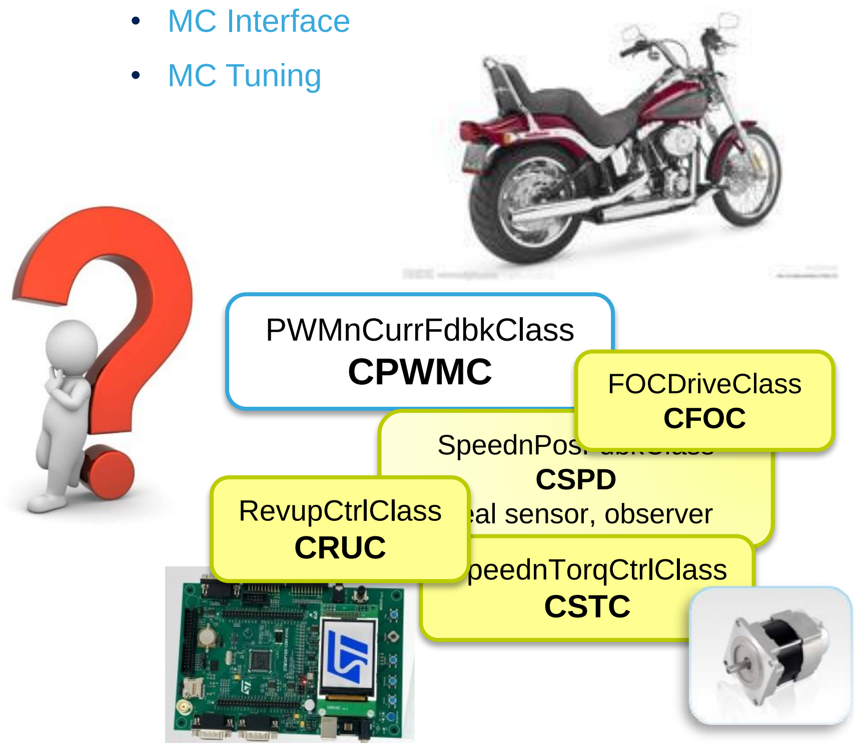
Agenda

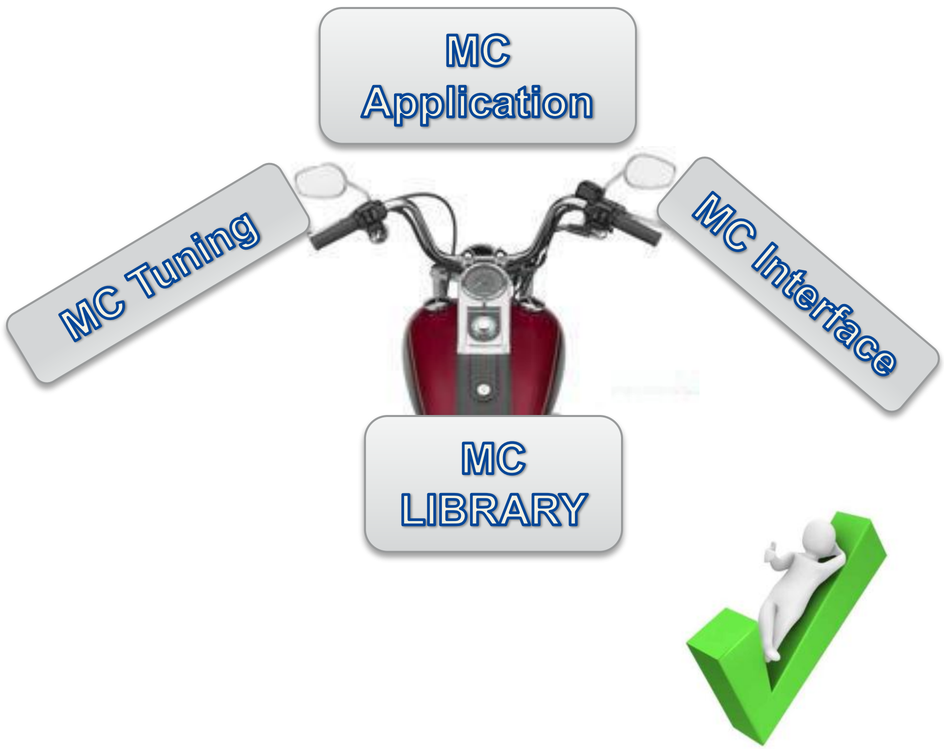
- 1st day – Afternoon
  - MC Application
    - Interface
    - Tuning
    - Tasks
    - Classes interaction
      - Current regulation
      - Ramp-up
      - Encoder alignment
  - Speed sensors updates:
    - Sensorless algorithm improvement
  - How to create User Project Interacting with MC Application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

MC Application

- The Motor Control Interface is the application built on top of the Motor Control Library this application is able to grant to the user layer the execution of a set of commands, named the MC Application Programming Interface (MC API).

