Comparison of AN1292 and Motor Control Application Framework

1 Comparison of AN1292 and MC Application Framework

1.1 Preface

This section of the document provides an analysis into differences between AN1292 software and the MC Application Framework supplied with motorBench™ Development Suite. The intention here is to retrospectively review the improvements and limitations of the new MC Application Framework code relative to the reference application note software from AN1292.

1.1.1 Self-commissioning

Self-commissioning is an element of the motorBench™ Development Suite that, with assistance from the dedicated dsPIC software, determines the following parameters of a PMSM motor:

1. Stator resistance (R)
2. Stator inductances (L_D and L_Q)
3. Back-EMF constant (Ke)
4. Friction torque (T_F)
5. Viscous Damping (B)
6. Inertia (J)

1.1.2 Autotuning

Autotuning is an element of the motorBench™ Development Suite that computes the PI controller gains for a specific hardware system based on the motor and hardware parameters.

1.1.3 MC Application Framework

The Motor Control Application Framework (MCAF) is a firmware package that consists of files used to generate code into any desired MPLAB X project. The MC Application Framework is “rendered” into a particular instance of application source code by motorBench™ Development Suite, based on configuration information, Self-Commissioning measurements, and the resulting calculations to tune motor control loops.

1.2 Areas of comparison

This section provides a series of topic-by-topic comparison analysis between AN1292 software and the MC Application Framework that is supplied with motorBench™ Development Suite.

1.2.1 Summary of changes

The following table summarizes at a high-level the details of comparison between AN1292 software and the Motor Control Application Framework (MC Application Framework or MCAF).

Feature	AN1292 software	MC Application Framework
Peak motor torque	+ Perceivably higher torque on small motors - Potential to damage hardware	+ Torque limited to motor specs - Perceivably lower torque
Tuning effort	- Manual effort required + Can be optimized for specific application	+ Automatic with limited user control + Works for a wide range of applications - Tuning is limited to constant loads (i.e. constant viscous damping over velocity)
Shaft-grab test	+ *Appears* difficult to stop the motor + Typically tuned for hand-grab test - No Overload/Stall detection	- Motor *appears* to give up easily due to the new Stall-detect feature - Tuned for a generic load testing
Restart after motor overload	- Not available	+ Automatically tries to start the motor 'n' number of times (can be disabled/changed)
Visual user interface	- Not available	+ LEDs indicate normal operation (heartbeat blink) or error codes
Motor stopping method	+ Motor stops quickly - Higher motor phase current when turned off; this can increase the bus voltage or damage hardware	+ Motor gently coasts down + Motor phase current is insignificant when motor is turned off - Motor takes more time to stop
Diagnostics and testing	- Limited by RTDM/DMCI performance + DMCI plugin available for public use	+ Supports X2C-Scope from the Linz Center of Mechatronics GmbH + RTDM/DMCI supported as well + Supports advanced test modes
CPU execution overhead time for MC code	+ Requires lesser CPU overhead	- Higher CPU overhead due to added features
Modular coding architecture	- Limited to the legacy code architecture	+ Easier to test and port to new hardware + Easier to read code - New architecture requires learning curve - Slightly longer time to compile code
Code base and release architecture	+ Individual code base released for each hardware combination - Difficult to maintain multiple code bases - No HAL, difficult to port for other hardware	+ Unified codebase supporting multiple hardware configurations + HAL allows for easy porting to other hardware - Requires motorBench™ Development Suite
Motor startup	+ Simple algorithm + Start up parameters conservatively tuned to work with multiple motors - Velocity overshoot occurs in most cases - Overcurrent fault can occur if incorrectly tuned	+ Smooth motor start up with no velocity overshoot / current surges + Successfully tested on multiple motors + Automatically tuned for specific motor and mechanical load - Requires mechanical parameters of the motor and load to be determined

* In the above table, (+) implies an advantage and (-) implies a limitation.

1.2.2 Motor current limitation

Motors and motor control board datasheets typically specify the maximum continuous current at which they can safely operate. If the motor currents exceed the specifications of either the motor or the motor control board, then this escalates the risk of damage due to overheating and thermal runaway. In addition to the continuous current rating, some of the motor datasheets and motor control board datasheets specify their peak current rating – this is the maximum current the motor / motor control board can handle for a brief duration of time without damage.

Motor drive software designed to work with a specific motor and a specific board should ideally limit the maximum continuous operating current to within the lowest of the continuous motor current, and a safe continuous board current.

		Part/Model Number
Specification	**Units**	**A0421046NC**
Supply Voltage	VDC	48
Continuous Stall Torque	oz-in	17
	Nm	0.12
Speed @ Cont. Torque	RPM	4000
Current @ Cont. Torque	Amps (A)	2.75
Continuous Output Power	Watts (W)	50.26
Motor Constant	oz-in/sqrt W	3.89
	Nm/sqrt W	0.027
Torque Constant	oz-in/A	6.179
	Nm/A	0.044
Voltage Constant	V/krpm	4.57
	V/rad/s	0.044
Terminal Resistance	Ohms	2.52
Inductance	mH	1.66
Max. Speed	RPM	4500
Peak Current	Amps (A)	8.25
Peak Torque	oz-in	51
	Nm	0.3601

Figure 1: Typical motor datasheet with peak and continuous current ratings

1.2.2.1 How does legacy AN1292 software implement this?

Legacy software from AN1292 limits the motor current during open-loop motor startup to a value defined by the macro Q_CURRENT_REF_OPENLOOP in the ***userparams.h*** file. Once the motor completes its startup phase and enters closed-loop normal mode of operation, motor current is limited by the output of the velocity controller. Since the velocity controller output is preset to saturate at 0x5000 (see macro SPEEDCNTR_OUTMAX in ***userparams.h*** file), the current reference for I_Q is limited to I_{Qsat} as calculated by the Equation (1).

$$I_{Qsat} = 0x5000 \times \frac{I_{peak}}{2^{15}} \quad \text{..... (1)}$$

Where,

I_{peak} is the peak current that can be measured by the ADC

For the MCLV2 board with a peak current capability design of 4.4A, maximum value of motor current, I_{Qsat}, based on Equation (1) comes out to be about $2.75A_{peak} = 1.94A_{RMS}$. If the end-application needs a

different current saturation limit, the value assigned to macro SPEEDCNTR_OUTMAX in the ***userparams.h*** file needs to be updated to the desired value as calculated by Equation (1).

1.2.2.2 How do we do it in MC Application Framework / motorBench™ Development Suite?

MC Application Framework limits the maximum motor current to the lowest amongst two:

1. Continuous current rating of the motor
2. Continuous current rating of the board.

This protects the board and motor from potential damage.

