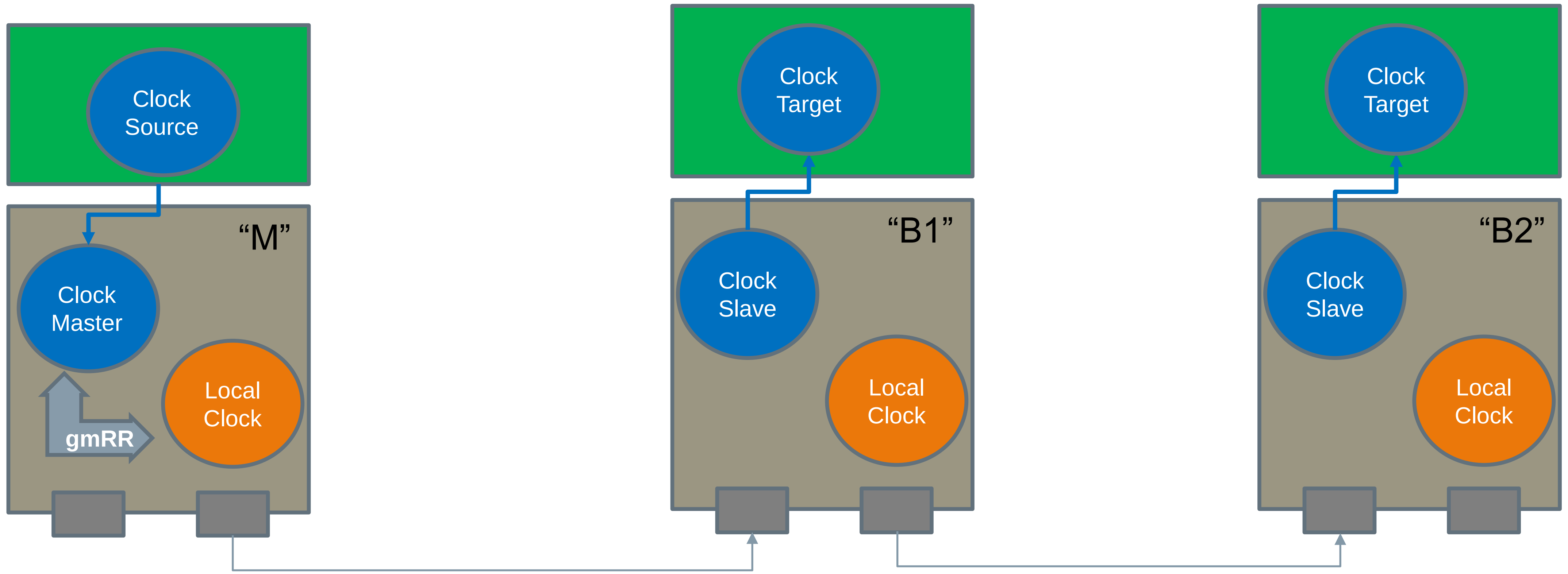
Controller Design for ClockSlaves

27.06.2022

Dragan Obradovic, Siemens AG

Agenda

Different Clocks and ClockMaster-Time Propagation



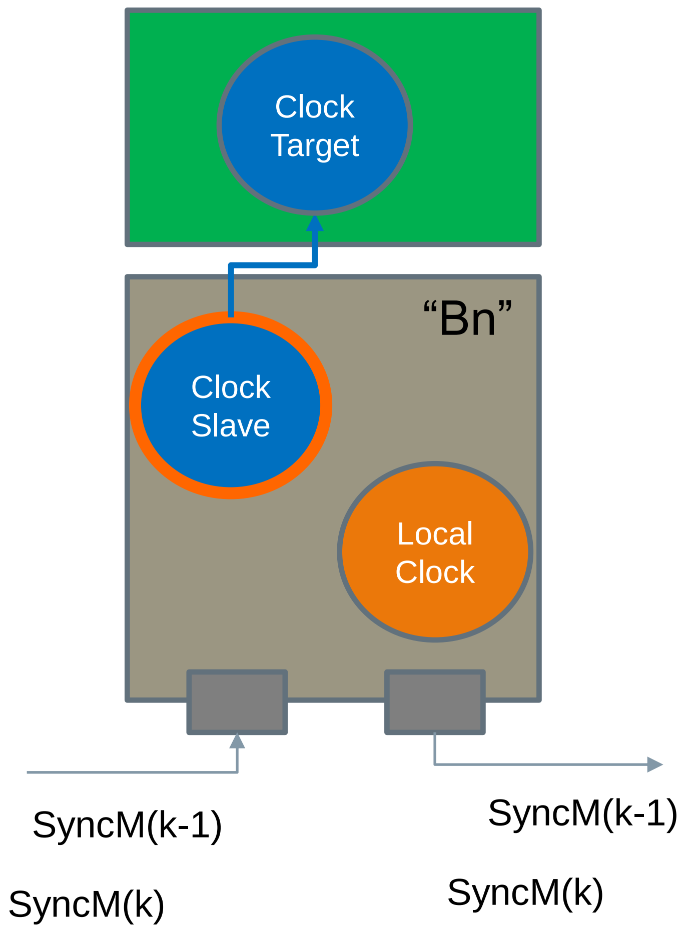
SyncM(CM_Time_orig, RR=gmRR, CM_Time_correction(@"M"))^k