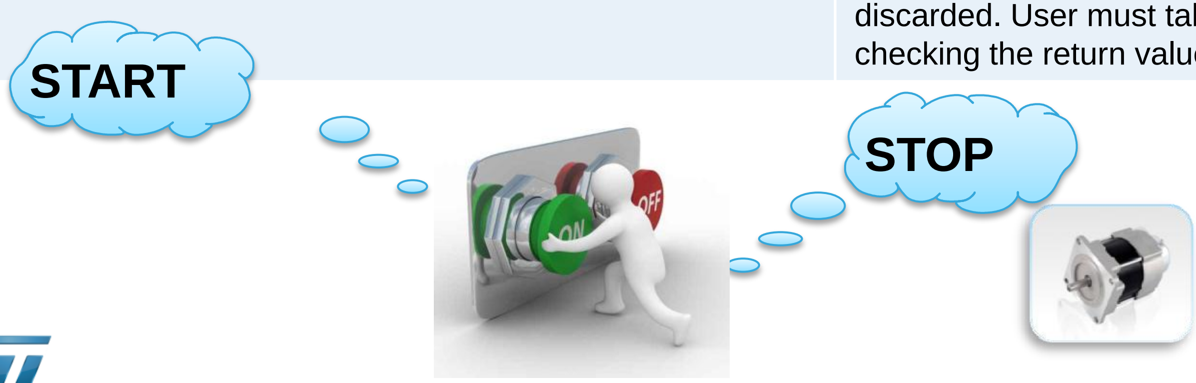
- The API is divided in two sections:
    - MC Interface
    - MC Tuning





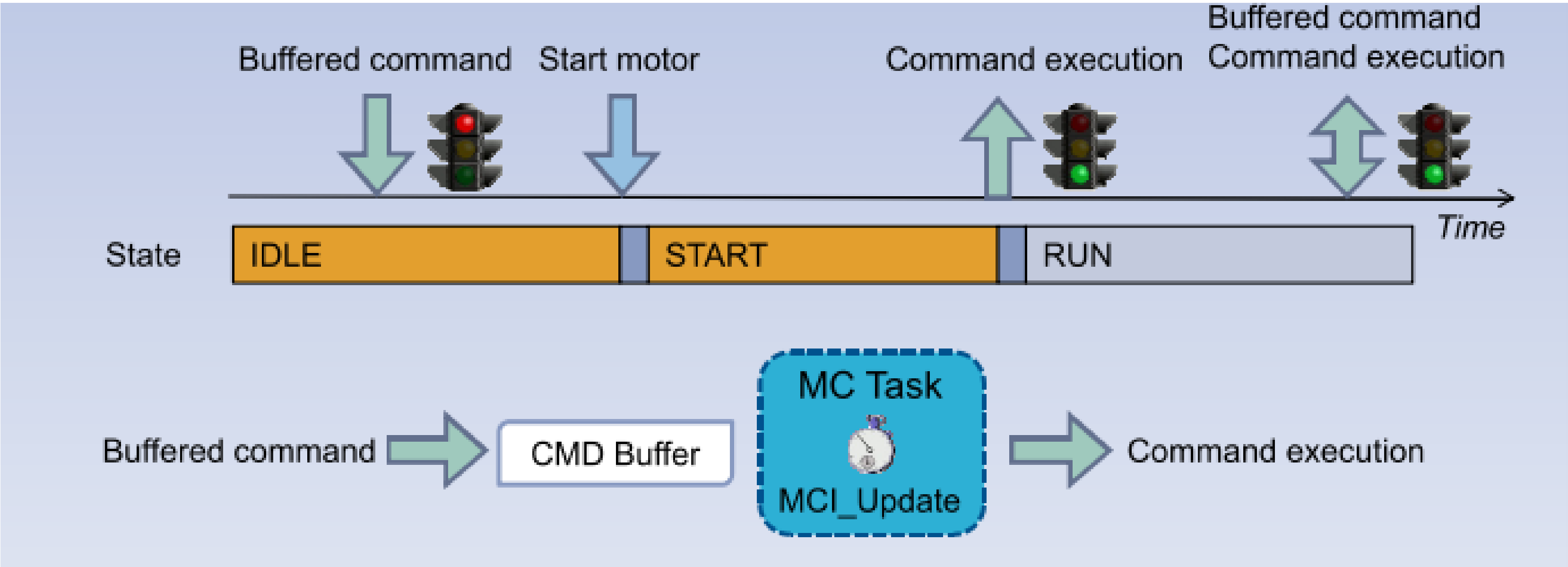
MC Interface commands

Method	Description
MCI_StartMotor	This is a user command used to start the motor. If is possible, the command is executed instantaneously otherwise the command is discarded. User must take care of this possibility by checking the return value.
MCI_StopMotor	This is a user command used to stop the motor. If is possible, the command is executed instantaneously otherwise the command is discarded. User must take care of this possibility by checking the return value.
MCI_FaultAcknowledged	This is a user command used to indicate that the user has seen the error condition. If is possible, the command is executed instantaneously otherwise the command is discarded. User must take care of this possibility by checking the return value.
MCI_EncoderAlign	This is a user command used to start the encoder alignment procedure. If is possible, the command is executed instantaneously otherwise the command is discarded. User must take care of this possibility by checking the return value.



MC Interface buffered commands

- Buffered commands don't become active as soon as it is called but it will be executed when machine is in a predefined state (Ex. RUN).



- If more that one buffered command is send before the execution only the last is considered.

MC Interface buffered commands

Method	Description
MCI_ExecSpeedRamp	This is a buffered command to set a motor speed ramp. This commands don't become active as soon as it is called but it will be executed when the oSTM state is START_RUN or RUN. User can check the status of the command calling the MCI_IsCommandAcknowledged method.
MCI_ExecTorqueRamp	This is a buffered command to set a motor torque ramp. This commands don't become active as soon as it is called but it will be executed when the oSTM state is START_RUN or RUN. User can check the status of the command calling the MCI_IsCommandAcknowledged method.
MCI_SetCurrentReferences	This is a buffered command to set directly the motor current references Iq and Id. This commands don't become active as soon as it is called but it will be executed when the oSTM state is START_RUN or RUN. User can check the status of the when the oSTM state is START_RUN or RUN. User can check the status of the command calling the MCI_IsCommandAcknowledged method.