1.2.2.3 Perceivable advantages of the MC Application Framework

Motor current limitation prevents damage to motor and the motor control board.

1.2.2.4 Perceivable disadvantages of the MC Application Framework

By imposing a hard-limit on the motor current, we are effectively limiting the peak mechanical torque produced by the motor to a conservative value even though the motor/board may be able to deliver significantly higher output for a short duration of time.

1.2.2.5 Scope for improvement

Instead of a hard-limit, apply a soft-limit on the motor current that allows the motor current to exceed the continuous motor/board specification by a certain margin for a brief duration of time.

1.2.3 Ease of tuning the application code for a new motor

Typically, the Microchip application note software projects are tuned for specific set of motors Ex: Hurst DMA0204024B101, Hurst DMB0224C10002, Motor-80, etc. While these software projects are good for demo purposes, they will need to be re-tuned before they can be used to run a different motor. This retuning process, while assuming the same hardware, requires updates to the following major components of the application note:

1. Sensorless estimator parameters
2. PI controller parameters for current control loops
3. PI controller parameters for velocity control loop
4. Motor startup parameters

1.2.3.1 Tuning the legacy AN1292 software for a new motor

In order to run the new motor using legacy AN1292 software, end user will have to manually update the ***userparams.h*** file based on the new motor parameters as described in the Application Note tuning guide document. To summarize, following are the high-level steps involved:

1. Obtain electrical parameters of the new motor from its datasheet. If the motor datasheet is unavailable/inaccurate/incomplete, use lab equipment such as a Digital Multimeter (DMM), oscilloscope and a hand-drill to obtain the electrical parameters of the motor
2. Use the motor electrical parameters to update the ***tuning_params.xls*** spreadsheet that is included with the software project
3. Manually copy a set of automatically calculated values from the ***tuning_params.xls*** spreadsheet into the ***userparams.h*** file. In certain cases, scaling/pre-division factors in the spreadsheet and the software may need to be adjusted as well
4. Switch to open-loop mode and update the motor startup parameters
5. Manually tune the PI controller parameters for the I_Q and I_D current control loops. Then, switch to closed-loop mode and tune the PI controller parameters for the velocity control loop
6. Update the slew-rate for velocity based on the motor and its mechanical load (SPEEDREFRAMP in ***userparams.h*** file)

1.2.3.2 Tuning motorBench™ Development Suite for a new motor

In order to run a new motor using motorBench™ Development Suite, user will only need to enter the basic name-plate details of the motor. Based on this information, motorBench™ Development Suite runs its Self-Commissioning process to determine the electrical and mechanical parameters of the motor. Then, it automatically calculates all of the application note parameters and generates the final application note software that can be readily used to run the new motor.

1.2.3.3 Perceivable advantages of motorBench™ Development Suite

Since motorBench™ Development Suite can automatically generate application software that is tuned and out-of-the-box ready to operate with the new motor, it saves the substantial time and effort required to manually perform the same tuning on the legacy AN1292 software.

1.2.3.4 Perceivable disadvantages of motorBench™ Development Suite

While the intention and design behind motorBench™ Development Suite is to support a majority of the motor types and mechanical loads, it cannot inherently support all unique hardware combinations that end users may have to work with.

In these special circumstances, as starting point, the end user can choose in the motorBench™ Development Suite the motor and load options that most closely represent the actual hardware. From this point, the end user can tweak the tuning and application parameters using interfaces provided within motorBench™ Development Suite to achieve the desired tuning performance. In addition to this, end users also have full access and control over the final generated software to make any necessary modifications to the software and tuning parameters.

1.2.4 Velocity and current controllers tuning characteristics

The PI controller gains of current and velocity controllers govern the PI controller response to disturbance and changes in velocity/current reference inputs. These parameters have to be tuned optimally in order to prevent control instability while still maintaining a robust and predictable response to disturbances.

1.2.4.1 How does legacy AN1292 software implement this?

PI controller parameters in the legacy AN1292 software were tuned manually based on the following hardware configuration:

- MCLV2 development board
- Hurst motor DMB0224C10002
- V_{BUS} = 24V, PWM dead time = 2µs, PWM frequency = 20kHz

The default tuning was finalized after iterative testing to ensure a reliable motor startup, smooth operation in field-weakening mode and provide an optimal response to a motor shaft being loaded by hand, also referred to as a “shaft-grab” test.

1.2.4.2 How do we do it in MC Application Framework / motorBench™ Development Suite?

PI controller parameters for current and velocity control loops in the MC Application Framework are automatically calculated by the motorBench™ Development Suite based on motor parameters and the preset configuration of Phase margin and PI phase at the crossover frequency. By default, these gains are conservatively tuned to provide a predictable behavior across a wide range of motors.

Table 1 summarizes the comparison between PI controller parameters used in legacy AN1292 software and those generated by motorBench™ Development Suite for the same hardware configuration with default values of Phase margin and PI phase at crossover.

Control loop	Parameter	Values		Characteristics
		Legacy AN1292 software	motorBench™ Development Suite	
Current	Bandwidth ω_C [rad/s]	2.84	1278	AN1292 has very low bandwidth and a large contribution of PI gain from the integral term.
	Phase margin [degrees]	103.50	79.99	
	PI phase at crossover [degrees]	88.70	45.01	
Velocity	Bandwidth ω_C [rad/s]	182.60	120.3	Integral term provides a very small contribution towards the PI controller gain in AN1292.
	Phase margin [degrees]	42.54	50	
	PI phase at crossover [degrees]	3.92	10	

