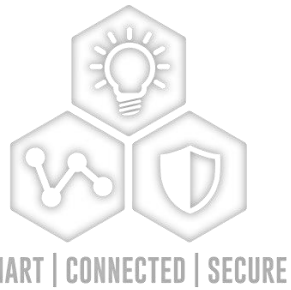
FTTM Change Summary for Comment Resolutions

A Leading Provider of Smart, Connected and Secure Embedded Control Solutions



Richard Tse

June 5, 2024

Major categories of changes

- Highlight “integrity” (per proposed PAR for P802.1ASed)
- Uniquely identify FTTM operation with single time input
- Change FTTM inputs from “domain” to “time”
- Only have one default algorithm
- Use 802.1Q’s state-machine structure
- Make states atomic w/o infinite state looping
- Define all variables/parameters

Highlight Integrity

- **Added subclauses on availability and on integrity and trust**

19.2.1 Availability of time

The continuous availability of time is enhanced by redundancy. For gPTP, this redundancy can be implemented by using multiple time domains, multiple time distribution paths, and multiple gPTP instances in Bridges and end stations.

19.2.2 Trust and Integrity of time

For this standard, a trusted time is one that passes a specified criterion that identifies it as being within a safe bound of a non-faulty time and is, thus, safe to use. This establishment of trust gives integrity to the time.

