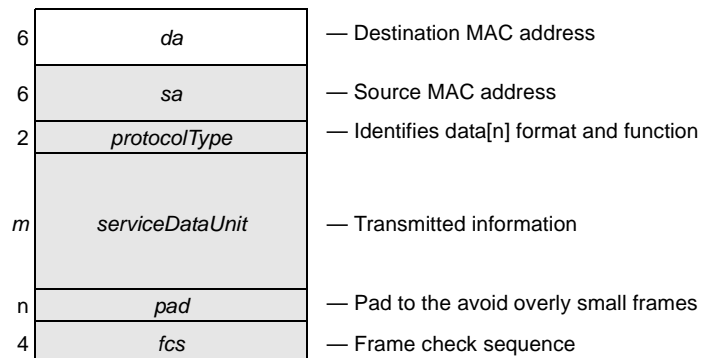


## 6. Frame formats

### 6.1 ClassA frames

#### 6.1.1 ClassA frame fields

A classA frame differs from other frames in the format of its multicast *da* (destination address), as illustrated in Figure 6.1.



**Figure 6.1—ClassA frame formats**

**6.1.1.1 *da*:** A 6-byte (destination address) field that specifies a multicast address associated with the stream.

**6.1.1.2 *sa*:** A 48-bit (source address) field that specifies the local station sending the frame. The *sa* field contains an individual 48-bit MAC address (see 3.11) as specified in 9.2 of IEEE Std 802-2001.

**6.1.1.3 *protocolType*:** A 16-bit field contained within the payload. When the value of *protocolType* is greater than or equal to 1536 ( $600_{16}$ ) the *protocolType* field indicates the nature of the MAC client protocol (type interpretation), selecting from values designated by the IEEE Type Field Register. When less than 1536 ( $0_{16}$ – $5FF_{16}$ ), the *protocolType* is interpreted as the length of the frame (length interpretation). The length and type interpretations of this field are mutually exclusive.

**6.1.1.4 *serviceDataUnit*:** An *m*-byte field the contains the service data unit provided by the client.

**6.1.1.5 *pad*:** If the sum of the other field lengths is less than 64 bytes, then the number of zero-valued *pad* bytes are sufficient to make a 64-byte frame. Otherwise, the *pad* field is not present.

**6.1.1.6 *fcs*:** A 4-byte (frame check sequence) field whose 32-bit CRC covers the frame's content.

## 6.2 clockSync frame format

### 6.2.1 clockSync fields

Clock synchronization (clockSync) frames facilitate the synchronization of neighboring clock span-master and clock span-slave stations. The frame, which is normally sent once each isochronous cycle, includes time-snapshot information and the identity of the network's clock master, as illustrated in 6.2. The gray boxes represent physical layer encapsulation fields that are common across all Ethernet frames.

6	<i>da</i>	— Destination MAC address
6	<i>sa</i>	— Source MAC address
2	<i>protocolType</i>	— Distinguishes RE frames from others (see 6.5.1)
1	<i>subType</i>	— Distinguishes clockSync from other RE frames (see 6.5.2)
1	<i>hopsCount</i>	— Hop count from the grand master
1	<i>syncCount</i>	— Sequence number for clockSync frames
1	<i>cycleCount</i>	— Cycle count for pacing
2	<i>systemTag</i>	— More-significant grand-master election precedence
8	<i>uniqueID</i>	— Less-significant grand-master election precedence
8	<i>lastFlexTime</i>	— Incoming link's frame transmission time (1 cycle delayed)
8	<i>deltaTime</i>	— Outgoing link's frame propagation time
8	<i>offsetTime</i>	— Cumulative offset times from the grand-master
4	<i>diffRate</i>	— Cumulative rate differences from the grand-master
4	<i>lastBaseTime</i>	— Incoming link's frame transmission time (1 cycle delayed)
4	<i>fcs</i>	— Frame check sequence

Figure 6.2—clockSync frame format

**6.2.1.1 *da*:** A 48-bit (destination address) field that specifies the station(s) for which the frame is intended. The *da* field contains either an individual or a group 48-bit MAC address (see 3.11), as specified in 9.2 of IEEE Std 802-2001.

**6.2.1.2 *sa*:** A 48-bit (source address) field that specifies the local station sending the frame. The *sa* field contains an individual 48-bit MAC address (see 3.11), as specified in 9.2 of IEEE Std 802-2001.

**6.2.1.3 *protocolType*:** A 16-bit field contained within the payload that identifies the format and function of the following fields (see 6.5.1).

**6.2.1.4 *subType*:** A 16-bit field that identifies the format and function of the following fields (see 6.5.2).

**6.2.1.5 *hopsCount*:** An 8-bit field that identifies the maximum number of hops between the talker and associated listeners.

**6.2.1.6 *syncCount*:** An 8-bit field that is incremented on each clockSync frame transmission.

**6.2.1.7 *cycleCount*:** A 7-bit field that equals  $(cycle \% 125)$ , where *cycle* represents units of 125  $\mu$ s within the transmitting station's *timeOfDay* value.

**6.2.1.8 *systemTag*:** A 16-bit field that has highest precedence in the grand-master selection protocols.

**6.2.1.9 *uniqueID*:** A 64-bit field that specifies the precedence of the grand clock master, specified in 6.2.3.

**6.2.1.10 *lastFlexTime*:** A 64-bit field that specifies the time within the source station when the previous clockSync frame was transmitted. The format of this field is specified in 6.2.4.

**6.2.1.11 *deltaTime*:** A 64-bit field that specifies the differences between clockSync receive and transmit times, as measured on the opposing link. The format of this field is specified in 6.2.4.

**6.2.1.12 *offsetTime*:** A 64-bit field that specifies the offset time within the source station. The format of this field is specified in 6.2.4.

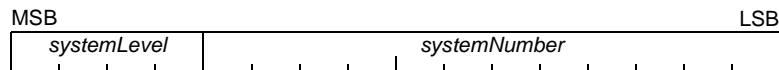
**6.2.1.13 *diffRate*:** A 32-bit field that specifies the *diffRate* value within the source station.

**6.2.1.14 *lastBaseTime*:** A 32-bit field that specifies the *timer1* value within the source station when the previous clockSync frame was transmitted.

**6.2.1.15 *fcs*:** A 32-bit (frame check sequence) field that is a cyclic redundancy check (CRC) of the frame.

## 6.2.2 *systemTag* subfields

The format of the 16-bit *systemTag* field is based on the format of the spanning tree protocol precedence value, as illustrated in Figure 6.3.



**Figure 6.3—*systemTag* subfields**

**6.2.2.1 *systemLevel*:** A 4-bit field that comprise a settable priority component that permits the relative priority of bridges to be managed.

**6.2.2.2 *systemNumber*:** A 12-bit field that comprise a locally assigned system identifier extension. (The term *systemID* is equivalent to 'system ID', as specified within IEEE Std 802.1D-2004.)

### 6.2.3 *uniqueID* fields

The format of the 64-bit *uniqueID* field is a unique identifier. For stations that have a uniquely assigned 48-bit *macAddress*, the 64-bit *uniqueID* field is derived from the 48-bit MAC address, as illustrated in Figure 6.4.