Table 1: Comparison on PI controller parameters

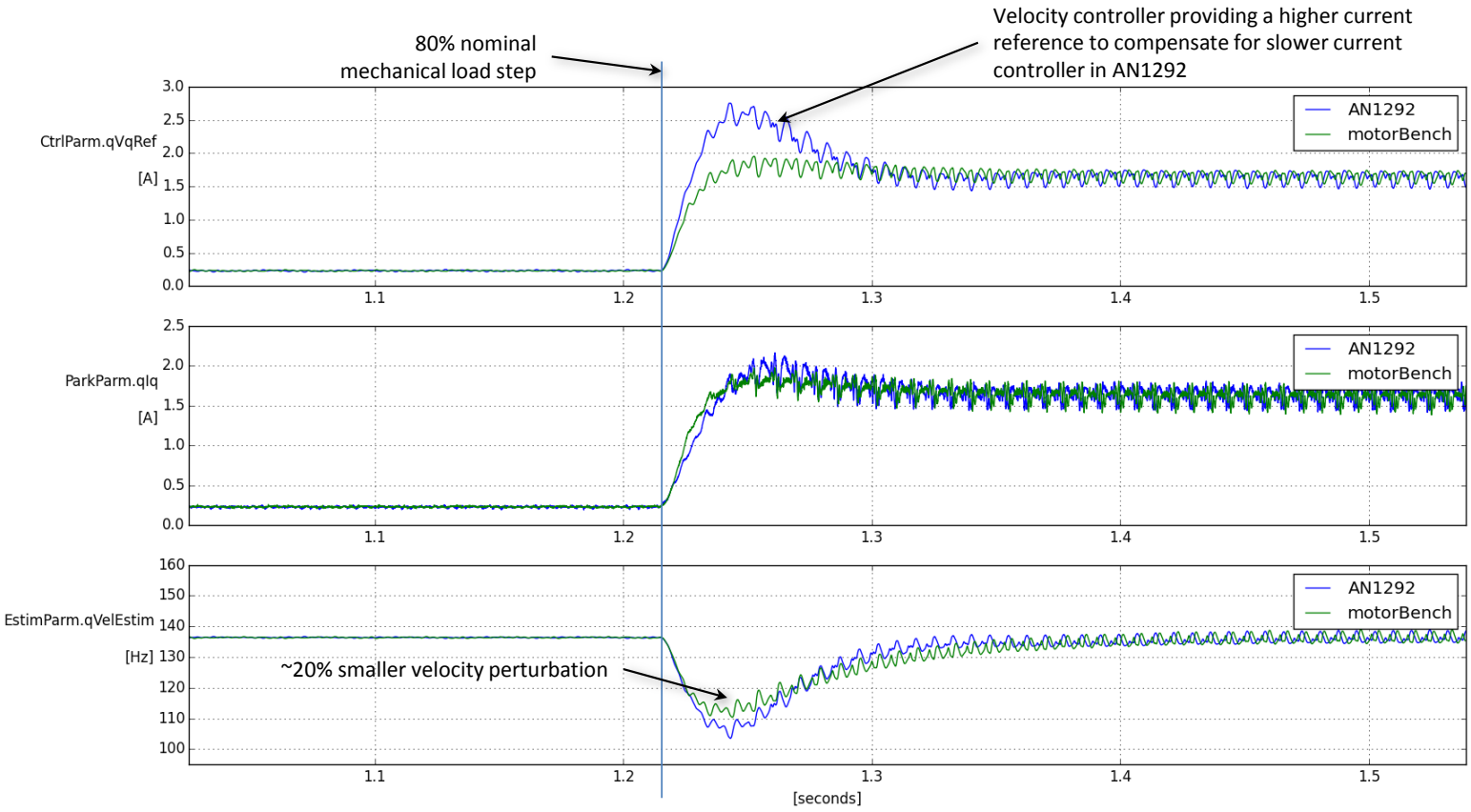

Figure 2: Comparative plot of current and velocity controller response to a mechanical step-load of 80% nominal rated load

Figure 2 shows plots of current and velocity controller responses of the stock AN1292 software and that of the motorBench™ Development Suite tuned AN1292 software. In both cases, the Hurst motor DMB0224C10002 was subjected to the same exact test condition - a mechanical step-load of 80% nominal rated load. A low-inertia bench-top dynamometer was used for this test in order to reduce the effect of added damping factor and inertia in de-tuning the velocity control loop.

1.2.4.3 Perceivable advantages of motorBench™ Development Suite tuned PI controller parameters

The PI controller gains calculated by motorBench™ Development Suite will work out-of-the-box across a wide range of motors.

1.2.4.4 Perceivable disadvantages of motorBench™ Development Suite tuned PI controller parameters

Tuning parameters obtained from the motorBench™ Development Suite is certainly a good starting point but it does not always address the unique goals of all applications out there. End users will have to manually tweak the output of motorBench™ Development Suite to get the precise controller response that they are looking for.

1.2.5 Motor startup

Motor startup routine provides the critical function of starting a motor from zero velocity and ramping up its velocity to the minimum value at which the sensorless position estimator can reliably operate.

1.2.5.1 How does legacy AN1292 software implement this?

The motor startup routine of AN1292 uses a simple three-step approach:

1. Lock phase
    a. Enable closed loop current control for Iq and Id
    b. Apply a DC current on Iq that is equal to the continuous current rating of the motor
    c. Wait for a predefined amount of time for the motor to align
2. Velocity ramp up phase
    a. Start incrementing the phase angle of applied voltage (i.e. forced angle) at a certain rate to achieve the desired velocity
    b. Increment the velocity to ramp up the motor to a minimum velocity at which the sensorless estimator operates reliably
3. Transition phase
    a. Switch the commutation angle from forced angle to estimated angle to complete the transition
    b. Reduce the angle offset that is added to the estimated motor angle

Figure 3 shows a plot of motor current and velocity during the startup phase with AN1292 software, MCLV2 board and Hurst motor (DMB0224C10002) with no mechanical load attached to it.

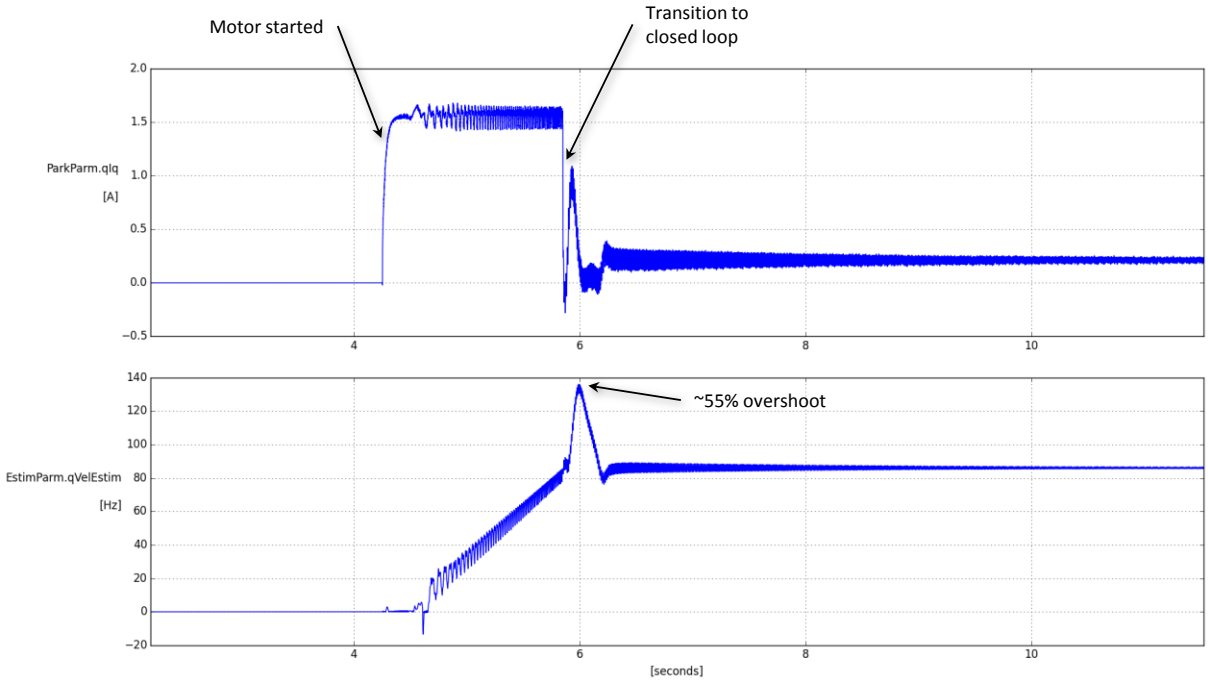

Figure 3: Plot of motor current and velocity at startup with AN1292 software and no mechanical load

