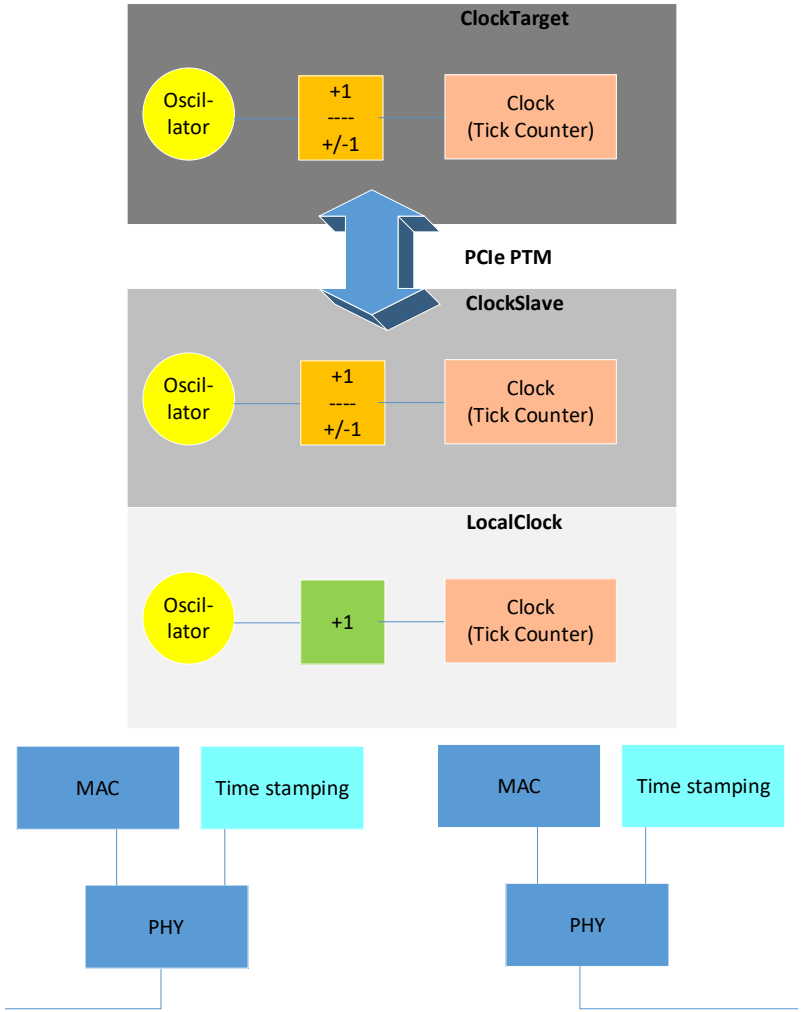
# Model for the synchronization of a ClockSlave

Rodrigo Ferreira Coelho, Siemens AG
Dragan Obradovic, Siemens AG
Günter Steindl, Siemens AG

V01

# Implementation model assumed for this presentation




The statement of this presentation are in general independent from this implementation model.

The error sources and thus the error contribution may differ depending on the chosen implementation.

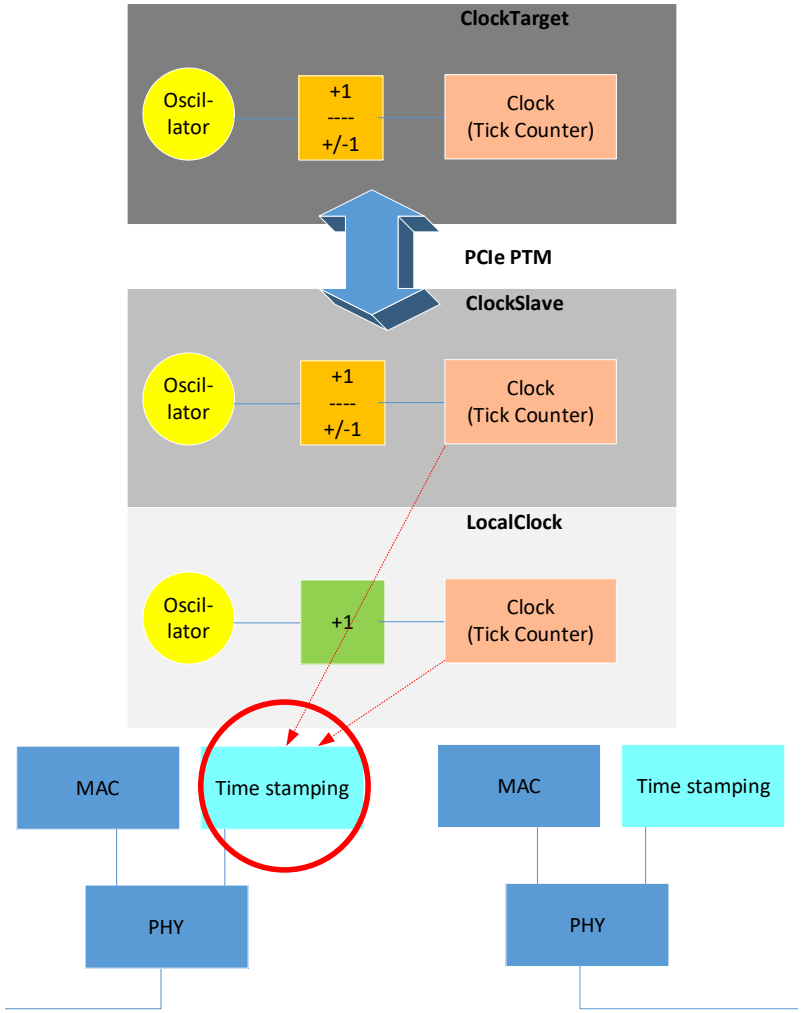
Example 1:
LocalClock, ClockSlave and ClockTarget may share an oscillator.