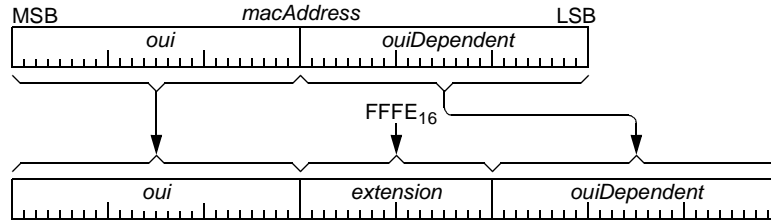


Figure 6.4—*uniqueID* format

**6.2.3.1 *oui*:** A 24-bit field assigned by the IEEE/RAC (see xx).

**6.2.3.2 *extension*:** A 16-bit field assigned to encapsulated EUI-48 values.

**6.2.3.3 *ouiDependent*:** A 24-bit field assigned by the owner of the *oui* field (see xx).

### 6.2.4 Time field formats

Time-of-day values within a frame are specified by 64-bit values, consistent with IETF specified NTP[B8] and SNTP[B9] protocols. These 64-bit values consist of two components: a 32-bit *seconds* and 32-bit *fraction* fields, as illustrated in Figure 6.5.

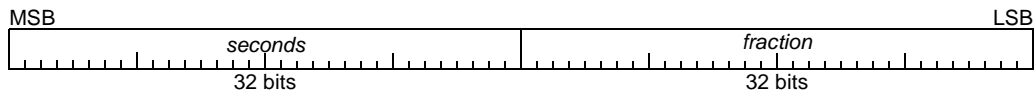


Figure 6.5—Complete seconds timer format

**6.2.4.1 *seconds*:** A 32-bit field that specifies time in seconds.

**6.2.4.2 *fraction*:** A 32-bit field that specified time offset within the second, in units of  $2^{-32}$  second.

The concatenation of 32-bit *seconds* and 32-bit *fraction* field specifies a 64-bit *time* value, as specified by Equation 6.1.

$$time = seconds + (fraction / 2^{32}) \tag{6.1}$$

Where:

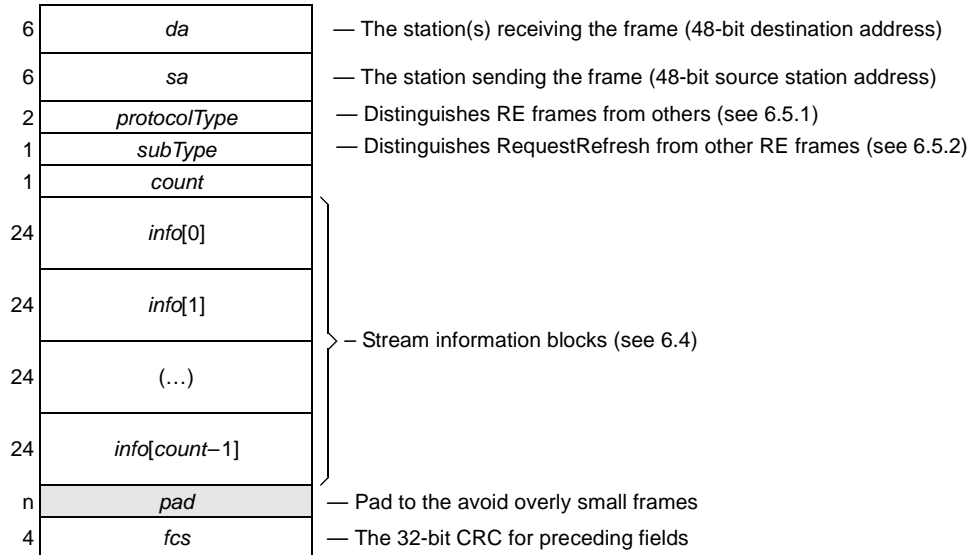
*seconds* is the most significant component of the time value (see Figure 6.5).

*fraction* is the less significant component of the time value (see Figure 6.5).

## 6.3 Subscription frame

### 6.3.1 Subscription frame fields

Subscription frames contain channel-acquisition information, as illustrated in Figure 6.6.



**Figure 6.6—Subscription frame format**

**6.3.1.1 *da*:** A 6-byte (destination address) field that normally specifies the destination address for the frame transmission, with unicast and multicast forms.

**6.3.1.2 *sa*:** A 6-byte (source address) field that normally specifies the source address for the frame transmission. If a bridge is present between the frame and its associated listener, the *sa* value identifies the bridge.

**6.3.1.3 *protocolType*:** A 2-byte field that normally specifies the frame length, or the format and function of the following fields (excluding the 4-byte *fcs* field). This RE assigned value distinguishes its frame formats from others (see 6.5.1).

**6.3.1.4 *subType*:** A 1-byte field that distinguishes the ResponseError frame from other frames defined within this working paper.

**6.3.1.5 *count*:** A 1-byte field that specifies the number of elements within the following *info*-block array.

**6.3.1.6 *info*:** A 24-byte array element that provides listener subscription information (see 6.4).

**6.3.1.7 *pad*:** If the sum of the other field lengths is less than 64 bytes, then the number of zero-valued *pad* bytes are sufficient to make a 64-byte frame. Otherwise, the *pad* field is not present.

**6.3.1.8 *fcs*:** The 4-byte (frame check sequence) field whose 32-bit CRC covers the frame's content. For RE content frames, the standard definition applies.

## 6.4 Common *info* field format

Many frame transports an array of one or more *info*[] fields, whose content is illustrated in Figure 6.7.

2	<i>command</i>	— Database action command
6	<i>talkerID</i>	— Multicast talker identifier
2	<i>plugID</i>	— Resource within the talker
6	<i>mcastID</i>	— Multicast destination label
2	<i>maxCycles</i>	— Delay from the talker
4	<i>maxBw</i>	— Maximum required bandwidth
2	<i>reserved</i>	— Reserved

**Figure 6.7—Common *info* field format**

**6.4.1 *command*:** A 2-byte field that differentiates between database-update actions.

**6.4.2 *talkerID*:** A 6-byte field that identifies the stream's talker.

**6.4.3 *plugID*:** A 16-bit field that specifies the plug identifier within the talker.

The concatenation of the 48-bit *talkerID* and 16-bit *plugID* fields forms a 64-bit *streamID* that uniquely identifies the classA multicast stream.

**6.4.4 *mcastID*:** A 6-byte (multicast identifier) field that routes frames between the talker and audience.

**6.4.5 *maxCycles*:** A 2-byte field that is updated by bridges, as the RequestRefresh flows from the talker to the listener, allowing the maximum number of delay cycles between the talker and listener stations to be known to the talker.

**6.4.6 *maxBw*:** A 4-byte field that specifies the level of negotiated classA bandwidth, measured in bytes of per-cycle content.

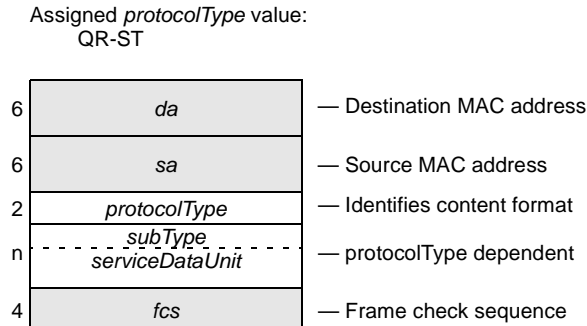
**6.4.7 *reserved*:** A 2-byte zero-valued field that is ignored.

## 6.5 Unique identifier values

### 6.5.1 *protocolType* identifier

NOTE—The following *protocolType*-assignment text will ultimately be updated with assigned values.

The clockSync (see 6.2) and subscription (see 6.3) frames are distinguished from other frames by their 16-bit distinct *protocolType* value, as illustrated in Figure 6.8. The following 1-byte *subType* field further distinguishes between these uses (see 6.5.2).



**Figure 6.8—*protocolType* field value**

### 6.5.2 *subType* identifier

Distinct *subType* identifiers distinguish between RE frame types, as specified by Table 6.1.

