7. Clock synchronization

7.1 Clock-synchronization overview

7.1.1 Clock synchronization services

Clock synchronization involves the transmission and reception of clockSync frames interchanged between adjacent-span stations, using the state machines defined within this clause. When considered as a whole, these provide the following services:

a) Election. The grand clock master is elected from among the grand-clock-master capable stations.

b) Isolation. Timeouts identify the boundaries, beyond which RE services are not supported.

c) Clock-sync. Clock-slave stations are synchronized to the grand master station's time reference.

7.1.2 Grand-master precedence

Grand-master precedence is based on the concatenation of multiple fields, as illustrated in Figure 7.1. The *portTag* value is used within bridges, but is not transmitted between stations.

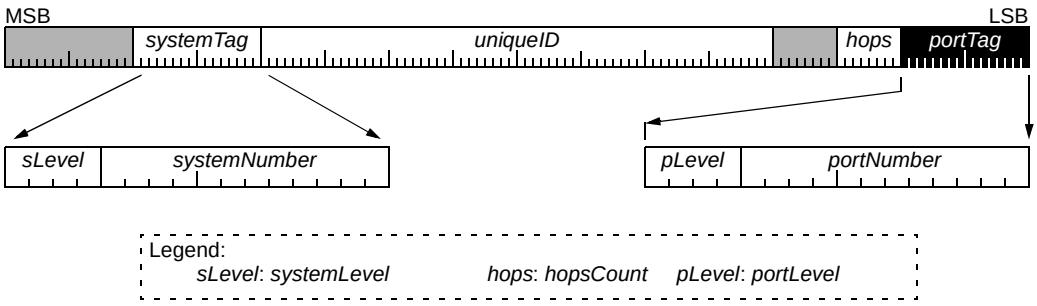


Figure 7.1—Grand-master precedence

7.1.3 Clock-synchronization agents

Clock-synchronization information conceptually flows from a grand-master station to clock-slave stations, as illustrated in Figure 7.2a. A more detailed illustration shows pairs of synchronized clock-master and clock-slave components, as illustrated in Figure 7.2b.

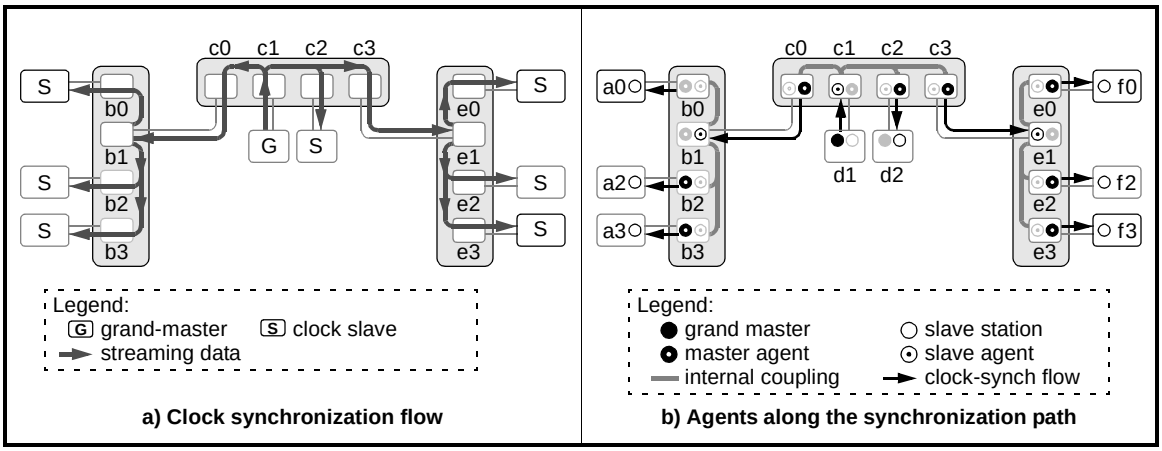


Figure 7.2—Hierarchical flows

7.1.4 Clock-synchronized pairs

Each bridge port provides clock-master and clock-slave agents, although both are never simultaneously active. External communications (see 7.2b) synchronize clock-slaves to clock-masters, as listed in Table 7.1.

