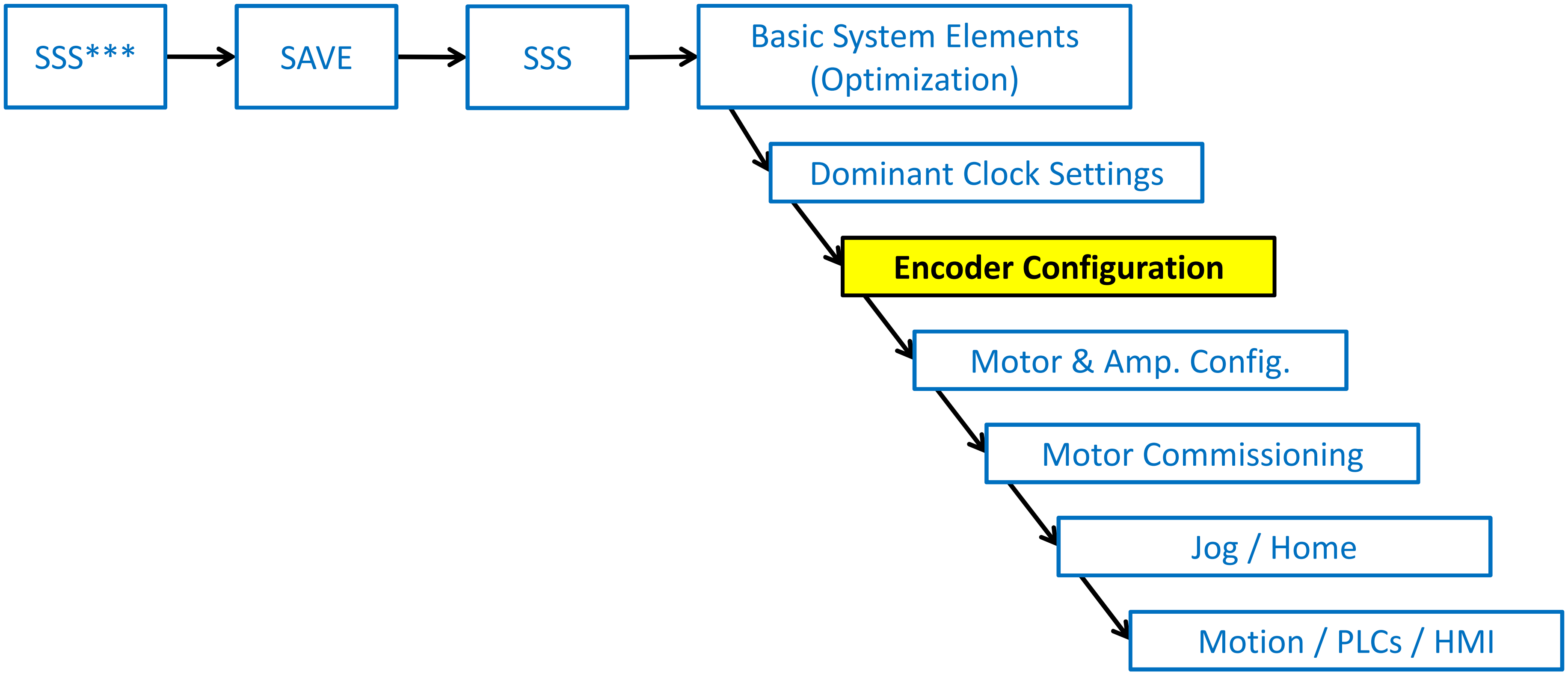
Power Brick LV
Encoder Configuration

System Configuration



Encoder Conversion Table ECT

- **The Encoder Conversion Table (ECT) pre-processes raw feedback data and scales it into double-precision floating point for use in the servo loop**