For gPTP, trust, and hence integrity, can be established through the comparison of the times coming from independent time sources and the observation that they match within the specified criterion. See 19.2.7.4 for an example of such a criterion.

Highlight integrity, operation with single time

- **Replaced “redundant” with “trust” in the algorithms**
- **Added the following to FTTM introduction (19.3)**

The availability and integrity scenarios supported by the FTTM are listed below.

— Enhanced availability and integrity scenario:

This scenario operates with multiple times and multiple domains.

In this mode, the FTTM supports increased availability and integrity.

— Enhanced availability and limited integrity scenario:

This scenario operates with multiple times and a single domain.

In this mode, the FTTM supports increased availability and potential time distribution integrity, but not GM integrity.

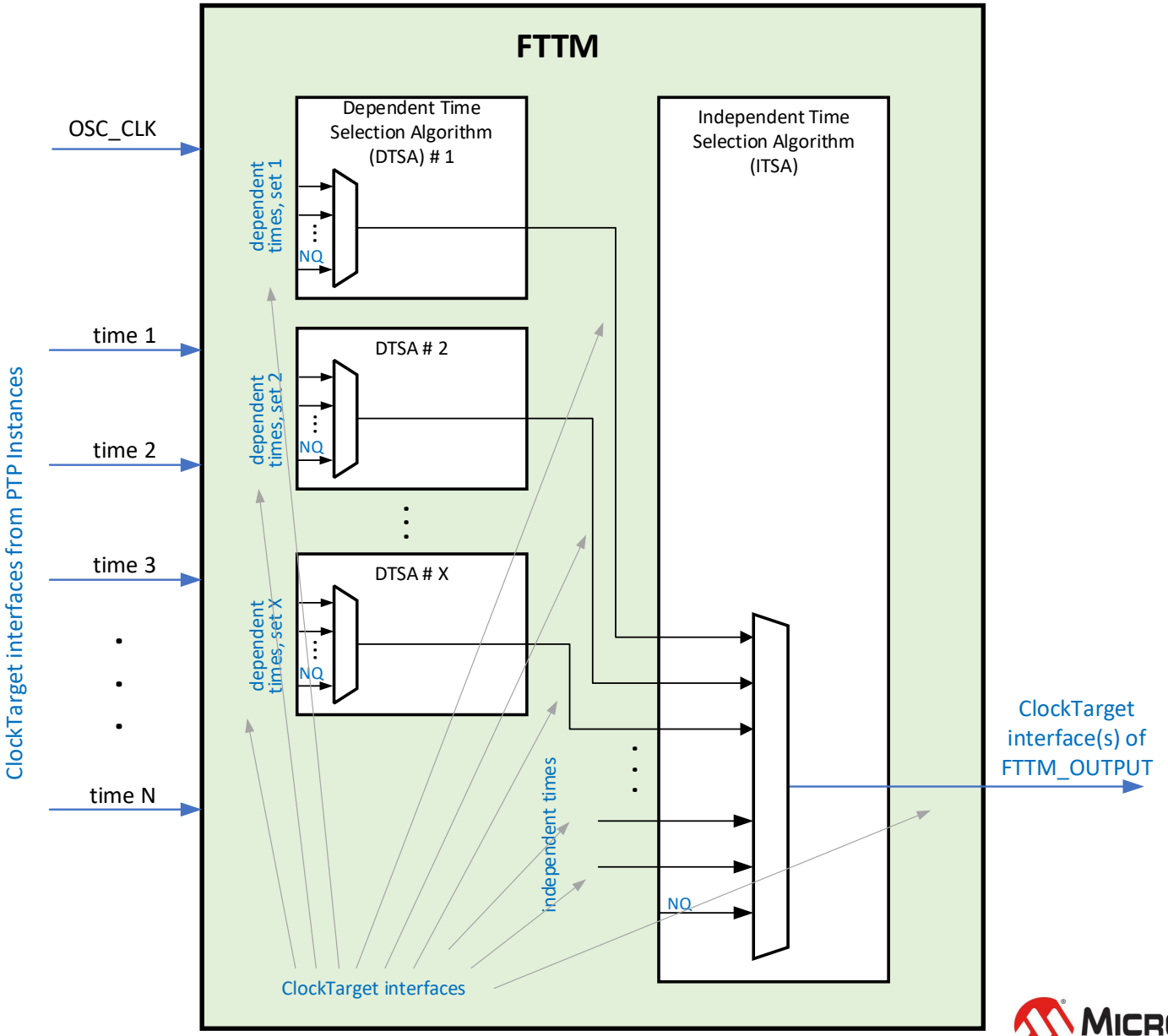
— Regular availability and no integrity scenario:

This scenario operates with a single time and, hence, a single domain.

In this mode, the FTTM does not support increased availability or integrity.

Change FTTM inputs from “domain” to “time”

- **Functional block diagram redrawn**
- **DDSA renamed to DTSA**
- **IDSA renamed to ITSA**



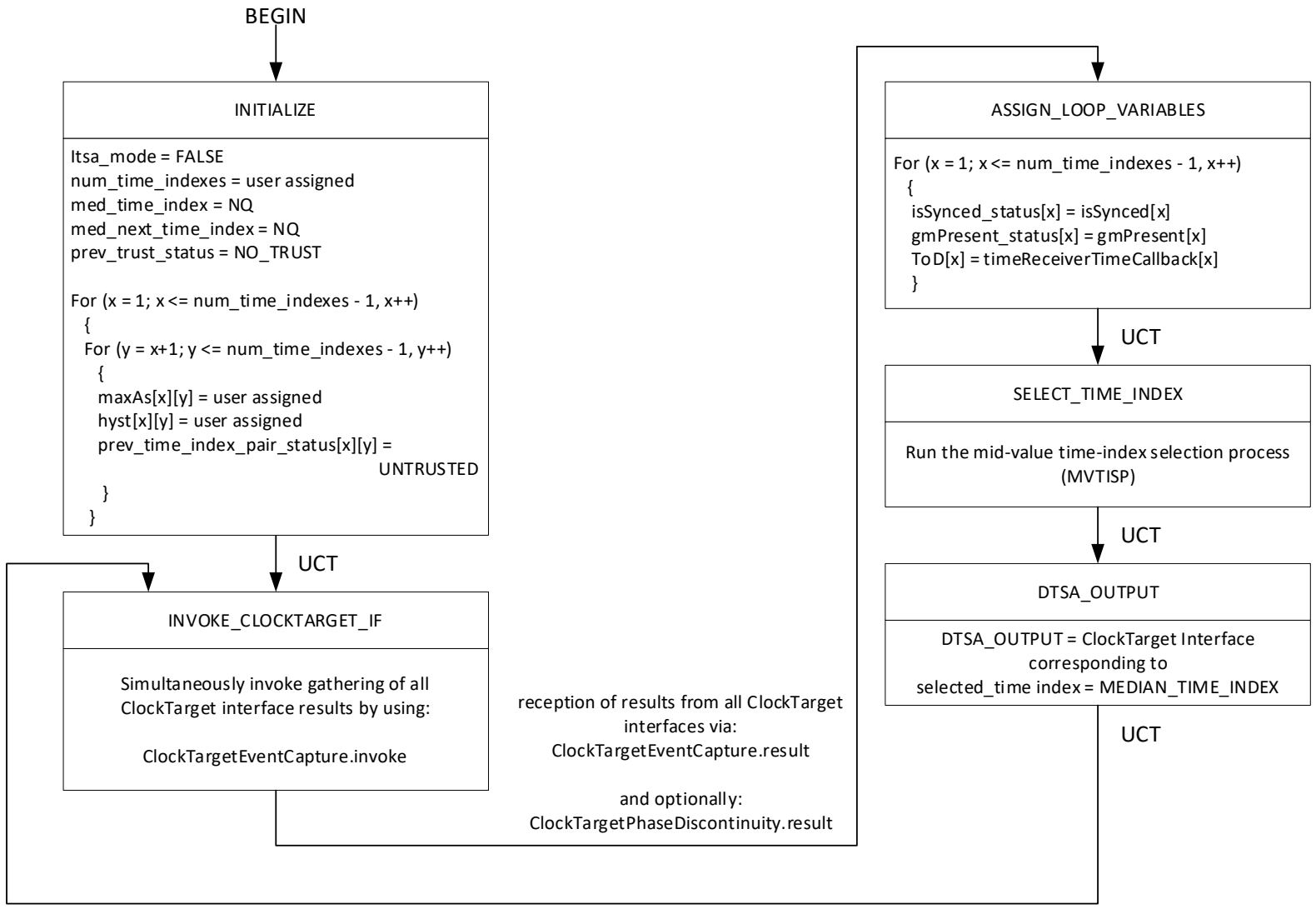
Only have one default “algorithm”

- **“Closest-pair algorithm” removed**
- **“Mid-value selection algorithm” retained and renamed to “Mid-value time-index selection process” (MVTISP)**
  - DTSA and ITSA are “algorithms”
  - MVTISP is the “process” used by the default DTSA and the default ITSA
  - “time-index” clarifies that the index is selected and not the time