Table 7.1—External clock-synchronization pairs

Grand master	Clock master agent	Clock slave agent	Clock slave	Type of synchronization
d1	–	c1	–	Station-to-bridge
–	c0	b1	–	Bridge-to-bridge
–	c3	e1	–	
–	b0	–	a0	Bridge-to-station
–	b2	–	a2	
–	b3	–	a3	
–	c2	–	d2	
–	e0	–	f0	
–	e2	–	f2	
–	e3	–	f3	

Internal communications distribute synchronized time from clock-slave agents b1, c1, and e1 to the other clock-master agents on bridgeB, bridgeC, and bridgeE respectively. However, bridge-internal port-to-port synchronization protocols are implementation-dependent and beyond the scope of this working paper.

Within a clock-slave, precise time synchronization involves adjustments of timer offset and rate values. The adjustments of the timer's offset is called offset synchronization (see 7.1.6); the adjustments of the timer's rate is called rate synchronization (see 7.1.8). Both involve calibration of local clock-master/clock-slave differences and the propagation of cumulative differences in the clock-slave direction, as described by the C code of Annex J.

Time synchronization yields distributed but closely-matched *timeOfDay* values within stations and bridges. No attempt is made to eliminate intermediate jitter with bridge-resident jitter-reducing phase-lock loops (PLLs,) but application-level phase locked loops (not illustrated) are expected to filter high-frequency jitter from the supplied *timeOfDay* values

7.1.5 Clock-synchronization intervals

Clock synchronization involves the processing of periodic events. Three distinct time periods are involved, as listed in Table 7.2. The clock-period events trigger the update of free-running timer values; the period affects the timer-synchronization accuracy and is therefore constrained to be small.

Table 7.2—Clock-synchronization intervals

Name	Time	Description
clock-period	< 20 ns	Time between timer-register value updates
send-period	10 ms	Time between sending of periodic clockSync frames between adjacent stations
slow-period	100 ms	Time between computation of clock-master/clock-slave rate differences

The send-period events trigger the interchange of clockSync frames between adjacent stations. While a smaller period (1 ms or 100 μs) could improve accuracies, the larger value is intended to reduce costs by allowing computations to be executed by inexpensive (but possibly slow) bridge-resident firmware.

The slow-period events trigger the computation of timer-rate differences. The timer-rate differences are computed over two slow-period intervals, but recomputed every slow-period interval. The larger 100 ms (as opposed to 10 ms) computation interval is intended to reduce errors associated with sampling of clock-period-quantized slow-period-sized time intervals.

7.1.6 Offset synchronization

Offset synchronization involves a subset of the time-synchronization components, as illustrated by white-colored boxes in Figure 7.5. Each clock consists of a progressing *timeOfDay* value, whose offset and rate are periodically adjusted. The free-running *flexTimer* timer is never reset; synchronization of stationE (with respect to stationD) is accomplished by adjustments to the *flexOffset* and *flexRate* values within stationE.