Execute a ramp

MC Interface get methods 1/2

Method	Description
MCI_IsCommandAcknowledged	It returns information about the state of the last buffered command.
MCI_GetSTMState	It returns information about the state of the related oSTM object.
MCI_GetMecSpeedRef01Hz	It returns information about the current mechanical rotor speed reference expressed in tenths of HZ.
MCI_GetAvrgMecSpeed01Hz	It returns information about last computed average mechanical speed, expressed in 01Hz (tenth of Hertz).
MCI_GetTorqueRef	It returns information about the current motor torque reference. This value represents actually the Iq current reference expressed in digit. To convert current expressed in digit to current expressed in Amps is possible to use the formula: Current(Amp) = [Current(digit) * Vdd micro] / [65536 * Rshunt * Aop].
MCI_GetCurrentsReference	It returns information about stator current reference in Curr_Components format.
MCI_GetControlMode	It returns the modality of the speed and torque controller.

MC Interface get methods 2/2

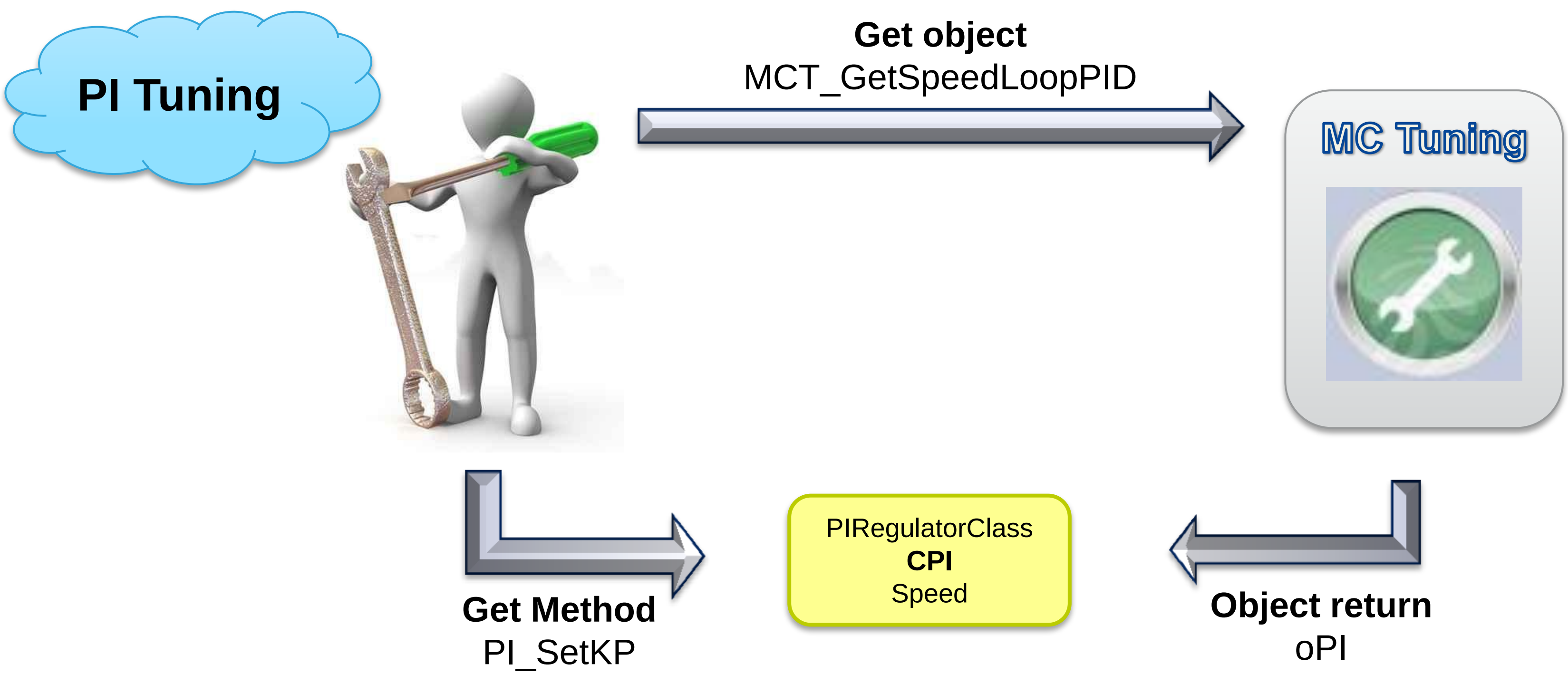
Method	Description
MCI_GetTorque	It returns information about current motor measured torque. This value represents actually the Iq current expressed in digit. To convert current expressed in digit to current expressed in Amps is possible to use the formula: Current(Amp) = [Current(digit) * Vdd micro] / [65536 * Rshunt * Aop].
MCI_GetPhaseCurrentAmplitude	It returns the motor phase current amplitude (0-to-peak) in s16A To convert s16A into Ampere following formula must be used: Current(Amp) = [Current(s16A) * Vdd micro] / [65536 * Rshunt * Aop].
MCI_GetPhaseVoltageAmplitude	It returns the applied motor phase voltage amplitude (0-to-peak) in s16V. To convert s16V into Volts following formula must be used: PhaseVoltage(V) = [PhaseVoltage(s16V) * Vbus(V)] /[sqrt(3) *32767].
MCI_GetImposedMotorDirection	It returns the motor direction imposed by the last command (MCI_ExecSpeedRamp, MCI_ExecTorqueRamp or MCI_SetCurrentReferences).

life.augmented

MC Tuning

- MCTuningClass acts as gateway to set/read data to/from objects (sensors, PI controllers…) belonging to the MC application.

