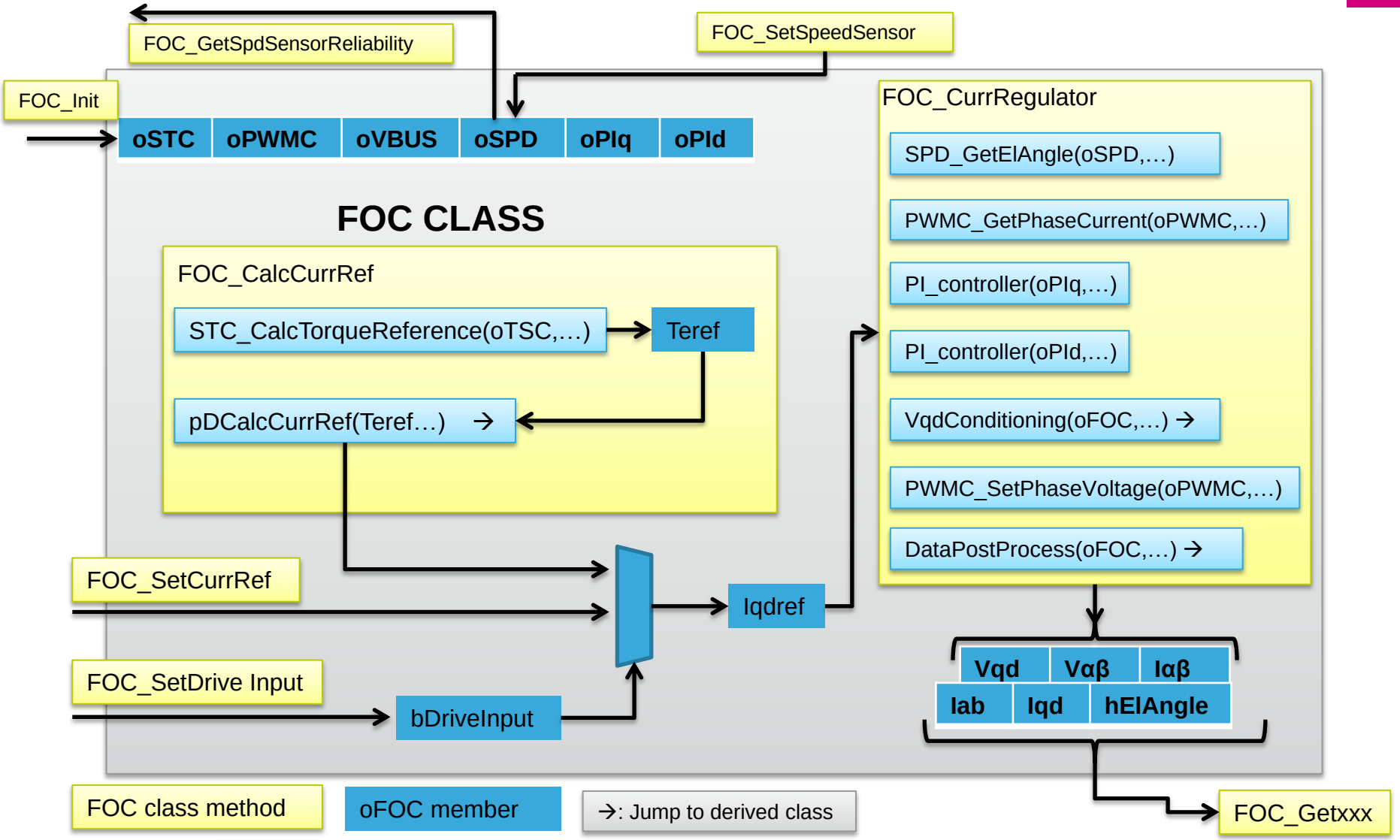
Current regulation

- Current regulation is handled by FOC class
- FOC additional methods are referred to derived classes:
  - SM class, no additional methods available
  - SMF, flux weakening
  - IMF, flux weakening + Internal PMSM MTPA strategy
  - IMFF, flux weakening + Internal PMSM MTPA strategy, feed-forward current regulation forward current regulation
- Current references are updated
  - Internally by FOC_CalcCurrRef, if bDriveInput is set to INTERNAL
  - Externally through FOC_SetIqdref, if bDriveInput is set to EXTERNAL