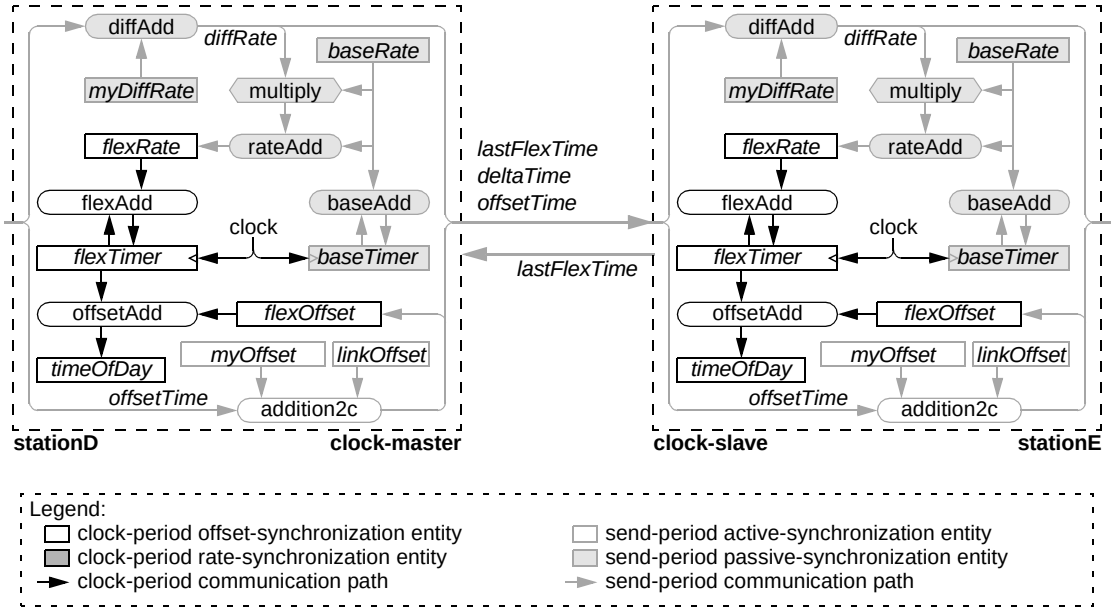


Figure 7.3—Offset synchronization

The offset-synchronization protocols interchange parameters periodically, possibly every 10 ms. The *lastFlexTime*, *deltaTime*, and *offsetTime* values are sent periodically from the clock-master to the clock-slave. The *lastFlexTime* is sent periodically from the clock-slave to the clock-master, providing information necessary for the clock-master to produce a *deltaTime* value for the clock-slave.

The offset-compensation protocols for stationE adjust its *myOffset* value so that the instantaneous values of *stationE.timeOfDay* and *stationD.timerOfDay* are the same. Computations are performed on clockStrobe reception and clockStrobe transmission.

As an option, an additional *linkOffset* value is available to manually compensate for mismatched transmit-link/receive-link duplex-cable delays on the clock-master side. The *linkOffset* value is expected be manually set when the cable mismatch is known through other mechanisms, such as specialized cable-characterization equipment.

The station's *offsetTime* value is constructed by adding the received *clockStrobe.offsetTime*, local *myOffset*, and local *linkOffset* values. This revised *clockStrobe.offsetTime* value is used within each station and is passed to the downstream neighbor (when such a neighbor is present).

7.1.7 Cascaded offsets

The concept of cascaded offset values can be better understood by considering a simple 3-bridge example, as illustrated in Figure 7.4. The slave-agent in bridgeB is synchronized to its neighbor grand-master via clockSync frames sent on the connecting bidirectional span. Within bridgeB, the clock-slave agent passes the time directly to the clock-master agent. The slave-agent in bridgeC is synchronized to its neighbor clock-master via clockSync frames sent on the connecting bidirectional span. Other ports are similarly synchronized, thus synchronizing the right-most clock-slave station to the left-most grand-master station.

Parameter					
name	grand-master	bridgeB	bridgeC	bridgeD	clock-slave
number	1	2	3	4	5
flexTimer	100	500	-300	200	400
myOffset	10	-400	800	-500	-200
flexOffset	10	-390	410	-90	-290
timeOfDay	110				

Representing:
myOffset[k+1] = flexTimer[k]–flexTimer[k+1];
flexOffset[k+1] = flexOffset[k]+myOffset[k+1];
timeOfDay[k] = flexTimer[k] + flexOffset[k];

Figure 7.4—Cascaded offsets (a possible scenario)

To simplify this illustration, consider only the seconds portion of the *flexTimer* value within each station or bridge. These values may differ dramatically, based (perhaps) on the power-cycling or topology formation sequence. Thus, the grand-master could have a *flexTimer* value of 100 while its bridgeB neighbor has a *flexTimer* value of 500.

The *myOffset* value within bridgeB will converges to the value of –400, representing the differences between grand-master and bridgeB *flexTimer* values. The *flexOffset* value received from the grand-master is added to this *myOffset* value, so that bridgeB's *flexOffset* becomes –390. The *flexTimer* and *flexOffset* values are added, to yield a resultant bridgeB *timeOfDay* value of 110, properly synchronized to the identical grand-master's value.

Similarly, bridgeC is synchronized to bridgeB, bridgeD to bridgeC, and the clock-slave to bridgeD.

7.1.8 Rate synchronization

Rate synchronization involves a subset of the time-synchronization components, as illustrated by white-colored boxes in Figure 7.5. The free-running *baseTimer* timer facilitate the determination of rate differences between the clock-master and clock-slave stations.