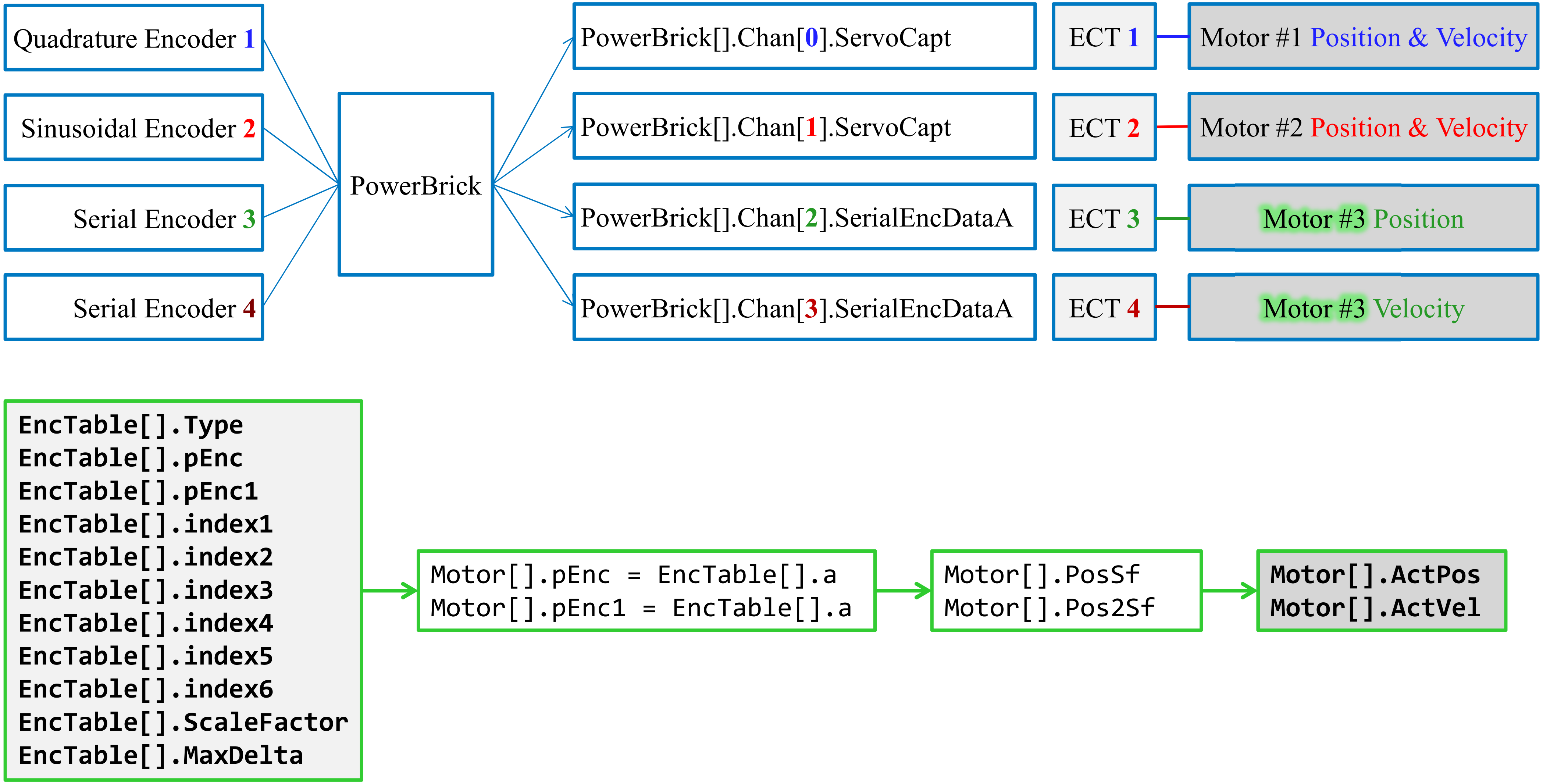
- **The result of this processing is also known as on-going position data**
  - As opposed to the one-time absolute position read aka absolute homing (typically on power-up)

- **The ECT can also be used for custom processing such as:**
  - Data manipulation (left/right shift)
  - Numerical manipulation
    - Single or double integration, or differentiation
  - Max change (filter)
  - Average-tracking (filter)
  - Sum and difference of various sources
  - Maximum change (velocity or acceleration) cap

Note

The ECT executes at servo rate before motor servo calculations

ECT Concept Example



ECT Main Structure

- **Type specifies the conversion method**
  - 0: End of table
  - 1: Single-word (32-bit) read
  - 2: Double-word (24-bit + 8-bit) read
  - 3: Software 1/T encoder extension
  - 4: Sinusoidal encoder arctangent extension
  - 5: Four-byte read
  - 6: Resolver direct conversion
  - 7: Extended hardware sinusoidal interpolation
  - 8: Addition of two sources
  - 9: Subtraction of two sources
  - 10: Triggered time base
  - 11: Floating-point read
  - 12: Single-word read with error check

Note

Specifying the end of the encoder conversion table minimizes CPU computations

- **pEnc specifies the primary source address**
- **pEnc1 specifies the secondary source address**
- **The indexes allow data or numerical manipulation(s)**
- **ScaleFactor specifies the units of the output (LSB)**

Note

All elements are Type dependent

```
EncTable[].Type
EncTable[].pEnc
EncTable[].pEnc1
EncTable[].index1
EncTable[].index2
EncTable[].index3
EncTable[].index4
EncTable[].index5
EncTable[].index6
EncTable[].ScaleFactor
EncTable[].MaxDelta
```

Quadrature On-Going Position Ex.