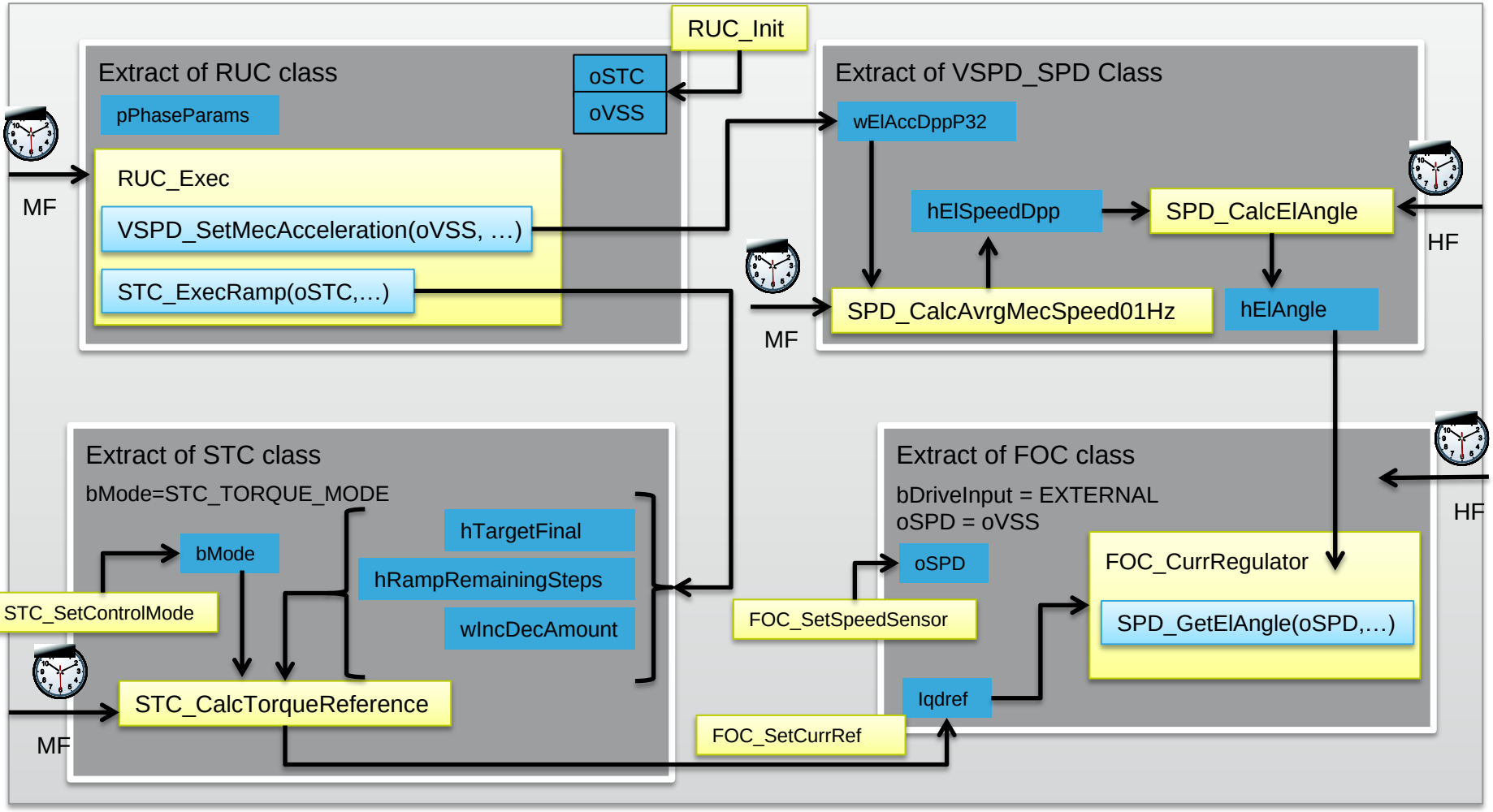
Current regulation block diagram



Sensor-less rev-up

- Sensor-less rev-up is composed by a configurable number of sections (5 max by default) with programmable duration.
- Speed and/or current amplitude increase linearly within the section
- It is carried out by the synergy of 4 different objects:
  - The rev-up controller (RUC class)
    - is the start-up master
    - It deals with sections duration
    - It configures the other objects at the beginning of each section
  - The speed and torque controller (STC class)
    - accounts for phase current amplitude ramps within the present section
  - The virtual speed sensor (VSS class)
    - is programmed by oRUC at the beginning of each section
    - feeds back a speed linearly increasing (programmable virtual acceleration) within the present section and – accordingly – a virtual electrical angle
  - The FOC drive (FOC class)
    - it's responsible for current regulation
    - Park transformations are computed on the electrical angle provided by oVSS
    - Current references are provided from the outside world
    - (oFOC set in EXTERNAL mode)