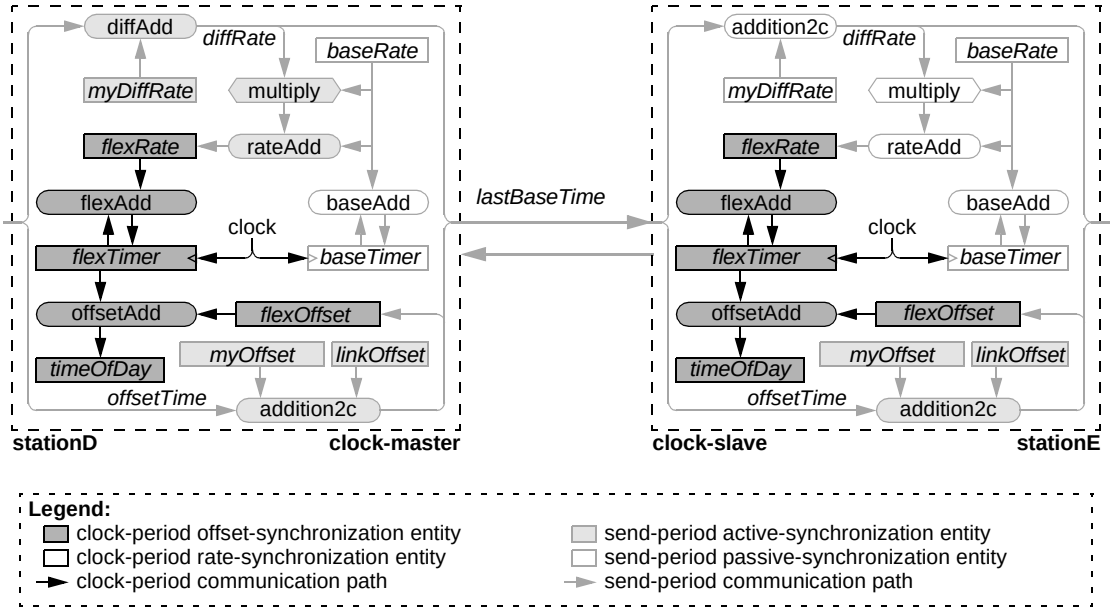


Figure 7.5—Rate synchronization

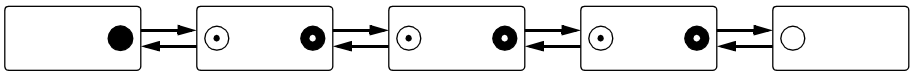
The rate-synchronization protocols interchange parameters periodically, but less frequently than the offset-synchronization protocols, possibly every 100 ms. The *lastBaseTime* value is sent periodically from the clock-master to the clock-slave. Nothing is returned from the clock-slave station.

The rate-compensation protocols for stationE adjust its *myDiffRate* value to accommodate for differences between the *stationD.baseTimer* and *stationE.baseTimer* rates. Computations are performed on clockStrobe reception and clockStrobe transmission.

The station's *diffRate* value is constructed by adding the received *clockStrobe.diffRate* and local *myDiffRate* values. This revised *clockStrobe.diffRate* value is used within each station and is passed to the clock-slave side neighboring station (if present).

7.1.9 Cascaded rate differences

The concept of cascaded rate values can be better understood by considering a simple 3-bridge example, as illustrated in Figure 7.6. Within this figure, the *myDiffRateN* and *diffRateN* represent parts-per-million (PPM) normalized values of *myDiffRate* and *diffRate* respectively.