As observed from Figure 3, motor current and velocity undergo transient disturbances during the startup phase. Figure 4 shows a plot of motor current and velocity during the startup phase with AN1292 software, MCLV2 board and Hurst motor (DMB0224C10002) with about 50% of nominal mechanical load.

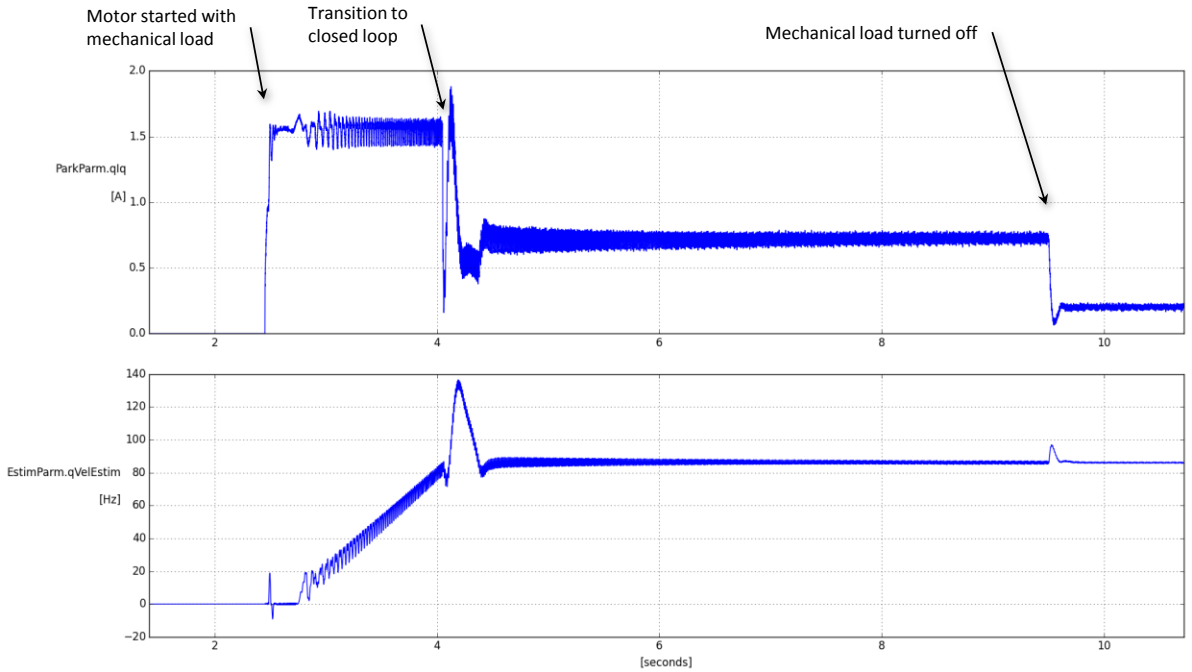

Figure 4: Plot of motor current and velocity at startup with AN1292 software and 50% of the nominal mechanical load

Comparing Figure 3 and Figure 4, we can observe that the magnitude of transient spike in motor current with 50% nominal motor load (Figure 4) increased by about 200% relative to the no-load condition (Figure 3).

1.2.5.2 How do we do it in MC Application Framework / motorBench™ Development Suite?

The MC Application Framework implements a newly-designed motor startup routine in order to:

1. Eliminate or reduce transient disturbances
2. Adapt to load changes during startup and
3. Improve the reliability of motor startup

See Motor startup section of the documentation for more details on the new motor startup algorithm.

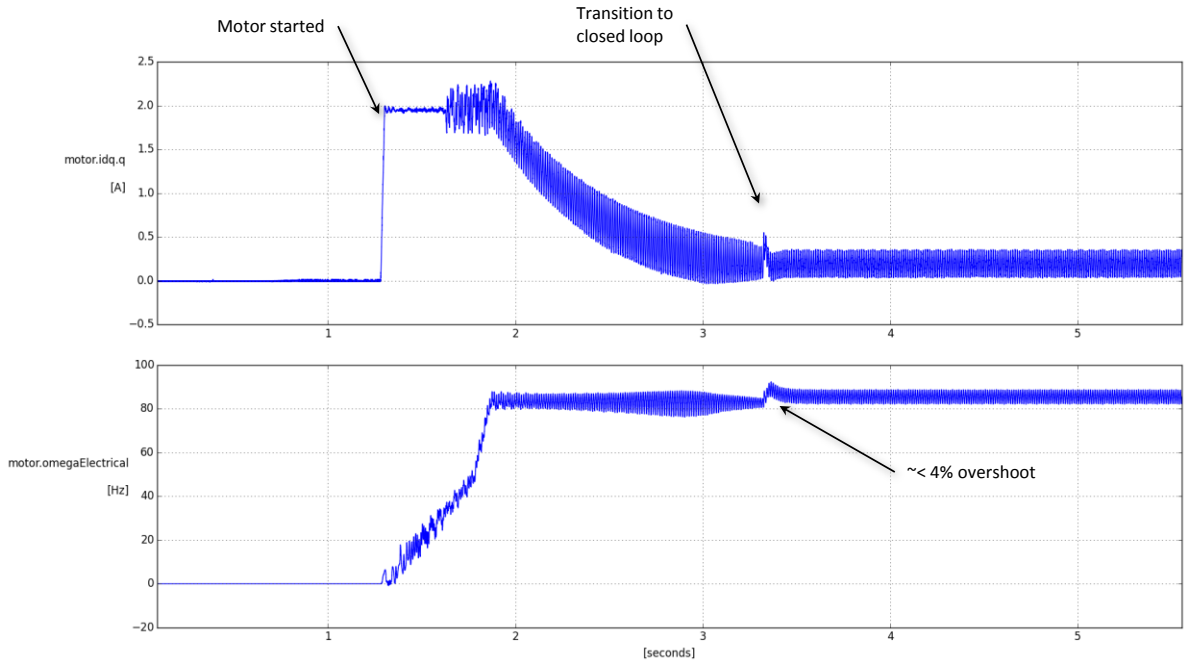

Figure 5: Plot of motor current and velocity at startup with MC Application Framework and no mechanical load

Figure 5 shows a plot of motor current and velocity during the startup phase with the MC Application Framework software, MCLV2 board and Hurst motor (DMB0224C10002) with no mechanical load attached to it. As observed from Figure 5, there is very little transient disturbance during and after motor startup.

