shown in isolation. In actual code, it would have to be part of an **`assign`** statement, or perhaps a procedural block.
- In sample 2, **`MODE`** is concatenated from a vector literal, a part select from **`MAIN_BUS`**, and scalar **`FLAG`**. Again, the assignment is shown in isolation.
- Sample 3 demonstrates a powerful feature of Verilog concatenation—the concatenated variables can occur on the *left-hand side* of an assignment.



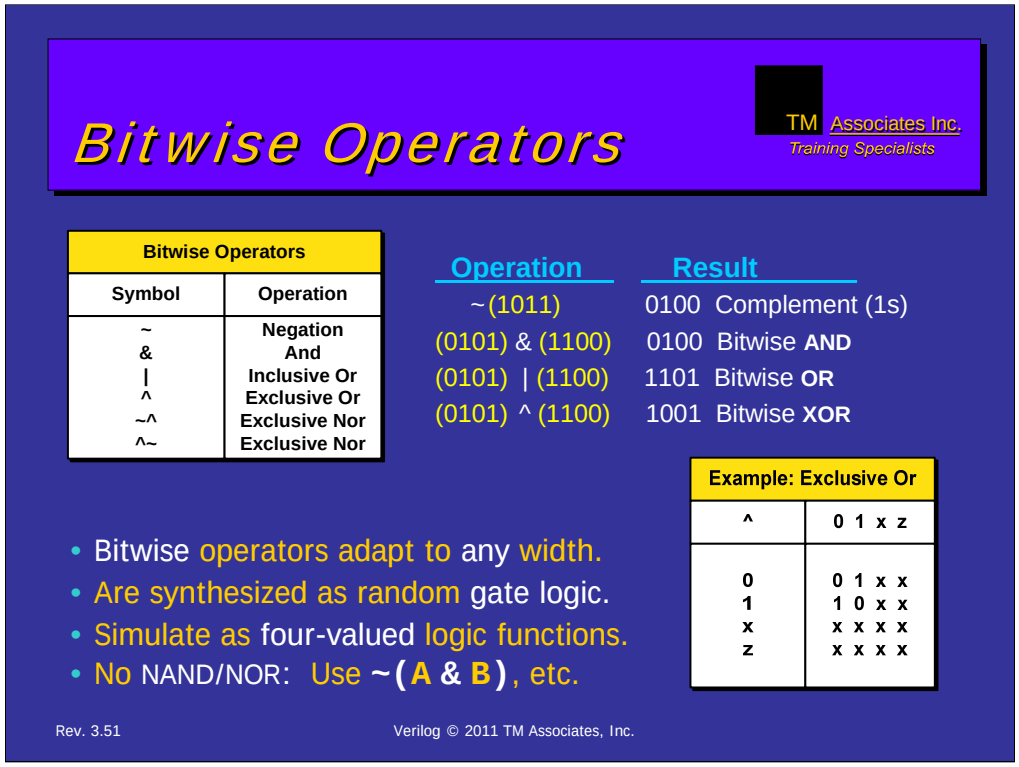
- **Replication** is a form of concatenation that joins together *multiple copies* of the curly-braced inner expression.
- The multiplier ***m*** must be a *constant* nonzero integer (with no **X** or **Z** bits). The second expression follows the previous rules for concatenations.

Code Samples:

- Sample 1 shows a concise method of *fanning out* a one-bit enable signal **EN** to 64 bused AND-gate inputs in an RTL Verilog description.
- Operator **&** is the bitwise AND operator, which we'll discuss shortly. Like a set of two-input AND gates, it ANDs together the individual pairs of bits.
- Sample 2 shows that concatenations can *nest*. This alternative method of specifying **MODE** is more *scalable* for wider vectors with runs of **0**s or **1**s.

Common Pitfall:

- In **Unit 2**, we encountered a *width-mismatch* warning due to invalid use of simple decimal numbers, like **1**. In Sample 2, we see a similar issue.
- Sample 2 results in an outright syntax error: the simple number **1** has no *explicit* bit width—it defaults to 32 or more bits, depending on the tool.
- The resulting error message in **Model*Sim*** is: *Illegal concatenation of an unsized constant.* To fix the error, replace **1** by the more explicit **1'b1**.