**Table 6.1—Assigned *subType* identifiers**

Value	Name	Row	See	Description
TBD	CLOCK_SYNC	1	6.2	Demarcates boundaries between isochronous cycles.
192-255	E1394	2	C.2.2	Encapsulated IEEE 1394 packet (or portion of 1394 packet)

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54

## 7. Timer synchronization

### 7.1 Synchronized time-of-day timers

#### 7.1.1 Assumptions

This working paper specifies a protocol to synchronize independent timers running on separate stations of a distributed networked system, based on concepts specified within IEEE Std 1588-2002. Although a high degree of accuracy and precision is specified, the technology is applicable to low-cost consumer devices. The protocols are based on the following design assumptions:

- a) Each end station and intermediate bridges provide independent clocks.
- b) All clocks are accurate, typically to within  $\pm 100$ PPM.
- c) Point-to-point transmit/receive duplex connections are provided.
- d) Transmit/receive propagation delays within duplex cables are well matched.

#### 7.1.2 Objectives

With these assumptions in mind, the time synchronization objectives include the following:

- a) Precise. Multiple timers can be synchronized to within 10's of nanoseconds.
- b) Inexpensive. For consumer A/V devices, the costs of synchronized timers are minimal. (GPS, atomic clocks, or 1PPM clock accuracies would be inconsistent with this criteria.)
- c) Scalable. The protocol is independent of the networking technology. In particular:
  - 1) Cyclical physical topologies are supported.
  - 2) Long distance links (up to 2 km) are allowed.
- d) Plug-and-play. The system topology is self-configuring; no system administrator is required.

#### 7.1.3 Strategies

Strategies used to meet these objectives include the following:

- a) Precision is achieved by calibrating and adjusting *timeOfDay* clocks.
  - 1) Offsets. Offset value adjustments eliminate immediate clock-value errors.
  - 2) Rates. Rate value adjustments reduce long-term clock-drift errors.
- b) Simplicity is achieved by the following:
  - 1) Concurrence. Most configuration and adjustment operations are performed concurrently.
  - 2) Feed-forward. PLLs are unnecessary within bridges, but possible within applications.
  - 3) Symmetric. Clock-master/clock-slave computations are similar (only slave results are saved).
  - 4) Periodic. Messages are sent periodically, rather than in timely response to other requests.
  - 5) Frequent. Frequent (every 10 ms) interchanges reduces needs for precise clocks.
- c) Balanced functionality.
  - 1) Low-rate. Complex computations are infrequent and can be readily implemented in firmware.
  - 2) High-rate. Frequent computations are simple and can be readily implemented in hardware.



## 7.1.4 Timer synchronization services

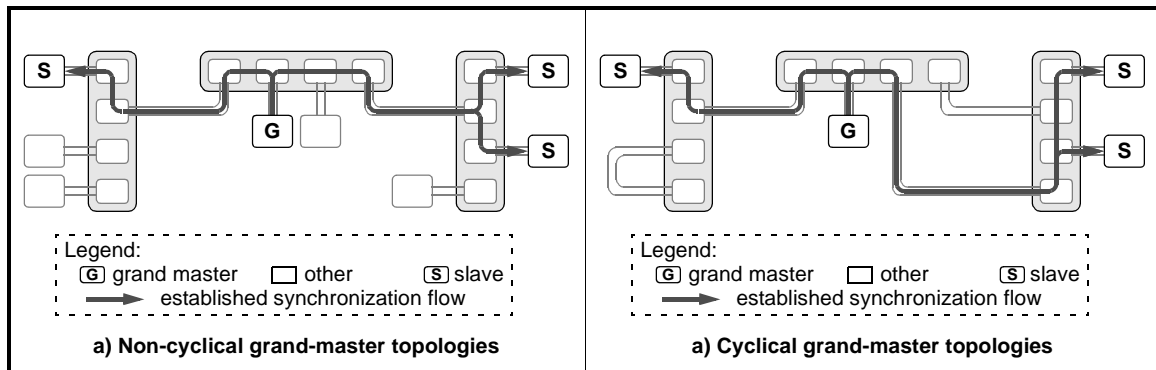
Clock synchronization involves the transmission and reception of clockSync frames interchanged between adjacent-span stations, using the state machines defined within this clause. When considered as a whole, these provide the following services:

- 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.

## 7.1.5 Grand-master selection

### 7.1.5.1 Grand-master selection

Clock synchronization involves streaming of clock-synchronization information from a grand-master timer to one or more slave timers. Although primarily intended for non-cyclical physical topologies (see Figure 7.1a), the synchronization protocols also function correctly on cyclical physical topologies (see Figure 7.1b), by activating only a non-cyclical subset of the physical topology.



**Figure 7.1—Timer synchronization flows**

In concept, the clock-synchronization protocol starts with the selection of the reference-timer station, called a grand-master station (oftentimes abbreviated as grand-master). Every RE-capable station is grand-master capable, but only one is selected to become the grand-master station within each network. To assist in the grand-master selection, each station is associated with a distinct preference value; the grand-master is the station with the “best” preference values.

As part of the grand-master selection process, stations forward the best of their observed preference values to neighbor stations, allowing the overall best-preference value to be ultimately selected and known by all. The station whose preference value matches the overall best-preference value ultimately becomes the grand-master.

### 7.1.5.2 Communicated preference values

The grand-master station observes that its precedence is better than values received from its neighbors, as illustrated in Figure 7.2a. A slave stations observes its precedence to be worse than one of its neighbors and forwards the best-neighbor precedence value to adjacent stations, as illustrated in Figure 7.2b. To avoid cyclical behaviors, a *hopsCount* value is associated with preference values and is incremented before the best-precedence value is communicated to others.

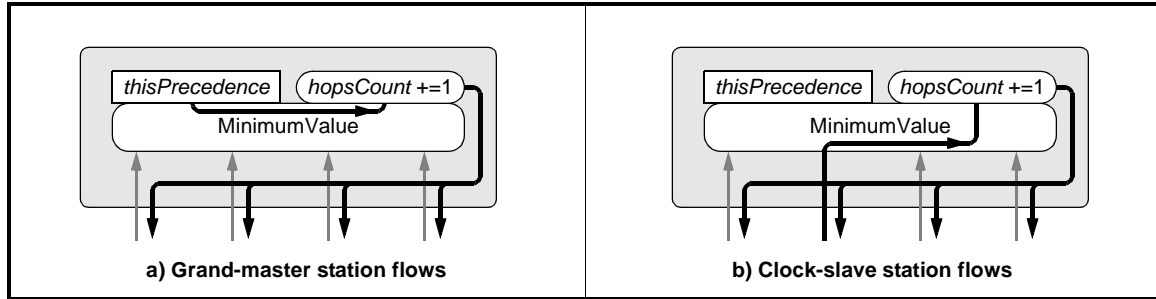


Figure 7.2—Grand-master precedence flows

The grand-master selection precedence includes multiple components, listed and described below (see 7.1.8). The *portTag* value is only needed within a bridge and is therefore not transmitted between stations.

- a) *systemTag*. A changeable value that is associated with each grand-master capable station. This value can specify grand-master preferences (e.g., a home gateway may be preferred).
- b) *uniqueID*. A unique value associated with each station, typically based on its MAC address. This value is used as a tie breaker, when two contenders have identical *systemTag* values.
- c) *hopsCount*. A value that is incremented when passing through stations. This is the tie breaker, when two ports receive identical *systemTag:uniqueID* values.
- d) *portTag*. A changeable value that is associated with each port on a grand-master capable station. This is the tie breaker, when two ports receive identical *systemTag:uniqueID:hopsCount* values.

