Controller Design in Time and Frequency Domains

19.08.2022

Dragan Obradovic & Günter Steindl, Siemens AG

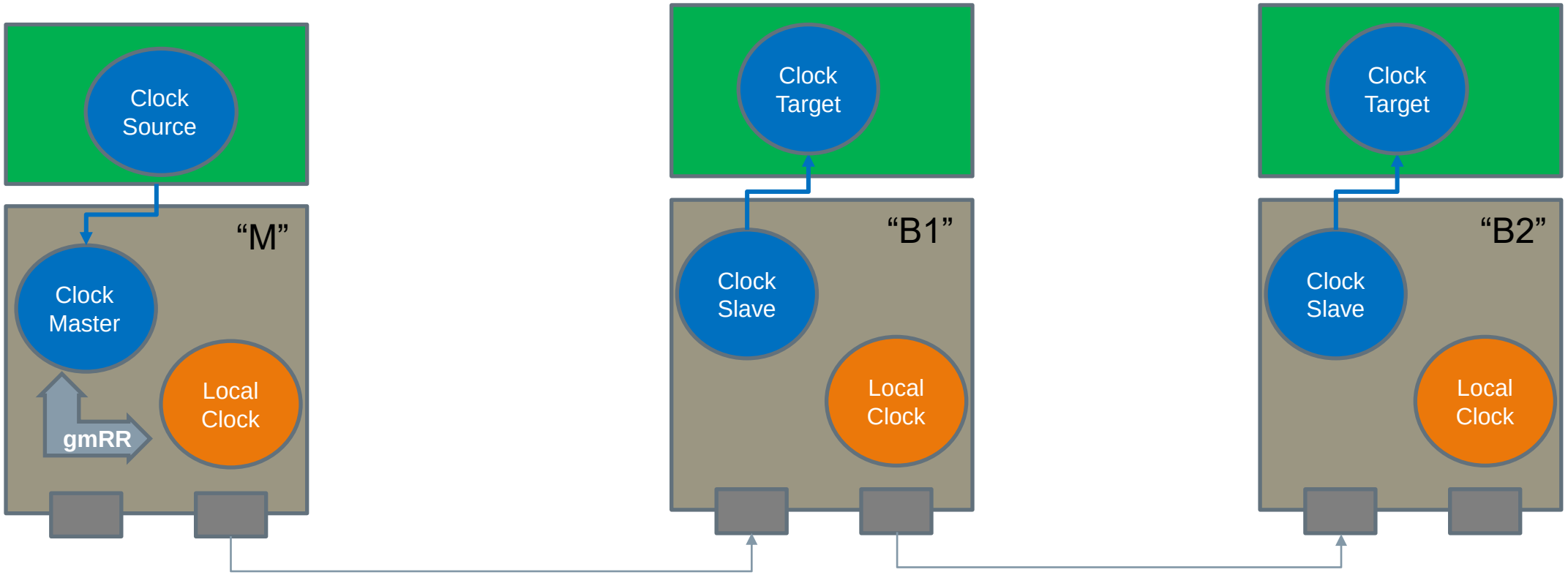
Agenda

1. **ClockSlave control problem formulation**

2. **Stability and Performance requirements**
   - **Typical requirements in time-domain**
   - **Typical requirements in frequency domain**

3. **ClockSlave control problem characteristics**

Different Clocks and ClockMaster-Time Propagation



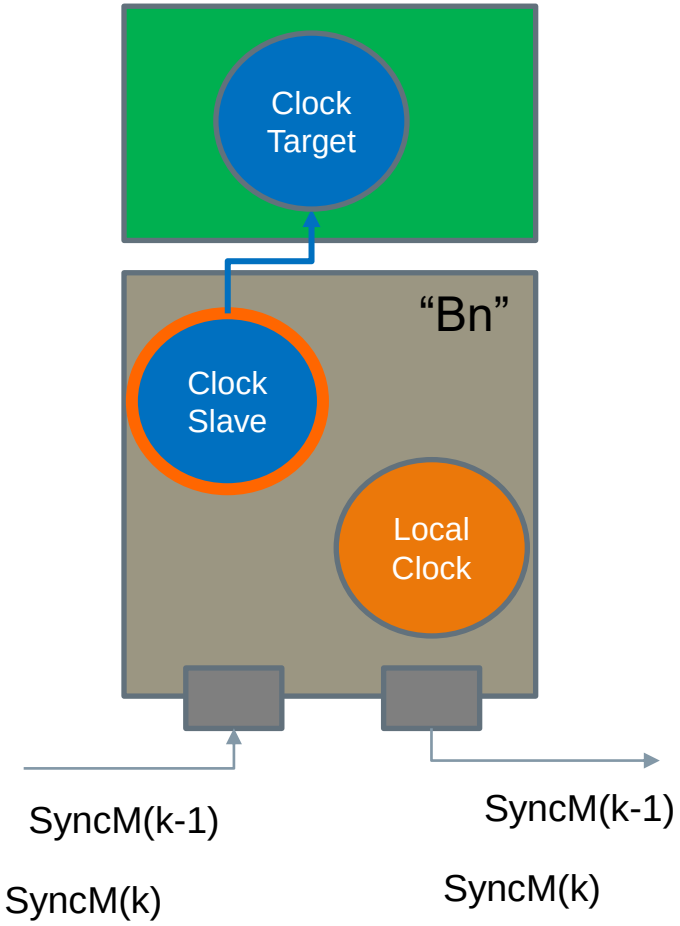
SyncM(CM_Time_orig, RR=gmRR, CM_Time_correction(@"M"))^k

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

"Time" = number of tics (counter value)