Parameter					
name	grand-master	bridgeB	bridgeC	bridgeD	clock-slave
number	1	2	3	4	5
crystal deviation	+10 PPM	+100 PPM	–100 PPM	–75 PPM	+75 PPM
myDiffRateN	0 PPM	–90 PPM	200 PPM	–25 PPM	–150 PPM
diffRateN	0 PPM	–90 PPM	110 PPM	+85 PPM	–65 PPM
flexTimer deviation	10 PPM				

Representing:
myDiffRateN[k+1] = flexRate[k]–flexRate[k+1];
diffRate[k+1] = diffRate[k]+myDiffRate[k+1];
flexTimerDeviation[k] = crystalDeviation[k] + diffRate[k];

Figure 7.6—Cascaded rate differences (a possible scenario)

The slave-agent in bridgeB is synchronized to its neighbor grand-master via clockSync frames sent on the connecting bidirectional span. Within bridgeB, the clock-slave agent passes the time directly to the clock-master agent. The slave-agent in bridgeC is synchronized to its neighbor clock-master via clockSync frames sent on the connecting bidirectional span. Other ports are similarly synchronized, thus synchronizing the right-most clock-slave station to the left-most grand-master station.

To simplify this illustration, consider only the parts-per-million (PPM) normalized rate values within each station or bridge. These values may differ significant, based (perhaps) on the nominal value or ambient temperature. Thus, the grand-master could have a crystal deviation of +10 while its bridgeB neighbor has a crystal deviation of +100.

The *myDiffRate* value within bridgeB will converges to the value of –90 PPM, representing the differences between grand-master and bridgeB crystal accuracies. The *diffRate* value received from the grand-master is added to the *myDiffRate* value, so that bridgeB's *diffRate* becomes –90 PPM. The *diffRate* and crystal deviation values are additive, yielding a resultant bridgeB *flexTimer* deviation of 10 PPM, properly synchronized to the identical grand-master's value.

Similarly, the rate of bridgeC is synchronized to bridgeB, bridgeD to bridgeC, and the clock-slave to bridgeD.

7.1.10 Rate-difference effects

If the absence of rate adjustments, significant *timeOfDay* errors can accumulate between send-period updates, as illustrated on the left side of Figure 7.7. The 2 ms deviation is due to the cumulative effect of clock drift, over the 10 ms send-period interval, assuming clock-master and clock-slave crystal deviations of −100 PPM and +100 PPM respectively.

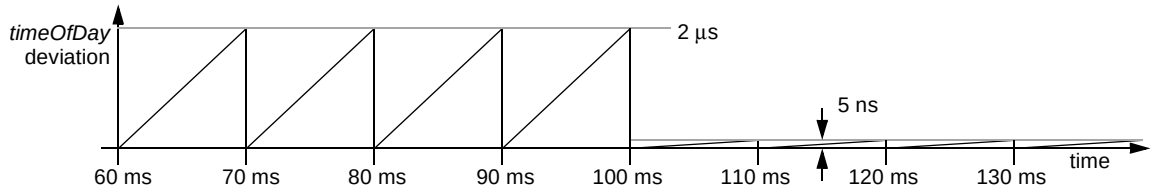


Figure 7.7—Rate-adjustment effects

While this regular sawtooth is illustrated as a highly regular (and thus perhaps easily filtered) function, irregularities could be introduced by changes in the relative ordering of clock-master and clock-slave transmissions, or transmission delays invoked by asynchronous frame transmissions. Tracking peaks/valleys or filtering such irregular functions are thought unlikely to yield similar *timeOfDay* deviation reductions.

The differences in rates could easily be reduced to less than 1 PPM, assuming a 200 ms measurement interval (based on a 100 ms slow-period interval) and a 100 ns arrival/departure sampling error. A clock-rate adjustment at time 100 ms could thus reduce the clock-drift related errors to less than 5 ns. At this point, the timer-offset measurement errors (not clock-drift induced errors) dominate the clock-synchronization error contributions.

7.1.11 *flexTimer* implementation example

The selection of the best time-of-day format is oftentimes complicated by the desire to equate the clock format granularity with the granularity of the implementation's 'natural' clock frequency. Unfortunately, the 'natural' frequency within a multimodal {1394, 802-100Mb/s, 802.3 1Gb/s} implementation is uncertain, and may vary based on vendors and/or implementation technologies.

The difficulties of selecting a 'natural' clock-frequency can be avoided by realizing that any clock with sufficiently fine resolution is acceptable. Flexibility involves using the most-convenient clock-tick value, but adjusting the timer advance rate associated with each clock-tick occurrence.

The same mechanism easily supports both near-arbitrary clocking rates and fine-grained rate-adjustments, needed to support timer-synchronization protocols, as illustrated in Figure 7.8. Within this figure, the shaded bytes represent values that can safely be hardwired to zero with insignificant loss of accuracy.