Example 2:
ClockSlave and ClockTarget may share an oscillator.

Additionally, the model is independent whether a PTP relay is implemented or not.

# Controlling the ClockSlave

# Controlling the ClockSlave
# Receive timestamp




Each received sync message is timestamped with both LocalClock and ClockSlave time.
This is either done getting these two timestamps by hardware or by translating the LocalClock timestamp into ClockSlave time by software.

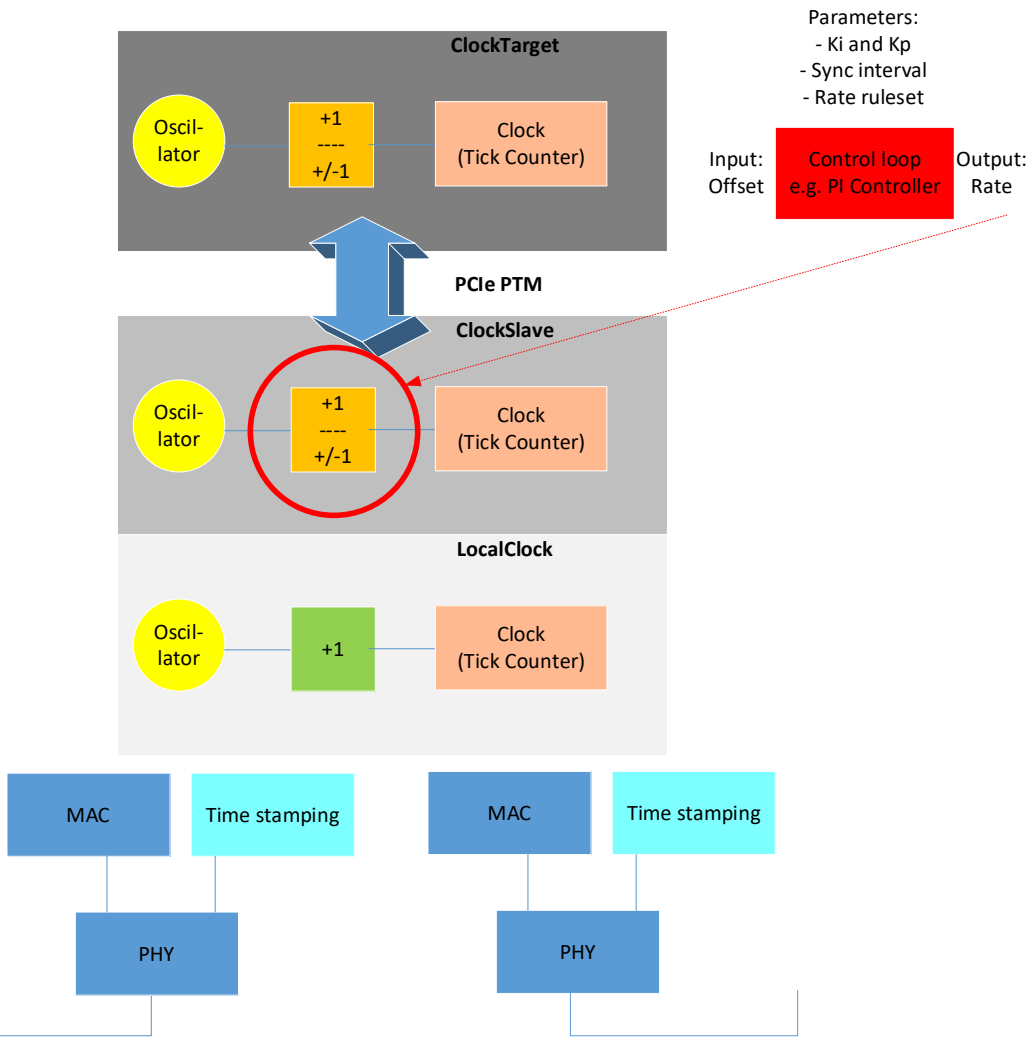
- if done by hardware, no additional error through “translation” in software