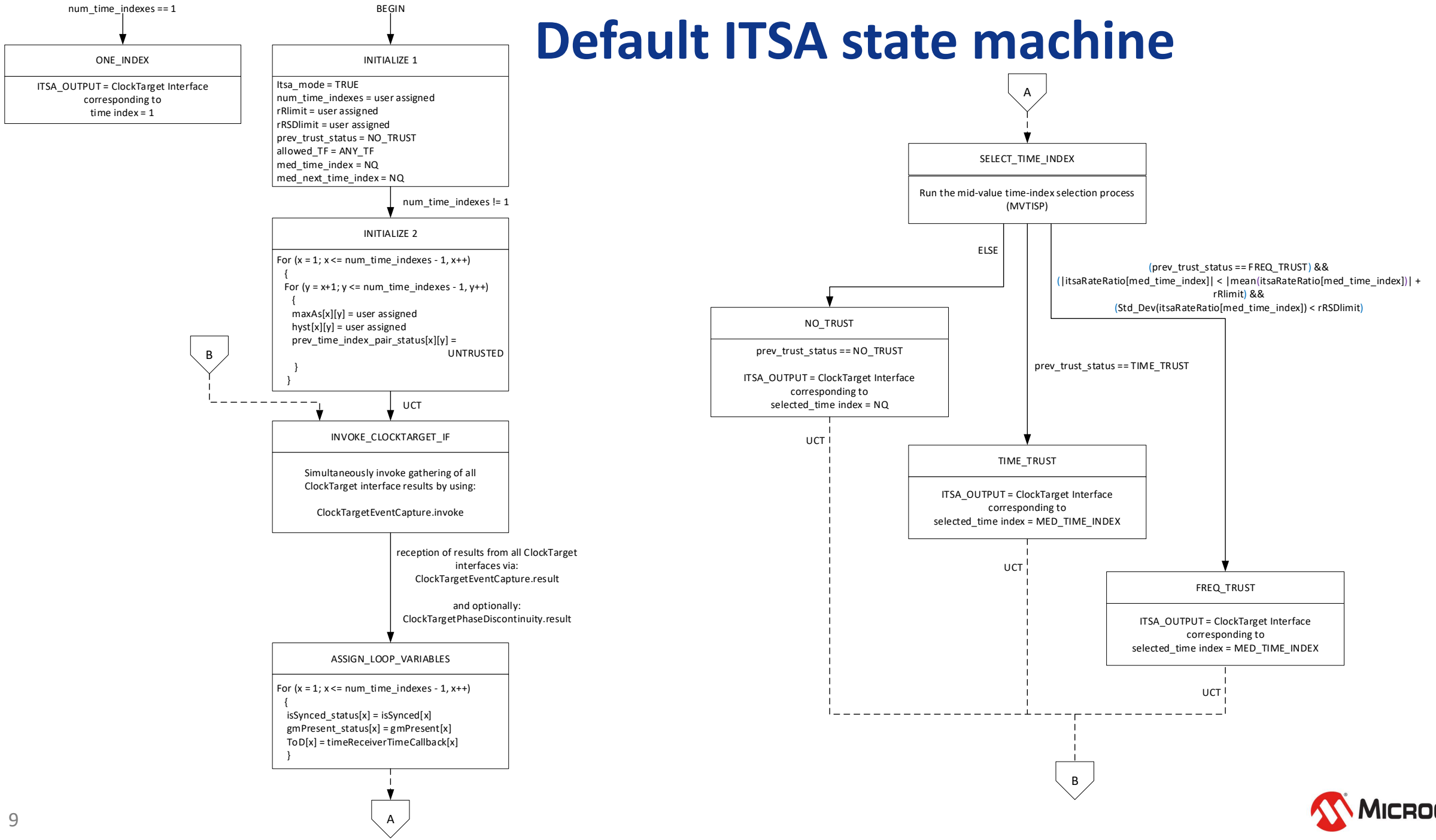
State-machines

- **Use 802.1Q's state-machine structure**
  - New state-machine defined for default DTSA
  - New state-machine defined for default ITSA
    - Formerly called the default FTTM state-machine
  - Mid-value time-index selection process (MVTISP)
    - Has no states
    - Is run in the SELECT_TIME_INDEX state of the new default DTSA/ITSA state-machines
    - Is shown by pseudo-code
- **States are atomic without infinite looping**
  - Default DTSA/ITSA state machines use the ClockTargetEventCapture application interface
  - ClockTargetEventCapture.invoke events are used to gather new time information (via ClockTargetEventCapture.response) and start a new round of state transitions

