12 Coding for Synthesis

At the level of abstraction where logic is coded in an HDL language, synthesis optimizations can only take a designer so far when meeting design requirements. At its very fundamental level, a synthesis tool will follow the coding structure and map the logic according to the architecture laid out in the RTL. Only for very regular structures such as FSMs, RAMs, and so forth can a synthesis tool extract the functionality, identify alternative architectures, and implement accordingly.

Aside from optimization, a fundamental guiding principle when coding for synthesis is to minimize, if not eliminate, all structures and directives that could potentially create a mismatch between simulation and synthesis. A good coding style typically ensures that the RTL simulation will behave the same as the synthesized netlist.

12.1 DECISION TREES

A decision tree refers to the sequence of conditions that are used to decide what action the logic will take. Usually, this breaks down to if/else and case structures.

12.1.1 Priority versus parallel

Implemented in *priority_decisionTree.sv* is a simple decision structure that is used to write to a register. The if/else structures are synthesized to a chain of muxes as shown in the synthesized design in Figure 12.1, and this creates the concept of priority. Those conditions that occur first in the if/else statement are given priority over others in the tree. A higher priority would correspond with the muxes near the end of the chain and closer to the register.

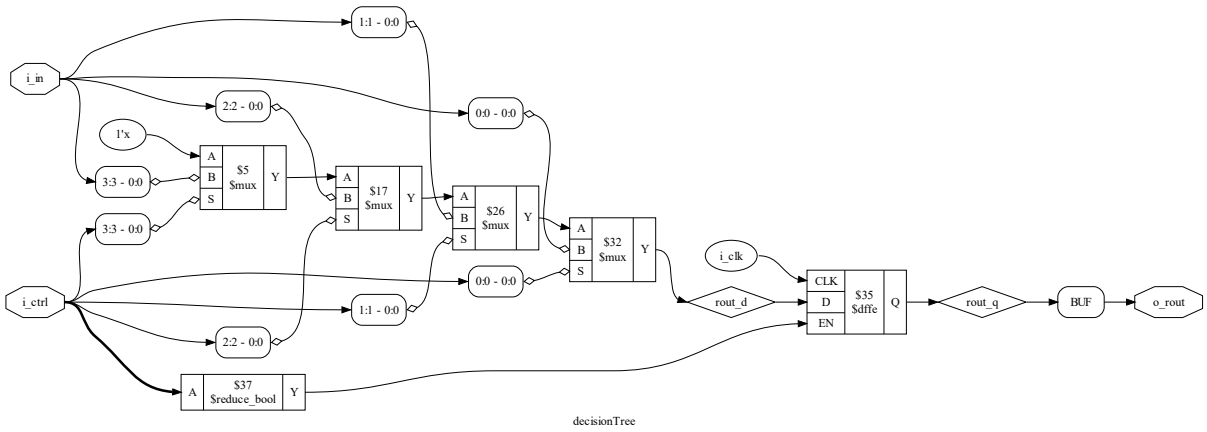

Figure 12.1: Simple priority with serialized mux structure.

Using a case statement as an alternative to if/else statements as implemented in *case_decisionTree.sv* results to a similar design being synthesized.

If it is required for the decision to be truly parallel, the synthesis directive “parallel_case” should be used. Note that the directive is a synthesis-only directive, and thus mismatches between simulation and actual implementation can occur. Frequent use of synthesis directives in general is bad design practice. It is better to code the RTL such that both the synthesis and simulation tools recognize the parallel architecture.

12.1.2 Full conditions

If the default condition in *case_decisionTree.sv* was omitted, the synthesis tool would feed the output of the register back around to the decision tree as a default condition so that if no conditions are met, the value does not change.

If it's not necessary to latch the value when none of the conditions are matched, the default value can be set to 0 as implemented in *default0_decisionTree.sv*. This will eliminate the need of an Enable but will increase the dynamic power consumption as the FF toggles more often.

A "full_case" synthesis directive can be used to tell the synthesis tool that all possible conditions have been covered by the case statement regardless of how the tool interprets the conditions. For the same reason as the parallel_case directive, using it is dangerous!

12.1.3 Multiple Control Branches

Keep all register assignments inside one single control structure to prevent the condition where a control branch is disconnected as shown in the textbook.

12.2 TRAPS

12.2.1 Blocking Versus Nonblocking

Use blocking assignments to model combinatorial logic.

Use nonblocking assignments to model sequential logic.

Never mix blocking and nonblocking assignments in one always block.

Violating these guidelines will likely lead to mismatches in simulation versus synthesis, poor readability, decreased simulation performance, and hardware errors that are difficult to debug.

12.2.2 For-Loops

For-loops should not be used to implement software-like iterative algorithms.

For-loops are often used as a short form to reduce the length of repetitive but parallel code segments.

12.2.3 Combinatorial Loops

Combinatorial loops are logic structures that contain feedback without any intermediate synchronous elements.

This type of behavior is rarely desirable and typically indicates an error in the design or implementation.

12.2.4 Inferred Latches

See textbook.

12.3 DESIGN ORGANIZATION

See textbook.