

Objectives

This proposal defines elaboration time system functions to query the inferred clocking event, disable expression, and enable condition:

- `$inferred_clock` returns the expression of the inferred clocking event.
- `$inferred_disable` returns the inferred disable expression.
- `$inferred_enable` returns the inferred enable condition.

The main motivation is to enable generic OVL-like checker libraries, where a single set of checkers can take action based on the context but also allow explicit clock/reset specification when needed. For example:

```
property p_never (test_expr, clk = $inferred_clock, rst =$inferred_disable)
```

This property can be used in procedural code with a clock and reset specified, within an always block inheriting its clock and reset, or within an always block overriding its clock and reset. This level of flexibility is not currently available without this extension.

One objection to this proposal may be that it is already possible to create functionality like this, by defining an unlocked property and having the designer specify a clock and reset where it is instantiated. But compare the two notations:

- OLD: `assert property(@(posedge clk) disable iff (rst) p_never(a));`
- NEW: `assert property(p_never (a, posedge clk, rst))`

The first is nonuniform and requires a separate syntax for the clock and reset, compared to the rest of the arguments. The second is much more convenient and intuitive for designers.

The situation gets even worse for multiclocked assertions, where without these extensions, one clock must be treated as the main clock and passed in with a separate syntax from any other clocks used:

- OLD: `assert property(@clka p(a, b, clkb));`
- NEW: `assert property p(a, b, clka, clkb);`

ADD

16.14.6 Inferred value functions (Note to the editor: Please adjust clause numbering as needed)

The following elaboration time system functions are available to query the inferred clocking event expression, disable expression, and enable condition.

- `$inferred_clock` returns the expression of the inferred clocking event.
- `$inferred_disable` returns the inferred disable expression.
- `$inferred_enable` returns the inferred enable condition.

The inferred clocking event expression is the current resolved event expression that can be used in a clocking event definition. It is obtained by applying clock flow rules to the point where `$inferred_clock` is called. If there is no current resolved event expression when `$inferred_clock` is encountered then an error shall be issued.

The inferred disable expression is the disable condition from the default disable declaration whose scope includes the call to `$inferred_disable` (See 16.15 [Note to editor: This is after introducing proposal #1648](#)). If the call to `$inferred_disable` is not within the scope of any default disable declaration, then the call to `$inferred_disable` returns `1'b0` (false).

The inferred enable condition is the expression defining the cumulative condition required to reach the current point within enclosing `if...else` and `case` blocks inside an `always` or `initial` block. This is the same as the contextually inferred enabling condition for verification statements (see 16.14.5). If there are no enclosing `if...else` or `case` blocks, then the cumulative condition is `1'b1` (true).

A call to an inferred expression function may only be used as the entire default value expression for a formal argument to a property or sequence declaration. A call to an inferred expression function shall not appear within the body expression of a property or sequence declaration. If a call to an inferred expression function is used as the entire default value expression for a formal argument to a property or sequence declaration, then it is replaced by the inferred expression as determined at the point where the property or sequence is instantiated. Therefore, if the property or sequence instance is the top-level property expression in a verification statement, the event expression that is used to replace the default argument `$inferred_clock` is that as determined at the location of the verification statement. If the property or sequence instance is not the top-level property expression in the verification statement then the event expression determined by clock-flow rules up to the instance location in the property expression is used as the default value of the argument.

Consider the following example:

```

module m(logic a, b, c, d, rst1, clk1, clk2);

logic rst;

default clocking @(negedge clk1); endclocking
default disable rst1;

property p_triggers(start_event, end_event, form, clk = $inferred_clock,
                    rst = $inferred_disable, en = $inferred_enable);
    @clk disable iff (rst)
    en throughout (start_event ##0 end_event[->1]) | => form;
endproperty

property p_multiclock(clkw, clkx = $inferred_clock, clky, w, x, y, z);
    @clkw w ##1 @clkx x | => @clky y ##1 z;
endproperty

a1: assert property (p_triggers(a, b, c));
a2: assert property (p_triggers(a, b, c, posedge clk1, 1'b0) );

always @(posedge clk2 or posedge rst) begin
    if (rst) ... ;
    else if (d)
        a3: assert property (p_triggers(a, b, c));
end

a4: assert property(p_multiclock(negedge clk2, , posedge clk1, a, b, c, d);

endmodule

```

The above code is equivalent to

```

module m(logic a, b, c, d, rst1, clk1, clk2);

logic rst;

```

```

a1: assert property @(negedge clk1) disable iff (rst1)
      1'b1 throughout a ##0 b[->1] | => c);
a2: assert property @(posedge clk1) disable iff (1'b0)
      1'b1 throughout a ##0 b[->1] | => c);

always @(posedge clk2 or posedge rst) begin
  if (rst) ... ;
  else if (d)
end

a3: assert property
  (
    @(posedge clk2) disable iff (rst1)
    (!bit'(rst!=1'b0) && d) |->
      (!bit'(rst!=1'b0) && d) throughout (a ##0 b[->1]) | => c
  );

end

a4: assert property @(negedge clk2) a ##1 @(negedge clk1) b | =>
      @(posedge clk1) c ##1 d);

endmodule

```

In assertion a1 the clock event is inferred from the default clocking, therefore `$inferred_clock` is **negedge** `clk1` for a1. In assertion a2 the event expression **posedge** `clk1` is passed to the formal argument `clk` in the instance of property `p_triggers`. Therefore, the `$inferred_clock` is not used for `clk` in that instance. In assertion a3 the clocking event is inferred from the event control of the **always** block, therefore `$inferred_clock` is **posedge** `clk2` for a3.

In assertion a4, as the property `p_multiclock` is instantiated in the **assert property** statement, `clkw` is replaced by the actual argument (**negedge** `clk2`), `clkx` by the default argument value `$inferred_clk` which is the default clocking clock (**negedge** `clk1`) at the location of the property instance in the assertion. The third clock, `clky`, is replaced by the actual argument (**posedge** `clk1`) because it is explicitly specified.

The disable condition `rst1` is inferred for assertions a1 and a3 from the default disable statement. Assertion a2 uses explicit reset value '0 in which case the **disable iff** statement could be omitted altogether in the equivalent assertion.

The inferred enabling condition for assertions a1 and a2 is `1'b1`, since they are not in the scope of any conditional statement; therefore, the **throughout** operator could also be omitted for these assertions. The inferred enabling condition for assertion a3 is `(!bit'(rst!=1'b0) && d)` because the assertion is in the **else** block of the `if (rst)` statement and `d` is from the `if` block and thus not negated.