### 7.1.6 Clock-synchronization agents

Clock-synchronization information conceptually flows from a grand-master station to clock-slave stations, as illustrated in Figure 7.3a. A more detailed illustration shows pairs of synchronized clock-master and clock-slave components, as illustrated in Figure 7.3b.

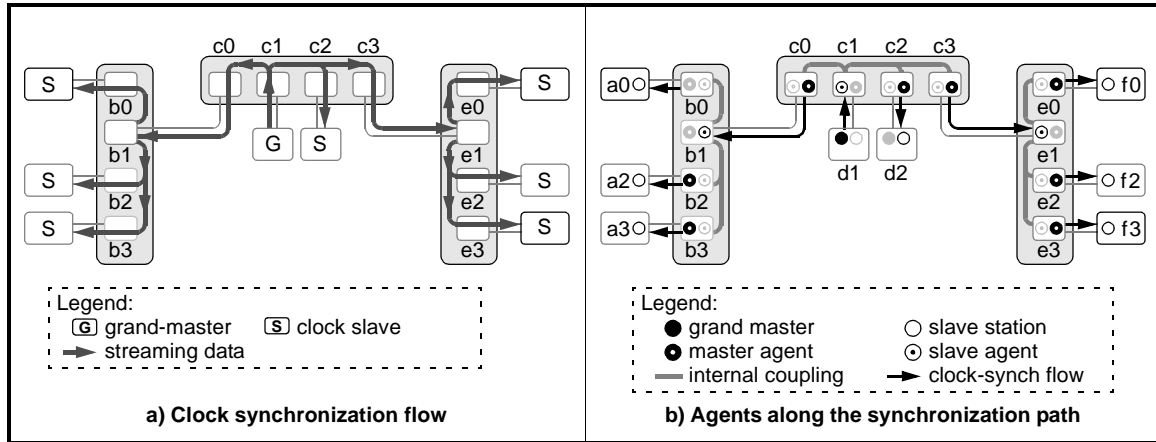


Figure 7.3—Hierarchical flows

### 7.1.7 Clock-synchronized pairs

Each bridge port provides clock-master and clock-slave agents, although both are never simultaneously active. External communications (see 7.3b) 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	

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54

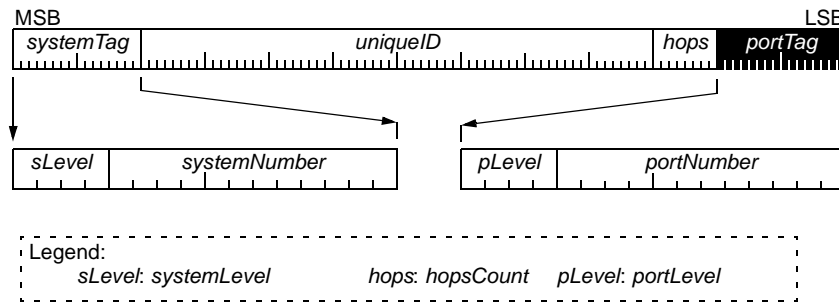
1 Internal communications distribute synchronized time from clock-slave agents b1, c1, and e1 to the other  
2 clock-master agents on bridgeB, bridgeC, and bridgeE respectively. However, bridge-internal port-to-port  
3 synchronization protocols are implementation-dependent and beyond the scope of this working paper.  
4

5 Within a clock-slave, precise time synchronization involves adjustments of timer offset and rate values. The  
6 adjustments of the timer's offset is called offset synchronization (see 7.1.12); the adjustments of the timer's  
7 rate is called rate synchronization (see 7.1.13). Both involve calibration of local clock-master/clock-slave  
8 differences and the propagation of cumulative differences in the clock-slave direction, as described by the C  
9 code of Annex J.

10  
11 Time synchronization yields distributed but closely-matched *timeOfDay* values within stations and bridges.  
12 No attempt is made to eliminate intermediate jitter with bridge-resident jitter-reducing phase-lock loops  
13 (PLLs,) but application-level phase locked loops (not illustrated) are expected to filter high-frequency jitter  
14 from the supplied *timeOfDay* values  
15

### 16 7.1.8 Grand-master precedence

17  
18 Grand-master precedence is based on the concatenation of multiple fields, as illustrated in Figure 7.4. The  
19 *portTag* value is used within bridges, but is not transmitted between stations.  
20

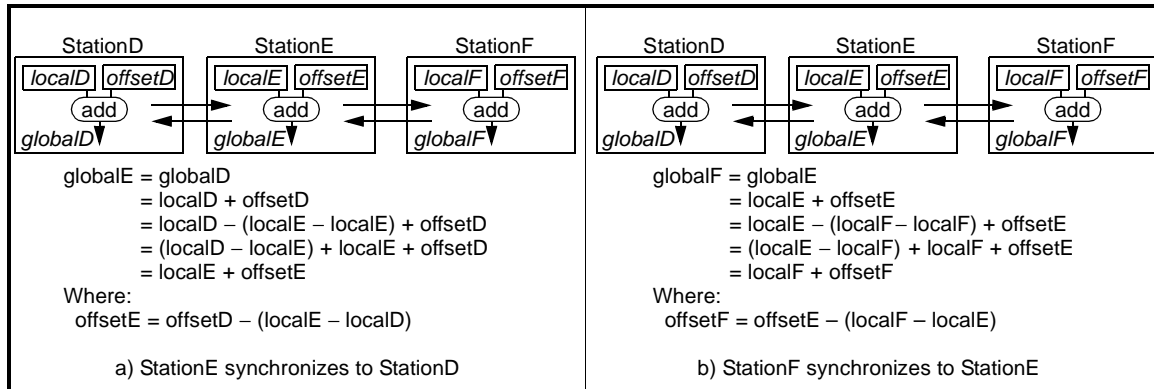


32 **Figure 7.4—Grand-master precedence**

33  
34 This format is similar to the format of the spanning-tree precedence value, except that a larger *uniqueID* is  
35 provided for compatibility with interconnects based on 64-bit station identifiers.  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54

### 7.1.9 Synchronization principles

Timer synchronization is based on the concept of free-running local times ( $localD$ ,  $localE$ , and  $localF$ ) with compensating offset values ( $offsetD$ ,  $offsetE$ , and  $offsetF$ ), as illustrated in Figure 7.5. Updates involve changes to the offset values, not the free-running local timer values. In this example, we assume that: StationE is synchronized to its adjacent StationD; StationF is synchronized to its adjacent StationE. As a result, StationF is indirectly synchronized to StationD (through StationE).



**Figure 7.5—Time synchronization principles**

The formulation of the  $offsetE$  value begins the assumption that the  $globalE$  and  $globalD$  times are identical. Addition of  $(localE - localE)$  and regrouping of terms leads to the formulation of the desired  $offsetE$  value, based on  $offsetD$  and  $(localE - localD)$  time difference values, as illustrated in Figure 7.5-a. Synchronization is thus possible using periodic transfers of  $offsetD$  values and computations of the  $(localE - localD)$  difference.

The formulation of the  $offsetF$  value begins the assumption that the  $globalF$  and  $globalE$  times are the identical. Addition of  $(localF - localF)$  and regrouping of terms leads to the formulation of the desired  $offsetF$  value, based on  $offsetE$  and  $(localF - localE)$  time difference values, as illustrated in Figure 7.5-b. Synchronization is thus possible using periodic transfers of  $offsetE$  values and computations of the  $(localF - localE)$  difference.

