

## Motivation

6.14.6 clarifies that *The triggered status of the sequence is set in the Observed region and persists through the remainder of the time step. In addition to using this method in assertion statements, it may be used in wait statements (see 9.4.3) or Boolean expressions outside a sequence context.* In addition, the LRM shows an example in program "check" with `if (e1.triggered)`.

**However, that are nuances as to how this .triggered method can and cannot be used**, even though it compiles and elaborates correctly. To demonstrate, let me use this simple example, with the key implication of incorrect usage identified in red and with the "<-----" pointer.

```
module trig;
  logic a=0, b=0, c=0, d, e;
  bit clk;
  sequence a_then_b;
  @(posedge clk) a ##[1:5] b;
  endsequence // req_ack
  initial forever #50 clk = !clk;

  // d is always set to 0 at @ (posedge clk)
  always @(posedge clk) d = a_then_b.triggered; // <-----BAD USAGE

  // e is always X in simulation
  always @(posedge clk) if(a_then_b.triggered) e=1; <----BAD USAGE

  // trig ??
  wire trig = a_then_b.triggered; // Illegal ??
  always @ (posedge clk) begin
    a0 : assert (std::randomize(a, b));
  end
  always begin
    wait(a_then_b.triggered)
    c=!c;
    @ (posedge clk);
  end
endmodule : trig
```

In the above example, **// d is always set to 0 at @ (posedge clk)**  
**always @(posedge clk) d = a\_then\_b.triggered; // <-----**

**// e is always X in simulation**  
**always @(posedge clk) if(a\_then\_b.triggered) e=1; <----**

We get those results because the sequence is clocked by the same (posedge clk) but its triggered property doesn't come true until the Observed region of the scheduler. But the @(posedge clk) block executes before that, in the Active region.

Other working solutions are:

```
event trig_ev;
always @(posedge clk) @(a_then_b.triggered) -> trig_ev;
```

```
// or
always @(a_then_b) ...
```

## ADD

Add after:

The process then displays the sequence that caused it to unblock, and then continues to execute at the statement labeled L2.

As the result of the scheduling mechanism of the triggered method, it is important to avoid constructs that lead to unexpected results. The following constructs demonstrate poor applications of the **triggered** method; these examples also apply to the **matched** method.

1) Improper usage of triggered in always procedure:

```
sequence a_then_b; @(posedge clk) a ##[1:5] b; endsequence
always @(posedge clk) begin : Improper_Usage_of_triggered
    d <= a_then_b.triggered; // <-----BAD USAGE
    if(a_then_b.triggered) e<=1; // <-----BAD USAGE
end : Improper_Usage_of_triggered
```

In the above case, d is always set to 0 and e is never updated because the triggered method is evaluated in the Observed region of the scheduler, but the **always** procedure executes before that, in the Active region.

2) Improper usage of triggered in continuous assignments

```
wire trig = a_then_b.triggered; /
```

The above case is illegal ?? results in ??

[Ben] I don't know the answer to this yet. Anyone knows?

Got this comment:

Not entirely useless—couldn't you use it like this?

```
A1: assert property (foo) else $display("Foo failed, and trig value is %d",trig);
```

Examples of the proper applications of the triggered method are shown below:

```
event trig_ev;
always begin
    wait(a_then_b.triggered);
    ...
end
always @(a_then_b.triggered) -> trig_ev;
always @(a_then_b) ...
```