One step:
The Origin time, the accumulated latency and the pdelay is added and the value compared with the ClockSlave timestamp

- Estimated ClockMaster time = Origin time + accumulated latency + pdelay
- Offset = ClockSlave timestamp - Estimated ClockMaster time

Two step:
Same as one step, but the calculation is done when the follow up message is received

# Controlling the ClockSlave Control loop




Each calculated Offset value (time interval e.g. in nanoseconds) is used as an input for a PI controller which calculates the “new” rate controlling the ClockSlave time.

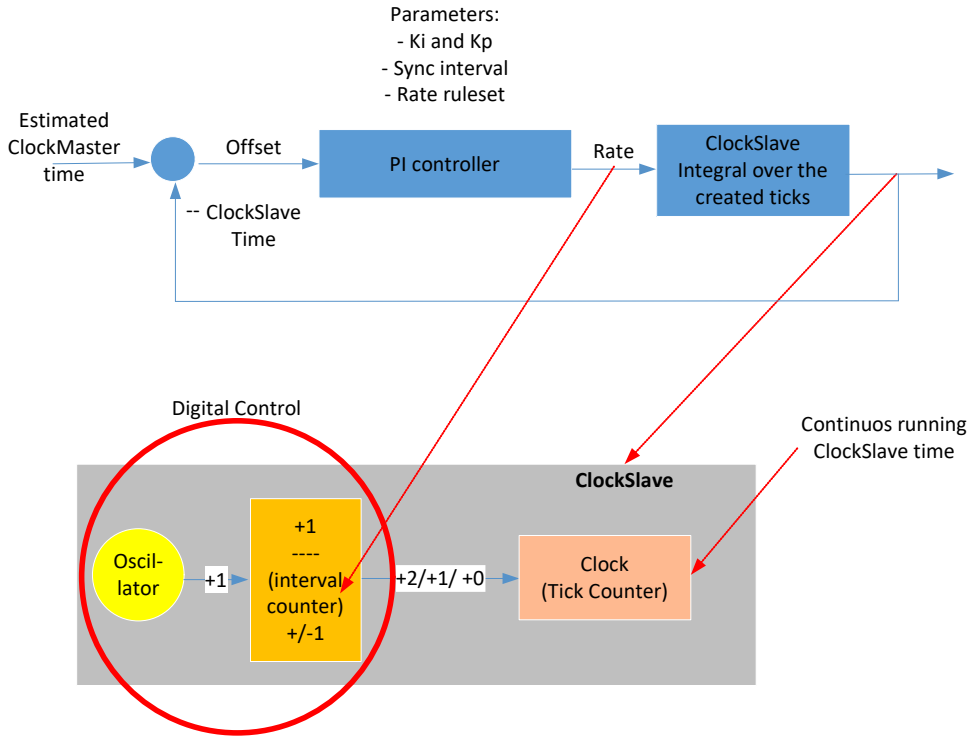
The PI controller ensures that the rules defined for the changes of the value of the ClockSlave time are kept.

Example:
To simplify the hardware implementation, this rate is expressed as a “number of ticks” (interval) and as “faster(+1) / slower(-1)” (action).

If the ClockSlave was to “slow”, then for each interval an additional “tick” will be added to ClockSlave time.

If the ClockSlave was to “fast”, then for each interval the “tick” will be skiped and thus the ClockSlave time is not incremented.

# Controlling the ClockSlave
# Control loop




Offset
Estimated ClockMaster time – ClockSlave time when receiving a sync message

- The PI controller is only executed when a sync message is received. Thus, the configured Rate stays constant until the next sync message is received.

Rate
Offset correction factor calculated as an interval in ticks in which an additional +1 is added or a +1 is skipped.

PI controller
The PI controller applies
- Ki and Kp

and
- uses the configured sync interval and the defined ruleset for allowed rate values.