SyncM(CM_Time_orig, **RR=gmRR*NRR(@B1), CM_Time_correction(@"M, B1"**))^k

"Time" = number of tics (counter value)

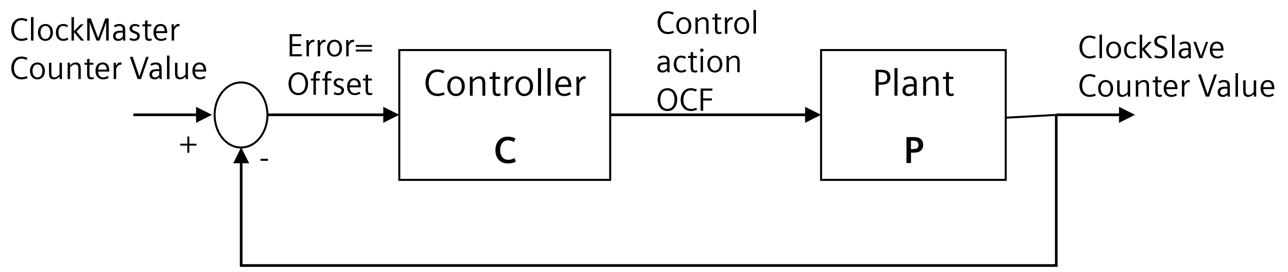
Control Problem: Make ClockSlave Track ClockMaster

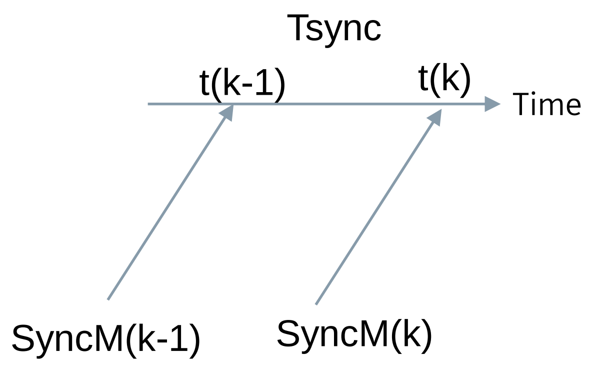
- This is a tracking problem, which is a standard control problem
- ➔ ClockSlave should track the ClockMaster counter value based on the information it receives from SyncMessages (after compensation of the Propagation Delay)
- Characteristics of this control problem
  - ClockSlave's counter value can only change gradually (periodic jumps of 1 extra tick allowed, large instantaneous jumps are not permitted)
  - ClockSlave's counter value (output) should be always available
  - ClockSlave is a SW-disciplined Clock. Hence, it is a disciplined counter connected to an oscillator (e.g. LocalClock).
  - Sync Messages are available periodically (with some deviation). Hence, the tracking controller can update the ClockSlave only at these instants.
  - Sync Messages content is noisy, their arrival interval can vary and they can even be lost. Hence, the control solution has to be robust!