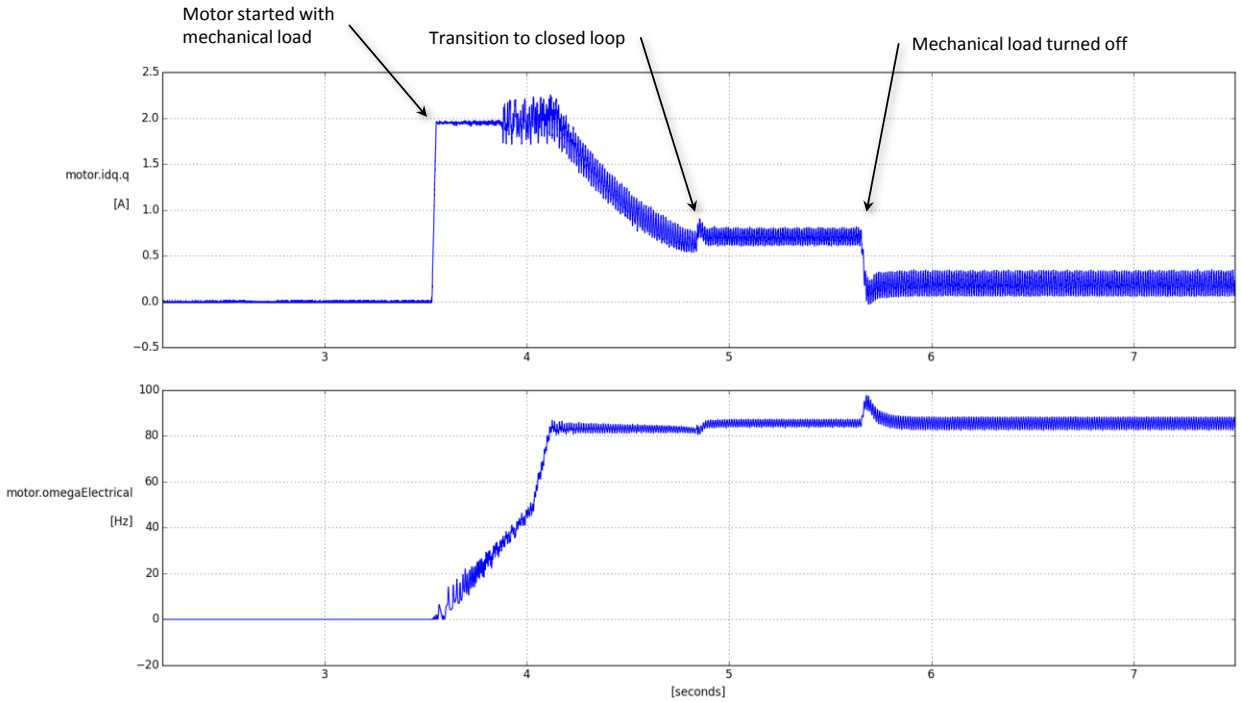

Figure 6: Plot of motor current and velocity at startup with MC Application Framework and 50% of nominal mechanical load

Figure 6 shows a plot of motor current and velocity during the startup phase with the MC Application Framework software, MCLV2 board and Hurst motor (DMB0224C10002) with about 50% nominal mechanical load.

1.2.5.3 Perceivable advantages of motorBench™ Development Suite approach

The newly-designed motor startup routine provides a smooth motor startup with minimal velocity overshoot / current transients. The startup routine parameters are automatically tuned for a specific motor and mechanical load combination.

1.2.5.4 Perceivable disadvantages of motorBench™ Development Suite approach

The new motor startup routine requires motorBench™ Development Suite in order to calculate the startup parameters. These startup parameters are based on mechanical parameters of the motor and that of the mechanical load that will be used during the motor startup. Changes in motor or load mechanical parameters will require retuning of the startup parameters using motorBench™ Development Suite.

1.2.6 Motor stall detection

Motor stall event can be defined as a condition where:

1. Estimated motor angle is no longer accurate enough to drive the motor synchronously and
2. As a result of this, the motor torque, flux and velocity control loops are no longer effective.

It is important for the control algorithm to detect motor stall conditions promptly and gracefully handle the after-effects of a motor stall. If the control algorithm fails to detect the motor stall condition, it could result in overcurrent and/or overvoltage faults which can potentially damage the hardware.

1.2.6.1 Stall detection logic in legacy AN1292 software

While software from older application notes such as AN1078 uses motor velocity as a primary source of information to detect motor stall condition, AN1292 does not implement this feature.

1.2.6.2 Stall detection logic in MC Application Framework

The MC Application Framework uses a newly designed Stall detection algorithm that uses the following vital control parameters in order to detect motor stall:

1. Variance of Back-EMF voltage from the PLL-based sensorless motor position estimator
2. Sign of the Back-EMF voltage from the PLL-based sensorless motor position estimator
3. Torque angle
4. Motor torque current value, i.e. I_Q
5. Motor velocity

Refer to the Motor Stall Detect Algorithm section of the documentation for more details.

1.2.6.3 Perceivable advantages

The new Stall detection algorithm used in MC Application Framework has been tested on multiple motors with and without mechanical loads to reliably detect motor stall condition. This feature provides a high degree of control reliability from the application perspective and also prevents potential damage to hardware that can result from an uncontrolled motor stall condition.

1.2.6.4 Perceivable disadvantages

In absence of any stall detect algorithm, motors operated using the legacy AN1292 software can, in certain circumstances, undergo 'cycle-slips' when loaded beyond their rated torque. When this happens, the rotor will stop spinning and vibrate with a large phase current. When the motor load is reduced at this point of time, the motor *may* automatically recover from this stall condition and continue spinning. Since this behavior is similar to an AC induction motor, it may appear more favorable to an untrained observer relative to the MC Application Framework.

1.2.7 Recovery from motor stall condition

Once a motor stall condition is detected, the control algorithm must initiate further steps to either:

1. Shutdown and then restart the motor, if required by the application or
2. Gracefully correct the stall condition and recover control of the motor.