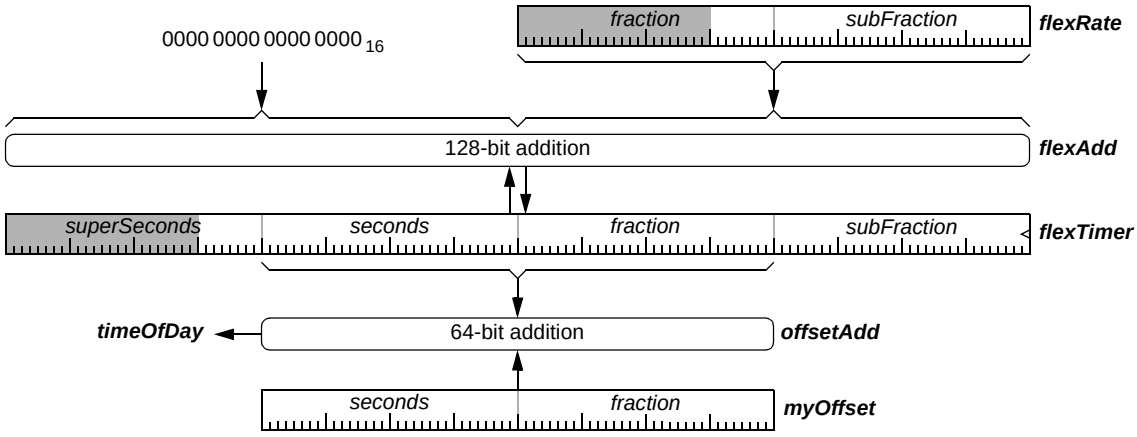


Figure 7.8—*flexTimer* implementation example

This illustration is not intended to constrain implementations, but to illustrate how the system's clock and timer formats can be optimized independently. This allows the *timeOfDay* timer format to be based on arithmetic convenience, timing precision, and years-before-overflow characteristics (see Annex E).

7.1.12 An alternative *baseTimer* implementation

An alternative implementation could implement the *baseTimer*-related circuitry in hardware. For such implementations, the associated firmware can be simplified, since the multiplies are eliminated from the most frequently executed loop (see Annex J).

A possible *baseTimer* hardware implementation is much simpler than the fully adjustable timer implementation, due to the absence of offset-compensation, rate-compensation, and seconds-accumulation hardware, as illustrated in Figure 7.9. Within this figure, the shaded bytes represent values that can safely be hardwired to zero with insignificant loss of accuracy.

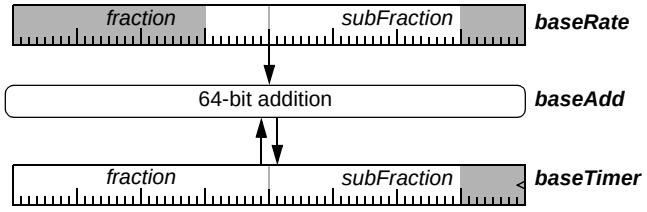


Figure 7.9—*baseTimer* implementation example

7.2 Terminology and variables

7.2.1 Common state machine definitions

The following state machine inputs are used multiple times within this clause.

CYCLES
: The number of isochronous cycles within each second; defined to be 8,000.

NULL
: Indicates the absence of a value and (by design) cannot be confused with a valid value.

queue values
: Enumerated values used to specify shared queue structures.
  - Q_CRX_SYNC—The identifier associated with the received clockSync frames.
  - Q_CTX_SYNC—The identifier associated with the transmitted clockSync frames.
  - Q_ARX_REQ*—The identifier associated with the received subscription request frames.
  - Q_ATX_REQ*—The identifier associated with the transmitted subscription request frames.
  - Q_ATX_RES*—The identifier associated with the transmitted ResponseError frames.
  - Q_ARX_STR*—The identifier associated with the talker agent’s streaming input.
  - Q_ATX_STR*—The identifier associated with the talker agent’s streaming output.

NOTE—Those queue identifiers with an ‘*’ are used in other clauses, but are described above. This allows all queue identification values in one location, rather than interleaving their definitions throughout this working paper.