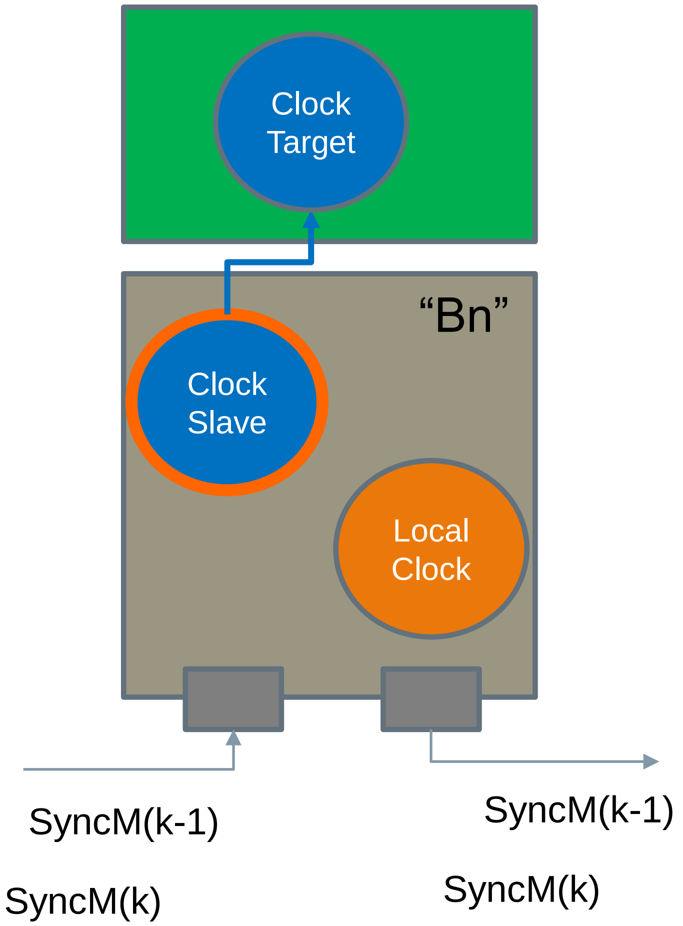


Closed Loop Control Implementation of ClockSlave tracking the ClockMaster

- OCF: Offset Compensation Factor







Plant: Counter of ClockSlave as controlled Counter of LocalClock

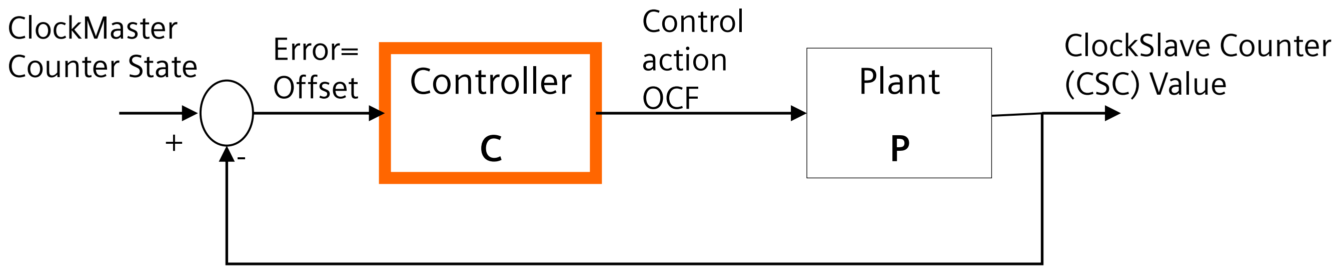
FRC: Free Running Clock (Local Clock)

Offset: difference of counter values, i.e. error

$f_{FRC}(t)$: frequency of FRC at time t

OCF:

- control signal kept constant between two successive Sync messages
- Scales the frequency of the Local Clock
- Can be used (its value (k-1)) to calculate the ClockSlave counter value at any time between two successive Sync messages [(k-1) , (k)]!



$$CSC(t(k)) = \int_0^{t(k)} OCF(t) \cdot f_{FRC}(t) \cdot dt =$$

$$= CSC(t(k-1)) + OCF(t(\,k-1)) \cdot \int_{t(k-1)}^{t(k)} f_{FRC}(t) \cdot dt$$

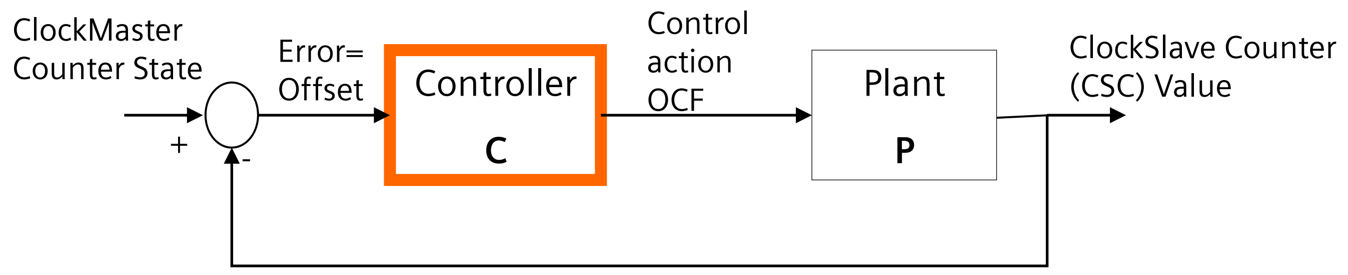
$$CSC(k) = CSC(k-1) + OCF(k-1) \cdot NumTicsFRC(within\ Tsync)$$

Controller: a Discrete Time PI controller

Offset is the error (err)

Nominal frequency f_{FRC} of the Free Running Clock known

Controller: discrete time PI controller with Tsync as sampling rate



$$OCF(t(k)) = K_p \cdot err(t(k)) + K_I \cdot \int_0^{t(k)} err(t) \cdot dt =$$

$$= K_p \cdot err(t(k)) \pm K_p \cdot err(t(k-1)) + K_I \cdot \int_0^{t(k-1)} err(t) \cdot dt + K_I \boxed{\int_{t(k-1)}^{t(k)} err(t) \cdot dt}$$

To be discretized with ZOH (constant value during the discretization interval).

$$OCF(k) = OCF(k-1) + K_p \cdot (err(k) - err(k-1)) + K_I \cdot err(k-1) \cdot Tsync$$