In certain applications, it may be undesirable for the motor to automatically restart after a stall condition. Hence, the decision to recover from a motor stall condition and the actual recovery behavior are usually dependent on the desired logic, as dictated by the end application.

1.2.7.1 Recovery logic in legacy AN1292 software

The legacy AN1292 software does not support recovery from motor stall condition. However, as discussed in Section 1.2.6.4, there may be unintended cases where the motor automatically recovers from a stall.

1.2.7.2 Recovery logic in MC Application Framework

The MC Application Framework implements an automatic recovery logic that is designed to shut-down the motor, wait for a predetermined duration of time for the motor to coast down and then attempt to restart the motor. The recovery logic is pre-programmed to retry the recovery sequence a specified number of times in a row before flagging a fault condition. The number of recovery retry attempts can be changed by the application designer. If the application desires not to recover automatically from a motor stall condition, this number can also be set to '0'.

1.2.7.3 Perceivable advantages

The recovery logic designed into the MC Application Framework relieves the end-application from the intricacies of the recovery process from a motor stall condition, while still providing it with an abstracted access into the stall detection and recovery processes.

1.2.7.4 Perceivable disadvantages

[none]

1.2.8 External User Interface (UI)

User Interface (UI) refers to the firmware logic within an application that uses human-interface elements such as LEDs, switches, test-point trace output, etc. as a means to provide feedback to the user / application designer about the status of the application. Application note software should utilize external interfaces to help application designers in development and debugging phases. Valuable real-time information regarding status, faults and other events can be displayed via these interfaces so that the application designer can avoid using more time and resource intensive debug tools.

1.2.8.1 External UI support in legacy AN1292 software

At the time when the legacy AN1292 software was developed, UI elements like LEDs were not available on Microchip motor control development boards (MCHV, MCLV and MCSM boards). Hence, the only UI elements that the legacy AN1292 software supports are the potentiometer and the two push-buttons.

UI elements like LEDs and additional test points were made available starting from the second generation motor control development boards (MCHV-2, MCLV-2 and Dual motor control board).

1.2.8.2 External UI in MC Application Framework

The MC Application Framework utilizes the push-buttons, potentiometer and LEDs to provide a new UI interface that supports following features:

1. “heart-beat” indication ↔ firmware is active and running normally
2. Motor direction indication
3. Fault code display using a base-4 LED blink sequence
4. Fault handling/clear using push button interface

For more details on the External UI provided in the Application Interface, refer to the External Interfaces section of the documentation.

1.2.8.3 Perceivable advantages of External UI in MC Application Framework

While using the MC Application Framework, status of the firmware can be determined easily without requiring the application designer to use external debugging tools.

1.2.8.4 Perceivable disadvantages

[none]

1.2.9 Motor stopping technique

Motors are electro-mechanical systems that, by the virtue of their moment of inertia, can store kinetic energy in addition generating torque that drives the mechanical load. In addition to this, mechanical loads themselves can have substantial amounts of energy stored in their moment of inertia. Hence, while stopping a motor from its spinning state, the motor control application will have to dissipate this kinetic energy in one of the following ways:

1. The simplest, slowest and safest approach: turn off the control voltage and allow the motor to coast down at its own pace. In this case, energy is gradually dissipated as heat within the motor and load. Comparatively, this technique applies the least amount of stress on the motor control hardware during the motor turn-off phase.
2. The quick and dirty approach: re-circulate the motor phase currents back into the motor windings by shorting out either all of the low-side or all of the high-side inverter switches. This will quickly dissipate the kinetic energy into the motor windings and inverter switches as heat and into the stator laminations as eddy current losses. However, this can result in large amplitude motor currents that can potentially damage the motor control hardware and/or the motor itself.
3. Dynamic braking approach: apply a controlled PWM duty cycle on the low side inverter switches to act like a chopper circuit and pump the energy into the inverter power supply. If this is not carefully controlled, it can increase the inverter bus voltage and cause potential damage to the motor control hardware.
4. Controlled braking / regeneration approach: use the velocity control loop to reduce the motor velocity down to zero. This will also result in regeneration and an increase in the inverter bus voltage. Hence, the velocity ramp down rate must be controlled to a safe value depending on the total moment of inertia of the motor + load in order to prevent excessive stress on the motor control hardware due to increase in the inverter bus voltage.

1.2.9.1 Motor stopping technique in legacy AN1292 software

In order to turn off the motor, the legacy AN1292 software re-circulates the motor phase currents back into the motor windings by shorting-out all of the low-side inverter switches (i.e. technique #2 discussed in Section 1.2.9). In applications where the motor and load have a large moment of inertia (ex: washing machine), turning off the motor using this technique can result in large motor phase currents, which, in turn leads to excessive stress on the motor control hardware and the motor itself.

1.2.9.2 How do we do it in MC Application Framework / motorBench™ Development Suite?

In order to turn off the motor, the MC Application Framework turns-off the control voltage and allows the motor to coast down at its own pace (i.e. similar to technique #1 discussed in Section 1.2.9). In addition to this, a small duty cycle is maintained on all three low-side PWM switches to keep the bootstrap capacitors charged up.

1.2.9.3 Perceivable advantages of the MC Application Framework approach

This approach applies least amount of stress on the motor control hardware.

Figure 7 shows the comparative plots of motor phase currents when the motor was turned-off with legacy AN1292 software and with MC Application Framework. The test setup used this case included an MCLV2 board, Hurst motor - DMB0224C10002 with an inertia load of 39.01µNms^2, V_{BUS} = 24V, PWM dead time = 2 µs, PWM frequency = 20 kHz. As observed from Figure 7, the AN1292 software causes a large current surge when the motor is turned-off while the MC Application Framework does not result in any motor phase currents under similar test conditions.