In concept, the  $offsetE$  value is adjusted first; its adjusted value is then used to compute the desired  $offsetF$  value. In actuality, the periodic computations of  $offsetE$  and  $offsetF$  values are performed concurrently.

7.1.10 Timer snapshot locations

Mandatory jitter-error accuracies are sufficiently loose to allow transmit/receive snapshot circuits to be located with the MAC, as illustrated in Figure 7.6a. Vendors may elect to further reduce timing jitter by latching the receive/transmit times within the PHY, where the uncertain FIFO latencies can be best avoided, as illustrated in Figure 7.6b.

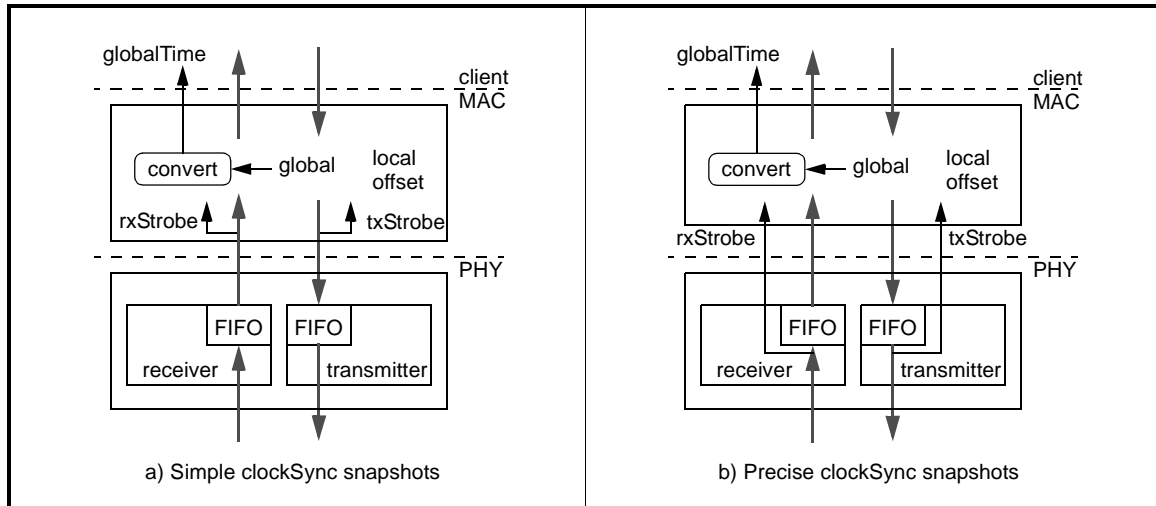


Figure 7.6—Timer snapshot locations

7.1.11 Clock-synchronization updates

7.1.11.1 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.

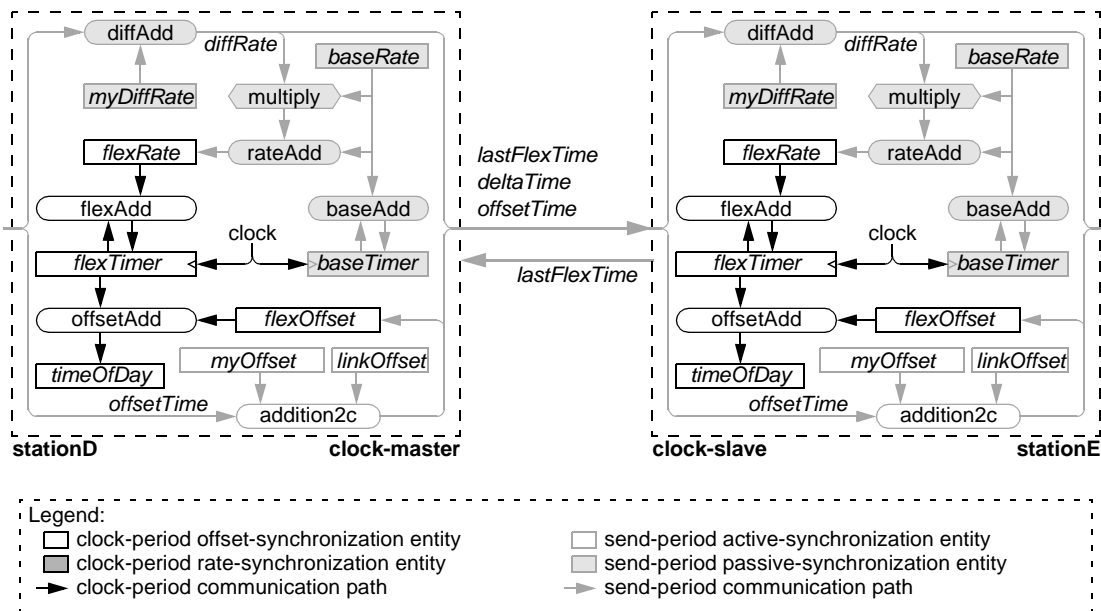
The slow-period events trigger the computation of timer-rate differences. The timer-rate differences are computed over two slow-period intervals, but recomputed every slow-period interval. The larger 100 ms (as

opposed to 10 ms) computation interval is intended to reduce errors associated with sampling of clock-period-quantized slow-period-sized time intervals.

## 7.1.12 Offset synchronization

### 7.1.12.1 Offset synchronization adjustments

Offset synchronization involves a subset of the time-synchronization components, as illustrated by white-colored boxes in Figure 7.9. 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 7.7—Offset synchronization adjustments**

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.

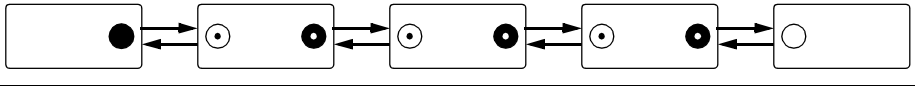
The offset-compensation protocols for stationE adjust its *myOffset* value so that the instantaneous values of *stationE.timeOfDay* and *stationD.timeOfDay* are the same. Computations are performed on clockStrobe reception and clockStrobe transmission.

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 to 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).

7.1.12.2 Cascaded offsets

The concept of cascaded offset values can be better understood by considering a simple 3-bridge example, as illustrated in Figure 7.8. 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	grand-master	bridgeB	bridgeC	bridgeD	clock-slave
name	grand-master	bridgeB	bridgeC	bridgeD	clock-slave
number	1	2	3	4	5
<i>flexTimer</i>	100	500	-300	200	400
<i>myOffset</i>	10	-400	800	-500	-200
<i>flexOffset</i>	10	-390	410	-90	-290
<i>timeOfDay</i>	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 7.8—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.



7.1.13 Rate synchronization

7.1.13.1 Rate synchronization adjustments

Rate synchronization involves a subset of the time-synchronization components, as illustrated by white-colored boxes in Figure 7.9. The free-running *baseTimer* timer facilitate the determination of rate differences between the clock-master and clock-slave stations.