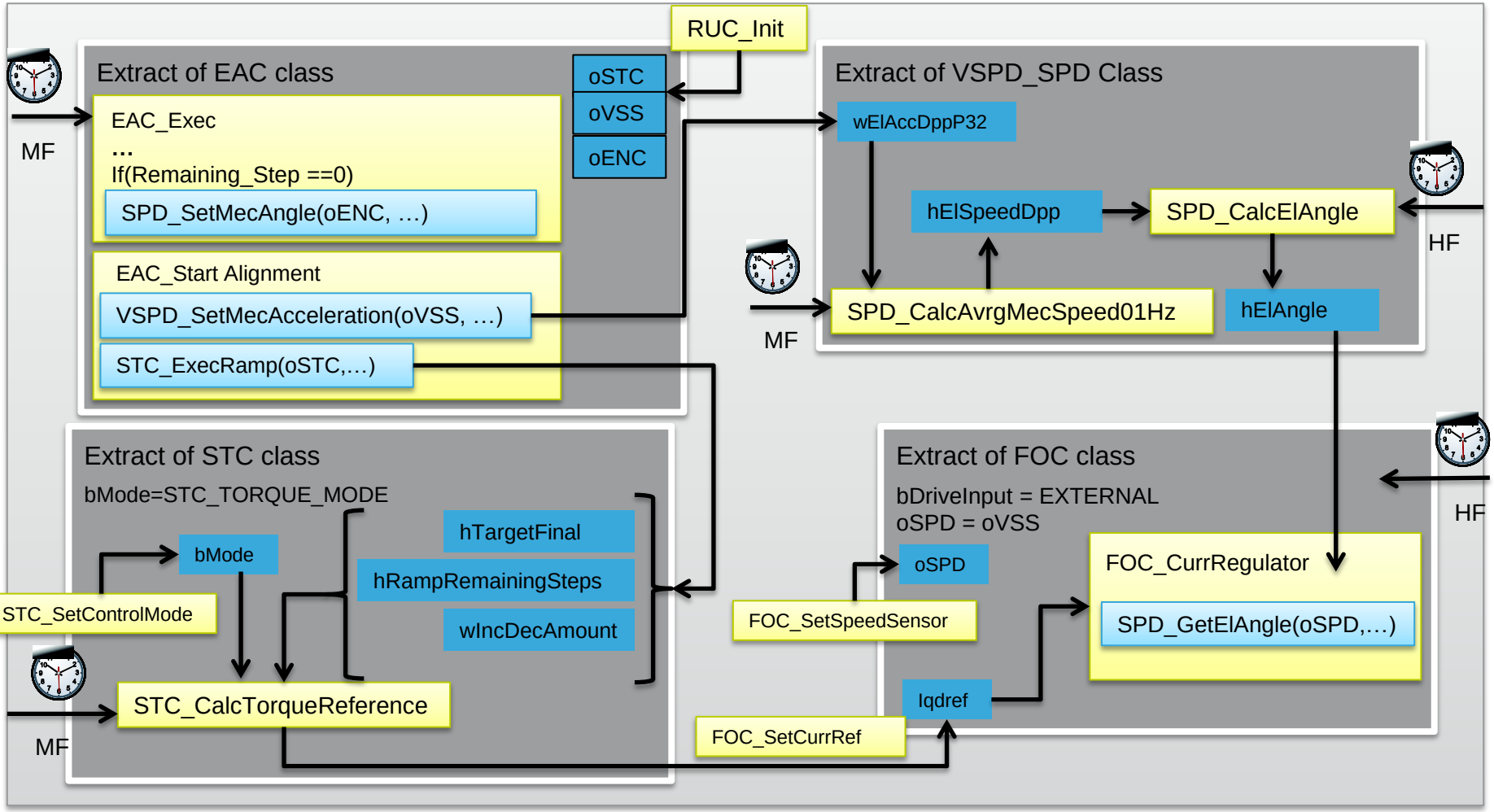
Sensor-less rev-up block diagram



Encoder alignment

- Very similar to sensor-less rev-up with single section
- It is carried out by the synergy of 4 different objects:
  - The encoder alignment controller (EAC class)
    - configures oSTC and oVSS at the beginning of the alignment
  - The speed and torque controller (STC class)
    - accounts for phase current amplitude ramp
  - The virtual speed sensor (VSS class)
    - feeds back a constant null speed linearly and – accordingly – a constant programmable electrical angle
  - The FOC drive (FOC class)
    - it's responsible for current regulation.
    - Park transformations are computed on the electrical angle provided by oVSS
    - Current references are provided from the outside world (oFOC set in EXTERNAL mode)