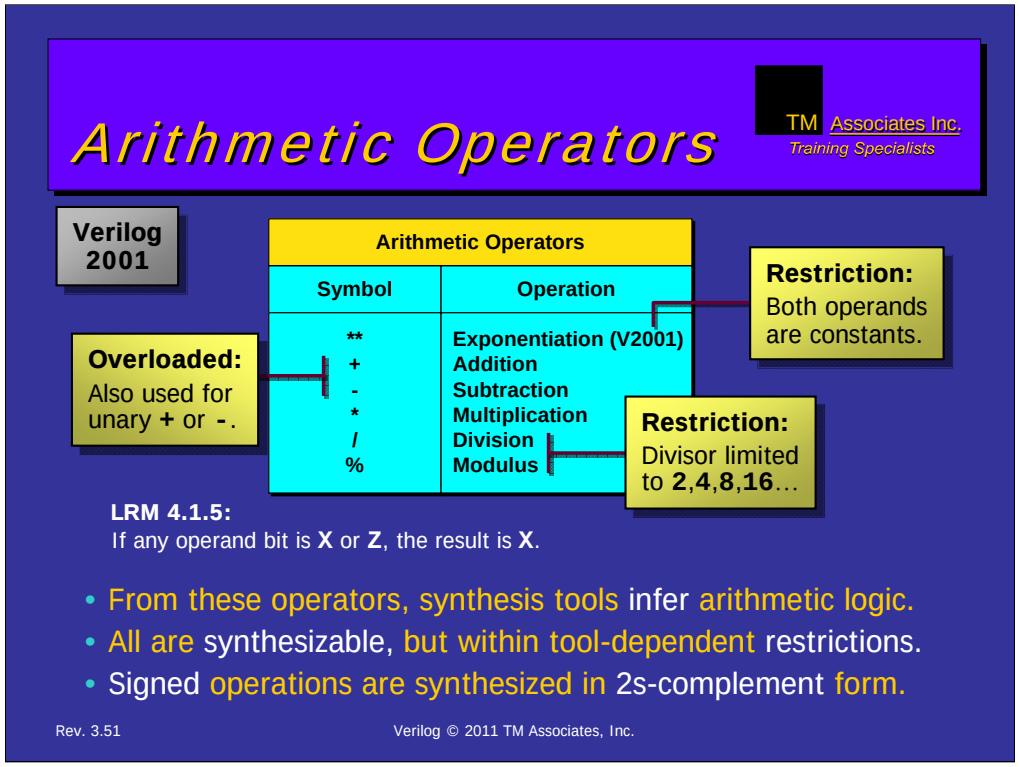
- Along with concatenation of bit vectors, Verilog's more *hardware-oriented* operators include six **bitwise** operators (of which **~^** and **^~** are synonyms).
- For example, the bitwise operation **A & B** will AND each bit of operand **A** with the corresponding bit of operand **B**, from the LSBs towards the MSBs.
- The *unary* **~** operator takes a *single* operand, and inverts each of its bits. This operator is typically synthesized as inverters, or inverting bubbles.
- All other bitwise operators are *binary*—they take *right* and *left* operands.

Synthesis Semantics:

- Synthesis tools typically implement these operators as random gate logic.
- Because these tools do significant *logic optimization,* the schematic you get may bear little resemblance to your code—but it *is* functionally equivalent.
- For instance, a *fast* implementation of **A & B** in standard-cell CMOS is often a 2:1 MUX, with **A** fed to input 1, **B** used as the select, and input 0 tied low.

Simulation Semantics:

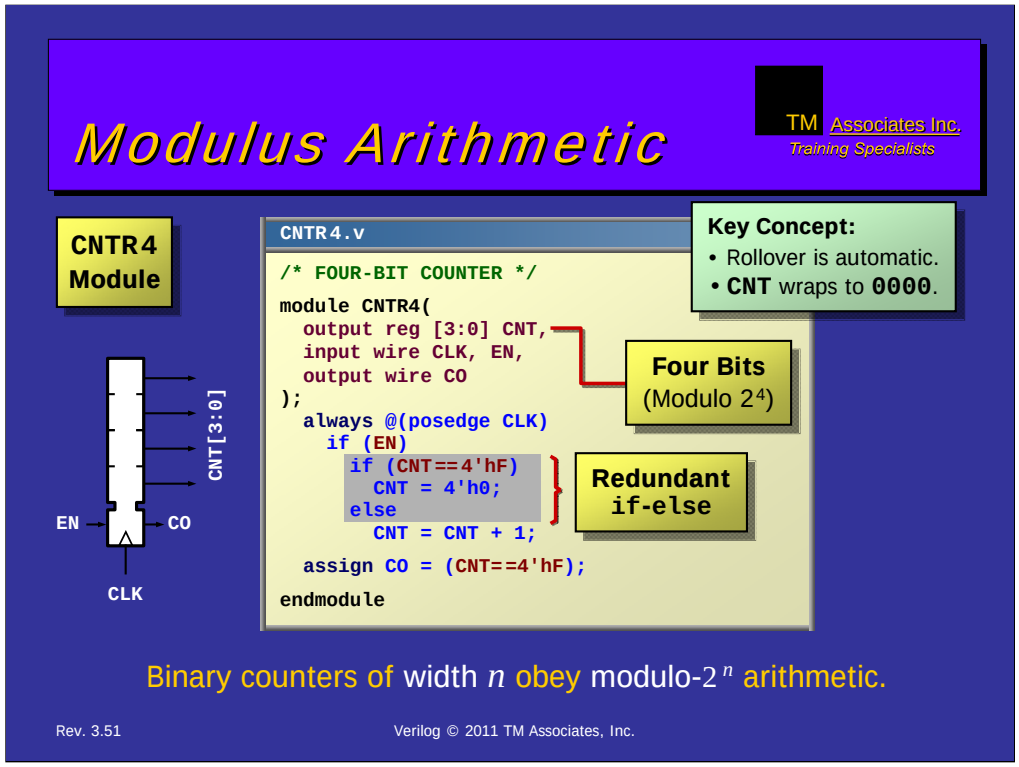
- Unlike synthesis tools, which can only AND together **0**s and **1**s, simulators can handle **X** and **Z** inputs as well. Thus, **0 & X** yields **0**, but **0 ^ X** yields **X**.
- Typically, such indeterminate **X** values will propagate through gate logic—often indicating a hardware *reset* problem as the root cause.



- HDL operators are like the *white filling* in the oreo-cookie model. They correspond to the *transfer* logic in register-transfer level (RTL) designs.
- The *arithmetic* operators can have a big impact on timing and area budgets. Hence, we want to fully understand the semantics of these six operators.

Synthesis Semantics:

- All arithmetic operators are *synthesizable*—with tool-specific restrictions.
- Synopsys ***Design Compiler***, for example, can synthesize **`2 ** DIN`**. But other tools may limit *both* the operands of **`**`** to *constant* expressions.
- When ***Design Compiler*** identifies an arithmetic operator in your HDL code, it *infers* the appropriate IP block, implementing it in the target technology.
- *Inferred* arithmetic logic—such as a multiplier—is always *combinational.* The synthesized logic is *parameterized* to fit the bit width of the operands.
- Alternatively, you can opt to *instantiate* a Synopsys-specific IP block, like an *n*-stage *pipelined* multiplier—at the sacrifice of losing code portability.
- FPGA-based tools like ***Precision*** do not rely on IP blocks, since arithmetic logic must be *tailored* to utilize FPGA-specific resources like carry chains.
- When ***Precision*** infers an arithmetic operation, it runs technology-specific, parameterized *module generators* to customize the logic to a target device.
- Thus, in **Unit 1** we saw a fast ORCA-specific implementation of **`CNTR 8`**.



- Ordinary binary counting in Verilog obeys the familiar rules of modulo-2 *n* arithmetic, where *n* is the width of the counter in bits [**LRM Clause 3.3.1**].
- This applies to Verilog arithmetic operations done on any bit-vector signal, whether of net or **`reg`** type.

Modulus Math:

- An everyday example of modulus arithmetic is the automobile *odometer,* which wraps around from 99,999 miles to 0 miles. Its *modulus* is 100,000.
- A four-bit counter has a modulus of 2 4. As the binary count advances with each clock, multiples of 16 are discarded, leaving only a count *remainder.*
- After 23 clock edges, for example, the four-bit count is 23 *modulo 16,* or 7.
- From a hardware perspective, only four flip-flops are synthesized for **`CNT`**. Thus, no flops are available in **`CNTR 4`** to store bit positions **`4`** and higher.

Conclusions:

- Rollover code, like the shaded **`if-else`** statement above, is thus redundant. Omit it, since a four-bit count will *automatically* roll over from hex **`F`** to **`0`**.
- Most synthesis tools, like Mentor Graphics ***Precision***, efficiently *prune out* such redundant code, yielding exactly the same logic for **`CNTR 4`** as before.
- But to prevent excess logic, possibly generated by legacy synthesis tools, use the **`CNTR 4.v`** version from **Unit 1**, and avoid redundant rollover code.

Summary

TM Associates Inc.
Training Specialists

For more information on our Verilog, SystemVerilog or VHDL courses please contact

Tom Wille
tw@tm-associates.com
503-656-4457
www.tm-associates.com