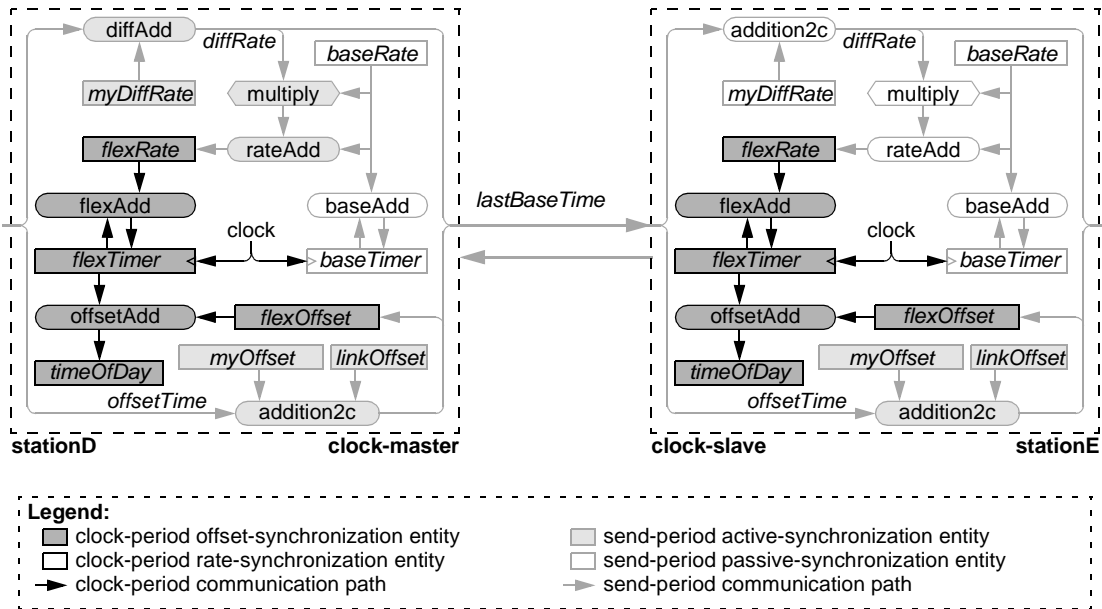


Figure 7.9—Rate synchronization adjustments

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).

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54

7.1.13.2 Cascaded rate differences

The concept of cascaded rate values can be better understood by considering a simple 3-bridge example, as illustrated in Figure 7.10. 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
<i>myDiffRateN</i>	0 PPM	-90 PPM	200 PPM	-25 PPM	-150 PPM
<i>diffRateN</i>	0 PPM	-90 PPM	110 PPM	+85 PPM	-65 PPM
<i>flexTimer</i> 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 7.10—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.

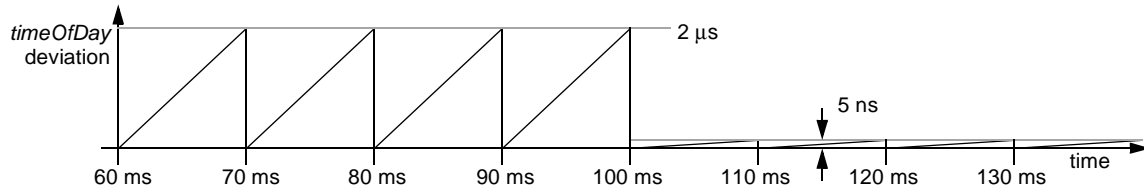
To simplify this illustration, consider only the parts-per-million (PPM) normalized rate values within each station or bridge. These values may differ significant, based (perhaps) on the nominal value or ambient temperature. Thus, the grand-master could have a crystal deviation of +10 while its bridgeB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

**7.1.14 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 7.11. 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.



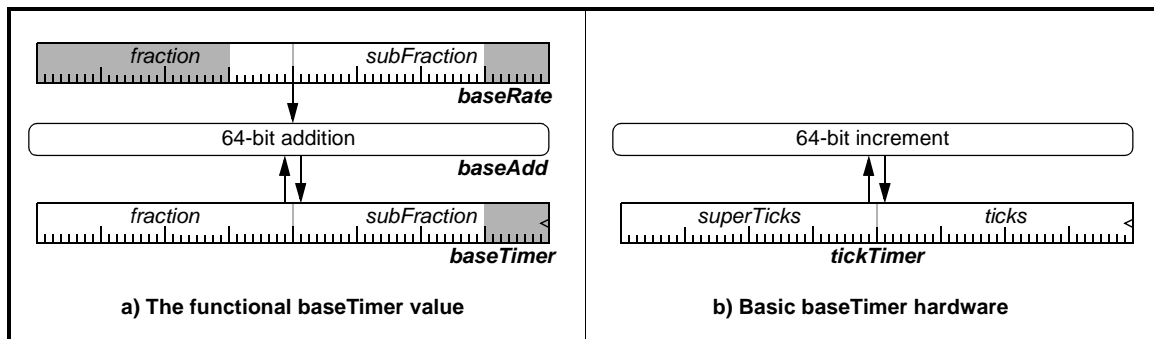
**Figure 7.11—Rate-adjustment effects**

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.

**7.1.15 baseTimer functionality**

The external formats within timeSync frames assumes the presence of *baseTimer*-related values. A direct-mapped hardware implementation involves a clocked *baseTimer* register and a precision adder, as illustrated in Figure 7.12a. Within this figure, the shaded bytes represent values that can safely be hardwired to zero with insignificant loss of accuracy.



**Figure 7.12—baseTimer implementation examples**

A simpler hardware implementations is to periodically increment a tick timer at an implementation-dependent rate, as illustrated in Figure 7.12a. Although this timer is improperly scaled, firmware can perform the multiplications that are necessary to provide the proper normalized external values.

### 7.1.16 flexTimer functionality

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 7.13. Within this figure, the shaded bytes represent values that can safely be hardwired to zero with insignificant loss of accuracy.

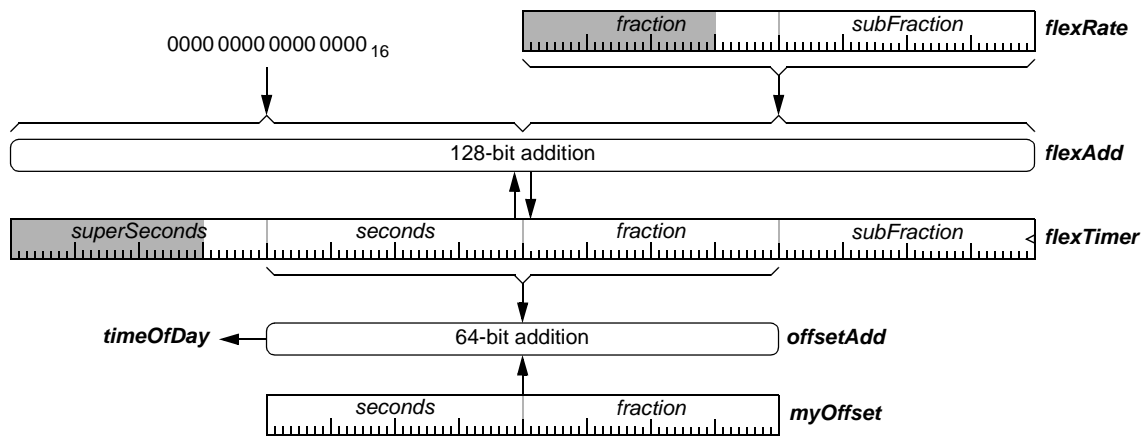


Figure 7.13—flexTimer implementation example

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).

Although a direct hardware implementation is possible, a simpler solution is to utilize a single *tickTimer* register (see 7.1.15). Since the external communication rates are low, firmware conversions between *tickTimer* and *flexTimer* values are expected to be feasible.

## 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\_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.

Q\_RX\_FRAME—Identifies the queue that supplies received frames.

Q\_RX\_SYNC—Identifies the queue where received timeSync frames are saved.

Q\_RX\_ELSE—Identifies the queue where received non-timeSync frames are saved.

Q\_TX\_FRAME—The identifier associated with the transmitted clockSync frames.

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.

### *core*

The contents of a timeSync frame, including the following components (see 6.2.1):

*clockDeviation*—Nominal frequency deviation of crystal-stabilized clock, in PPM.