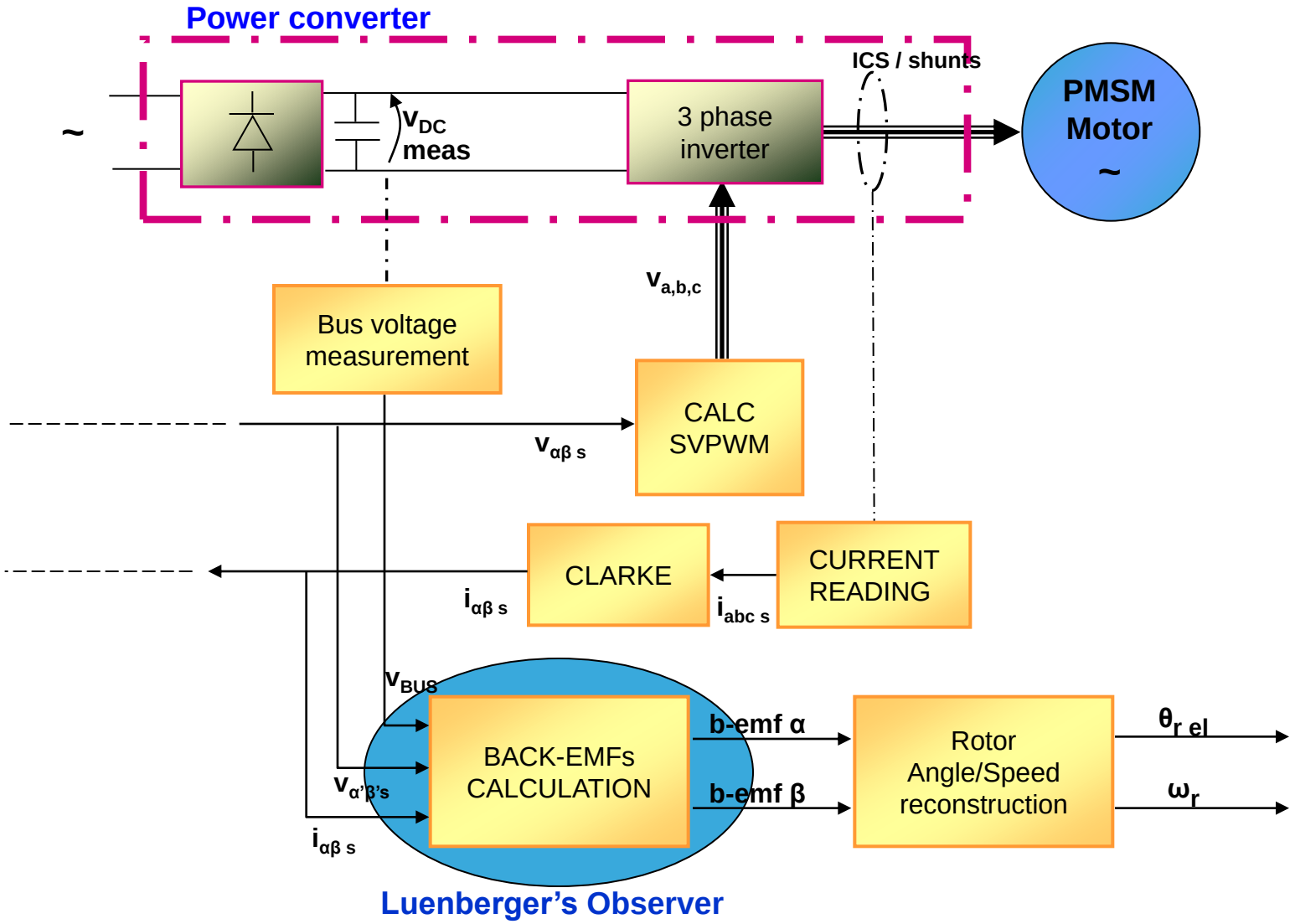
Encoder alignment block diagram



Sensorless algorithm add-ons

- State observer block diagram (review)
- PLL rotor angle extrapolation (review)
- CORDIC rotor angle extrapolation
- Redundant b-emf reliability check
- Smooth start→run switchover

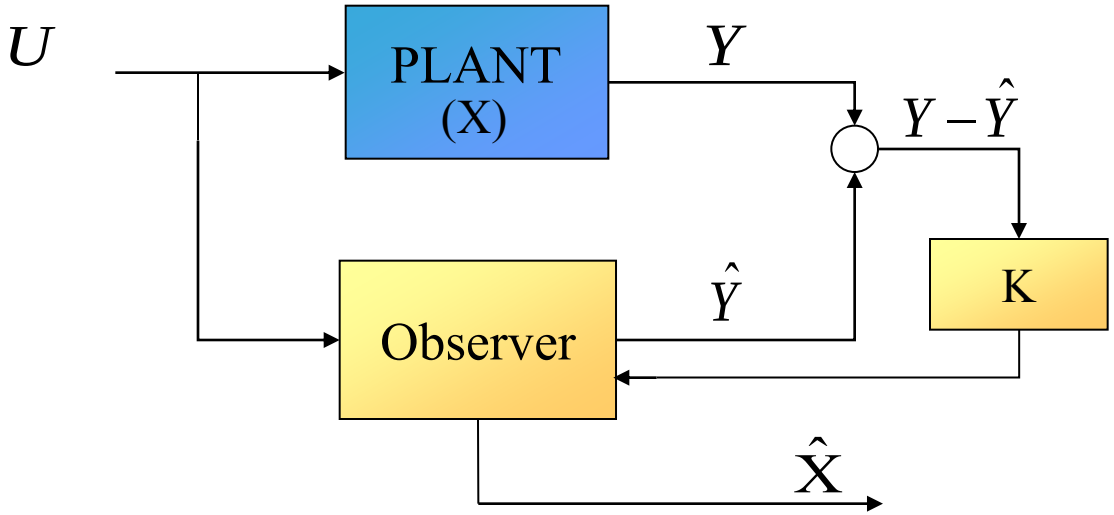
Sensorless Block diagram



B-EMFs calculation: the observer

- In control theory a system is **observable**, if it is possible to fully reconstruct the system state from its output measurements

- The **state observer** is a system that provides an estimation of the internal state of the observed system given its input and output measurements.



Angle/speed reconstruction

- After the two B-emf alpha and Beta have been produced, rotor position information must be extracted

- Using e_α and e_β definitions:

$$e_\alpha = \Phi_m p\omega_r \cos(p\omega_r t)$$

$$e_\beta = -\Phi_m p\omega_r \sin(p\omega_r t)$$

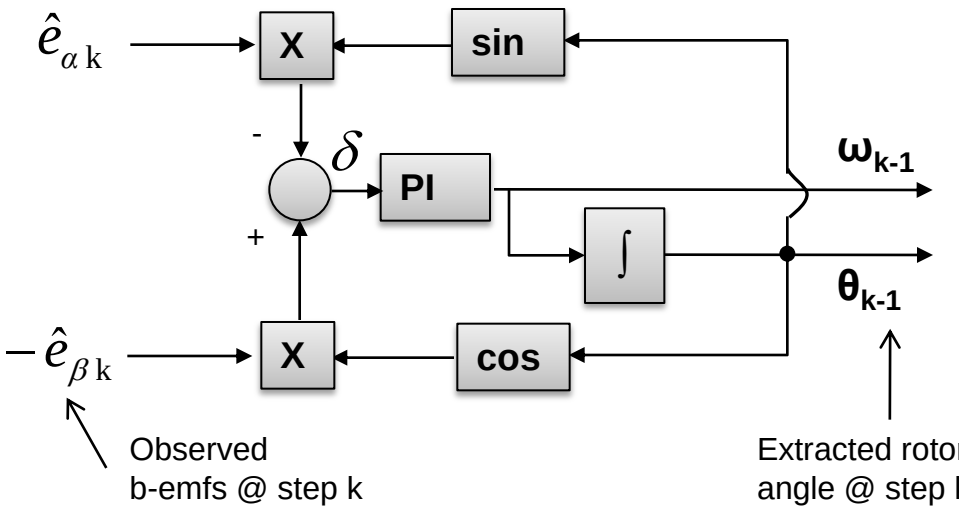
rotor position could be computed as

$$\theta_r = p\omega_r t = arctg\left(-\frac{\hat{e}_\beta}{\hat{e}_\alpha}\right)$$

through CORDIC algorithm.
This straight forward open-loop approach could be sensitive to the noise possibly present on Bemfs (especially speed)

We recommend to select on ST MC Workbench "Observer + PLL"

PLL(v2.0 / v3.2 option) basics



Contained Info about rotor angle @ step k

$$\hat{e}_{\alpha k} = A\cos\hat{\theta}_k$$

$$-\hat{e}_{\beta k} = A\sin\hat{\theta}_k$$

Discrete-time model @ step k

- How it works?

$$\delta = -\hat{e}_{\alpha k}\cdot\sin\theta_{k-1} - \hat{e}_{\beta k}\cdot\cos\theta_{k-1} =$$

$$= -A\cos\hat{\theta}_k\cdot\sin\theta_{k-1} + A\sin\hat{\theta}_k\cdot\cos\theta_{k-1} = A\sin\left(\hat{\theta}_k - \theta_{k-1}\right) \cong A\left(\hat{\theta}_k - \theta_{k-1}\right)$$

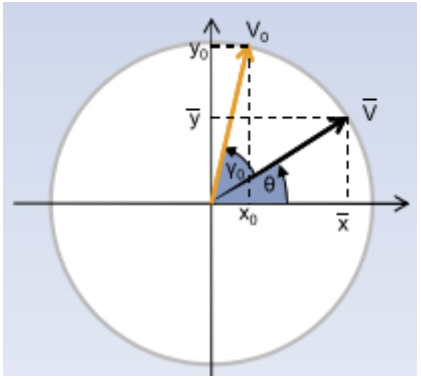
- By keeping δ = 0, it will happen that $\theta_{k-1} = \hat{\theta}_k$ (prediction);

- PI gains are calculated by the GUI ($\omega_n = \omega_{\max}\ ; \xi = 0.707$)

No tuning required for PLL gains (P and I)

CORDIC (v3.2 option), basics

- COordinate Rotation DIgital Computer: a successive approximation algorithm to calculate trigonometric functions; it only requires additions, subtractions, bitshifts and a small LUT.

