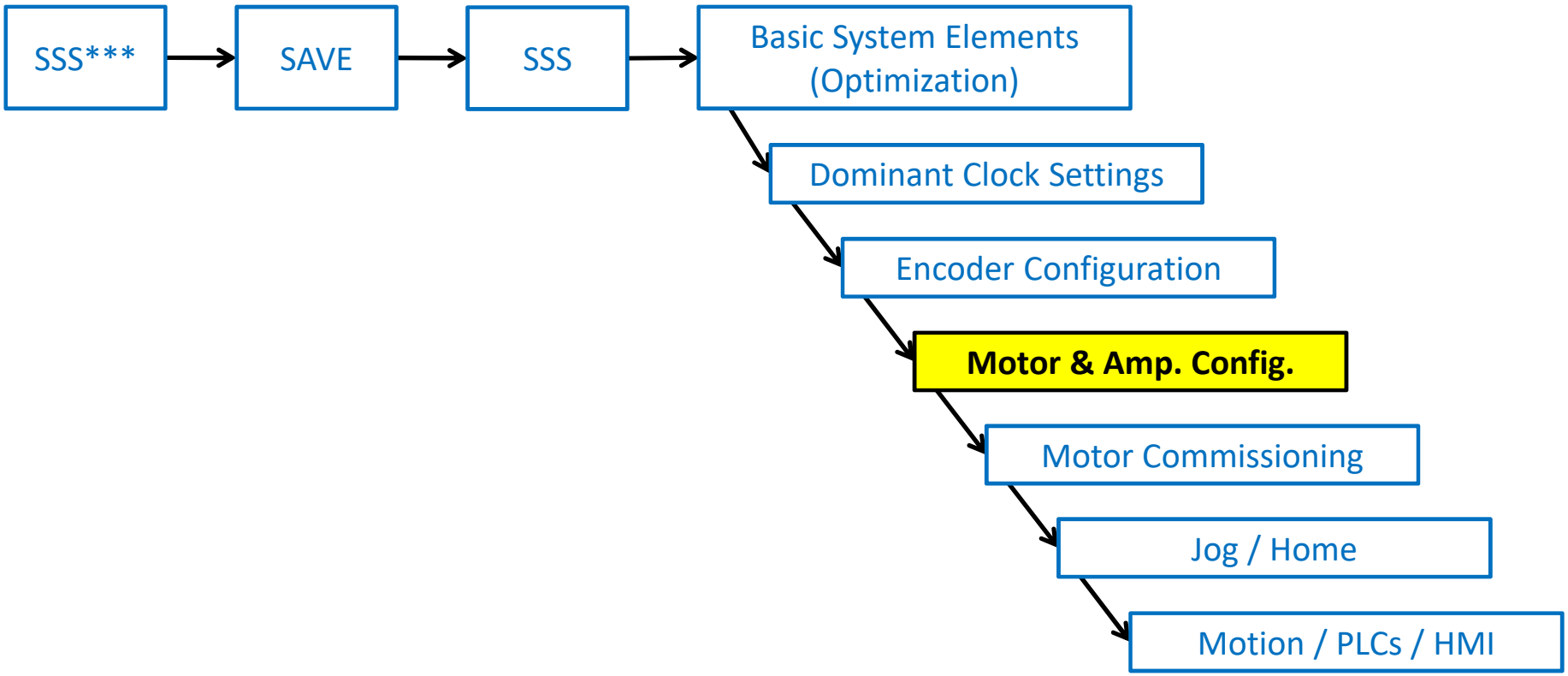
Power Brick LV
Motor & Amplifier Configuration

System Configuration



Brick LV Structure Elements

- **The BrickLV structure elements consist of two main categories**
  - Global elements BrickLV. which affect all the channels
  - Channel specific elements BrickLV.Chan[]. which only affect the indexed channel.