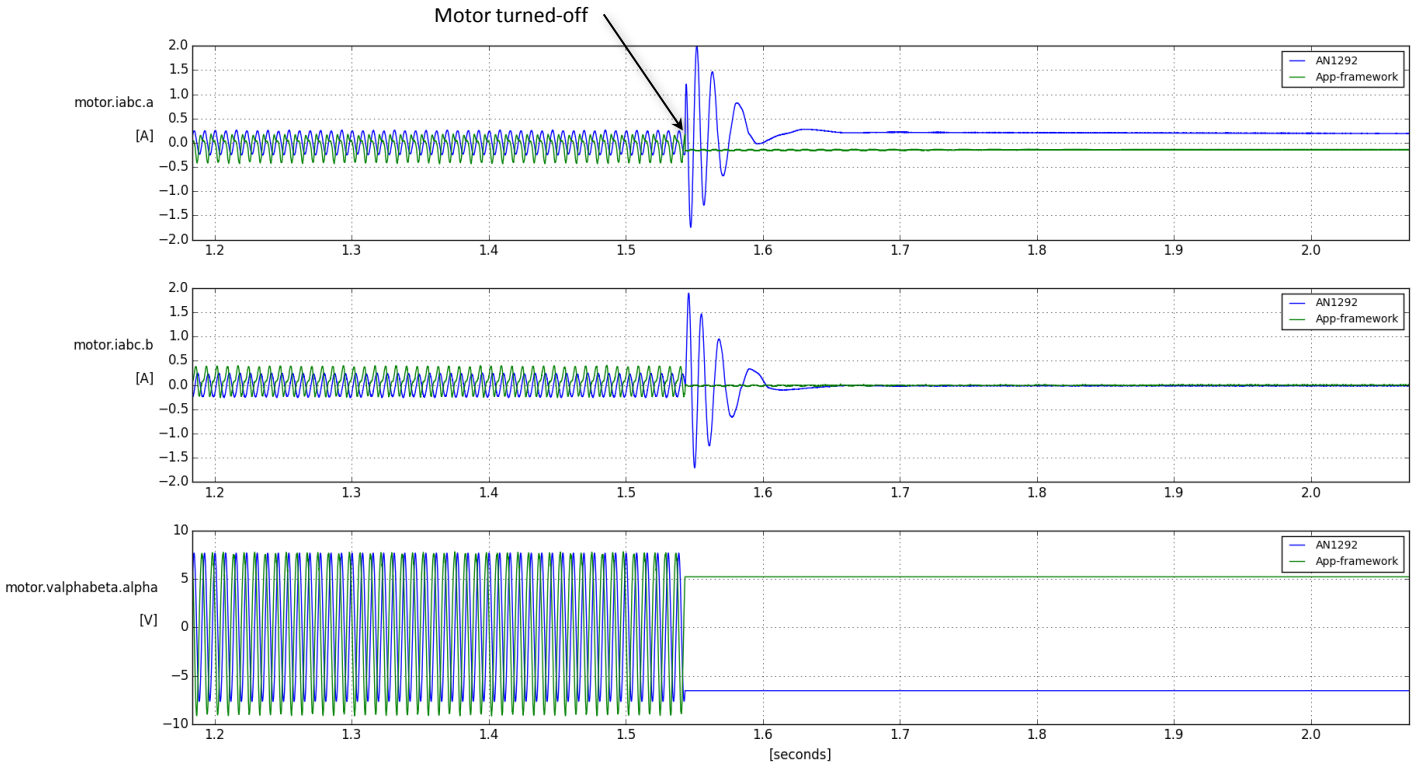

Figure 7: Comparative plots of motor phase currents when the motor is turned-off

1.2.9.4 Perceivable disadvantages of the MC Application Framework approach

Compared to other motor stopping techniques, this approach does require a longer time frame to bring the motor to a complete stop.

1.2.10 Modular code architecture

Modular code architecture refers to software architecture where the source code is well organized into independent modules, each with a specific purpose. It is important for Application note software to be modular for the following important reasons:

1. End-users of Application note software typically modify it before using it in their end-application. Modular software allows end-users to easily identify and modify individual modules of the software rather than searching for specific sections of the source code
2. Modular source code is easy to maintain and improves readability for end-users
3. End-users can easily run unit tests on any modifications that they make

1.2.10.1 Code architecture in legacy AN1292 software

1.2.10.1.1 Data structure and data access

The legacy AN1292 software is designed to have a set of eight core-data structures that hosts all of the data variables required by the application note and a set of application note specific functions that implicitly use these structures as global variables, as shown in Figure 8. Some of these functions are written in assembly language and these assembly functions access the core-data structures using ASM30 API included within their corresponding *.inc files.

1.2.10.1.2 Hardware access

In line with the idea of modular coding architecture, initialization routines for all dsPIC functions are contained in a single source file, i.e. ***periph.c***. However, many of the application note functions in the

rest of the software project access dsPIC special function registers directly and hence, these functions have hardware dependency by design.

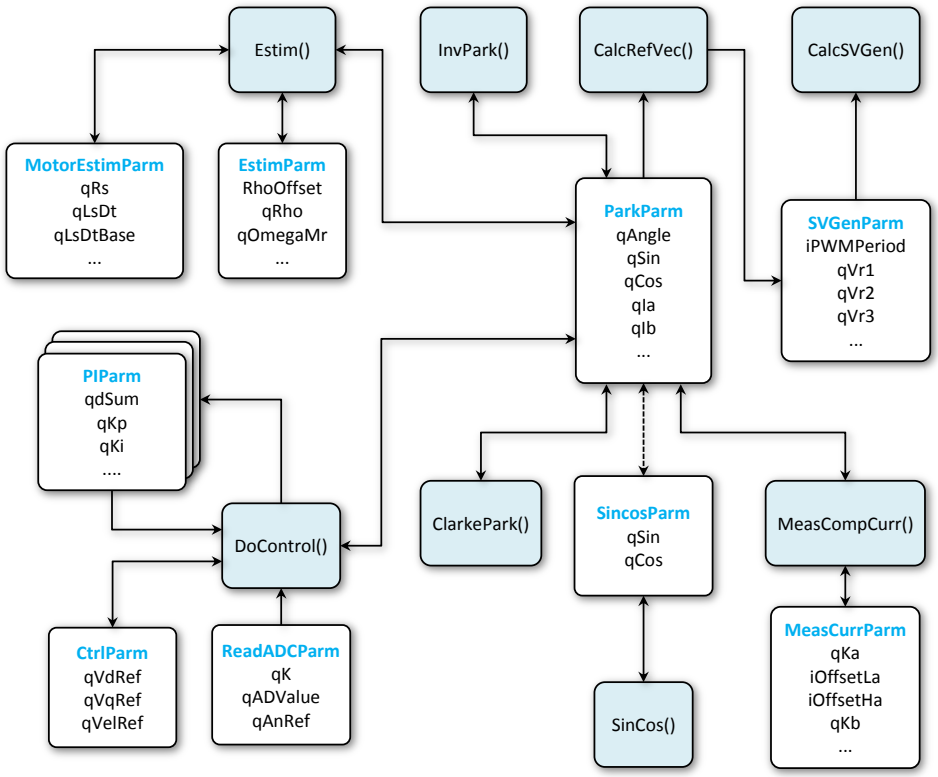

Figure 8: Architecture of core-data structure and access in AN1292 software

1.2.10.1.3 Source code and file organization

In addition to the main(), the ***pmsm.c*** file hosts CalculateParkAngle(), DoControl(), InitControlParameters and the ADC ISR. Rest of the application note functions are hosted by individual files.

1.2.10.1.4 State logic handling for User Interface

The control logic that handles user interface actions such as turning the motor on/off and doubling the motor speed are implemented using software bit-flags and controlled within blocking while-loops within the main().

