Note that the assertion wouldn't be written the other way around. That is, inA==0 does not imply that done is 1. For instance, it's possible that inA is always 0 unless specific values are being sent to be summed. Thus if there were several states between when values were being sent, inA would be 0 but done should not be asserted. Remember that done is a signalling variable indicating to the thread connected on the output that it can load the output value. This is also captured in the next-state and output logic of the FSM. This assertion will be checking all of that logic.

Consider the situation now where done is asserted which should mean that sum becomes 0. done is asserted at clock D and the value of sum will be visible as a result of clock E.

```
1  property doneImpliesSUMbecomes0;
2    @(posedge ck) disable iff (~reset_l)
3      done |=> (sum == 0);
4  endproperty
```

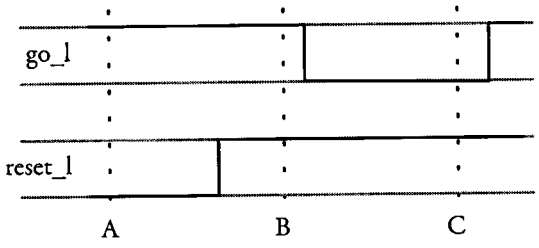

Figure 9.13 — reset_l Timing Relationship

Contrasting the last two properties, the first deals with two conditions that exist at the same time in the datapath. The second deals with two conditions that exist one clock tick apart. This can be seen in the timing diagram where the first is only active at clock D and the second spans clocks D and E.

Actions timed from reset are often specified using the deassertion (trailing) edge of the reset signal as the starting point. Figure 9.13 shows the timing of the reset_l signal with respect to the earliest timing of the go_l signal of the sumItUp thread; the go_l signal should not be asserted during a state where reset was active. The relationship of reset and go_l can be checked with the following property:

```
1  property goFollowsReset;
2    @(posedge ck)
3      $rose(reset_l) |-> (go_l == 1);
4  endproperty
```

In this case, we don't want the sumItUp thread to start during a state where reset was active. This sequence states that when we see the trailing edge of reset_l, then at clock tick B, go_l should be not-asserted (1).

Remember from the sumItUp thread implementation that reset_l is directly connected to the asynchronous reset of the sum register. The relationship of reset and sum can be checked with the following property:

```
1  property sumGetsReset;
2    @(posedge ck)
3      $rose(reset_l) |-> (sum == 0);
4  endproperty
```

In this case we check for the trailing edge of reset which is observed at clock tick B. At that point, sum should be 0 as a result of its asynchronous reset. Note that since the actions of