

The `verify` command contains an option, `-selections number`, which controls the size of the random sampling of input selections in the `autorestrict` algorithm. The number ranges from 1 (small sampling) to 100 (large sampling). The default is 50.

If you wish to change the size of the random sampling of input selections while running a query with the `autorestrict` algorithm, you can do so by invoking the `verify` command as follows:

```
verify queryname -algorithm autorestrict -selections new number
```

Clock Extraction

Clock extraction is a sound optimization for verification runs in FormalCheck if a design is synchronous and has only *one* active clock edge (either rising or falling). When applied, FormalCheck “extracts the clock” by examining the design behavior *only* on the *active* clock edge. Disregarding consideration of the inactive clock edge may yield substantial savings in time and memory consumed by the verification run. Additionally, any failure error trace produced will have half the length of the original trace. Clock extraction is a simple, push-button optimization that can be applied to queries that take excessive computation time or memory.

Apart from optimization of the verification run, clock extraction can be used to check if the design is indeed dependent on *one* active clock edge. FormalCheck flags an error message if some signal in the design is not conditioned on the specified active clock edge. Consequently, clock extraction is not applicable to designs that are conditioned on both the rising and the falling edges of the clock, multiple clocks, or designs that contain level-sensitive latches. Also, clock extraction is only allowed with a (1 1) clock.

Cadence recommends that you use clock extraction to verify a design whenever possible. You should apply it early in the verification process to save redoing the queries to make them clock extractable.

This section describes how to do the following tasks:

- Apply clock extraction for the verification of queries.
- Debug queries that terminate due to violations of the clock extraction audit.
- Restrict the scope of the query run to a bug hunting effort on designs for which clock extraction is not completely feasible.

Applying Clock Extraction

Clock extraction is applied by the following command

FormalCheck User Guide

Running Verification

```
project -configure -extractclock <clock-name>
```

This command applies to the entire project. Therefore, all subsequent verifications will run with clock extraction. All previously run queries will be outdated.

The results of verification may change when clock extraction is applied, unless clock extraction is taken into account while expressing the properties in the query. A query, whose properties contain expressions that are not conditioned on the active edge of the clock, monitors the design behavior both on the active edge and the inactive edge. Additionally, the state variables specified in FormalCheck may be conditioned on both edges of the clock. Therefore, monitoring the design on both edges of the clock is different from considering the clock extracted design, thereby producing different verification outcomes.

To ensure that the verification with clock extraction yields the same results as verification without clock extraction, the expression in the properties comprising the queries must be conditioned on the same active clock edge. The rules for the properties are listed below

- the expressions following the `always` and `eventuallyAlways` qualifier must be checked only on the active edge of the clock. For example, when extracting the rising edge of the clock, the property `p1` is written as below.

```
property -create p1 {
    always {count <15 || clk != rising}
}
```

- the expressions following the `after`, `never`, `unless`, `unless_after`, and `eventually` qualifier must be conditioned on the active edge of the clock. For example, when extracting the rising edge of the clock, the property `p2` is written as below.

```
property -create p2 {
    after {req == 1 && clk == rising}
    eventually {grant == 1 && clk == rising}
    unless {req == 0 && clk == rising}
}
```

Additionally, all state variables also need to be conditioned on the same active edge. When clock extraction is applied, the inputs are assumed to be synchronous with the clock. Therefore, you can also remove constraints that specify synchronous inputs when applying clock extraction.

When the property expressions or FormalCheck state variables are not conditioned on the active edge of the clock as recommended above, and clock extraction is applied on the design, then FormalCheck monitors only the design on the active edge of the clock. (Results may vary if you verify the same query on the design without clock extraction). FormalCheck will produce a warning stating which property or state variable is not extractable. This warning is produced in the files `verify.out` or `verify.stdout`. See [“Debugging”](#) on page 138 for more information.

Verification Results

When you use clock extraction, FormalCheck checks the design to assess whether or not it is *safe* to extract the clock prior to running the verification. If the check fails, the query run terminates with the following message:

```
ERROR: The clock cannot be extracted from this design.
```

Note: In production environments, design teams often know in advance if a design is clock-extractable, and if so, write your queries with this in mind.

The performance of clock extraction on benchmark circuits suggests a speed improvement of approximately 30% to 40% for the *BDD* algorithm. For the *explicit state* algorithm, the speed improvement is approximately 50%. Often, the verification run is able to explore all reachable states, thus allowing for query convergence without running out of memory.

When a Query Fails

When a query fails, the counter-example does not contain the clock signal (it has been extracted). Therefore, every *tick* mark on the waveform display coincides with the occurrence of the active edge of the clock at any clock-cycle.

Debugging

If the query run terminates the following message is displayed:

```
ERROR: The clock cannot be extracted from this design
```

When this occurs, you should inspect the files `verify.out` and `verify.stdout` for clues to explain the failure. It is likely that the query termination is due to one or more of the following conditions:

- FormalCheck does not agree with your assessment that the design is safe for clock extraction.
- The primary clock signal is not controlled by a (1 1) clock
- Some state variables in the design are **not** all changing only on the active clock edge.
- The design contains multiple clocks, gated clocks, asynchronous logic, or level-sensitive latches.
- The design contains spuriously inferred latches.

Understanding Audit Failures

Consider the following portion of a VHDL design, named `example.vhd`. Running any query using clock extraction on this design results in termination of the query because the design is conditioned on both edges of a common clock.

```
65 reg_A: PROCESS (clk, Inp)
66 BEGIN
67 IF (clk'EVENT AND clk = '0') THEN
68 Areg <= Inp;
69 END IF;
70 END PROCESS reg_A;
71
72 reg_B: PROCESS (clk, Inp)
73 BEGIN
74 IF (clk'EVENT AND clk = '1') THEN
75 Breg <= Inp;
76 END IF;
77 END PROCESS reg_B;
```

Asynchronous Reset

As a special case, a design with an asynchronous reset is considered to be extractable when the reset signal is assigned to a constant value (for example, when the reset is always deasserted). The initial state in this case is either the default value that FormalCheck sets or the design may be reset to a specific reset state, as explained in [Appendix E, “Advanced Options for Verification.”](#)

Improving FormalCheck Efficiency

This section includes tips on how to reduce verification run time.

- Reduce the number of unrelated properties per query—Verification run time improves as the localization reduction increases. Since the reduction algorithm prunes the design model relative to the properties being checked, the largest pruning (and shortest run time) is achieved when all the properties in a query refer to the same design variables. For this reason, it is recommended that unrelated properties be placed into separate queries. It is faster to verify several small queries in sequence than to verify a single large query containing unrelated properties.