- **Each category (global or channel) consists of:**
  - Saved Setup Elements
  - Non-saved Setup Elements (automatically reset)
  - Status (read only)

- **Noteworthy Structure Elements**
  - BrickLV.Chan[].TwoPhaseMode
    - =0 for Brushless/Brushed
    - =1
  - BrickLV.Reset
    - Clear amplifier faults
    - Apply changes made to structure elements

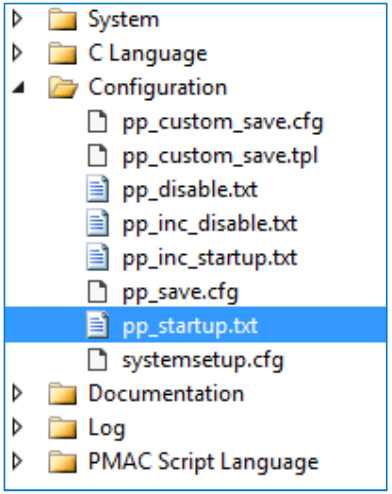
BRICKLV STRUCTURE ELEMENTS 206
Global Saved Setup Elements 207
BrickLV.MonitorPeriod 207
Global Non-Saved Setup Elements 208
BrickLV.Config 208
BrickLV.Monitor 210
BrickLV.Reset 212
Global Status Elements 213
BrickLV.BusOverVoltage 213
BrickLV.BusUnderVoltage 213
BrickLV.OverTemp 214
Channel Saved Setup Elements 215
BrickLV.Chan[j].I2tWarnOnly 215
BrickLV.Chan[j].TwoPhaseMode 216
Channel Status Elements 217
BrickLV.Chan[j].I2tExcess 217
BrickLV.Chan[j].OverCurrent 218
BrickLV.Chan[j].ActivePhaseMode 218
BrickLVVers 219

Power-On Reset PLC

- **Power Brick LV PowerOnResetPLC**
  - Necessary for proper functionality
  - Must be launched on startup
  - Can launch other PLCs from within
  - Clears amplifier faults
  - Saves any changes made to the BrickLV Elements

pp_startup.txt

```
ENABLE PLC PowerOnResetPLC
```

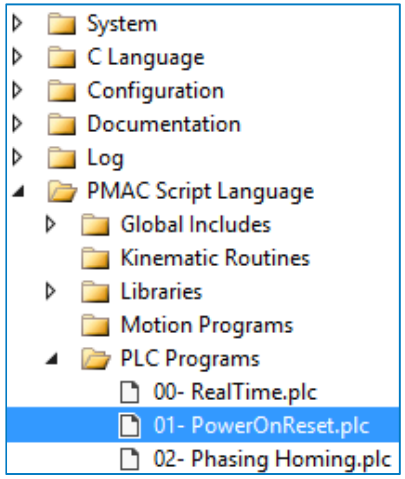



```
OPEN PLC PowerOnResetPLC
Sys.WDTReset = 5000 / (Sys.ServoPeriod * 2.258)
CALL DelayTimer(0.005)

BrickLV.Reset = 1
WHILE (BrickLV.Reset == 1){}
IF (BrickLV.Reset == 0)
{
   Sys.WDTReset = 0

   // HERE, ENABLE PLCs

   DISABLE PLC PowerOnResetPLC
   CALL DelayTimer(0.005)
}
ELSE
{
   // RESET FAILED, TAKE ACTION
   KILL 1..8
   DISABLE PLC 0,2..31
   SEND 1"RESET FAILED !!!"
   Sys.WDTReset = 0
   DISABLE PLC PowerOnResetPLC
   CALL DelayTimer(0.005)
}
CLOSE
```



Data Unpacking

➢ Input & Output Data Packing

- The Gate data must be ALWAYS be unpacked with the Power Brick LV

```
PowerBrick[0].Chan[0].PackOutData = 0
PowerBrick[0].Chan[1].PackOutData = 0
PowerBrick[0].Chan[2].PackOutData = 0
PowerBrick[0].Chan[3].PackOutData = 0

PowerBrick[0].Chan[0].PackInData = 0
PowerBrick[0].Chan[1].PackInData = 0
PowerBrick[0].Chan[2].PackInData = 0
PowerBrick[0].Chan[3].PackInData = 0
```

Note

If an 8-axis Power Brick LV, then PowerBrick[1] elements must also be set to 0.

Power Brick LV Motor Support

- **The Power Brick LV drives directly the following types of motors**
  - 3-phase Brushless
    - AC/ DC, Rotary, Linear
  - 2-phase Stepper w/o encoder
    - Typically, Rotary
  - 2-phase Stepper w/ encoder
    - Typically, Rotary
  - DC Brush
    - Rotary or linear actuator such as voicecoil

Brushless Motor Configuration

- **Having implemented the following:**
  - System optimization settings
  - Dominant clock settings
    - Sys.pAbortAll
  - Data Unpacking
  - BrickLV Structure Elements
    - **BrickLV.Chan[].TwoPhaseMode = 0** for brushless
  - PowerOnResetPLC, and executed once
    - Automatic on download
  - Verified encoder feedback
    - Set up Absolute Position Read