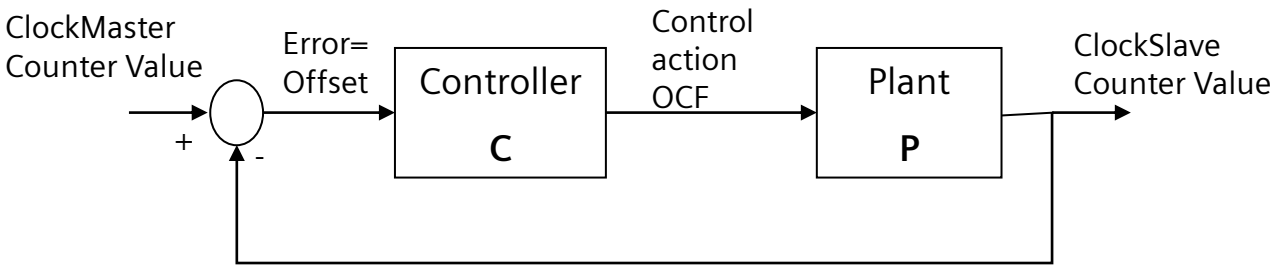
Control Problem: Make ClockSlave Track ClockMaster

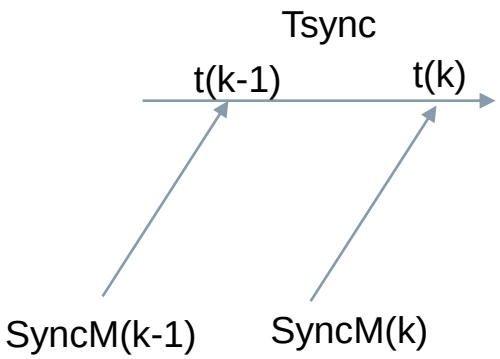
- This is a tracking problem, which is a standard control problem
- ➔ ClockSlave should track the ClockMaster counter value based on the information it receives from SyncMessages (after compensation of the Propagation Delay)
- Characteristics of this control problem
  - ClockSlave's counter value can 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!

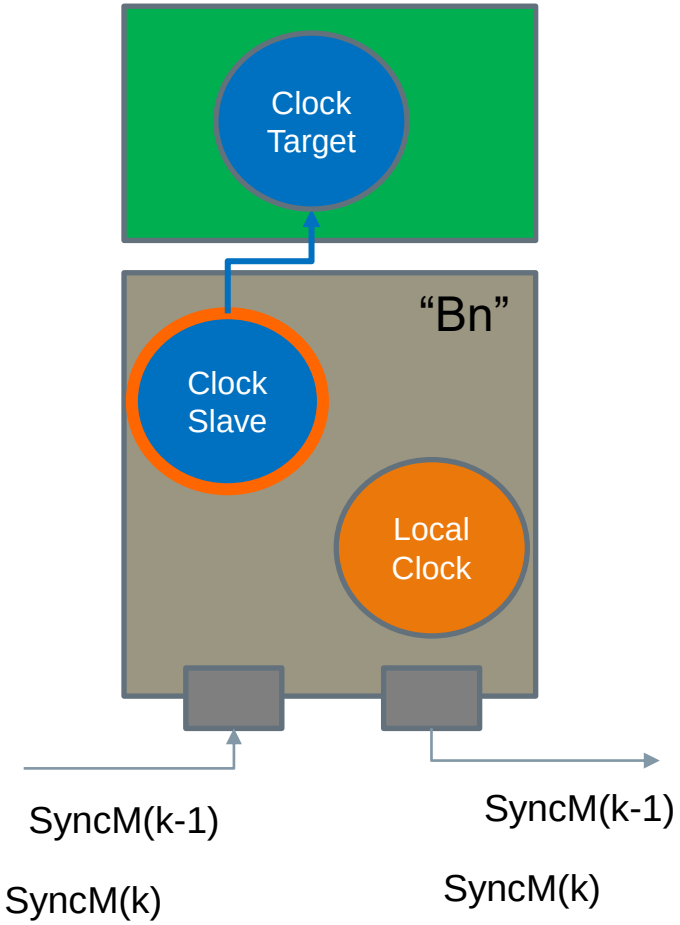


Closed Loop Control Implementation of ClockSlave tracking the ClockMaster

- OCF: Offset Compensation Factor



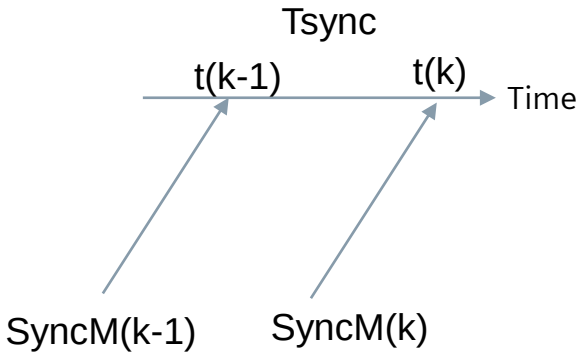


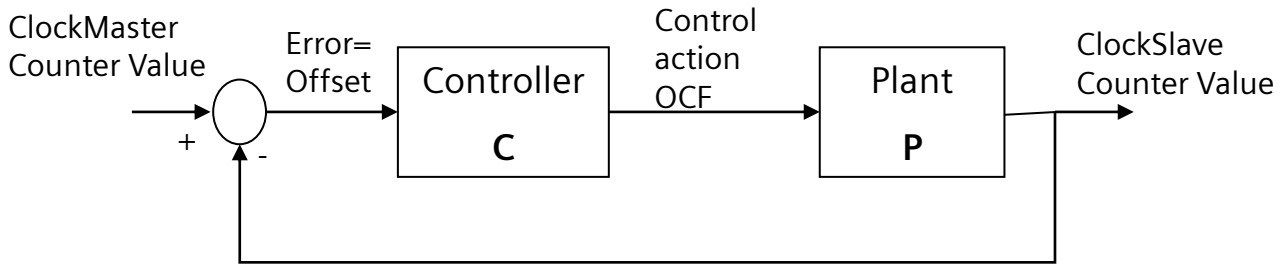


Closed Loop Control Implementation of ClockSlave tracking the ClockMaster

What is expected from the controller:

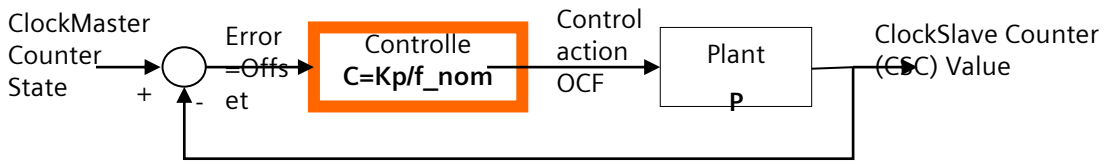
- **Stability**: after perturbations the system goes back to the normal behavior (e.g. signals do not explode, …). Stability is not guaranteed solely by defining bandwidth and maximum gain over frequencies!
- **Performance**: “good tracking properties” (e.g. fast, limited overshoot, limited settling time, noise-insensitiveness, robust to modelling errors and plant changes …)
  - All the above are easy to understand in time domain, but
  - Some of the above characteristics easier to “measure” in frequency domain



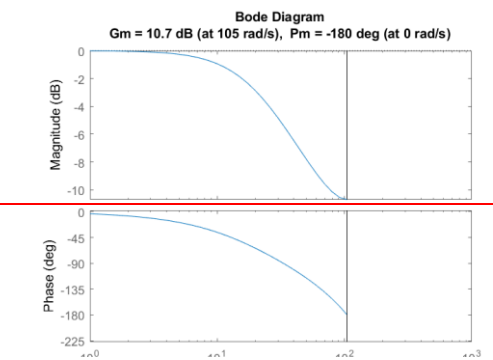


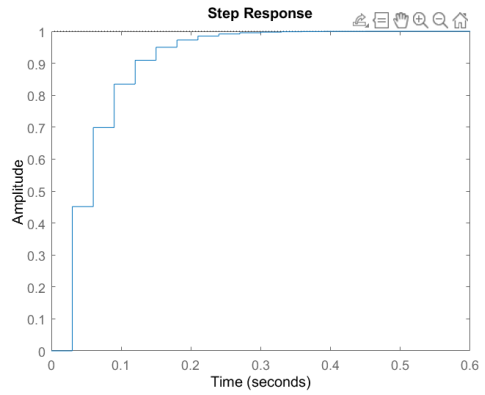
Simple example: Proportional controller (Kp only)

Let us look at the proportional controller and change the feedback loop from negative to positive

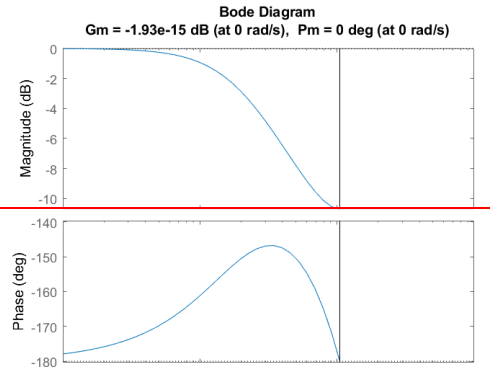


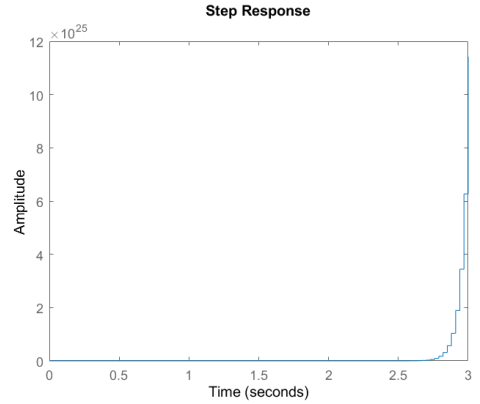
Negative Feedback





Positive Feedback





Frequency Response Magnitude plots identical → same maximum gains and same bandwidth!

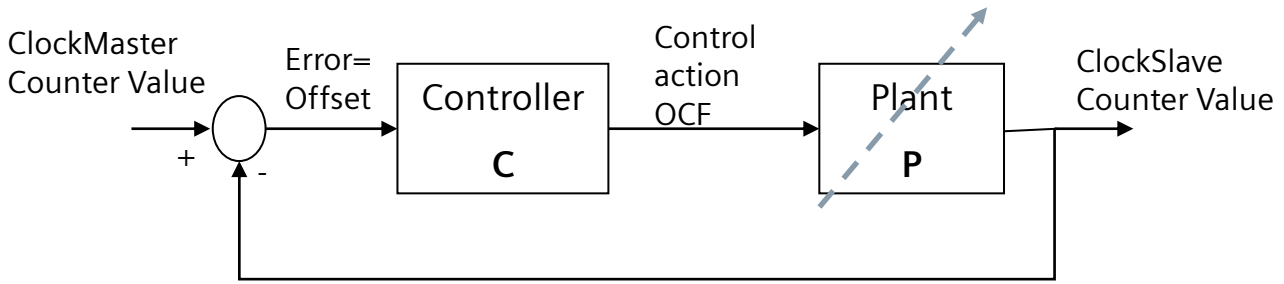
But the time response with respect to the step input explodes in the case of positive feedback

Maximum gain and bandwidth are not enough for guaranteeing stability! → Phase information over frequencies is as important as the magnitude information!