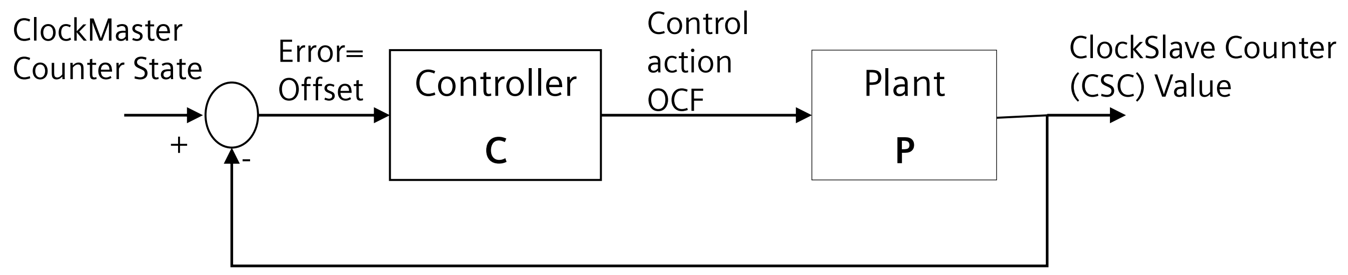
$$= OCF(k-1) + K_p \cdot (err(k) - err(k-1)) + K_I \cdot err(k-1) \cdot NumTicsFRC(within\, Tsync)/f_{FRC}$$

Controller: a Discrete Time PI controller

Offset is the control error (err)

Nominal frequency f_{FRC} of the Free Running Clock known

Controller: discrete time PI controller with Tsync as sampling rate



$$OCF(k) = OCF(k-1) + K_p \cdot (err(k) - err(k-1)) + K_I \cdot err(k-1) \cdot Tsync$$

$$= OCF(k-1) + K_p \cdot (err(k) - err(k-1)) + K_I \cdot err(k-1) \cdot NumTicsFRC(within\ Tsync)/f_{FRC}$$

Question: How is OCF(k) implemented? We cannot change the frequency but only the clock (the number of ticks)!

→ Solution: OCF is used to calculate the OCF_Interval

→ OCF_Interval tells you when to add or subtract one tick from the ClockSlave clock!

Control Signal: from OCF to OCF_Interval

$$Controller:\ OCF(k) = OCF(k-1) + K_p \cdot (err(k) - err(k-1)) + K_I \cdot err(k-1) \cdot NumTicsFRC / f_{FRC}$$

f_{FRC}: frequency of FRC, without compensation

$f_{CS} = OCF \cdot f_{FRC}$: The ClockSlave frequency obtained by scaling the frequency of FRC

Goal:

- find the time interval of length T after which the two counters (of FRC and CS clocks) differ by 1 tick, and
- express this time interval in the number of tics of the Free Running Clock (e.g. Local Clock)

$$T \cdot f_{CS} = T \cdot f_{FRC} \pm 1$$

$$T = \frac{\pm 1}{f_{CSC} - f_{SFRC}} = \frac{\pm 1}{f_{FRC}} \cdot \frac{1}{OCF - 1} \Rightarrow \quad \boxed{OCF_{interval} = \frac{\pm 1}{OCF - 1}}$$

(in praxis can the number of updates be limited)

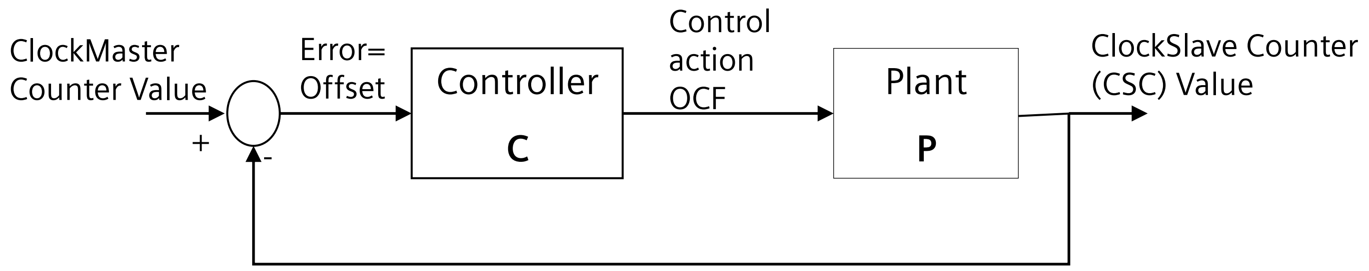
→ If OCF>1, add 1 extra tick after the OCF_interval,

→ If OCF<1, subtract 1 tick after the OCF_interval,

→ If OCF=1, do nothing (no correction needed)

Tuning of the Controller Parameters

$$Controller\colon\ OCF(k) = OCF(k-1) + K_p \cdot (err(k) - err(k-1)) + K_I \cdot err(k-1) \cdot NumTicsFRC/f_{FRC}$$



There are two parameters pro controller → in a line with 100 Slaves there are 200 parameters

Our Plant is an integrator with a large gain (the FRC frequency). Idea: find good parameters for the integrator with gain one, and scale them with the FRC frequency.