Digital control for the ClockSlave time counter
Implementing the rate as offset correction

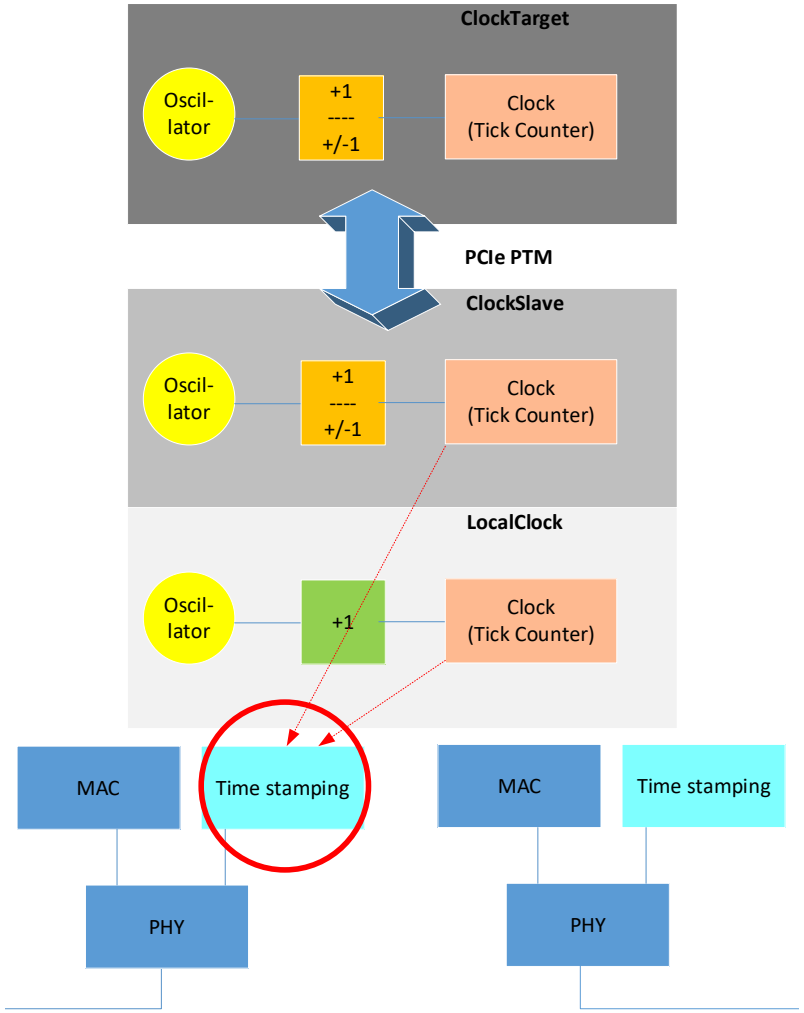
# Controlling the ClockSlave
# Conclusion

This model doesn’t calculate (and thus not use) a rate ratio between the ClockMaster and the ClockSlave oscillator for the control of the ClockSlave time.

It just calculates the estimated offset between the estimated ClockMaster time and the ClockSlave time and tries to minimize this offset to zero.

# Rate ratio calculation

# Rate ratio calculation
# Possible ways




The rate ratio between the ClockMaster and the ClockSlave can be calculated in multiple ways.

Example 1:
Neighbor rate ratio as specified in 802.1AS

Example 2:
Using the sync message receive timestamp similar to the concept explained at slide 4.
With each sync message, both, the receive timestamp and the estimated ClockMaster time are stored.
A sliding window of at least 200ms is used to calculate the time interval for the stored estimated ClockMaster times and associated the LocalClock times.
The ratio between these intervals is used as rate ratio.

Example 3:
“Using the ClockSlave time while synchronized”
Use of synchronized ClockSlave time (implicit applied rate ratio) as hardware optimization in conjunction with residence times << 1ms for the implementation of one step.

# Rate ratio calculation
# Filtering

The calculated rate ratio values are filtered in multiple ways:

1. Is the rate ratio value in an allowed range (e.g. values above 250ppm are assumed erroneous if +/-50ppm oscillators are used)

2. A median filter – median out of the last seven values – is applied

# Rate ratio calculation
# Conclusion

## ClockSlave control

This model doesn’t calculate (and thus not use) a rate ratio between the ClockMaster and the ClockSlave oscillator for the control of the ClockSlave time.

It just calculates the estimated offset between the estimated ClockMaster time and the ClockSlave time and tries to minimize this offset to zero.

## Rate calculation (for pdelay and residence time compensation)

It does calculate the rate ratio concurrently for pdelay and residence time compensation based on the stored timestamps / time intervals.

-> This model works if **no** pdelay messages are used due to exact measured pdelay value in case of aerospace or in-car too.

# Conclusion

The offset between ClockMaster and a ClockSlave
and
the rate ratio between them can be calculated and controlled in multiple ways.

Not all of them may fit for industrial automation!