Controller Requirement: Closed Loop Stability

Main (normative) requirement for the controller: STABILITY

- **Stability**: can be checked in different ways:´→
  - looking at the poles of the closed-loop transfer function $TF(s) = \frac{PC(s)}{1+PC(s)}$ in the “s” or “z” domains
  - Inspection of the frequency plots of TF: Bode Plot, Nyquist Plot, …
  - Many different ways of choosing controller parameters to guarantee stability (model based, time-response based, …) for the given plant P
  - What if P is not completely known or it can change? Can we make stability robust to these changes?

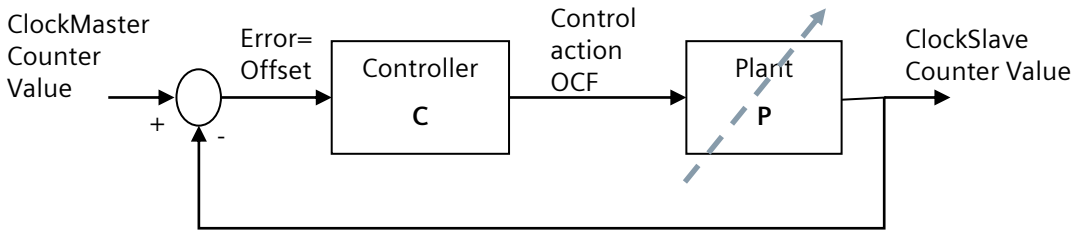


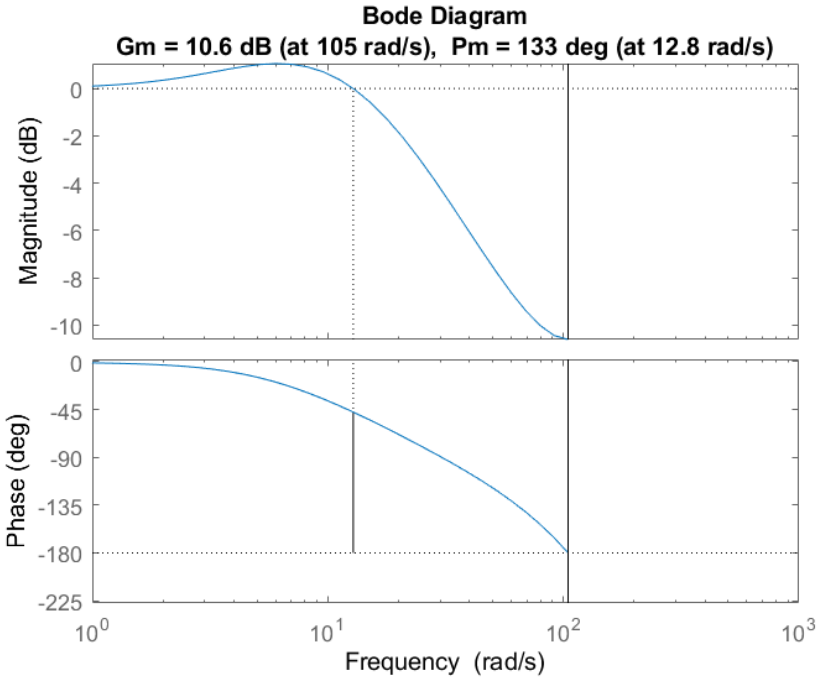
Controller Performance: Stability Robustness

Performance of the controller: STABILITY ROBUSTNESS (could be normative, t.b.d.)

- **Stability Robustness**: can be checked in the frequency domain, it tells us how much the gain and/or the phase of the open loop (e.g. due to the plant change or its modelling error) can change without making the system unstable

- → These are so called Gain-Margin and Phase-Margin



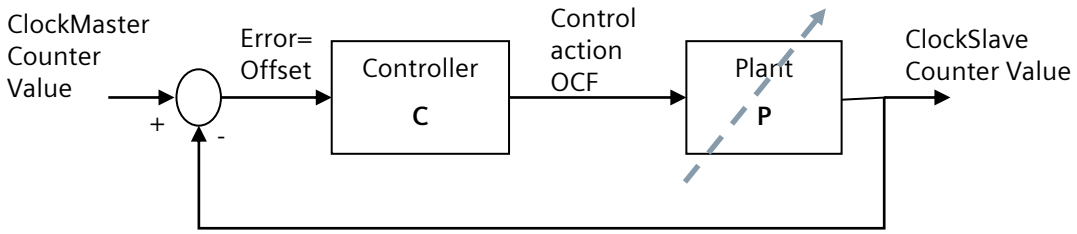


ClockSlave with the PI controller: Gain and Phase Margins

Controller Performance: Stability Robustness (time continuous systems)

Gain margin is defined as the amount of change in open-loop gain required to make the closed-loop system unstable. The gain margin is the difference between 0 dB and the gain at the phase cross-over frequency (determined by phase of −180°). If the gain margin is positive, then the closed-loop system is stable.

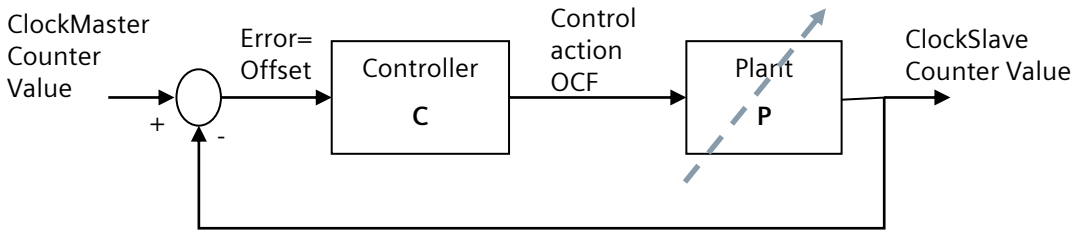
Phase margin is defined as the amount of change in open-loop phase required to make the closed-loop system unstable. The phase margin is the difference between 180° and the phase at the gain cross-over frequency (where the gain is 0 dB). If the phase margin is positive, the closed-loop system is stable.