There are many methods for tuning the parameters of a single PID controller. But, we had another goal:

Goal: Find the common parameters for all Slave elements which guarantee 1µs performance, in the presence of :

- Noise (stamping and quantization) errors
- Frequency drift of the Master and Slave clocks
- …

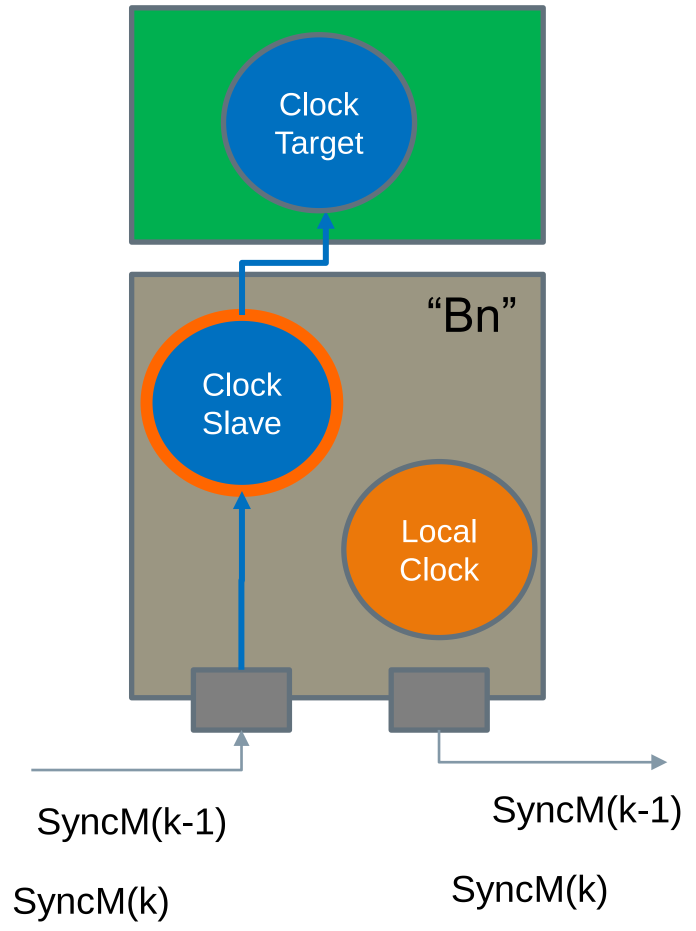
Done through (a lot of) simulations (for the Siemens Profinet case)!

Application of the controlled ClockSlave

Standard Application:

- Use the ClockSlave only for providing the Master Clock Information to the Application

→ The ClockSlave at element “n” does not influence the estimation of nRR and RR at that element, nor the update of the Sync message content before it is forwarded to the next Slave element “n+1”



Application of the controlled ClockSlave

Additional Application:

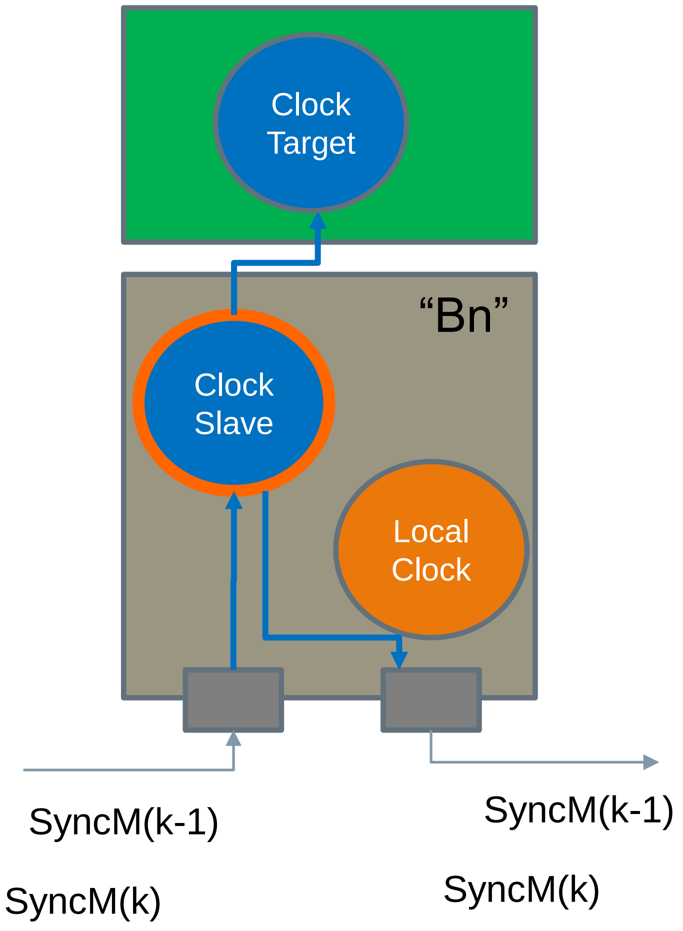
- Use the ClockSlave also for updating the Sync Message with the encountered Delays

→ Master Time Correction at "Bn" consists of:

- pDelay compensation
- Residence Time (RT) compensation

Two ways of calculating the pDelay and RTdelay:

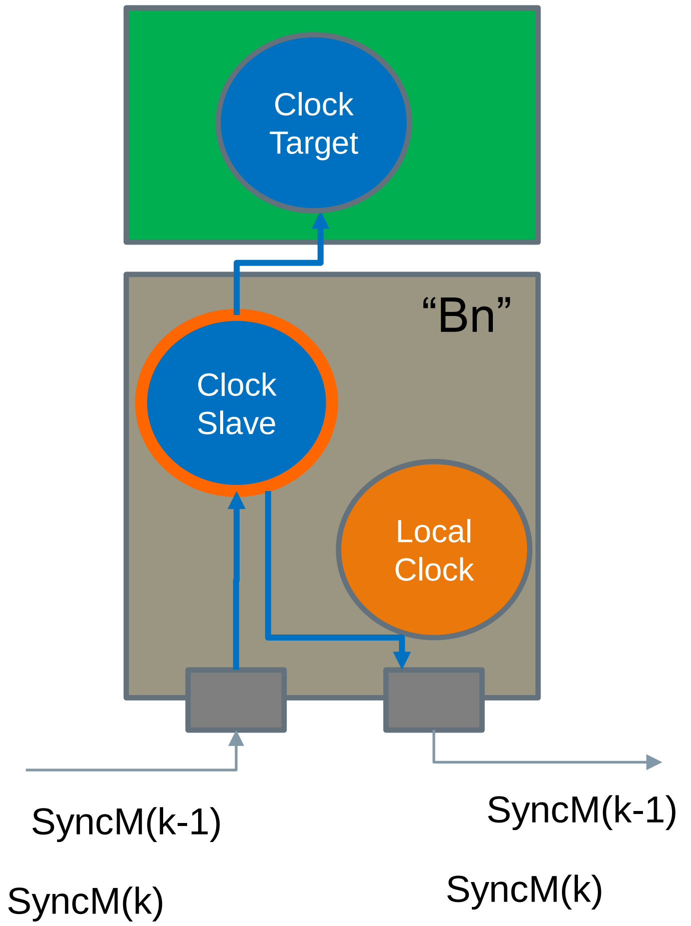
1. Use LocalClock time scaled to the Master Time by RR(M,B_n). ClockSlave is not used in this case!
2. *Use ClockSlave estimate of the ClockMaster for both pDelay and RTdelay calculations. This looks like a mixture of a Transparent and a Boundary Clock.*



Application of the controlled ClockSlave for pDelay and Residence Delay calculation

Use ClockSlave estimate of the ClockMaster for both pDelay and RTdelay calculations. This looks like a mixture of a Transparent and a Boundary Clock.

- Should be used only when the slave is in the “inSync” state (e.g. when the difference between the ClockMaster and ClockSlave counters is under the threshold)
- *If ClockSlave clock is used in the current and in the previous Slave elements (or the previous element is the GrandMaster) for the delay compensation, then the nRR and RR should be equal to 1 (within limits)!*
- *Hence, nRR and RR need not be calculated! If calculated, their deviation from 1 can be an additional information whether the element “n” is in the “inSync” state (t.b.d.)*