- **We can now configure a brushless motor**
  - Common brushless motor setup elements
  - PWM Scale Factor
  - On-Going Phase Position
  - I2T Protection

Brushless Motor Configuration

Common Brushless Motor Setup Elements

```
Motor[1].PhaseCtrl = 4                                  // UNPACKED ADCs COMMUTATION
Motor[1].PhaseOffset = 683                              // 3-PHASE MOTOR
Motor[1].pLimits=PowerBrick[0].Chan[0].Status.a         // =0 TO DISABLE
Motor[1].AdcMask = $FFFC0000                            // 14-BIT ADCs (BRICK LV)
Motor[1].AmpFaultLevel = 1                              // HIGH TRUE (BRICK LV)
```

PWM Scale Factor (Motor[].PwmSf)

- If motor voltage >= bus voltage, **Motor[].PwmSf = 16384**
- If motor voltage < bus voltage, **Motor[].PwmSf = MotorVoltage * 16384 / BusVoltage**

Brushless Motor Configuration

- **On-Going Phase Position (Motor[].PhasePosSf)**
  - Depends on encoder type, and resolution and motor poles, or magnetic pitch.
  - E.g. for a quadrature encoder:

```
Motor[].pPhaseEnc = ACC24E3[].Chan[].PhaseCapt.a
Motor[].PhasePosSf = 2048 * NoOfPolePairs / (256 * CountsPerRev)  // ROTARY MOTOR
Motor[].PhasePosSf = 2048 * RESmm / (256 * ECLmm)                 // LINEAR MOTOR
```

  - Where:
    - **NoOfPolePairs** is the number of pole pairs of a rotary motor
    - **CountsPerRev** is the number of raw quadrature encoder counts per revolution of a rotary motor
    - **ECLmm** is the linear motor electrical cycle length or magnetic pitch (e.g. 60.96 mm)
    - **RESmm** is the linear encoder resolution (a.k.a. pitch) in the same unit as ECLmm (e.g. 1 µm = 0.001 mm)

Brushless Motor Configuration

➢ I2T Settings (Thermal Protection)

- Need to know continuous and peak current values. Lowest between the PBLV and motor are chosen
- Need to know allowed time at peak current (in seconds)
- Need to know amplifier max. adc reading in amps

```
Motor[].MaxDac = RmsPeakCur * 32768 * SQRT(2) * COSD(30) / PeakMaxAdc
Motor[].I2TSet = RmsContCur * 32768 * SQRT(2) * COSD(30) / PeakMaxAdc
Motor[].I2tTrip = (POW(Motor[].MaxDac,2) - POW(Motor[].I2TSet,2)) * TimeAtPeak
```

- Where
  - **RmsPeakCur** and **RmsContCur** are the lower peak and continuous RMS current values between the motor and the amplifier
  - **TimeAtPeak** is the maximum time allowed at the chosen peak current
  - **PeakMaxAdc** is the adc sensors scale in peak amps (amplifier spec)

Note

If the current limits are given as peak values, there is no need to multiply by $\sqrt{2}$

Axis Current Rating	Max ADC
0.25/0.75A	**1.6925 A**
1/3A	**6.770 A**
5/15A	**33.85 A**

Brushless Motor Commissioning

- **Having implemented the following:**
  - Encoder setup and verification
  - Amplifier and motor configuration

- **We can now commissioning a brushless motor (next presentation) with the Power Brick LV**

PROCEDURE	DESCRIPTION
Current loop tuning	Current loop tuning
Motor phasing	Establishing a phase reference
Open loop test	Verify output/encoder polarity
Position loop tuning	Servo loop tuning
Absolute power-on phasing*	Motion–less phase reference

***If absolute feedback device**

Stepper Motor (w/o encoder) Configuration

- **Having implemented the following:**
  - System optimization settings
  - Dominant clock settings
    - Sys.pAbortAll
  - Data Unpacking
  - BrickLV Structure Elements
    - **BrickLV.Chan[].TwoPhaseMode = 1** for stepper
  - PowerOnResetPLC, and executed once
    - Automatic on download
  - Set up an encoder conversion table for the direct-micro-stepping technique

- **We can now configure a stepper motor**
  - Common stepper motor setup elements
  - I2T Protection
  - PWM Scale Factor
  - Direct Magnetization Current
  - Max Command Output