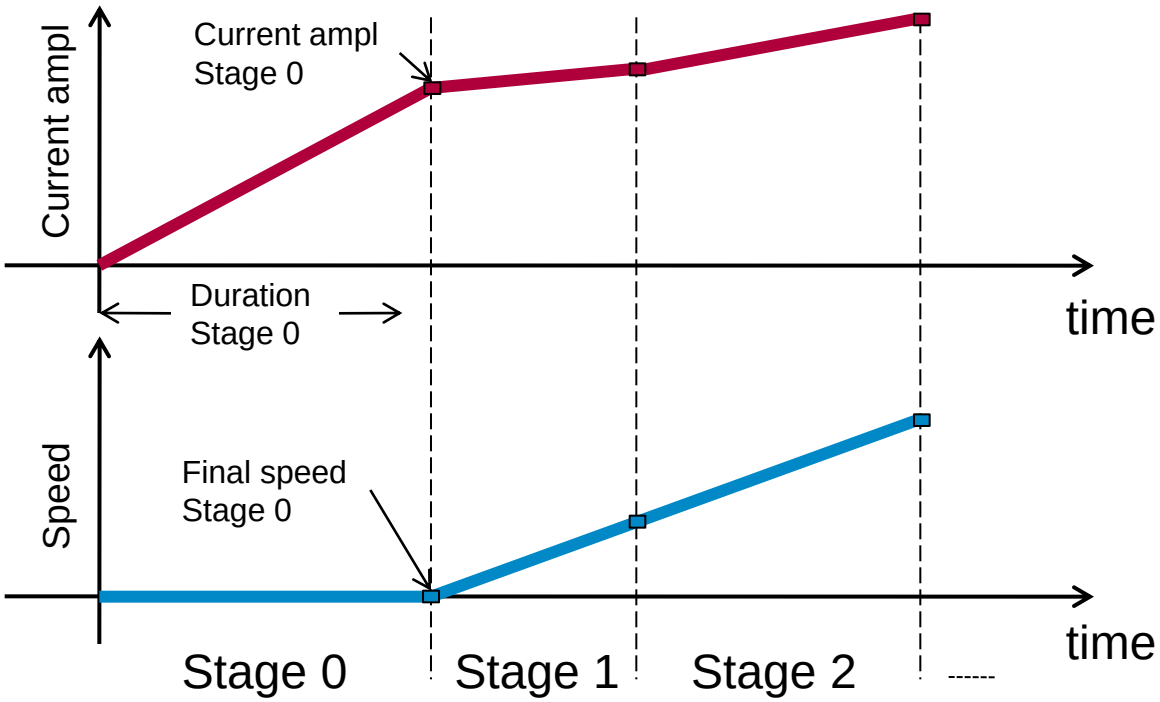


- How it work ? It's standard math algorithm and please download and read the manual from the website.
- CORDIC, considerations:
  - It has been experimented that – due to inherent noiseness of the cordic algorithm – a much greater speed variance threshold is to be allowed;
  - On the other hand, due to inherent readiness, cordic has not the initial delay it takes for the PLL to lock;
  - GUI parameter 'bemf quality factor' set to 1(max) means the observed bemf is perfectly clean and sinusoidal;
  - GUI parameter 'maximum application acceleration' is to be considered the real one, not α_{MAX};
  - α_{MAX} = MaxApplicationAcceleration/BemfQualityFactor

Start-up (Rev-up)

- Motor start-up (rev-up) is composed by a variable number of stages (5 by default)

- For each of the stages is possible to define duration, final current amplitude reference and final forced speed (STMCWB, Drive management → Start-up parameters in case of advanced profile)



Start-up needs to be tuned case by case (depends on motor inertia, load and frictions)
Refer to tutorial

Start-up output conditions 1/2

- For switching from the rev-up (START state) to the Field Oriented Control (RUN state) it is necessary both observer and PLL converged (no convergence required by CORDIC)

- Two statical index are computed on the observed speed FIFO with the periodicity of the medium frequency task:

$$\mu = \frac{\sum_{i=1}^{64} x_i}{64} \qquad \sigma^2 = \frac{\sum_{i=1}^{64} (x_i - \mu)^2}{64}$$

Start-up output conditions 2/2

- The algorithm is assumed to be converged if:

$$\sigma^2 \leq \mu^2 \cdot \textbf{\textit{Variance threshold;}}$$

Estimated speed band tolerance lower limit *** forced speed <** ***Estimated*** **speed;**

Estimated **speed <** ***estimated speed band tolerance upper limit*** *** forced speed;**

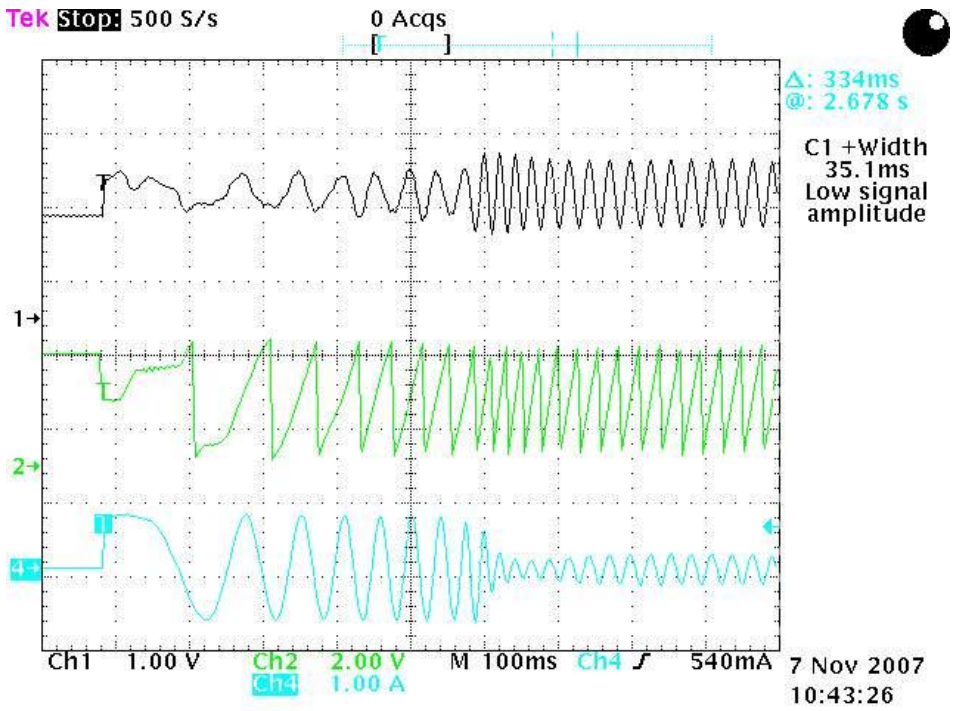
$$\mu > \textbf{\textit{Minimum start-up output speed.}}$$

for a number of *consecutive successful start-up output test*
(italic stands for STMCWB → Drive management parameters)

- If the above conditions are found before the end of the rev-up, the main state machine moves from START to START_RUN; otherwise it goes into FAULT state (“Rev-up failed”)