Default DTSA state machine



Default ITSA state machine



Mid-value time-index selection process

```
// ###########################################################################
// Gather the current skews between the ToDs of all the time indexes.
// ###########################################################################
For (x = 1; x <= num_time_indexes - 1, x++) {
    For (y = x + 1, y <= num_time_indexes, y++) {
        ToD_Diff[x][y] = |ToD[x] - ToD[y]|
        }
    }


// ###########################################################################
// Clear status before starting a new round of time index comparisons.
// ###########################################################################
trust_status = NO_TRUST
time_index_pair_status[x][y] = UNTRUSTED for all x and all y
time_index_status[x] = UNTRUSTED for all x
num_sorted = 1
exclude_time_index[x] = FALSE for all x


// ###########################################################################
// Find all trusted time indexes, considering hysteresis.
// ###########################################################################
For (x = 1, x <= num_time_indexes - 1, x++) {
    For (y = x + 1, y <= num_time_indexes, y++) {
        if ((ToD_Diff[x][y] <= maxAs[x][y] &&
             prev_time_index_pair_status[x][y] == UNTRUSTED) ||
             (ToD_Diff[x][y] <= (maxAs[x][y] + hyst[x][y]) &&
              prev_time_index_pair_status[x][y] == TRUSTED)) &&
            (isSynced_status[x] && gmPresent_status[x]) &&
            (isSynced_status[y] && gmPresent_status[y]))
        {
             // trust found for the pair
             trust_status = TIME_TRUST
             time_index_pair_status[x][y] = TRUSTED
             time_index_status[x] = TRUSTED
             time_index_status[y] = TRUSTED
             prev_time_index_pair_status[x][y] = TRUSTED
        }
        else
        {
             // trust not found for the pair
             prev_time_index_pair_status[x][y] = UNTRUSTED
        }
    }
}
```

```
// ###########################################################################
// If trust_status = TIME_TRUST, find time index with the mid-value ToD.
// ###########################################################################
// Trusted times detected.
If {trust_status == TIME_TRUST)
{
  // Update previous trust status.
  prev_trust_status = TIME_TRUST

  // Sort all trusted time indexes in order of their ToD, from smallest
  // to largest using two loops.
  // Outer loop iterates over all time indexes.
  For (x = 1, x <= num_time_indexes, x++) {
    min_value = 2^48 seconds  // Start with ToD value that is larger
                              // than any possible gPTP ToD value.
    // Inner loop finds and records the time index with the minimum ToD
    // value and excludes it from further iterations of the outer loop.
    For (y = 1, y <= num_time_indexes, y++) {
       if (time_index_status[y] == TRUSTED &&
           exclude_time_index[y] == FALSE &&
           ToD[y] <= min_value)
       {
         min_value = ToD[y] // record latest min ToD value found in the
                            // inner loop
         ordered_time_index[num_sorted] = y  // record latest time index
                                             // with min ToD value
       }
    }
    // Exclude latest time index with the min ToD value and add to sort index
    exclude_time_index[ordered_time_index[num_sorted]] = TRUE
    num_sorted = num_sorted + 1
  }

  // Get median trusted time index and the trusted time index with
  // the next higher ToD.
  // The lower time index is selected if the number of trusted time
  // indexes is even.
      med_time_index      = ordered_time_index[INT((num_sorted)/2)]
      med_next_time_index = ordered_time_index[INT((num_sorted)/2)+1]
}

// If itsa_mode, check for transition to or sustaining frequency trust state.
// If time trust just lost or if already using frequency trust, keep
// existing median time index values if corresponding PTP Instance
// is still valid.
// Set previous trust status to FREQ_TRUST.
else if {itsa_mode == TRUE &&
         prev_trust_status == (TIME_TRUST || FREQ_TRUST) &&
         isSynced_status[med_time_index) == TRUE &&
         gmPresent_status(med_time_index) == TRUE)
{
  med_time_index = med_time_index
  med_next_time_index = med_next_time_index
  prev_trust_status = FREQ_TRUST
}

// No time trust or frequency trust so set output time indexes to NQ
// and clear previous trust status to NO_TRUST.
else
{
  med_time_index = NQ
  med_next_time_index = NQ
  prev_trust_status = NO_TRUST
}
```

Define all variables/parameters

- **Management objects proposed in 14.23**
- **Define all variables for MVTISP**
  - Mostly done (see 19.3.3.2.1)
    - Need to define object types for all the vectors and arrays
- **Define all variables for default DTSA/ITSA state-machines**
  - Still in progress…

Outlook

- **Plan to finish the following within a few weeks:**
  - Have all variables for default DTSA/ITSA described with appropriate object types in a couple of weeks
  - Do further updates based on comments received from this meeting or subsequent interactions
- **Other:**
  - Need help on YANG model
  - Do we need objects to connect FTTM inputs to DTSA/ITSA, DTSAs to ITSA, and ITSA to FTTM output?
    - Management objects for this are proposed in 14.23

Thank You

MICROCHIP