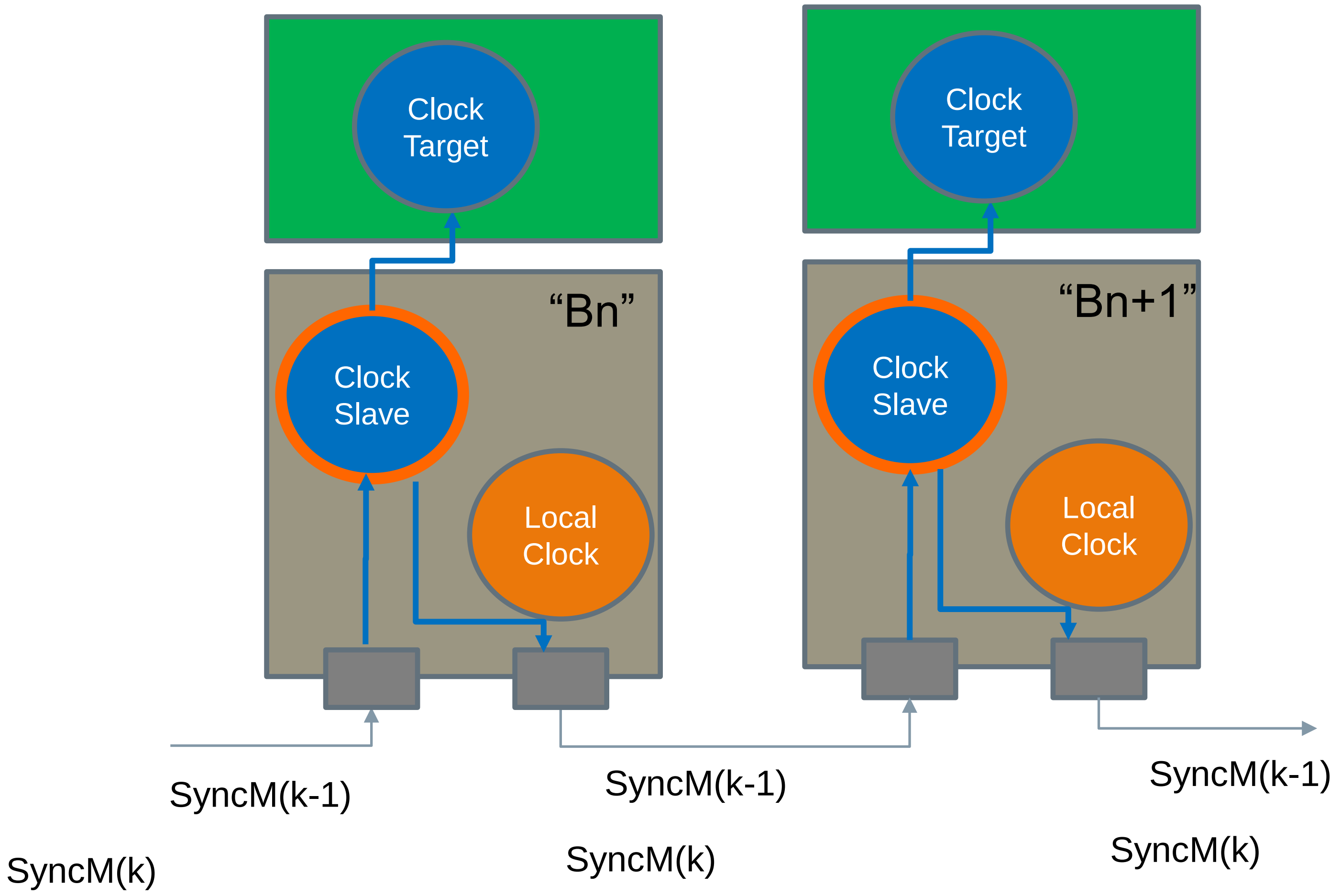


ClockSlave for Delay calculation → Propagation of the CS control loop signals over Slave elements

Sync Message at the output of “Bn” will depend on the control loop at this element (on the calculated OFC(Bn))

Sync Message at the output of Bn+1 will depend on the own control loop (OFC(Bn+1)) and also on the control loop at the previous element (OFC(Bn))

Question: how does OFC(Bn) propagates over elements?