Algorithm conversion, a typical start-up

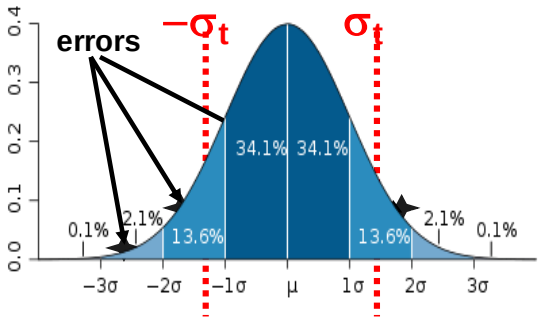
- Following capture shows a typical start-up with Shinano motor



- Angle converged in about 150ms, output conditions are verified after 500ms

Rotor locked detection

- Variance of speed measurement increases when a confused information about B-emf is fed to PLL/Cordic;
- Tipically this happens at low speed
- If in RUN state $\sigma^2 \geq \mu^2 \cdot$ *Variance threshold*



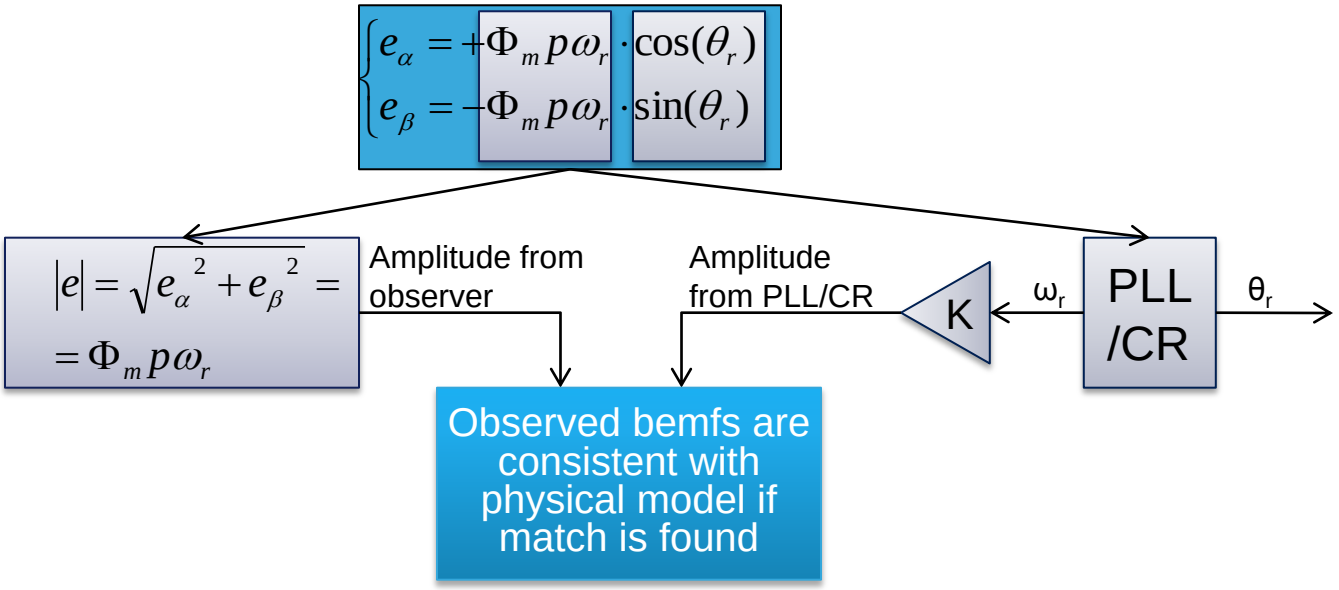
for *Max measurement errors before fault* consecutive times, main state machine goes into Fault (“Speed feedback”)

Since the rotor locked detection is based on statistics, it could fail in particular conditions (load suddenly applied)

Enlarge threshold (up to 50%) to accept more noisy speed estimations (lower in speed, weaker speed regulation)

Redundant bemf check, alg

- An additional algorithm runs now along with the speed variance check
- Foundation:
  - motor bemfs have 2 contents:
    - sinusoidal modulation, syncronized with rotor angle
    - amplitude modulation, proportional with rotor speed



Only necessary to be tuned if previous detection fails (rotor vibrating and sensor-less algorithm not realizing it)