7.2.2 Common state machine variables

One instance of each variable specified in this clause exists in each port, unless otherwise noted.

currentTime
A value representing the current time.

7.2.3 Common state machine routines

ClockSyncArrived(*stationInfoPtr, portInfoPtr*)
Snapshots the clockSync frame arrival time, on specified station and port (see Annex J).
ClockSyncDeparted(*stationInfoPtr, portInfoPtr*)
Snapshots the clockSync frame departure time, on specified station and port (see Annex J).
ClockSyncTransmit(*stationInfoPtr, portInfoPtr, clockSyncPtr*)
Forms a clockSync frame for transmission (see Annex J).
ClockSyncReceive(*stationInfoPtr, portInfoPtr, clockSyncPtr, rateAdjust*)
Processes a clockSync frame after reception (see Annex J).
Dequeue(*queue*)
Returns the next available frame from the specified queue.
frame—The next available frame.
NULL—No frame available.
Enqueue(*queue, frame*)
Places the frame at the tail of the specified queue.
Min(*value1*, *value2*)
Returns the numerically smaller of two values.
QueueEmpty(*queue*)
Indicates when the queue has emptied.
TRUE—The queue has emptied.
FALSE—(Otherwise.)
TimerTick(*stationInfoPtr*)
Updates *flexTimer* (and *baseTimer*) entities on each clock tick (see Annex J).

7.2.4 Variables and literals defined in other clauses

This clause references the following parameters, literals, and variables defined in Clause TBD:

TBDs

7.3 Clock synchronization state machines

7.3.1 ClockCore state machine

7.3.1.1 ClockCore state machine definitions

The following state machine inputs are used multiple times within this clause:

None.

7.3.1.2 ClockCore state machine variables

One instance of each variable specified in this clause exists in each port, unless otherwise noted.

clockPeriod
The duration of a synchronized timer update interval.
clockPeriod < 20 ns
currentTime
See 7.2.2.
clockDeviation
The deviation from nominal frequency of the station-local crystal-stabilized clock.
msCount
A count that is incremented at the end of each 1 millisecond interval.
msTime
The start time of the current 1 millisecond timing interval.
nominalFrequency
The nominal frequency of the station-local crystal-stabilized clock.
tickTime
A time snapshot taken at the start of each clockPeriod interval.

7.3.1.3 ClockCore state machine routines

TimerTick(*stationInfoPtr*)
See 7.2.3.

7.3.1.4 ClockCore state table

The ClockAgent state machine calls other C-code routines, as specified in Table 7.3. A purpose of the ClockAgent state machine is to ensure correctness of the other routines, by ensuring their indivisible executions. The notation used in the state table is described in 3.4.

Table 7.3—ClockCore state table

Current		Row	Next	
state	**condition**		**action**	**state**
START	—	1	clockPeriod = 1.0 / (nominalFrequency * (1.0 + (deviation / 1000000.)))	START
FIRST	(currentTime – tickTime) >= clockPeriod	1	TimerTick(siPtr); tickTime = currentTime;	FIRST
	(currentTime – msTime) >= .001	2	msTime = currentTime; msCount += 1;	
	—	3	—	

Row START-1: Compute the *clockPeriod*, based on the nominal frequency and deviation.

Row FIRST-1: Update the *flexTimer* and *baseTimer* once every *clockPeriod* interval.
Row FIRST-2: Update the millisecond counter once every millisecond.
Row FIRST-3: Otherwise, no operations are performed.

7.3.2 ClockPort state machine

7.3.2.1 ClockPort state machine definitions

The following state machine inputs are used multiple times within this clause.

None.