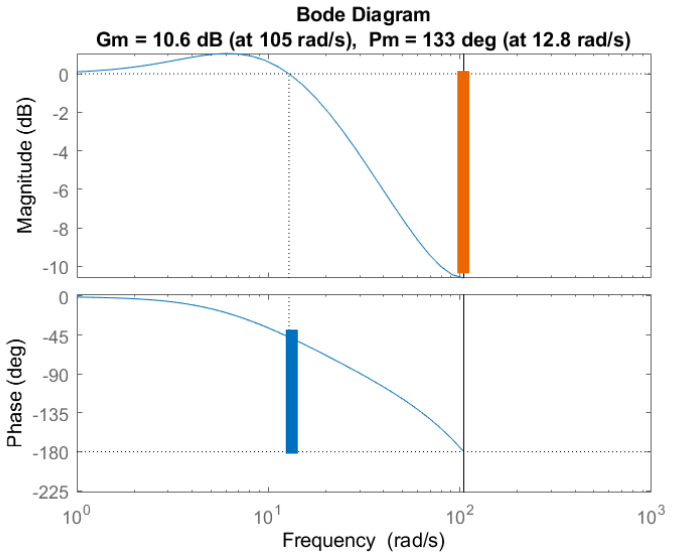


Controller Performance: Stability Robustness (time discrete systems)

Gain margin is defined as the amount of change in open-loop gain required to make the closed-loop system unstable. The gain margin is the difference between 0 dB and the gain at the phase cross-over frequency (where $\angle CP(\exp(j\omega Ts)) = -180°$). If the gain margin is positive, then the closed-loop system is stable.

Phase margin is defined as the amount of change in open-loop phase required to make the closed-loop system unstable. The phase margin is the difference between 180° and the phase at the gain cross-over frequency (where the gain is 0 dB, i.e. where $|CP(\exp(j\omega Ts))| = 1$). If the phase margin is positive, the closed-loop system is stable.
Phase margin → defines how large delays in the open-loop can be tolerated!



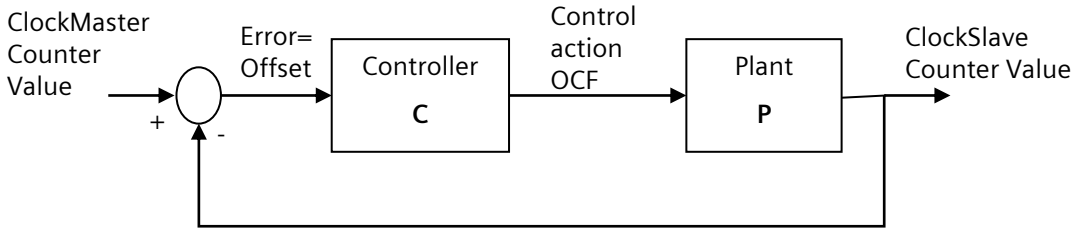


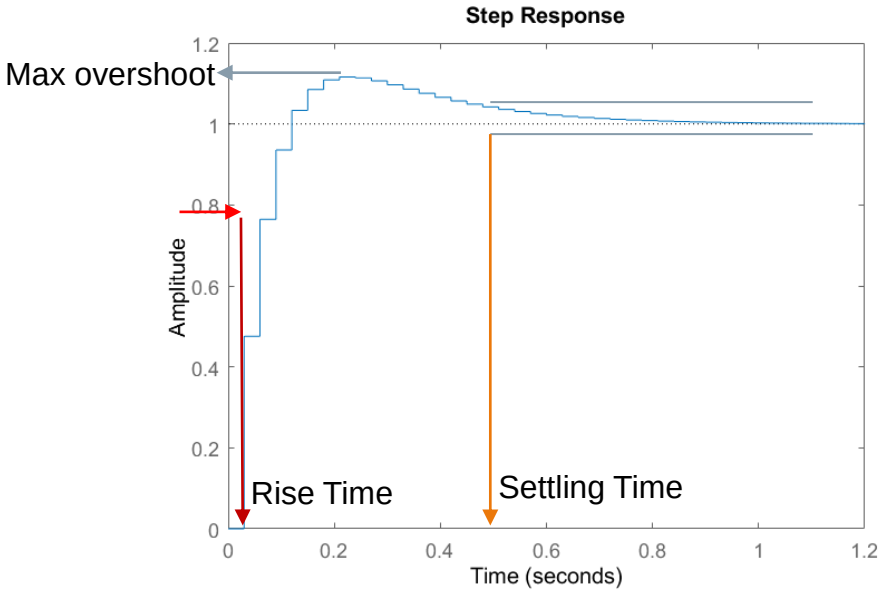
ClockSlave with the PI controller: Gain and Phase Margins

Controller Performance: Input Response

Performance of the controller: INPUT RESPONSE (could be normative, t.b.d.)