Agenda

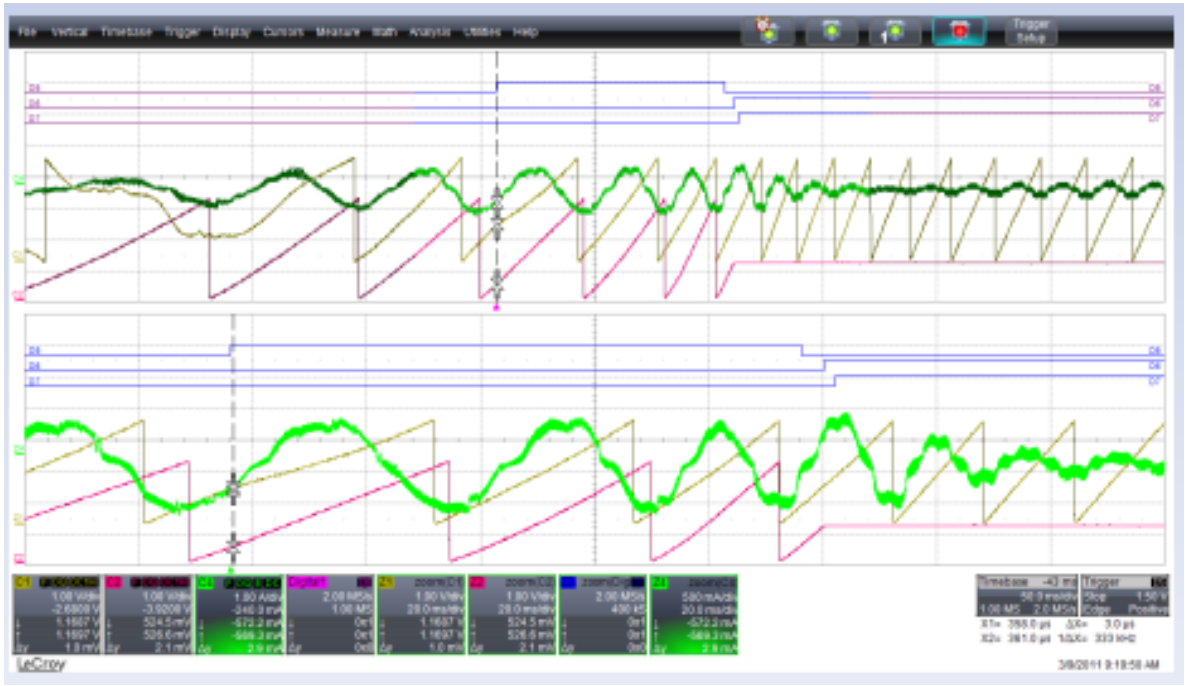
- 1st day – Afternoon
  - MC Application
    - Interface
    - Tuning
    - Tasks
    - Classes interaction
      - Current regulation
      - Ramp-up
      - Encoder alignment
  - Speed sensors updates:
    - Sensorless algorithm improvement
  - How to create User Project Interacting with MC Application
  - Dual motor control
    - Resources sharing
    - Supported configurations
    - Code size efficiency
  - Current reading sensor update

Start-Run switchover

- Any sensorless method based on bemf detection (STO, VI, KF, 6step ton/toff) requires a start-up procedure. In FOC, switching from start-up to run mode is a delicate matter:
  - speed loop integral sum should be properly initialized(already done in v2.0 but with some limitations)
  - there could be a sharp discontinuity between VSS angle and Observed angle;
  - combined effects are:
    - glitches on current regulation;
    - noise on observed bemfs -> speed feedback error
    - discontinuity on produced Te

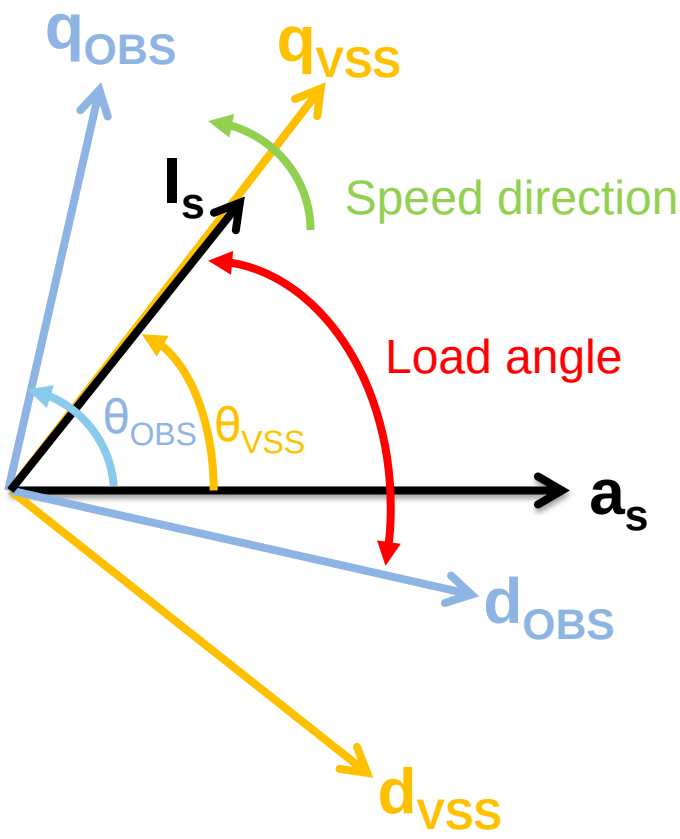
Start-run switchover, how it works

- How it works:
  - open loop startup (VSS angle);
  - observer converged (OBS angle now reliable);
  - transition starts (GUI defined duration);
  - a third angle ‘TR angle’ is computed, which moves - during the transition - from VSS to OBS. The algorithm is a sort of time-weighted compensation. ‘TR angle’ is used for FOC;
  - transition ends. TR angle = OBS angle now
  - startup end. Total duration **CAN’T** be greater than ‘Speed ramp duration’.



Start-run switchover, how it works

- physical representation



- θ_{OBS} should lead θ_{VSS}

Start-run switchover, benefits

- it smooths currents regulation/torque;
- experimentally – it improves bemf detection (start-up conditions not always get along with linearized PMSM model);
- towards end of transition and until RUN state, motor is actually in torque mode;
- as a consequence, it’s easier to properly initialize the speed loop integral sum.
- can be disabled if not giving good results(just setting transition duration = 0)