- The MCTuningClass allows the user to obtain objects of the MC application and apply methods on them.



MC Tuning get object 1/2

MCT get object methods	Desciption
MCT_GetFOCDrive	It returns the FOCDrive object
MCT_GetSpeedLoopPID	It returns the speed control loop PI(D) object
MCT_GetIqLoopPID	It returns the Iq current control loop PI(D) object
MCT_GetIdLoopPID	It returns the Id current control loop PI(D) object
MCT_GetFluxWeakeningLoopPID	It returns the Flux Weakening control loop PI(D) object
MCT_GetPWMnCurrFdbk	It returns the PWMnCurrFdbk object
MCT_GetRevupCtrl	It returns the Rev-up controller object
MCT_GetSpeednPosSensorMain	It returns the Main Speed'n Position sensor object. Main position sensor is considered the one used to execute FOC one used to execute FOC
MCT_GetSpeednPosSensorAuxiliary	It returns the Auxiliary Speed'n Position sensor object. Auxiliary position sensor is considered the one used to backup/tune the main one

MC Tuning get object 2/2

MCT get object methods	Desciption
MCT_GetSpeednPosSensorVirtual	It returns the Virtual Speed'n Position sensor object. Virtual position sensor is considered the one used to rev-up the motor during the start-up procedure required by the state-observer sensorless algorithm
MCT_GetSpeednTorqueController	It returns the Speed'n Torque Controller object
MCT_GetStateMachine	It returns the State Machine object
MCT_GetTemperatureSensor	It returns the Temperature sensor object
MCT_GetBusVoltageSensor	It returns the Bus Voltage sensor object
MCT_GetBrakeResistor	It returns the Brake resistor object
MCT_GetNTCRelay	It returns the NTC Relay object