Maximum: +100

Minimum: -100

*clockFrequency*—The nominal frequency of the synchronized timer.

*clockFrequency* > 50 MHz

*diffRate*—The cumulative rate difference from the grand-master station.

*flexOffset*—The cumulative value difference from the grand-master station.

*linkOffset*—The specified delay differences between transmit and receive links.

*myDiffRate*—The measured rate difference between the local and remote ports.

*myOffset*—The measured time-offset difference between the local and remote ports.

*offsetCount*—A running count that is incremented approximately once every 10 ms.

*offsetTicks*—A *timerTicks* snapshot taken at the start of each 10 ms interval.

*precedence*—The currently observed grand-master precedence value.

*rateCount*—A running count that is incremented approximately once every 100 ms.

*slaveCount*—A snapshot of *offsetCount*, received from a grand-master slave port.

*slavePort*—The slave port identifier associated with an upstream grand-master station.

*systemTag*—The most-significant portion of this station's grand-master precedence.

*tickOffset*—A scalar parameter for the conversion between *timerTicks* and *flexTimer* values.

*tickRate*—A scalar parameter for the conversion between *timerTicks* and *flexTimer* values.

*tickTime*—A time snapshot taken at the start of each *clockPeriod* interval.

*timerTicks*—A running count that is incremented at the end of each *clockPeriod* interval.

*uniqueID*—The more-significant portion of this station's grand-master precedence

### *frame*

The contents of a timeSync frame, including the following components (see 6.2.1):

*da*, *sa*, *protocolType*, *subType*, *hopsCount*, *syncCount*, *cycleCount*, *systemTag*,  
*uniqueID*, *lastFlexTime*, *deltaTime*, *offsetTime*, *diffRate*, *lastBaseTime*, *fcs*.

See 6.2.1.

*rxTimerTicks*—The value of *core.timerTicks*, sampled when this frame was received.

### *port*

The contents of a timeSync frame, including the following components (see 6.2.1):

*offsetCount*—An accounting value whose comparison to *core.offsetCount* triggers processing.

*portPri*—A priority differentiator for the port.

*portID*—A unique identifier for the port.

*propTime*—An average cable propagation delay measurement for the attached link.

*rateCount*—An accounting value whose comparison to *core.rateCount* triggers processing.

*syncCount*—The last observed *frame.syncCount* value, saved for consistency checks.

*timeSyncAllowed*—Indicates when a timeSync frame is enabled for transmission.

*txTimerTicks*—The saved *core.timerTicks* value, when timeSync was last transmitted.

### *timerTicks*

See 7.2.2.

**7.2.3 Common state machine routines***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.)

**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 TimerTicks state machine****7.3.1.1 TimerTicks state machine definitions**

The following definitions are used within this subclause:

NULL

A constant that indicates the absence of a slave-port identifier.

TICKS\_PER\_SEC

The nominal number of tick periods in every cycle.

**7.3.1.2 TimerTicks state machine variables**

Only one instance of each state-machine variable exists in each bridge.

*core**currentTime*

See 7.2.2.

*formed*

A computed grand-master precedence value, based on this station's parameters.

7.3.1.3 TimerTicks state table

The TimerTicks state machine provides the timer values for other state machines, as specified in Table 7.3. The notation used in the state table is described in 3.4.

Table 7.3—TimerTicks state table

Current		Row	Next	
state	condition		action	state
START	$(currentTime - core.tickTime) \geq 1.0 / (core.clockFrequency * (1.0 + (core.clockDeviation / 1000000.)))$	1	core.timerTicks += 1; core.tickTime = currentTime;	START
	$(core.timerTicks - core.offsetTicks) \geq 0.010 * TICKS\_PER\_SEC$	2	core.offsetTicks = core.timerTicks; core.offsetCount += 1; formed = PreForm(core.systemTag, core.uniqueID, 0, 0, 0);	NEXT
	—	3	—	START
NEXT	$(core.offsetCount - core.slaveCount) \geq 5$	1	core.slavePort = NULL;	NEAR
	—	2	—	
NEAR	Best(formed, core.precedence)    core.slavePort == NULL	1	core.slavePort = NULL; core.precedence = formed; core.slaveCount = core.offsetCount;	FINAL
	—	2	—	
FINAL	$(core.offsetCount \% 10) == 0$	1	core.rateCount += 1;	START
	—	2	—	

**Row START-1:** The *timerTicks* register is incremented once every *clockPeriod* interval. (The *timerTicks* register is the timer used to snapshot all arrival and departure times.)

**Row START-2:** The *offsetCount* register is incremented approximately once every 10 ms interval. (Changes in the *offsetCount* register triggers transmissions of periodic *timerSync* frames.)

**Row START-3:** Otherwise, no updates are performed.

**Row NEXT-1:** If the slave port receives no heartbeats, the slave-port identifier is released.

**Row NEXT-2:** Otherwise, no updates are performed.

**Row NEAR-1:** If this station has the best preference, or no slave port exists, this station has precedence.

**Row NEAR-2:** Otherwise, the cumulative preference value remains unchanged.

**Row FINAL-1:** The *routeCount* register is incremented at 1/10 of the *offsetCount* update rate. (Changes in the *routeCount* register triggers computations of new rate-of-change values.)

**Row FINAL-2:** Otherwise, no updates are performed.



## 7.3.2 TimerRxLatch state machine

### 7.3.2.1 TimerRxLatch state machine definitions

The following definitions are used within this subclause:

Q\_RX\_ELSE  
Q\_RX\_FRAME  
Q\_RX\_SYNC  
See 7.2.1.

### 7.3.2.2 TimerRxLatch state machine variables

The following variables are used within this subclause:

*core*  
*frame*  
See 7.2.2.

### 7.3.2.3 TimerRxLatch state machine routines

The following routines are used within this subclause:

*Dequeue(queue)*  
*Enqueue(queue, frame)*  
See 7.2.3.

### 7.3.2.4 TimerRxLatch state table

The TimerRxLatch state machine associates a time stamp within incoming frames, as specified in Table 7.4. The notation used in the state table is described in 3.4.

**Table 7.4—TimerRxLatch state table**

Current		Row	Next	
state	condition		action	state
START	(frame = Dequeue(Q_RX_FRAME)) != NULL	1	—	FINAL
	—	2	—	START
FINAL	frame.protocolType == TIME_SYNC	1	frame.rxTimerTicks = core.timerTicks; Enqueue(Q_RX_SYNC, frame);	START
	—	2	Enqueue(Q_RX_ELSE, frame);	

**Row START-1:** A new frame has arrived and is available for processing.

**Row START-2:** Wait for a new frame to arrive.

**Row FINAL-1:** The timeSync frames are immediately time-stamped and enqueued for special processing.

**Row FINAL-2:** Other frames are processed in their normal fashion.

1       **7.3.3 TimerTxLatch state machine**

2  
3       **7.3.4 TimerRxLatch state machine**

4  
5       **7.3.4.1 TimerRxLatch state machine definitions**

6  
7       The following definitions are used within this subclause:

8  
9            Q\_TX\_FRAME

10          Q\_TX\_SYNC

11            See 7.2.1.

12  
13       **7.3.4.2 TimerRxLatch state machine variables**

14  
15       The following variables are used within this subclause:

16  
17            *core*

18            *frame*

19            *port*

20            See 7.2.2.

21  
22       **7.3.4.3 TimerTxLatch state machine routines**

23  
24       The following routines are used within this subclause:

25  
26            *Dequeue(queue)*

27            *Enqueue(queue, frame)*

28            *QueueEmpty(queue)*

29            See 7.2.3.

**7.3.4.4 TimerTxLatch state table**

The TimerTxLatch state machine associates a time stamp within incoming frames, as specified in Table 7.5. The notation used in the state table is described in 3.4.