7.3.2.2 ClockPort state machine variables

One instance of each variable specified in this clause exists in each port, unless otherwise noted.

frame
The contents of a clockSync frame.
lastInterval
A saved value of *rateInterval*, when the last rate-interval update was scheduled to occur.
rateInterval
A counter that increments on transitions of 100 ms rate-update intervals.
rateCount
A milliseconds-snapshot taken during the clockSync receive processing.
The *rateCount* value paces the relatively infrequent rate-update computations.
rxClockLast
The previously observed value of *rxClockSync*, used to detect changes in this toggling value.
rxClockSync
An indication whose value is toggled on the PHY-sensed arrival of each clockSync frame.
This value is toggled before a frame can be dequeued from the Q_CRX_SYNC queue.
rxCount
A milliseconds-snapshot taken during clockSync receive processing.
The *rxCount* value paces the detection of clockSync-silence timeouts.
sendCount
A milliseconds-snapshot taken during the clockSync transmission processing.
The *sendCount* value paces the normal clockSync frame transmissions.
selectCount
A value that tracks *siPtr–>selectCount*, to facilitate detection of station-precedence changes.
sinkCount
A milliseconds-snapshot taken during the clockSync reception and timeout processing.
The *sinkCount* value paces the infrequent clockSync-reception timeout processing.
txClockSync
An indication whose value is toggled on the PHY-sensed departure of each clockSync frame.
This value is toggled shortly after a frame has departed from the Q_CTX_SYNC queue.
txClockLast
The previously observed value of *txClockSync*, used to detect changes in this toggling value.

7.3.2.3 ClockPort state machine routines

ClockSyncArrived(*stationInfoPtr, portInfoPtr*)
ClockSyncDeparted(*stationInfoPtr, portInfoPtr*)
ClockSyncTransmit(*stationInfoPtr, portInfoPtr, clockSyncPtr*)
ClockSyncReceive(*stationInfoPtr, portInfoPtr, clockSyncPtr, rateAdjust*)
ClockSyncTransmit(*stationInfoPtr, portInfoPtr, clockSyncPtr*)
See 7.2.3.
Dequeue(*queue*)
Enqueue(*queue*, *frame*)
See 7.2.3.

7.3.2.4 ClockPort state table

The ClockPort state machine calls other C-code routines, as specified in Table 7.4. A purpose of the ClockPort state machine is to ensure correctness of the other routines, by ensuring their timely and indivisible executions. The notation used in the state table is described in 3.4.

Table 7.4—ClockPort state table

Current		Row	Next	
state	**condition**		**action**	**state**
START	(frame = Dequeue(Q_CRX_SYNC)) != NULL	1	rxCount = sendCount;	NEAR
	rxClockSync != rxClockLast	2	ClockSyncArrived(siPtr, piPtr); rxClockLast = rxClockSync	START
	txClockSync != txClockLast	3	ClockSyncDeparted(siPtr, piPtr); txClockLast = txClockSync	
	selectCount != siPtr–>selectCount && (msCount – sendCount) >= 1	4	selectCount = siPtr–>selectCount; sendCount = siPtr–>msCount; ClockSyncTransmit(siPtr, piPtr, &frame); Enqueue(Q_CTX_SYNC, frame);	
	(siPtr–>msCount – sendCount) >= 10	5		
	(siPtr–>msCount – rateCount) >= 100	6	rateInterval += 1;	
	(siPtr–>msCount – sinkCount) >= 50	7	ClockSyncReceive(siPtr, piPtr,NULL, 0); sinkCount = siPtr–>msCount;	
	—	8	—	
NEAR	lastInterval != rateInterval	1	ClockSyncReceive(siPtr, piPtr, &frame, 1); lastInterval = rateInterval;	FINAL
	—	2	ClockSyncReceive(siPtr, piPtr, &frame, 0);	
FINAL	—	1	rxCount = sinkCount = siPtr–>msCount;	START

Row START-1: When a clock-sync frame arrives, mark its arrival time and process.
Row START-2: Process the PHY-generated signal to determine when the clockSync frame arrived.
Row START-3: Process the PHY-generated signal to determine when the clockSync frame departed.
Row START-4: Transmit quickly when the grand-master selection is changing.
Row START-5: Transmit routinely when the grand-master selection has stabilized.
Row START-6: Trigger the rate adjustments on approximate 100 ms intervals.
Row START-7: A port timeout occurs in the continued absence of clockSync frame arrivals.
Row START-8: Otherwise, wait for the next event to occur.

Row NEAR-1: Restart the rate interval condition after the last rate-measurement completion.
Row NEAR-2: Otherwise, process the received clockSync frame without rate-interval measurements.

Row FINAL-1: Restart the receive-timeout counter after processing each clockSync frame.