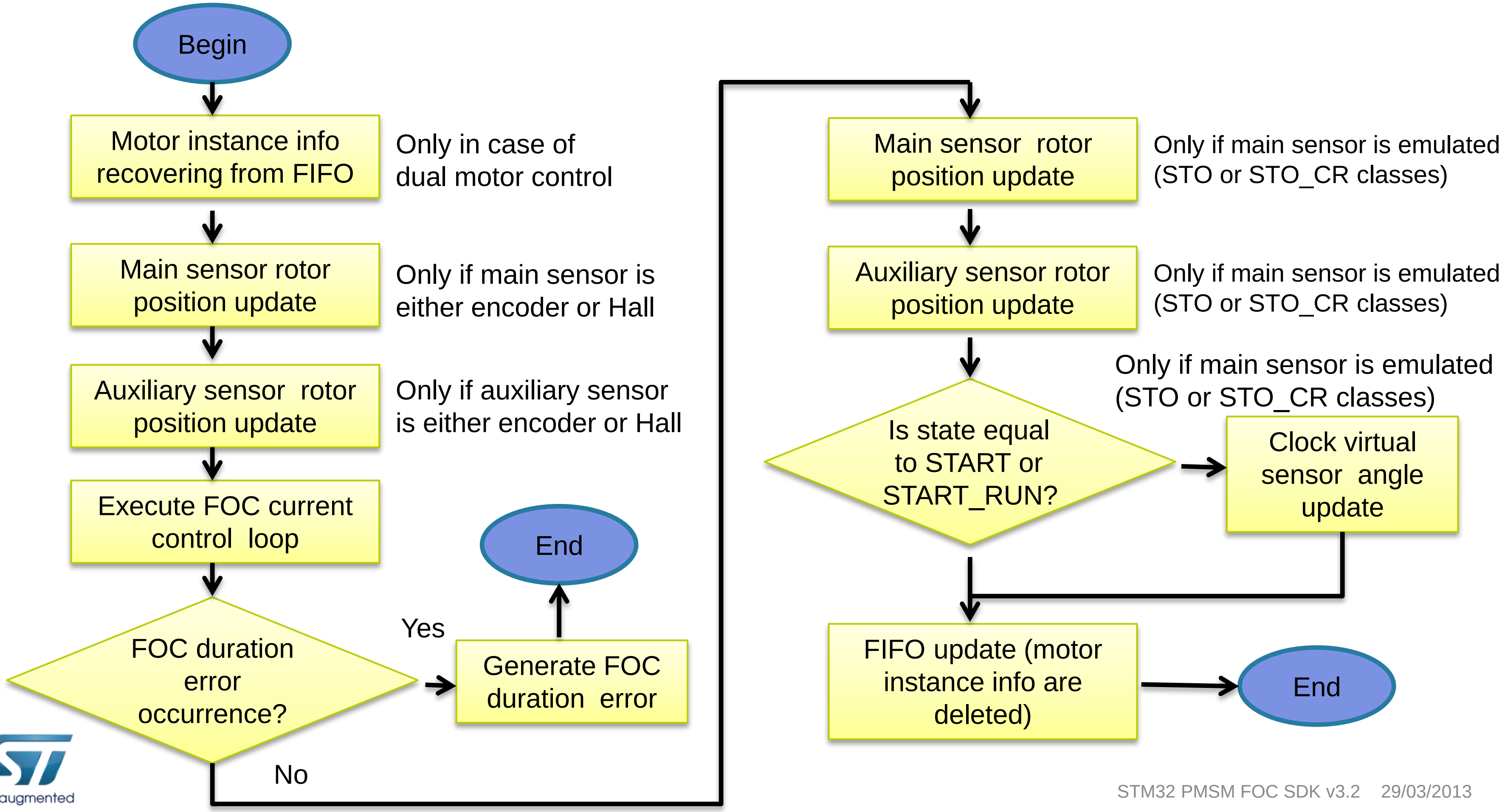
Tasks description

- Five tasks are currently used in the default project (ordered by priority)
  - 'High frequency' task
    - Clocked by ADC(s) JEOC interrupt(s), executes motor control duties requiring high frequency rate and precise timing (e.g. FOC current control loop)
  - 'Safety' task
    - Executed each 500us, it handles through state machine object the fault generation management
  - 'Medium frequency' task
    - Executed at configurable rate (SPEED_LOOP_FREQUENCY_HZ, Drive parameters.h'). Processes requiring a precise timing are here executed (e.g. speed loop)
  - 'Low frequency' task
    - Executed every 10ms, it includes duties not requiring a very precise timing and/or needing a low refresh rate (e.g. boot capacitors charge time counting)
  - 'User Interface' task
    - Executed each 100ms, LCD and keyboard refresh

life.augmented

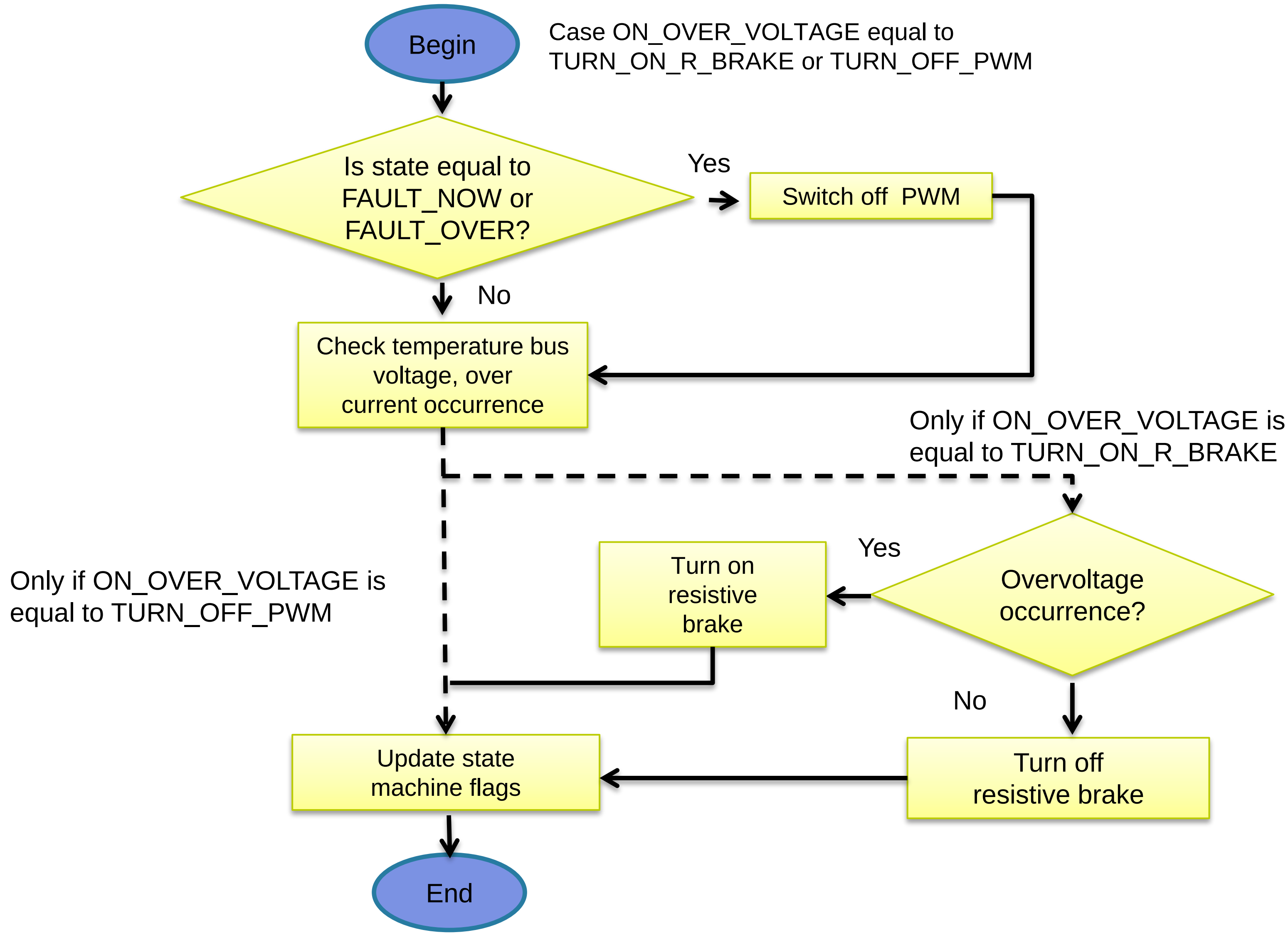
High frequency task

- The high frequency task executes for a given motor those duties requiring a high frequency rate and a precise timing (e.g. FOC loop).
- It is triggered by ADC JEOC interrupt which is sanctioning the end of the related motor phase currents reading.
- This trigger is only available in states START, START_RUN, IDLE_ALIGNMENT, ALIGNMENT, thus the high frequency task is actually executed only in these states while it is not triggered otherwise