**Table 7.5—TimerTxLatch state table**

Current		Row	Next	
state	condition		action	state
START	(QueueEmpty(Q_TX_FRAME)) && port.timeSyncAllowed	1	—	FINAL
	—	2	—	START
FINAL	(frame = Dequeue(Q_TX_SYNC)) != NULL	1	port.txTimerTicks = core.timerTicks; Enqueue(Q_TX_FRAME, frame);	START
	—	2	—	

**Row START-1:** A new frame has arrived and is available for processing.

**Row START-2:** Wait for a new frame to arrive.

**Row FINAL-1:** The timeSync frames are immediately time-stamped and saved for special processing.

**Row FINAL-2:** Other frames are processed in their normal fashion.

## 7.3.5 TimerRxCompute state machine

### 7.3.5.1 TimerRxCompute state machine definitions

The following definitions are used within this subclause:

*Q\_TX\_SYNC*  
See 7.2.1.

### 7.3.5.2 TimerRxCompute state machine variables

The following variables are used within this subclause:

*core*  
See 7.2.2.

*formed*  
A computed grand-master precedence value, based on the frame-supplied parameters.

*frame*

*port*  
See 7.2.2.

*rxDeltaTime*  
A temporary value representing time-snapshot differences on the receiving link.

*txDeltaTime*  
A temporary value representing time-snapshot differences on the transmitting link.

### 7.3.5.3 TimerRxCompute state machine routines

The following routines are used within this subclause:

*Best(test, base)*  
Indicates whether the *test* is smaller than the *base* precedence, as defined by Equation 7.1.

$$(\text{test.hi} < \text{base.hi} \mid\mid (\text{test.hi} == \text{base.hi} \ \&\& \ \text{test.lo} \leq \text{base.lo})) \quad (7.1)$$

*Compensate(core, rate)*  
Indicates whether the *test* is smaller than the *base* precedence, as defined by Equation 7.1.

$$(\text{core.tickOffset} -= (\text{rate} - \text{core.flexRate}) * \text{core.tickRate}) \quad (7.2)$$

*FlexTime(tickTime, tickRate, tickOffset)*  
Derives a 64-bit time-of-day value from the implementation-dependent *tickTime* value.

*PreForm(sys, uid, hops, tag)*  
Forms a 128-bit precedence value from its component fields, as defined by Equation 7.1.

$$\text{PreForm}(\text{sys}, \text{uid}, \text{hops}, \text{pri}, \text{pid}) \quad (7.3)$$

```
{
    doubleInt value;

    value.hi = (sys << 24) | (uid >> 40);
    value.lo = (uid << 24) | (hops << 16) | (pri << 12) | pid;
    return(value);
}
```

### 7.3.5.4 TimerRxCompute state table

The TimerRxCompute state machine processes received timeSync frames while participating in the grand-master selection protocol, as specified in Table 7.6. The notation used in the state table is described in 3.4.

**Table 7.6—TimerRxCompute state table**

Current		Row	Next	
state	condition		action	state
START	(QueueEmpty(Q_TX_SYNC)) && port.timeSyncAllowed	1	port.rxTime1 = port.rxTime0; port.rxTime0 = FlexTime(core.rxTimerTicks, core.tickRate, core.tickOffset); port.syncCount = (port.syncCount + 1) % 256; formed = PreForm(frame.systemTag, frame.uniqueID, frame.hopsCount, port.portPri, port.portID);	PREP
	—	2	—	START
PREP	frame.syncCount != port.syncCount	1	port.syncCount = frame.syncCount;	START
	—	2	frame.rxDeltaTime = rxDeltaTime = port.rxTime1 - frame.lastFlexTime; txDeltaTime = frame.deltaTime; port.propTime = (txDeltaTime + rxDeltaTime)/2;	NEXT
TEST	Best(formed, core.precedence)    core.slavePort == port.portID	1	core.slavePort = port.portID; core.precedence = formed; core.slaveCount = core.offsetCount; core.myOffset = (txDeltaTime - rxDeltaTime)/2; core.flexOffset = core.linkOffset + frame.offsetTime + core.myOffset;	NEAR
	—	2	—	START
NEAR	port.rateCount != core.rateCount	1	port.rateCount = core.rateCount; core.myDiffRate = TBD();	FINAL
	—	2	—	
FINAL	—	1	rate = core.diffRate; core.diffRate = frame.diffRate + core.myDiffRate; Compensate(core, rate);	START

**Row START-1:** A new frame has arrived; compute this arrival time; save previous arrival time.

**Row START-2:** Wait for a new timeSync frame to arrive.

**Row PREP-1:** Discard timeSync frames when their sequence numbers are unexpected.

**Row PREP-2:** Compute *rxDeltaTime* and save the observed *txDeltaTime* values.

1 **Row TEST-1:** The best precedence slave port is marked and the keep-alive timeout is cleared.

2 **Row TEST-2:** Otherwise, no common timer values are updated.

4 **Row NEAR-1:** At the end of the lower-rate interval, the rate differences are calculated.

5 **Row NEAR-2:** Otherwise, no rate differences are calculated.

7 **Row FINAL-1:** Update the offset and rate values.

### 9 **7.3.6 TimerTxCompute state machine**

#### 11 **7.3.6.1 TimerTxCompute state machine definitions**

13 The following definitions are used within this subclause:

15 NULL

16 A constant that indicates the absence of a slave-port identifier.

17 Q\_TX\_FRAME

18 See 7.2.1.

#### 19 **7.3.6.2 TimerRxCompute state machine variables**

21 The following variables are used within this subclause:

23 *core*

24 *frame*

25 *port*

26 See 7.2.2.

#### 28 **7.3.6.3 TimerRxCompute state machine routines**

30 The following routines are used within this subclause:

32 *Enqueue(queue, frame)*

33 *QueueEmpty(queue)*

34 See 7.2.3.

**7.3.6.4 TimerTxCompute state table**

The TimerTxCompute state machine provides the arguments for the timeSync frame, as specified in Table 7.7. The notation used in the state table is described in 3.4.

**Table 7.7—TimerTxCompute state table**

Current		Row	Next	
state	condition		action	state
START	(QueueEmpty(Q_TX_FRAME)) && port.offsetCount != core.offsetCount	1	port.offsetCount = core.offsetCount;	SEND
	—	2	—	START
FORM	core.slavePort == NULL	1	core.flexRate = frame.offsetTime = core.myOffset; rate = core.diffRate; core.diffRate = frame.diffRate = 0; Compensate(core, rate);	FINAL
	—	2	frame.offsetTime = core.flexOffset; frame.diffRate = core.diffRate;	
FINAL	—	1	frame.lastFlexTime = FlexTime(port.txTimerTicks, core.tickRate, core.tickOffset); frame.deltaTime = rxDeltaTime; frame.lastBaseTime = FlexTime(port.txTimerTicks, core.baseRate, 0); Enqueue(Q_TX_FRAME, frame)	SEND

**Row START-1:** When space is available and the time has arrived, transmit the next timeSync frame.

**Row START-2:** Wait until the next timeSync transmission time.

**Row FORM-1:** The grand-master station processing is distinct.

The grand-master sets the timer offset value and assumes a zero-valued cumulative rate difference.

**Row FORM-2:** The slave station processing is distinct.

The slave provides cumulative timer offset and rate-difference values.

**Row FINAL-1:** When space is available and the time has arrived, transmit the next timeSync frame.