1. **As long as signal_a is up, signal_b should not be asserted. Write an assertion.**

```
property signal_b_asserted_when_signal_a_deasserted;
  @(posedge clk) disable_iff(!reset)
    ($fell(signal_a) |-> signal_b);
endproperty
```

`@(posedge clk)` specifies that the assertion triggers on the positive edge of the clock.

`disable_iff(!reset)` ensures that the assertion is disabled during the reset condition.

`($fell(signal_a) |-> signal_b)` states that whenever signal_a falls (goes from 1 to 0), signal_b should be asserted (1). This ensures that signal_b is asserted when signal_a is deasserted.

2. **The signal_a is a pulse; it can only be asserted for one cycle, and must be deasserted in the next cycle.**

```
property pulse_property;
  @(posedge clk) disable_iff(!reset)
    ($rose(signal_a) |-> ##1 !signal_a);
endproperty
```

`@(posedge clk)` specifies that the assertion triggers on the positive edge of the clock.

`disable_iff(!reset)` ensures that the assertion is disabled during the reset condition.

`$rose(signal_a)` checks for a rising edge of signal_a, meaning it was deasserted in the previous cycle.

`|-> ##1 !(signal_a)` specifies that in the next cycle (##1), signal_a must be deasserted.

3. **Signal_a and signal_b can only be asserted together for one cycle; in the next cycle, at least one of them must be deasserted.**

```
property both_asserted_one_cycle_either_deasserted_next_cycle;
  @(posedge clk) disable_iff(!reset)
    ($stable({signal_a, signal_b}) && (##1 !$stable({signal_a, signal_b})));
endproperty
```

`@(posedge clk)` specifies that the assertion triggers on the positive edge of the clock.

`disable_iff(!reset)` ensures that the assertion is disabled during the reset condition.

`$stable({signal_a, signal_b})` checks if both signal_a and signal_b are stable (unchanged) for the current cycle.

`(##1 !$stable({signal_a, signal_b}))` specifies that in the next cycle `(##1)`, at least one of them must change (be deasserted).

4. **Signal_a must not be asserted before the first signal_b (maybe asserted on the same cycle as signal_b)**

```
property no_signal_a_before_first_signal_b;
  @(posedge clk) disable_iff(!reset)
    (!$rose(signal_a) until (signal_b));
endproperty
```

`@(posedge clk)` specifies that the assertion triggers on the positive edge of the clock.

`disable_iff(!reset)` ensures that the assertion is disabled during the reset condition.

`$rose(signal_a)` checks for a rising edge of signal_a.

`until (signal_b)` specifies that the rising edge of signal_a should not occur until the rising edge of signal_b.

```
//OR

property no_signal_a_before_first_signal_b;

  @(posedge clk) disable_iff(!reset)
    ($rose(signal_b) |-> signal_a);
endproperty
```

`$rose(signal_b)` checks for a rising edge of signal_b.
`|->` signal_a specifies that if there is a rising edge of signal_b, then signal_a is allowed to be asserted.

5. **At posedge of clock if signal A is asserted, then in the next cycle signal B should go high & signal C should be stable. The signal C should not change until signal B goes low. Once the Signal B goes low in the same cycle the signal C should change.**

```
// Sequence S1: C should be stable for one or more cycles, intersecting with
B for one or more cycles
sequence S1;
  ($stable(c) [*1:$] intersect b[*1:$]);
endsequence

// Property ABC: If A rises, then in the next cycle, B should be true, C
should be stable,
// followed by the condition specified in S1. After S1, in the next cycle, C
should change and B should fall.
property property_A_B_C;
  @(posedge clk)
  $rose(a) |=> b && $stable(c) ##1 S1 ##1 $changed(c) && $fell(b);
endproperty
```

```
Sequence S1:
($stable(c) [*1:$] intersect b[*1:$]): C should be stable for one or more
cycles ($stable(c) [*1:$]), intersecting with B for one or more cycles
(intersect b[*1:$]).

Property property_A_B_C:
@(posedge clk): This property is triggered on the positive edge of the clock.
$rose(a) |=>: If signal A rises...
b && $stable(c) ##1 S1 ##1 $changed(c) && $fell(b): Then, in the next cycle,
B should be true (b), C should be stable ($stable(c)), followed by the
condition specified in sequence S1 (##1 S1), and after S1, in the next cycle,
C should change ($changed(c)) and B should fall ($fell(b)).
```

6. **Write an assertion to check divide by 2 circuit output.**

```
property divide_by_2_assertion;
  // Property triggered on the positive edge of the clock
  @(posedge clock)

  // Disable the property check during reset
  disable iff (reset)

  // Check that the current state 'q' is not equal to its previous state
  (q != $past(q, 1));
endproperty
```

```
disable iff (reset): The assertion is disabled when the reset condition is
active, preventing false positives during the reset period.

(q != $past(q, 1)): This part of the property checks that the current state q
is not equal to its previous state ($past(q, 1)). This condition ensures that
the output toggles between different states on each rising edge of the clock.
```

7. **The active-low reset must be low for at least 6 clock cycles.**

```
property reset_duration;
  @(posedge clk) disable_iff(!reset)
    (reset |-> ##[6:$] !reset);
endproperty
```

`(reset |-> ##[6:$] !reset)` asserts that whenever reset is true `(|->)`, `for` the next 6 cycles `(##[6:$])`, reset must remain true, `and` after that, it should become false `(!reset)`.

8. **signal_a must not be asserted between signal_b and the following signal_c (from one cycle after the signal_b until one cycle after the signal_c)**

```
property no_a_between_b_and_c;
  @(posedge clk) disable_iff(!reset)
    !((signal_b |-> ##[1:$] signal_a) && (signal_c |-> ##[1:$] signal_a));
endproperty
```

`((signal_b |-> ##[1:$] signal_a) && (signal_c |-> ##[1:$] signal_a))` checks that `if` signal_b is asserted, then `for` the subsequent cycles `(##[1:$])`, signal_a should be false. Similarly, `if` signal_c is asserted, then `for` the subsequent cycles `(##[1:$])`, signal_a should be false.

`!` negates the entire condition, ensuring that signal_a is `not` asserted between signal_b `and` the following signal_c.

9. **If signal_a is down, signal_b may only rise for one cycle before the next time that signal_a is asserted.**

```
property one_cycle_rise_before_next_a;
  @(posedge clk) disable_iff(!reset)
    ((!signal_a) |-> (signal_b |-> ##1 !$changed(signal_b)) until
(signal_a));
endproperty
```

`(!signal_a |-> (signal_b |-> ##1 !$changed(signal_b)) until (signal_a))` checks that `if` signal_a is down `(!signal_a)`, then signal_b is allowed to rise `for` exactly one cycle `(##1 !$changed(signal_b)) before` the next `time` that signal_a is asserted.

10. signal_a must not be asserted together with signal_b or with signal_c.

```
property no_assertion_together;
  @(posedge clk) disable_iff(!reset)
    !(signal_a && (signal_b || signal_c));
endproperty
```

`!(signal_a && (signal_b || signal_c))` checks that signal_a is **not** asserted together **with** either signal_b **or** signal_c. If signal_a is asserted, the entire condition evaluates to false **if** either signal_b **or** signal_c is also asserted.