Stepper Motor (w/o encoder) Configuration

➢ **Common Stepper Motor Setup Elements**

```
Motor[].PhaseCtrl = 6                                  // UNPACKED, MICRO-STEPPING
Motor[].PhaseOffset = 512                              // 2-PHASE MOTOR
Motor[].pLimits=PowerBrick[].Chan[].Status.a           // =0 TO DISABLE
Motor[].AdcMask = $FFFC0000                            // 14-BIT ADCs (BRICK LV)
Motor[].AmpFaultLevel = 1                              // HIGH TRUE (BRICK LV)

Motor[].PhaseMode = 1                                  // DISABLE THIRD HARMONIC INJECTION
Motor[].PhasePosSf = 0
Motor[].pAbsPhasePos = PowerBrick[].Chan[].PhaseCapt.a
Motor[].PowerOnMode = 2
Motor[].SlipGain = Sys.PhaseOverServoPeriod / (Motor[].Stime + 1)
Motor[].AdvGain = 1/16*Sys.PhaseOverServoPeriod*(0.25/Sys.ServoPeriod/Sys.PhaseOverServoPeriod)

Motor[].Servo.Kp = 1 / Motor[].PosSf
Motor[].Servo.Kvfb = 0
Motor[].Servo.Kvifb = 0
Motor[].Servo.Kvff = 1 / Motor[].PosSf
Motor[].Servo.Kviff = 0
Motor[].Servo.Kaff = 1 / Motor[].PosSf
Motor[].Servo.Ki = 0
```

Stepper Motor (w/o encoder) Configuration

➢ I2T Settings (Thermal Protection)

- Need to know continuous and peak current values. Lowest between the PBLV and motor are chosen
- Need to know allowed time at peak current (in seconds)
- Need to know amplifier max. adc reading in amps
- MaxDac is computed differently with direct micro-stepping, a global variable is used to complete I2T Settings

```
GLOBAL CalcMaxDac
Motor[].MaxDac = RmsPeakCur * 32768 * SQRT(2) * COSD(30) / PeakMaxAdc
Motor[].I2TSet = RmsContCur * 32768 * SQRT(2) * COSD(30) / PeakMaxAdc
Motor[].I2tTrip = (POW(CalcMaxDac,2) - POW(Motor[].I2TSet,2)) * TimeAtPeak
```

- Where
  - **RmsPeakCur** and **RmsContCur** are the lower peak and continuous RMS current values between the motor and the amplifier
  - **TimeAtPeak** is the maximum time allowed at the chosen peak current
  - **PeakMaxAdc** is the adc sensors scale in peak amps (amplifier spec)

Note

If the current limits are given as peak values, there is no need to multiply by $\sqrt{2}$

Axis Current Rating	Max ADC
0.25/0.75A	**1.6925 A**
1/3A	**6.770 A**
5/15A	**33.85 A**

Stepper Motor (w/o encoder) Configuration

- **PWM Scale Factor (Motor[].PwmSf)**
  - If motor voltage >= bus voltage, **Motor[].PwmSf = 16384**
  - If motor voltage < bus voltage, **Motor[].PwmSf = MotorVoltage * 16384 / BusVoltage**

- **Direct Magnetization Current (Motor[].IdCmd)**
  - Typical setting

```
Motor[1].IdCmd = Motor[1].I2TSet / 2
```

  - Can be modified subsequently to optimize heat dissipation (in standstill)

Stepper Motor (w/o encoder) Configuration

➢ Maximum Command Output (Motor[].MaxDac)

- With direct micro-stepping, MaxDac is computed as follows

```
GLOBAL MaxRPM = 1500
GLOBAL StepAngle = 1.8
Motor[].MaxDac = MaxRpm / 60000 * (360 / (4 * StepAngle)) * 2048 * Sys.ServoPeriod
```

➢ Need to reach top speed?

- If the product **Motor[].MaxDac * Motor[].SlipGain is ≥ 512** then
  - Reduce servo frequency Or increase **Motor[].Stime**
  - Until condition is satisfied

Maximum Achievable Speeds

The direct microstepping technique has a maximum speed of 1024 microsteps per servo cycle, and 512 microsteps per phase cycle.