- **Quadrature incremental encoder. E.g. Rotary with 5,000 lines per revolution.**
  - Default x4 decode processing (typical), controlled by **PowerBrick[].Chan[].EncCtrl**.
  - Produces 4 x 5,000 = 20,000 counts/revolution.
  - 1/T sub-count interpolation performed in the PowerBrick.
  - Sampling rate controlled by **PowerBrick[].EncClockDiv**. Default is 3.125 MHz.
  - Should increase for higher resolution/speeds if **PowerBrick[].Chan[].CountError** is set to 1.
  - Encoder loss detection

```
EncTable[1].Type = 1
EncTable[1].pEnc = PowerBrick[0].Chan[0].ServoCapt.a
EncTable[1].pEnc1 = Sys.Pushm
EncTable[1].index1 = 0
EncTable[1].index2 = 0
EncTable[1].index3 = 0
EncTable[1].index4 = 0
EncTable[1].index5 = 0
EncTable[1].index6 = 0
EncTable[1].ScaleFactor = 1 / 256
EncTable[1].MaxDelta = 0
```

```
Motor[1].ServoCtrl = 1   // MUST ACTIVATE TO SEE COUNTS
Motor[1].pEnc = EncTable[1].a
Motor[1].pEnc1 = EncTable[1].a
```

Sinusoidal On-Going Position Ex.

- **Sinusoidal encoder. E.g. Rotary with 1,000 sine/cosine lines per revolution.**
  - Standard interpolation x16384. (ACI x65536).
  - Produces 1,000 x 16,384 = 16,384,000 counts/rev
  - Arctangent calculation performed in the PowerBrick.
  - Sampling rate controlled by **PowerBrick.EncClockDiv**, PowerBrick**[].AdcEncClockDiv**. Default is 3.125 MHz.
  - Should increase for higher resolution/speeds if **PowerBrick[].Chan[].CountError** is set to 1.
  - ADC Bias calibration.
  - Encoder loss detection using Sum of squares.

```
EncTable[1].Type = 1
EncTable[1].pEnc = PowerBrick[0].Chan[0].ServoCapt.a
EncTable[1].pEnc1 = Sys.Pushm
EncTable[1].index1 = 0
EncTable[1].index2 = 0
EncTable[1].index3 = 0
EncTable[1].index4 = 0
EncTable[1].index5 = 0
EncTable[1].index6 = 0
EncTable[1].ScaleFactor = 1
EncTable[1].MaxDelta = 0
```

```
Motor[1].ServoCtrl = 1   // MUST ACTIVATE TO SEE COUNTS
Motor[1].pEnc = EncTable[1].a
Motor[1].pEnc1 = EncTable[1].a
```

```
Motor[1].EncType = 6
PowerBrick[0].Chan[0].AtanEna = 1
```

Serial On-Going Position Ex.

- **Serial encoder. E.g. Rotary with 20-bit single-turn (starting from bit 4), and 16-bit multi-turn data.**
  - Single-turn data is sufficient for on-going position.
  - Data size specified by **PowerBrick[].Chan[].SerialEncCmd**.
  - PMAC expects data to be left most shifted to handle rollover gracefully
  - Transmission speed dictated by **PowerBrick[].SerialEncCtrl**. Encoder dependent.
  - Produces $2^{\text{single-turn}} = 1,048,576$ counts/rev
  - Encoder errors in **SerialEncDataB**.

```
EncTable[1].Type = 1
EncTable[1].pEnc = PowerBrick[0].Chan[0].SerialEncDataA.a
EncTable[1].pEnc1 = Sys.Pushm
EncTable[1].index1 = 12
EncTable[1].index2 = 4
EncTable[1].index3 = 0
EncTable[1].index4 = 0
EncTable[1].index5 = 0
EncTable[1].index6 = 0
EncTable[1].ScaleFactor = 1 / EXP2(12)
EncTable[1].MaxDelta = 0
```

```
Motor[1].ServoCtrl = 1   // MUST ACTIVATE TO SEE COUNTS
Motor[1].pEnc = EncTable[1].a
Motor[1].pEnc1 = EncTable[1].a
```

```
PowerBrick[0].Chan[0].SerialEncEna = 1
```

Note

Any one of the Power Brick manuals provides explicit examples which cover most serial encoder cases with the Gate3

Direct Micro-Stepping On-Going Position Ex.

