Since the result starts simulation with value 0, and $past returns 0 for times before the start of simulation, the first assertions pass.

For another example we return to the sumItup thread of Chapter 5.1. The protocol timing diagram of the thread is repeated here in Figure 9.12. Based on this, the following properties can be written.

```
1  property goAssertedForOneClock;
2    @(posedge ck) disable iff (~reset_l)
3      $fell(go_l) |=> $rose(go_l);
4  endproperty
```

This sequence starts when go_l has fallen. In the timing diagram, this would be detected as a result of clock edge B because preponed values of go_l are 1 at clock edge A and 0 at clock edge B. Then, at clock edge C (one clock edge later as specified by the |=> implication) we should see go_l rise to 1 again.

This will work, however, the following assertion will also check the same condition:

```
1  property goAssertedForOneClockAlt;
2    @(posedge ck) disable iff (~reset_l)
3      ~go_l |=> go_l;
4  endproperty
```

That is, if go_l is asserted now, it will be not-asserted in the next state. These sequences are equivalent because you are only checking a pulse that is one clock period wide.

The $rise, $fall, and $changed functions are useful when you want to time a sequence from when an expression originally changes; you want the sequence to be edge-triggered. Consider a signal Q that, when asserted, is only asserted for four consecutive ticks; it becomes not-asserted after that. This would be captured by the following assertion:

```
1  property fourQ;
2    @(posedge ck) disable iff (~reset_l)
3      $rose(Q) |=> Q[*3] ##1 ~Q;
4  endproperty
```

In this case, the sequence only starts when Q rises. Then starting at the next tick, Q should remain asserted for three more consecutive ticks and then it should become not-asserted. This would not work with the following replacement for line 3:

```
1      Q |=> Q[*3] ##1 ~Q;
```

This is because this sequence would start anytime that Q is TRUE. Given that it would also start on the second through fourth Qs in the sequence, there wouldn't be three more Qs following the first. Thus these sequences would fail even though there is nothing wrong with the system. Not cool.

It is important to know that done is asserted only if inA is 0. i.e., that the output signalling is working. This would be detected at clock D with the following assertion:

```
1  property doneImpliesINAis0;
2    @(posedge ck) disable iff (~reset_l)
3      done |-> (inA == 0);
4  endproperty
```