Example: For a standard 100-pole (1.8°) stepper motor with a 5 kHz servo update rate, and a 20 kHz phase update rate. The maximum achievable speed can be computed as follows:

A 1.8° full-step motor has a ¼ x 1.8° = 7.2° commutation cycle, therefore:

Servo limitation:

$$\text{MaxRpm} = 1{,}024\ \frac{\text{mstep}}{\text{cycle}} \times 5{,}000\ \frac{\text{cycle}}{\text{sec}} \times \frac{7.2}{2{,}048}\ \frac{\text{deg}}{\text{mstep}} \times \frac{1}{360}\ \frac{\text{rev}}{\text{deg}} \times 60\ \frac{\text{sec}}{\text{min}} = 3{,}000\ \text{rpm}$$

Phase Limitation:

$$\text{MaxRpm} = 512\ \frac{\text{mstep}}{\text{cycle}} \times 20{,}000\ \frac{\text{cycle}}{\text{sec}} \times \frac{7.2}{2{,}048}\ \frac{\text{deg}}{\text{mstep}} \times \frac{1}{360}\ \frac{\text{rev}}{\text{deg}} \times 60\ \frac{\text{sec}}{\text{min}} = 6{,}000\ \text{rpm}$$

Therefore, the maximum achievable speed (servo limitation) is 3,000 rpm. Higher speeds will require increasing the update rate(s) correspondingly.

Stepper Motor (w/o encoder) Configuration

- **Having implemented the following:**
  - Encoder Conversion Table for Direct Micro-Stepping
  - Amplifier and motor configuration

- **We can now commissioning a stepper motor w/o encoder (next presentation) with the Power Brick LV**

PROCEDURE	DESCRIPTION
Current loop tuning	Current loop tuning

Stepper Motor (w/ encoder) Configuration

- **Having implemented the following:**
  - System optimization settings
  - Dominant clock settings
    - Sys.pAbortAll
  - Data Unpacking
  - BrickLV Structure Elements
    - **BrickLV.Chan[].TwoPhaseMode = 1** for stepper
  - PowerOnResetPLC, and executed once
    - Automatic on download
  - Verified encoder feedback
    - Set up Absolute Position Read

- **We can now configure a stepper motor**
  - Common stepper motor setup elements
  - PWM Scale Factor
  - I2T Protection
  - On-going phase position

- **Control Strategy**
  - Traditionally, the direct micro-stepping technique is used for commutation and only servo position acquired from the encoder
  - But for best performance, this configuration (described in this presentation) is implemented as a high pole-count with both commutation and servo positions coming from the encoder

Stepper Motor (w/ encoder) Configuration

Common Stepper Motor Setup Elements

```
Motor[1].PhaseCtrl = 4                                  // UNPACKED ADCs COMMUTATION
Motor[1].PhaseOffset = 512                              // 2-PHASE MOTOR
Motor[1].pLimits=PowerBrick[0].Chan[0].Status.a         // =0 TO DISABLE
Motor[1].AdcMask = $FFFC0000                            // 14-BIT ADCs (BRICK LV)
Motor[1].AmpFaultLevel = 1                              // HIGH TRUE (BRICK LV)
```

PWM Scale Factor (Motor[].PwmSf)

- If motor voltage >= bus voltage, **Motor[].PwmSf = 16384**
- If motor voltage < bus voltage, **Motor[].PwmSf = MotorVoltage * 16384 / BusVoltage**

Stepper Motor (w/ encoder) Configuration

➢ I2T Settings (Thermal Protection)

- Need to know continuous and peak current values. Lowest between the PBLV and motor are chosen
- Need to know allowed time at peak current (in seconds)
- Need to know amplifier max. adc reading in amps

```
Motor[].MaxDac = RmsPeakCur * 32768 * SQRT(2) * COSD(30) / PeakMaxAdc
Motor[].I2TSet = RmsContCur * 32768 * SQRT(2) * COSD(30) / PeakMaxAdc
Motor[].I2tTrip = (POW(Motor[].MaxDac,2) - POW(Motor[].I2TSet,2)) * TimeAtPeak
```

- Where
  - **RmsPeakCur** and **RmsContCur** are the lower peak and continuous RMS current values between the motor and the amplifier
  - **TimeAtPeak** is the maximum time allowed at the chosen peak current
  - **PeakMaxAdc** is the adc sensors scale in peak amps (amplifier spec)

Note

If the current limits are given as peak values, there is no need to multiply by $\sqrt{2}$

Axis Current Rating	Max ADC
0.25/0.75A	**1.6925 A**
1/3A	**6.770 A**
5/15A	**33.85 A**

Stepper Motor (w/ encoder) Configuration

➢ On-Going Phase Position (Motor[].PhasePosSf)