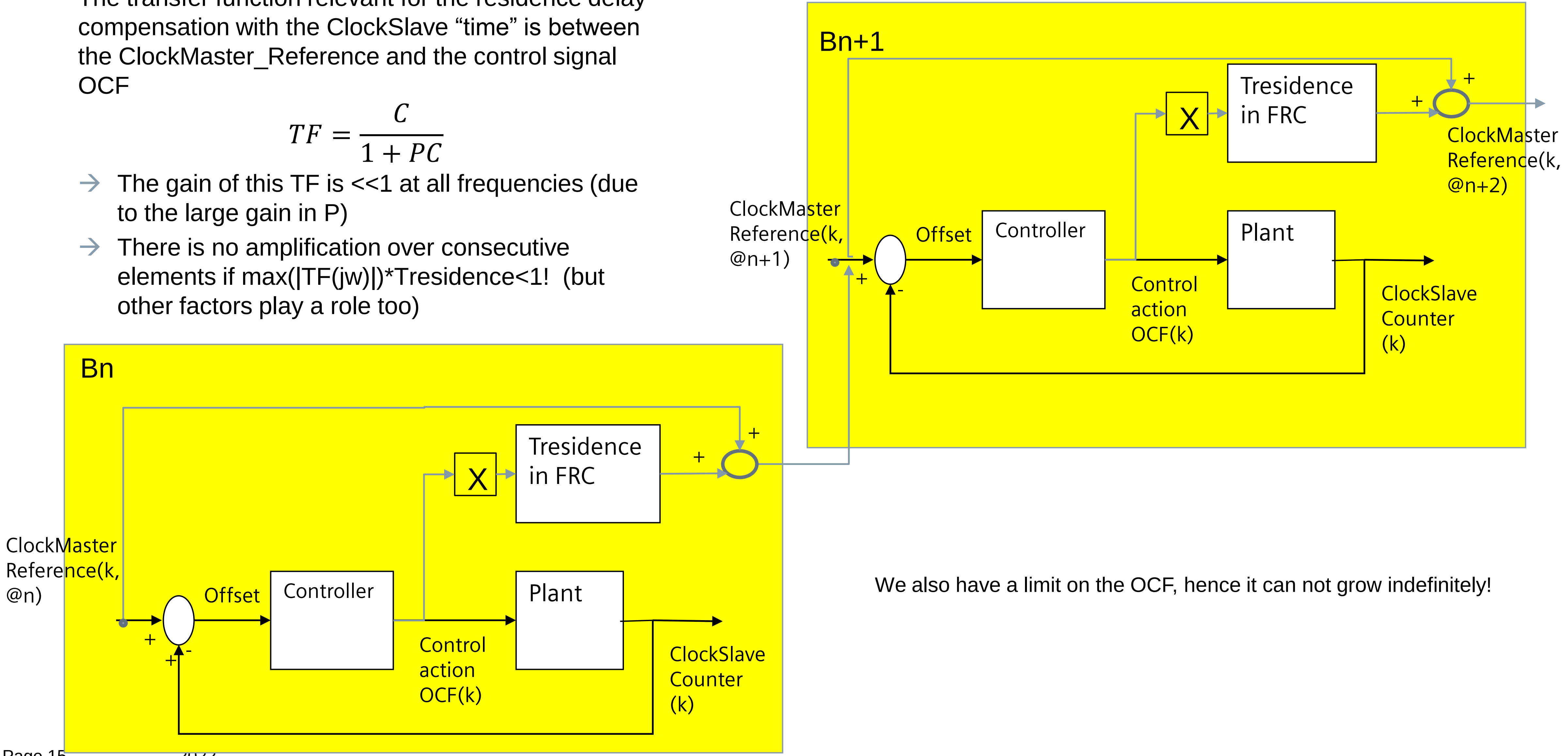


Using controlled clock to update Sync Message

The transfer function relevant for the residence delay compensation with the ClockSlave "time" is between the ClockMaster_Reference and the control signal OCF

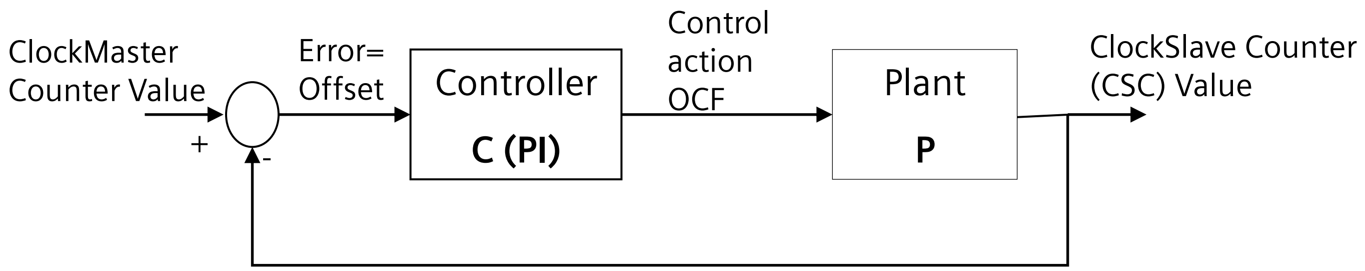
$$TF = \frac{C}{1 + PC}$$

- The gain of this TF is <<1 at all frequencies (due to the large gain in P)
- There is no amplification over consecutive elements if max(|TF(jw)|)*Tresidence<1! (but other factors play a role too)



We also have a limit on the OCF, hence it can not grow indefinitely!

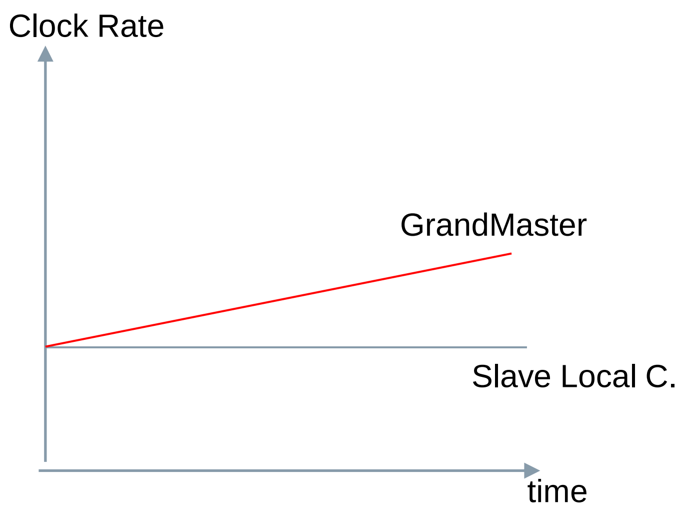
Can the PI-controlled ClockSlave help with delay compensation if Master Frequency is drifting? YES!



Case: the frequency of the GM clock has a constant drift in one direction (for some time), while the frequency of the slave’s Local Clock is constant

- The Offset (error) will have the same sign (positive).
- Kp will react proportionally to the current offset (pushes OCF up)
- Ki will further amplify this increase of OCF the longer the drift is present (due to the cumulation of the errors with the same sign)

- ➔ The Ki action “senses” that there is a drift and tries to compensate it! This is done without explicitly learning the drift!
- ➔ Hence, OCF has a built-in compensation of the drift, RR does not have it!
- The compensation is not ideal because of noise, delays, etc. But it has the correct sign!


Clock Rate
GrandMaster
Slave Local C.
time