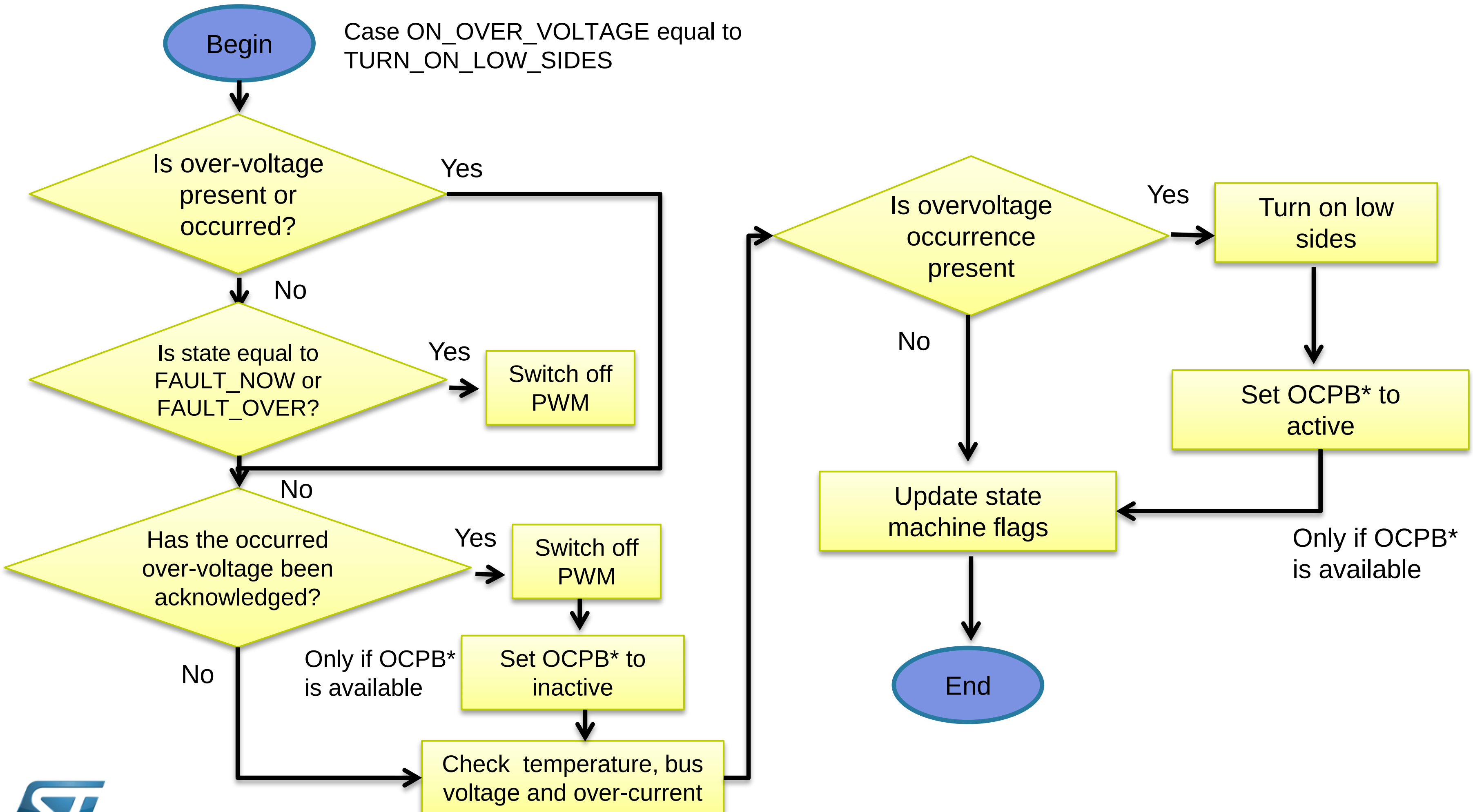
Safety task 1/2

- If - in case of over voltage - FW is configured so as to switch on Rbrake or turn off PWM:



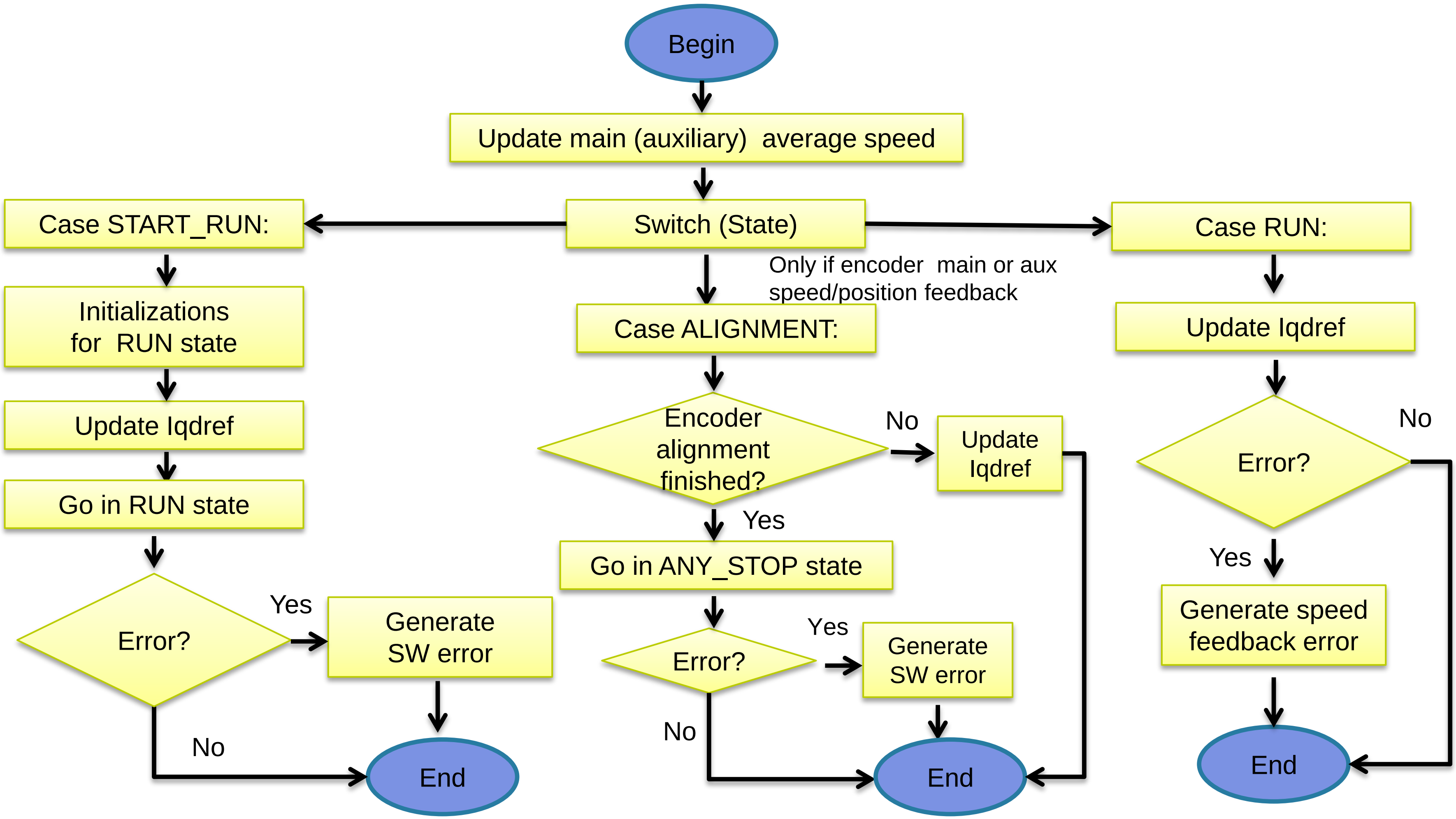
Safety task 2/2

- The safety task executes in sequence, each 500us, the safety checks to each of the drives.
- Actions to be taken in case of over-voltage (turn on low side switches, turn off PWM or brake resistor turn-on) are here managed.
- If - in case of over voltage - FW is configured so as to close low side switches:

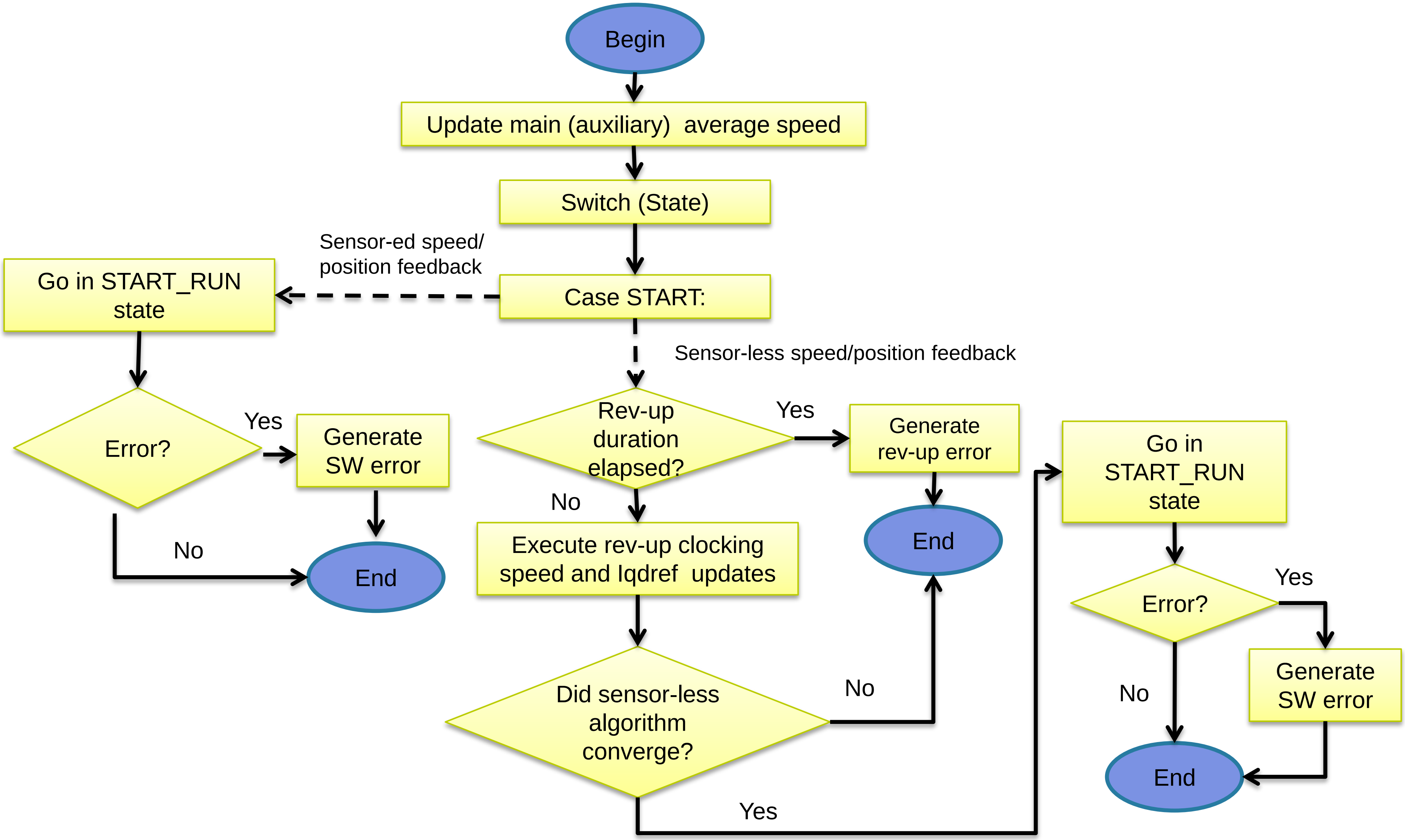


Medium frequency task 1/2

- The medium frequency task executes - in sequence- the medium frequency tasks related to each of the drives. Duties requiring a specific timing (e.g. speed controller) are here executed @ configurable frequency (SPEED_LOOP_FREQUENCY_HZ, Drive parameters.h')

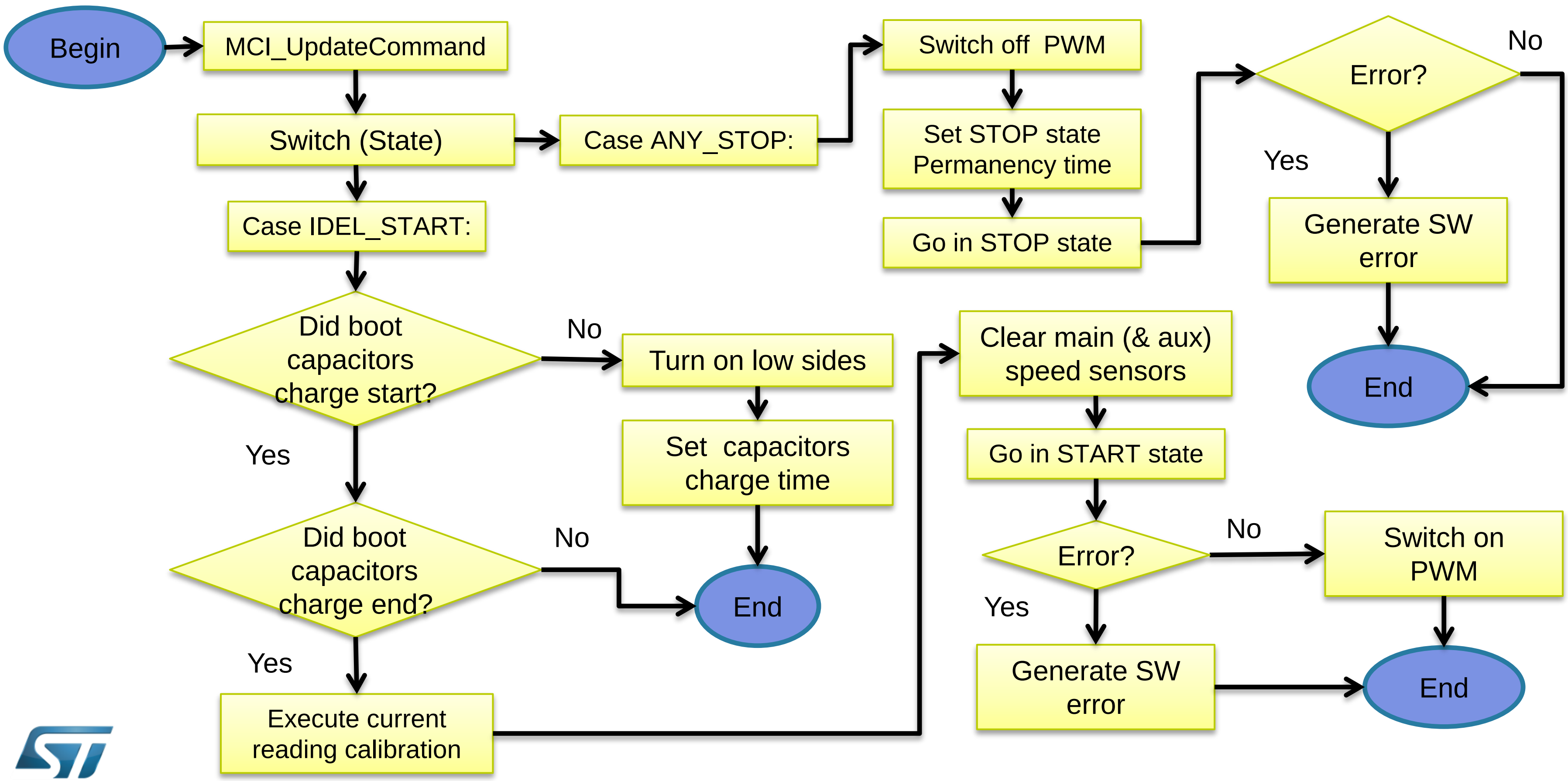


Medium frequency task 2/2



Low frequency task 1/2

- It executes - in sequence- the low frequency tasks related to each of the drives.
- It includes those duties not requiring a precise timing and/or needing a low refresh rate (e.g. stop state permanency time or boot capacitors charge time counting).
- Execution rate is 100Hz, priority should be set just above background (main) priority (e.g.
- tskIDLE_PRIORITY+1 for FreeRTOS based applications)
- User commands such as ‘run’ or ‘stop’ motor are also processed in this task.



Low frequency task 2/2