- **Input Response**: typical inputs are steps, ramps, etc. This is a time domain specification, and there are many controller optimization tools which look for controller parameters which provide required response.
- Example: Step response is described with RiseTime (to reach XX% of the step, the max overshoot, the settling time (when the signal is reaches and stays in the +- YY% error), …



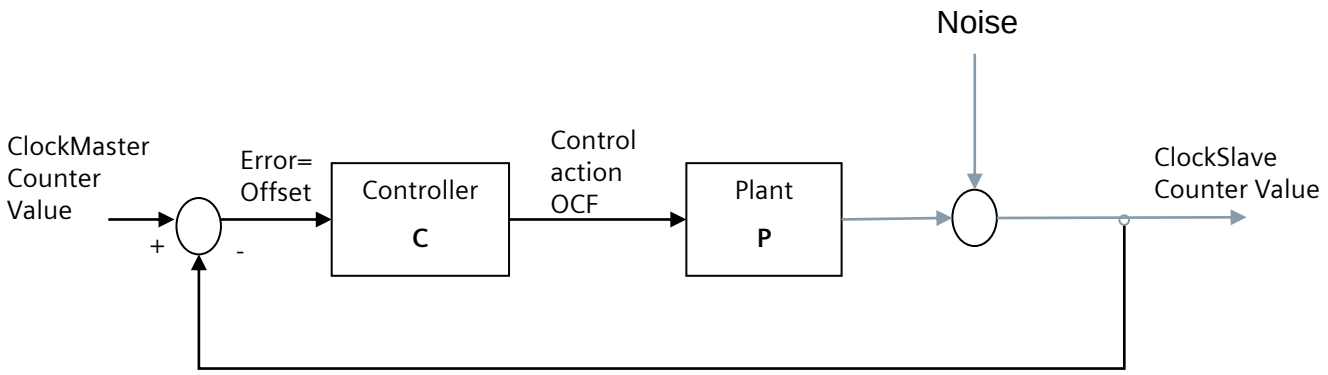


ClockSlave with the PI controller: Step Response

Controller Performance: Noise Cancellation

Performance of the controller: NOISE CANCELLATION (could be normative, t.b.d.)

- **Balance tracking with (measurement) noise cancellation**: there two different transfer functions CMCV→CSCV (closed-loop TF1) and noise→CSCV (sensitivity TF2)
- We would like to independently specify them but their sum is equal to one! → they are not independent
- Noise can also be at other locations in the loop



$$\mathrm{TF1} = \frac{CP}{1+CP}$$

$$\mathrm{TF2} = \frac{1}{1+CP}$$

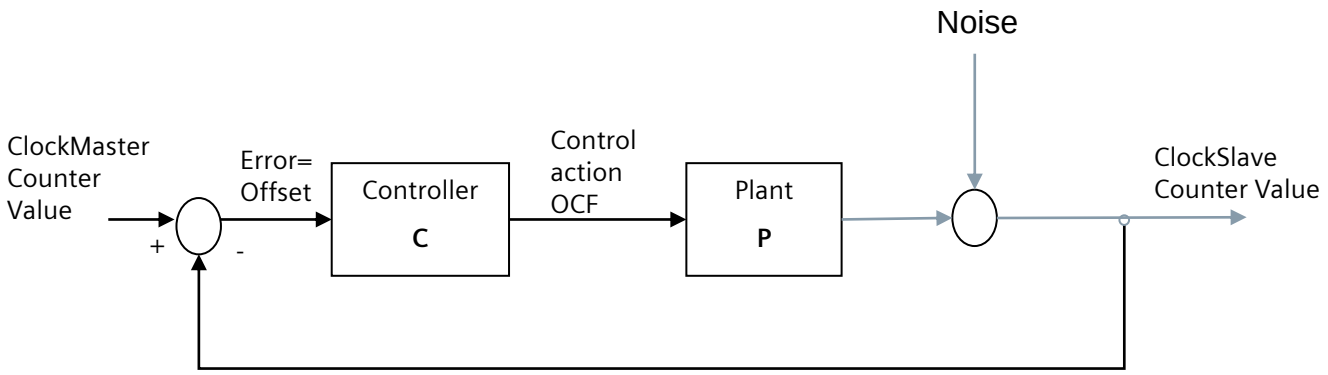
$$\mathrm{TF1} + TF2 = 1$$

$$CSCV = \frac{CP}{1+CP} \cdot CMCV + \frac{1}{1+CP} \cdot Noise$$

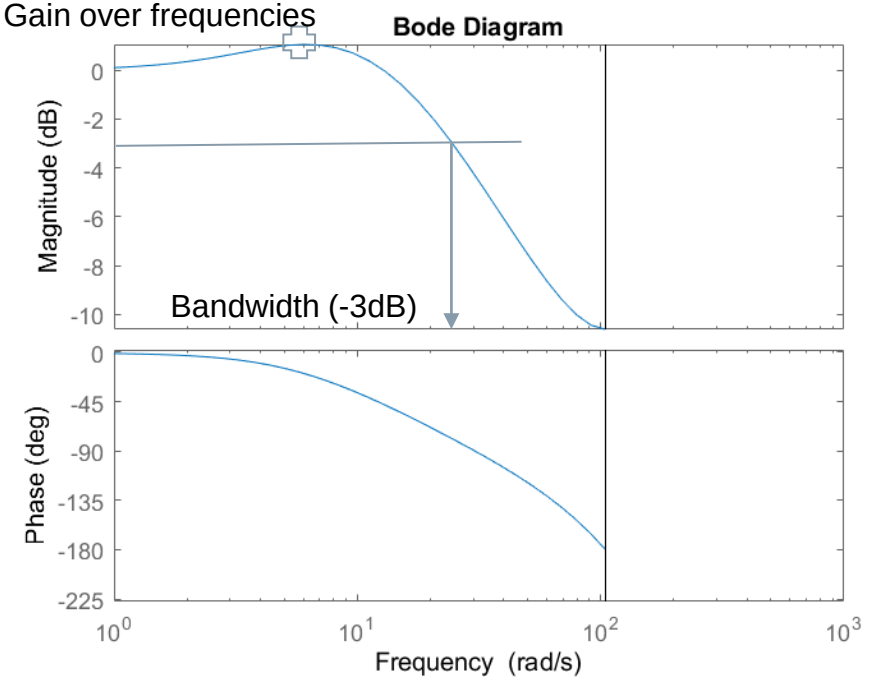
Controller Performance: Other Frequency Domain Charactersitcs

Performance of the controller: Other Frequency Domain Characteristics (could be normative, t.b.d.)

