re: ballot comment #15, maintenance request 0286

802.1Q-Rev Project Editor, Mick Seaman

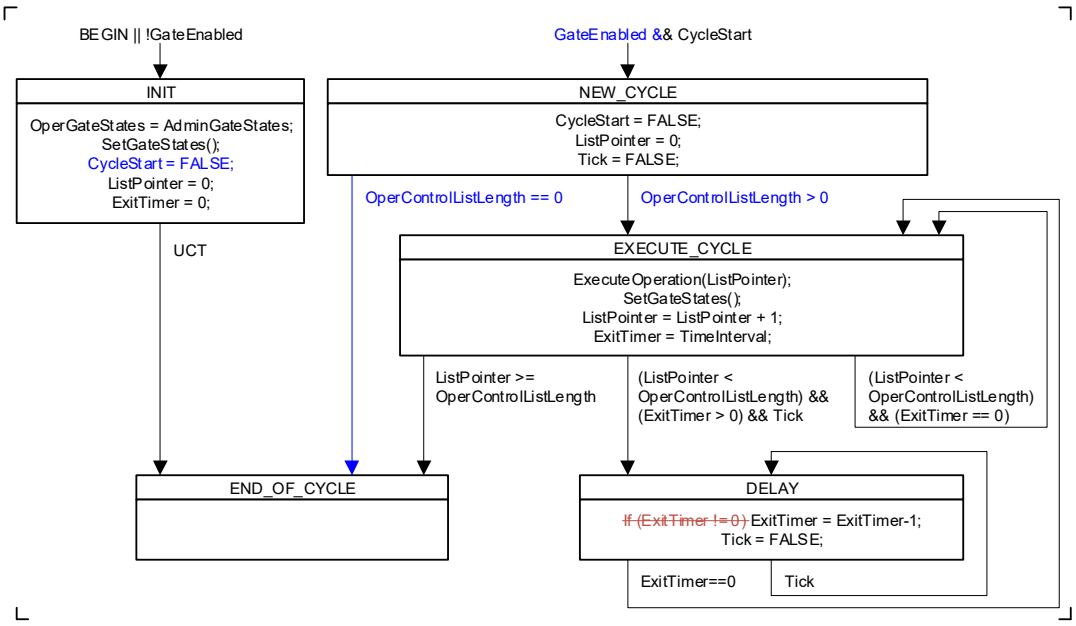
This document is a project editor's discussion of, and proposed resolution to, ballot comment #15, which itself was a request to implement maintenance request 0286. This comment and the others received on this ballot can be found in the proposed disposition of comments (later revisions of the proposed disposition may be founfd in the same directory):

https://www.ieee802.org/1/files/private/q-rev-drafts/d0/802-1Q-rev-d0-4-pdis-v1.pdf

An earlier draft of this document was discussed during the May 2021 interim sessions of the Maintenance Task Group (the BRC for this ballot).

I propose that we adopt the replacement Figure 8-20 shown in this document (with insert and ~~strikeout~~ for suggested technical changes) for the next WG recirculation ballot. Notes on the changes from the P802.1Q-Rev/D0.4 version follow the figure, and include notes on the changes from the first proposed change. Minor editorial change suggestions for readability are not shown. These include consistency of test conditions in transitions and consistency of variable setting order in states. Since the actions in each state are specified as occuring atomically and the state machine definition further requires that the entry condition to the state be true when that atomic execution takes place,, these readibility changes have no technical effect.

Please note that there is a substantive issue outstanding, which is (or will be) the subject of a further maintenance item: see discussion point 5 below.



The proposed replacement Fig 8-20 (above) differs slightly from the recorded resolution of maintenance request.

1. The state machine conventions (see Annex E of 802.1Q) do not specify behavior when both UCT and another condition are True for exit from a state. Other descriptions of this state machine logic have maintained that the choice of next state is undefined when multiple conditions apply. Annex E does specify the use of ELSE, which would resolve the issue in the present case. However explicitly representing the two cases of a zero and non-zero list would seem to be clearer in this case.

2. Given that clarification it would be possible to simply not leave the NEW_CYCLE state at all if the list length is zero, but this would risk leaving the state at an unaligned time when the list length was changed, and the END_OF_CYCLE state (which has no action, and could have been omitted) was presumably included for clarity of documentation.

3. The state machine behavior in the case of !GateEnabled and CycleStart both being True is undefined [BEGIN is special by convention (and it and open transitions in general should be better described in Annex E, but that should be left for another day) and does not suffer from this problem. In general state machines should follow the rule that correctness is a local property, and in this case of this machine the issue is not resolved within the set of related state machines as none of the machines appears to control the relationship between GateEnabled and CycleStart. I believe it is the case that NEW_CYCLE should not be entered if GateEnabled is False. The open entry to NEW_CYCLE is then best qualified GateEnabled && CycleStart. To guard against a randomly time start occuring due to a transition to GateEnabled True revealing a previous CycleStart it would also seem wise to reset CycleStart in INIT. Notionally the open transition to INIT continuously executes the INIT actions so that the variables set/reset in that states actions will have the assigned values up to the point that BEGIN becomes False and GateEnabled becomes True.

4. The test (ExitTimer != 0) was unnecessary in state DELAY. The state will not be entered if ExitTimer is not greater than zero. If there is any worry about failure to evaluate ExitTimer == 0 to exit the state after the state actions set Tick = FALSE then the Tick exit could be qualified with (ExitTimer != 0) , but that would be an unusual interpretation of the machines.

5. **The issue:** the current understanding of the editor this is that the desired timing of each successive ExecuteOperation(..) is meant to be (for some at least) in synchronized time, while the Tick specified is an approximation (and unless the Tick occurs at the finest granularity of synchronized time possible the whole notion of counting down using a Tick that is not itself synchronized with synchronized time would by its nature yield an approximate time, even if the duration of the Tick was precise). The Editor does not see how this issue can be fixed by simple changes to the figure and it probably has much wider consequences, outside of the scope of the current ballot resolution