- The ongoing phase position for stepper motors with encoders is set up similarly to brushless motors
- The number of poles pairs involved in the ongoing phase position settings is computed as follows:
  - Number of pole pairs = 360 / (Step Angle * 4)
  - E.g. A 1.8° step motors yields 50 pair poles

```
Motor[].pPhaseEnc = ACC24E3[].Chan[].PhaseCapt.a
Motor[].PhasePosSf = 2048 * NoOfPolePairs / (256 * CountsPerRev)  // ROTARY MOTOR
```

- Where:
  - **NoOfPolePairs** is the number of pole pairs of a rotary motor
  - **CountsPerRev** is the number of raw quadrature encoder counts per revolution of a rotary motor

Stepper Motor (w/ encoder) Configuration

- **Having implemented the following:**
  - Encoder setup and verification
  - Amplifier and motor configuration

- **We can now commissioning a stepper motor w/ encoder (next presentation) with the Power Brick LV**

PROCEDURE	DESCRIPTION
Current loop tuning	Current loop tuning
Motor phasing	Establishing a phase reference
Open loop test	Verify output/encoder polarity
Position loop tuning	Servo loop tuning
Absolute power-on phasing*	Motion–less phase reference

***If absolute feedback device**

DC Brush Motor Configuration

- **Having implemented the following:**
  - System optimization settings
  - Dominant clock settings
    - Sys.pAbortAll
  - Data Unpacking
  - BrickLV Structure Elements
    - **BrickLV.Chan[].TwoPhaseMode = 0** for brush
  - PowerOnResetPLC, and executed once
    - Automatic on download
  - Verified encoder feedback
    - Set up Absolute Position Read

- **We can now configure a brush motor**
  - Common brush motor setup elements
  - PWM Scale Factor
  - I2T Protection

DC Brush Motor Configuration

Common Brush Motor Setup Elements

```
Motor[].PhaseCtrl = 4                                  // UNPACKED ADCs COMMUTATION
Motor[].PhaseOffset = 512                              // 2-PHASE MOTOR
Motor[].pLimits=PowerBrick[0].Chan[0].Status.a         // =0 TO DISABLE
Motor[].AdcMask = $FFFC0000                            // 14-BIT ADCs (BRICK LV)
Motor[].AmpFaultLevel = 1                              // HIGH TRUE (BRICK LV)

Motor[].PhaseMode = 3                                  // NO THIRD HARMONIC, DIRECT CURRENT OFF
Motor[].PhasePosSf = 0                                 // NO COMMUTATION
Motor[].pAbsPhasePos = Sys.pushm                       // POINT TO SCRATCH MEMORY
Motor[].PowerOnMode = 2                                // DUMMY POWER-ON PHASING
```

PWM Scale Factor (Motor[].PwmSf)

- If motor voltage >= bus voltage, **Motor[].PwmSf = 16384**
- If motor voltage < bus voltage, **Motor[].PwmSf = MotorVoltage * 16384 / BusVoltage**

DC Brush Motor Configuration

➢ I2T Settings (Thermal Protection)

- Need to know continuous and peak current values. Lowest between the PBLV and motor are chosen
- Need to know allowed time at peak current (in seconds)
- Need to know amplifier max. adc reading in amps

```
Motor[].MaxDac = RmsPeakCur * 32768 / PeakMaxAdc
Motor[].I2TSet = RmsContCur * 32768 / PeakMaxAdc
Motor[].I2tTrip = (POW(Motor[].MaxDac,2) - POW(Motor[].I2TSet,2)) * TimeAtPeak
```

- Where
  - **RmsPeakCur** and **RmsContCur** are the lower peak and continuous RMS current values between the motor and the amplifier
  - **TimeAtPeak** is the maximum time allowed at the chosen peak current
  - **PeakMaxAdc** is the adc sensors scale in peak amps (amplifier spec)

Note

If the current limits are given as peak values, there is no need to multiply by $\sqrt{2}$

Axis Current Rating	Max ADC
0.25/0.75A	**1.6925 A**
1/3A	**6.770 A**
5/15A	**33.85 A**

DC Brush Motor Commissioning

- **Having implemented the following:**
  - Encoder setup and verification
  - Amplifier and motor configuration

- **We can now commissioning a Brush motor (next presentation) with the Power Brick LV**

PROCEDURE	DESCRIPTION
Current loop tuning*	Current loop tuning
Open loop test	Verify output/encoder polarity
Position loop tuning	Servo loop tuning

***Requires special settings before tuning**