- **System Bandwidth:** region of frequencies where controller has enough gain to act (e.g. track input). Ideally noise on input signal should be small in this region
- **Max frequency gain**: maximum amplification of sinus signals (Hinf norm of the system)
- …



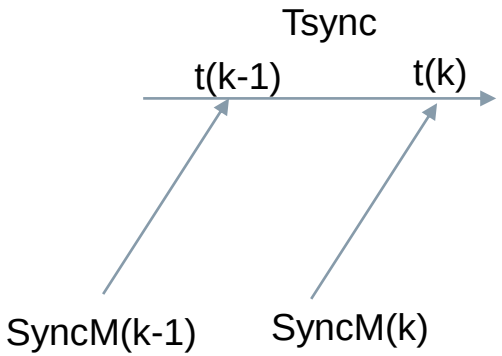
Max Gain over frequencies

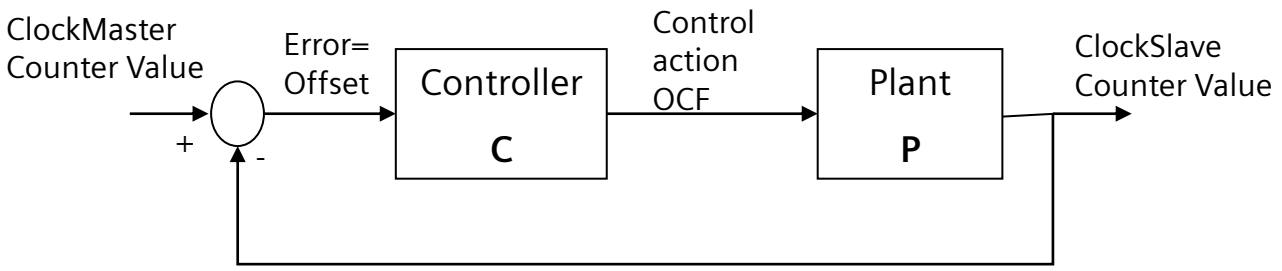


Controller in ClockSlave

This controller is influenced by the global ClockSync system parameters:

- It is a discrete time controller (FOH), whose sample time is Tsync → the controller "does not" see any changes (i.e. dynamics) in the input signals with frequencies > 1/(2*Tsync), which is the Nyquist frequency, and therefore it cannot track them!
- The "ClockMaster Counter Value" input signal is noisy, and the noise increases along the line of network elements (i.e. different controllers see different noise) → The controller might track the noise if it is large and not filtered!
- The "ClockMaster Counter Value" input signal provides a delayed information about the true MasterTime, this delay increases over the line topology (i.e. different controller see different delays)
- The design of the controller(s) will/could depend on the usage of the controller (only for the local application or also for delay compensation in Sync-Messages, where the controller actions in element "N" influence control actions in the following elements – see the next 2 slides), and from the requirement that either all controllers should be the same or that their parameters should depend on the element position in the line topology
- …



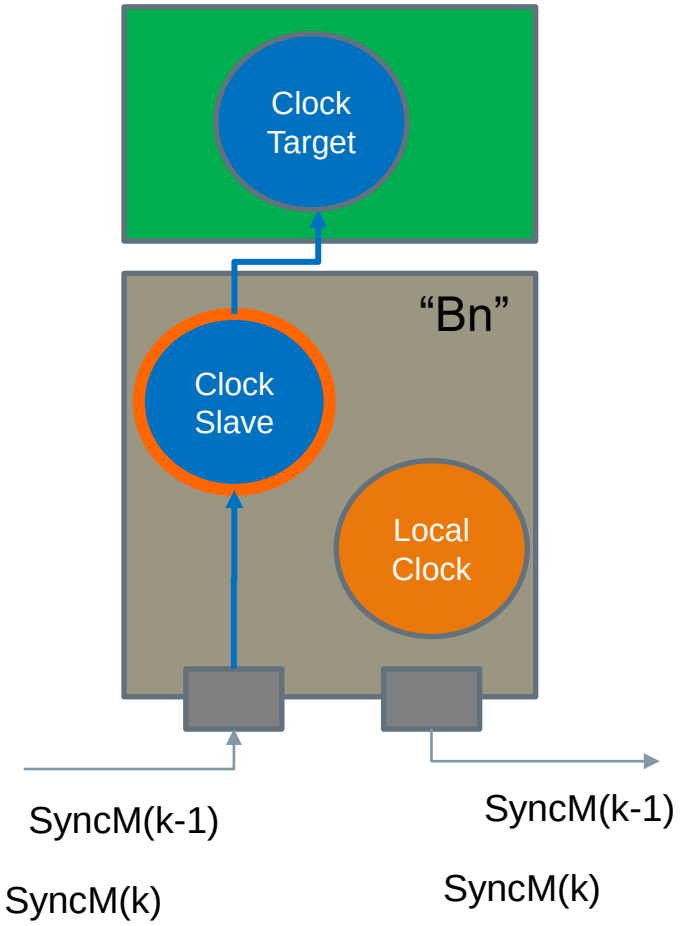


Application of the controlled ClockSlave

Standard Application:

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

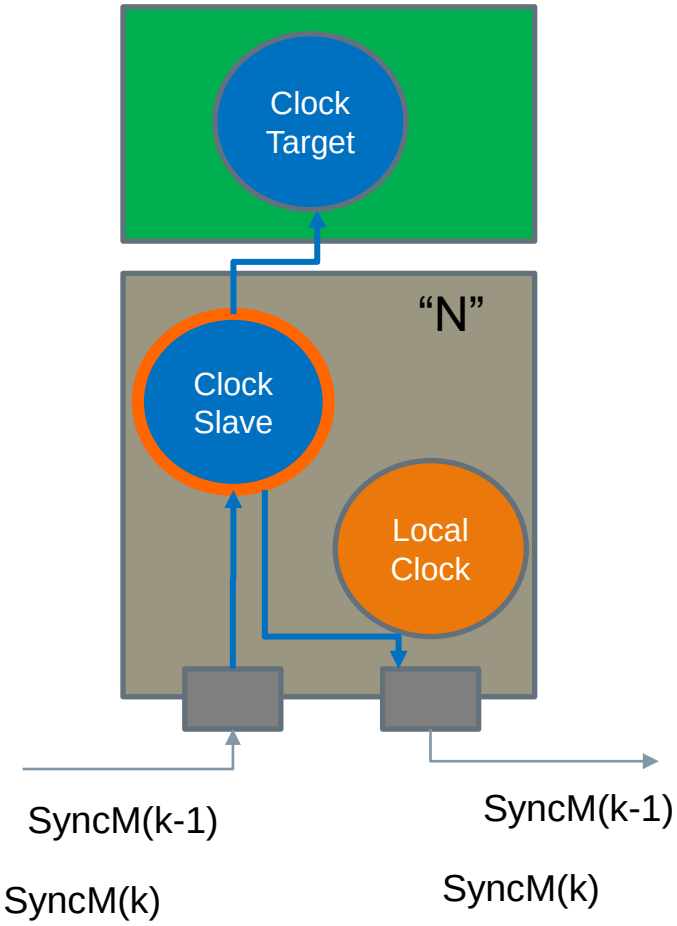
→ The ClockSlave at element “n” does not influence the estimation of nRR and RR at that element, nor the update of the Sync message content before it is forwarded to the next Slave element “n+1”



Application of the controlled ClockSlave

Additional Application:

- Use the ClockSlave also for updating the Sync Message with the encountered Delays
- ➔ <u>The ClockSlave at “N” influences the ClockSlave at “N+1” because it is used to calculate the MasterTime which should be tracked!</u>



→ Master Time Correction at “N” 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.*

Controller in ClockSlave: What should be Normative!

What is the purpose of making some controller characteristic normative?

- If not satisfied, the system will not work (no synchronization)! → they are necessary but not sufficient! ← I like this one better!
- If satisfied, it will definitely work! →they are sufficient

NORMATIVE requirements

1. Stability of the ClockSlave closed loop
2. Stability Margins (t.b.d., e.g. how to characterize delays in open loop such as the interval between Sync and the FollowUp messages)

Informative (recommendations):

- Choose HW which limits drifts and stamping errors (see the Siemens example of G. Steindl)
- Short Sync Interval enables tracking of fast changes in MasterClock frequencies
- Increasing Signal2Noise Ratio: time intervals of interest should be >> stamping noise, potentially filter MasterTime estimate before presenting it to the controller. Leave to the end user to choose controller performance criteria and the controller design method!
- If HW and system parameter choices different from functioning example(s), test in simulation/ HW!

To be Continued

Thank you for your attention!

Questions?

My Questions:

- How do we continue?
  - Do we start from a working solution (e.g. Siemens solution) and do a sensitivity analysis w.r.t. different system parameters?
  - Continue trying to improve filtering (e.g. use Kalman Filter for filtering the MasterTime input to the control loop)?
  - …
- Do we introduce a digital twin of the system (a fast, easy to parameterize simulation tool) for the end user to test their solution? Can IEEE or some other entity offer this (this is practically a certification step)? Who can guarantee that the system model (e.g. drift model, stamping error description, …) is correct?

Extra: I am writing a “white paper” about …

Goal of the document:

- Specify all necessary inputs needed for simulations runs, i.e. how to “guarantee” that simulations with two different simulators can be compared (we are reviewing our “old” ClockSync simulator):
  - Sync and p-delay message intervals
  - 1 or 2 step time forwarding via sync messages (with/out FollowUp messages)
  - Controller parameters in ClockSlaves
  - Method for estimating and compensating delays (updating time info in Sync messages):
    - Using RR, where RR calculated via nRRs (based on p-delay messages)
    - Using RR, where RR calculated using Sync messages
    - Without using RR, but using the ClockSlave time (best Master Estimate Time at slaves)
  - Jitter and drift characteristics
- List all inputs for known implementation(s) where 100 (originally 64) elements satisfied the 1µs requirement …
- Initialization of the system (when the messages start being sent and forwarded)
- Necessary logic (without implementing all state machines)
- …

Extra: I am writing a “white paper” about …

More about the document

- I will need input from the “clock-synchronization expert group” and also from anybody else who is interested in the topic and would like to contribute and comment
- The first version will have a lot questions …

Time plan:

- First version at the latest on the 2.09.2022 (then I am on vacation)
- First version with many questions
- I hope to have a first complete version (with at least one complete specification of the simulation with parameters corresponding to known functioning system) by the end of October
- Comparison / alignment with Geoff’s simulator
- Common decision how to proceed: do we have a trusted simulator (accessible to many, e.g. as a web service) where different ideas can be tested fast and reliably, or is there a different way for calculating worst case (with acceptable probability) performance, such as having analytic formulas (e.g. like in Siemens publications)
- …