- **The Direct Micro-Stepping technique for driving stepper motors with a Power Brick LV requires a special encoder conversion table entry**
  - Utilizes internally generated pseudo-feedback for both commutation (phase) and servo positions.
  - Produces **360 * 512 / Step Angle** counts/rev
    - E.g. 1.8° step angle motor yields 102,400 cts/rev

```
EncTable[1].Type = 11
EncTable[1].pEnc = Motor[1].PhasePos.a
EncTable[1].pEnc1 = Sys.Pushm
EncTable[1].index1 = 5
EncTable[1].index2 = 0
EncTable[1].index3 = 0
EncTable[1].index4 = 0
EncTable[1].index5 = 255
EncTable[1].index6 = 1
EncTable[1].ScaleFactor =  1 / (256 * (EncTable[1].index5 + 1) * EXP2(EncTable[1].index1))
```

```
Motor[1].ServoCtrl = 1   // MUST ACTIVATE TO SEE COUNTS
Motor[1].pEnc = EncTable[1].a
Motor[1].pEnc1 = EncTable[1].a
```

Scaling to User Engineering Units

- **Many legacy (non-turbo & turbo) PMAC are used to raw encoder counts**
  - Jogging, following error limit, home offset etc.. all in raw encoder counts
  - Development software (equivalent to the IDE) position windows scaled manually
  - Scaled to engineering units in the coordinate system axis definition

- **With Power PMAC, it is strongly suggested to scale raw encoder counts to engineering units**
  - Allows jogging in engineering units
  - All structure elements specified in motor units (m.u.) become in user engineering units e.g. degrees, inches, mm
    - Elements such as FeFatalLimit, HomeOffset, InPosBand etc..
  - IDE position windows display in user engineering units (without any properties manual scaling)
  - This scaling should include encoder resolution and any coupling (gearing) to reflect the engineering units at the load

- **This scaling is best performed using**
  - Position scale factor, **Motor[].PosSf**

    AND
  - Velocity scale factor, **Motor[].Pos2Sf**

Caution

Changing PosSf and Pos2Sf has significant implications on numerous element settings including tuning gains. It should be done prior to configuring and commissioning a motor

Scaling to User Engineering Units Ex.

- **Motor #1 is a direct drive rotary motor scaled to degrees**
  - 17-bit single-turn serial encoder producing 2^{17} = 131,072 counts/revolution

```
Motor[1].PosSf = 360 / 131072            // DEGREES  PER COUNT
Motor[1].Pos2Sf = Motor[1].PosSf         // SAME ENCODER USED FOR VELOCITY
```

- **Motor #2 is a geared rotary motor (single feedback) scaled to inches**
  - 23-bit single-turn serial encoder producing 2^{23}=8,388,608 counts/revolution
  - 10:1 gearbox (10 revolutions per inch)

```
Motor[2].PosSf = 1 / (10 * 8388608)      // INCHES  PER COUNT
Motor[2].Pos2Sf = Motor[2].PosSf         // SAME ENCODER USED FOR VELOCITY
```

- **Motor #3 is a linear direct (load) drive motor scaled to millimeters**
  - 1 nm resolution yielding 1,000,000 counts/millimeter

```
Motor[3].PosSf = 1 / 1000000             // MILLIMETERS  PER COUNT
Motor[3].Pos2Sf = Motor[3].PosSf         // SAME ENCODER USED FOR VELOCITY
```

Scaling to User Engineering Units Ex.

- **Motor #4 is a geared rotary motor (dual feedback) scaled to millimeters**
  - 500-line quadrature shaft encoder producing 2,000 counts/revolution
  - 10:1 gearbox (10 revolutions per mm)
  - Linear load encoder with 50 nm resolution, or 20,000 counts/mm

```
Motor[4].PosSf = 1 / 20000              // MILLIMETERS PER COUNT
Motor[4].Pos2Sf = 1 / (10 * 2000)       //
```

- **Motor #5 is a 1.8° stepper motor (direct micro-stepping) scaled to degrees**
  - The direct micro-stepping technique yields **360 * 512 / Step Angle** counts/rev

```
Motor[5].PosSf = 360 / 102400           // DEGREES  PER COUNT
Motor[5].Pos2Sf = Motor[5].PosSf        //
```

Note: With the motors scaled using PosSf and Pos2Sf, the IDE position windows display is now in engineering units. No need to re-scale

Note: With absolute encoders, remember to set Motor[].AbsPosSf = Motor[].PosSf

Verifying Encoder Feedback

- **Encoder verification**
  - Ideally, done by moving the motor/encoder by hand (if possible)
  - Is it counting up and down (positive and negative)?
    - Typically, looking at the position window
  - Is it counting correctly?
    - As expected by the processing/resolution
    - Measure roughly 1 revolution or inches/mm
  - Is it repeatable?
    - Going back to same location produces same number

Note

Raw encoder data can be read using **EncTable[].PrevEnc** multiplied by **EncTable[].ScaleFactor**. Multiply by **Motor[].PosSf** to compute **Motor[].ActPos**

Caution

Encoder verification is a critical step in commissioning a motor. Subsequent configuration steps depend on the encoder counting properly and correctly