# Verilog Errors

Each example here consists of a design and a test-bench. The design and/or the test-bench contain common errors or problems that need fixing. They are also opportunities for you to develop your skills in terms of:

- Reading synthesisable Verilog
- Translating Verilog to circuit diagrams
- Reading and modifying test-benches
- Working with compilation and simulation
- Interpreting wave-forms

#### v1 - 8-bit incrementor

The module add4 (correctly) implements an 8-bit incrementor. The test-bench (incorrectly) claims that it does not work.

Fix the test-bench.

### Discussion:

- What is the underlying error?
- Does the compiler warn you about the problem?
- How will you avoid this type of error?

### v2 - 1-bit Flip-Flop

The module ff is supposed to implement a 1-bit Flip-Flop. The test-bench (correctly) claims that it does not work.

Fix the flip-flop.

#### Discussion:

- What does the waveform look like? Can you explain it?
- What is the underlying error?
- Does the compiler try to help you avoid this type of error?
- What happens if you use always\_ff in the original (broken) FF, rather than always?

# v3 - 1-bit Flip-Flop with clock-enable

The module ff is supposed to implement a 1-bit Flip-Flop with a clock-enable - as with a normal flip-flop it only updates q at the rising edge of the clock, but it also requires that ce is asserted. The test-bench (correctly) claims that it does not work, but the test-bench itself is also not correct.

Fix the flip-flop and the test-bench.

### Discussion:

- Did the compiler suggest there is a problem?
- The test-bench is correct that the FF does not work, but still contains an error. How can you detect errors in test-benches?
- Both the FF and bench-mark use examples of poor style. Try to recognise them, and the errors they might cause.
- The test-bench never modifies clock\_enable. Why might that be a problem?

### v4 - Or gate

The module or\_gate.v is supposed to implement an "or" gate, but the test-bench (correctly) claims that it does not work.

Fix the or gate.

#### Discussion:

- Did the compiler suggest there is a problem?
- Does replacing always with always\_comb change the way the compiler views it?
- Why is the module called or\_gate rather than just or?
- How many different ways or styles of specifying the or gates logic can you come up with?
- Which style do you think is clearest and/or least error prone?

### v5 - 4-bit combinatorial adder

The module fadd implements a 1-bit full-adder, which the module add4 then uses to build a 4-bit adder. The test-bench (correctly) indicates that the 4-bit adder does not work.

Fix the 4-bit adder.

### Discussion:

- The adder is wrong due to an incorrect assumption about the semantics of buses (multi-wire signals) in Verilog. What is it?
- Assume you had to build a module add32.v which adds 32-bit values, but still had to rely on the fadd primitive. How might you do it?
- The test-bench for add4.v correctly identifies this problem through an exhaustive testing strategy. Would this strategy still work for add32.v?

# v6 - 16-bit Hamming weight

The modules hamming2, hamming4, hamming8, and hamming16 each calculate the Hamming weight of 2, 4, 8, or 16 bits. The Hamming weight is just a count of the number of set bits in the input - for example, the hamming weight of 0101 is 2, 1110 is 3, and 1111 is 4. The programming style uses recursive composition of modules to build the wider circuits out of the smaller circuits.

The test-bench claims that hamming16 works, but this is incorrect - both the circuit *and* the test-bench contain errors.

Fix the test-bench, then fix the circuit.

Discussion: - Both the test-bench and circuit were written by the same person. Why might this cause this type of error? - The circuit was written before the test-bench. Why might this cause this type of error? - The failure case is unlikely because it is rare. How might the test-bench be written to look for these cases? - What is the advantage of this hierarchical composition style? - How might you adapt the test-bench and test approach for a 64-bit hamming circuit?

# v7 - Composite ALU

The module add\_sub\_logic constructs a 4 operation ALU around a single 16-bit adder (provided by add16) - this is very similar to the construction of the bit-slice ALU in lectures, except the carry-chain is included embedded in a Verilog multi-bit addition. In this case the design-under test is correct, but the test-bench is wrong, though very close to being right.

Fix the test-bench.

#### Discussion:

- In this case the circuit and test-bench were developed independently, and it was initially unclear which one was correct. What could (or should) help resolve such situations?
- The case op==3 in add\_sub\_logic has some edge-cases. What types of inputs (values of a and b) would be worth testing?
- The total set of possible inputs is very large. How might you create an "interesting" set of inputs that might trigger failure edge cases for all ops?