1.2.10.1.5 Adding/Removing/Disabling modules

Legacy AN1292 software is not modular enough to support adding/removing/disabling core modules like the estimator, startup routine and SVM routine without breaking the code.

1.2.10.2 Code architecture in MC Application Framework

For more details on the architecture of MC Application Framework, refer to Section 3 - Architecture.

1.2.10.2.1 Data structure and data access

The MC Application Framework uses a hierarchical approach to storing data in structures. It uses smaller individual data structures that are dedicated to host variables specific to a function. For example,

- `idqCmd` holds variables that provide reference values for I_D and I_Q
- `vabc` holds variables that provide the desired phase voltages V_A, V_B and V_C
- `estimator` holds variables related to the position and velocity estimator routine.

Individual data structures that are related to the control and functioning of a particular motor are grouped under one higher-level structure. As shown in Figure 9 for a single-motor system, this higher-level structure is the `motor` structure. Similarly, all of the smaller individual structures that are used for system-level tasks are maintained under one `systemData` structure.

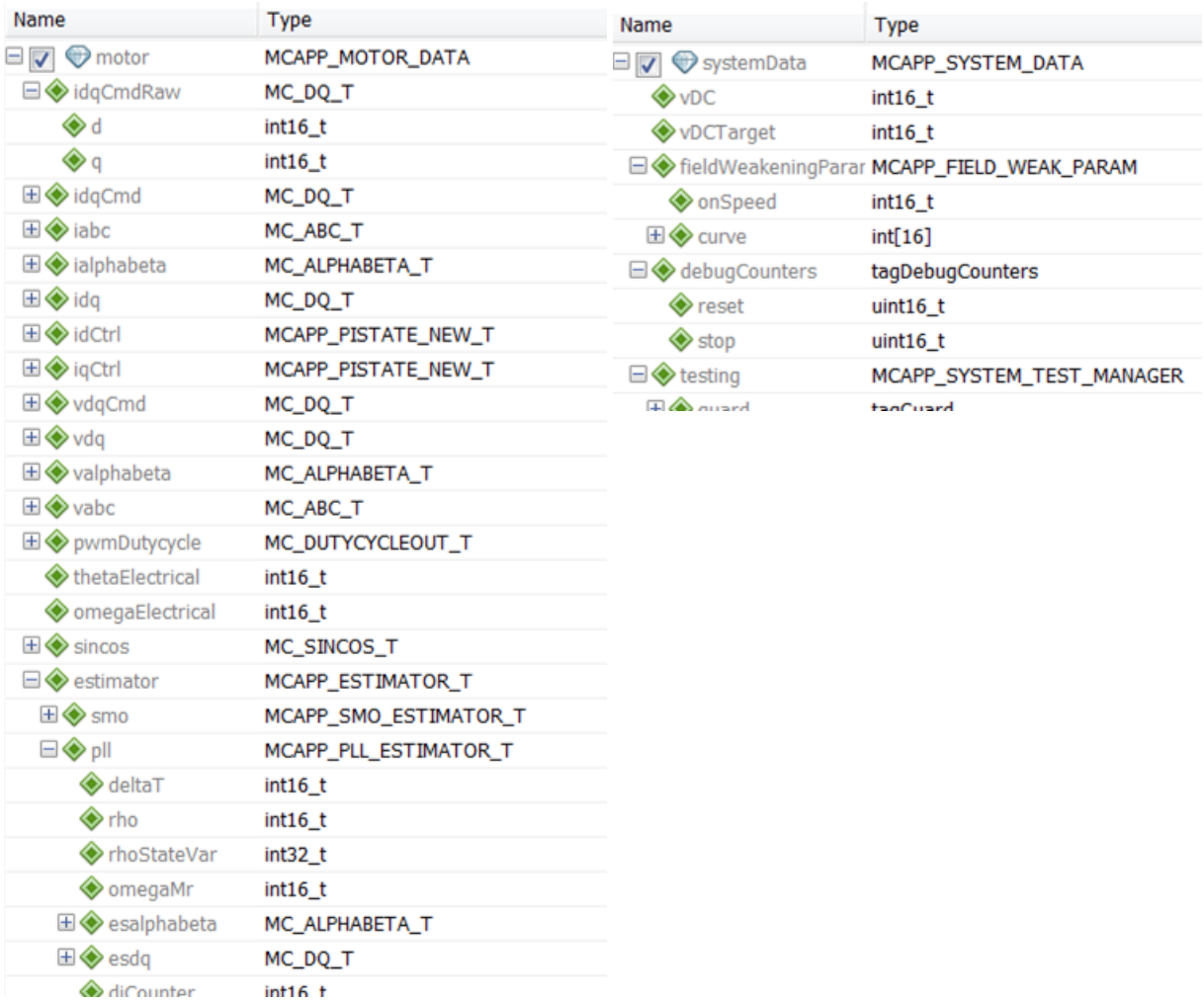

Figure 9: Hierarchy of data structure in the MC Application Framework

The details that are shown in Figure 9 are for representation purposes only and are subject to change. Refer to the latest software documentation for more accurate details on the variable names and organization of the data structures.

1.2.10.2.2 Hardware access

All access to hardware, i.e. dsPIC SFRs, in the MC Application Framework is channeled through the Hardware Abstraction Layer (HAL).

1.2.10.2.3 Source code and file organization

MC Application Framework uses individual modules (= one *.c file + one *.h file) to host functions and their related support elements like constants, type definitions and function declarations.

1.2.10.2.4 State logic handling for User Interface

MC Application Framework uses state machine approach to implement User Interface related logic.

1.2.10.2.5 Adding/Removing/Disabling modules

MC Application Framework is designed to allow most of its individual modules/features to be disabled or replaced with other variants, provided that they share the same interface. For instance, the MC Application Framework supports multiple variants of estimators – each of which can be individually selected, disabled or replaced with another variant.

1.2.10.3 Perceivable advantages of the MC Application Framework

Compared to the legacy AN1292, MC Application Framework has following advantages

1. MC Application Framework is easier to read and understand, due to the following features:
    a. Standardized coding and code commenting guidelines
    b. Highly modular architecture
    c. Consistent function/variables/constants naming system
2. MC Application Framework is easier for end-users to modify and maintain
3. Hardware Abstraction Layer allows for easy porting to customer hardware

1.2.10.4 Perceivable disadvantages of the MC Application Framework

MC Application Framework introduces a software architecture to the end-users that is different from the legacy AN1292 software. This change will require existing end-users who are already familiar with AN1292 software to go through a learning-curve, which will require some additional time and effort initially.

1.2.11 Parameter calculations

Application note source code projects are typically made up of two important parts:

1. Actual source code that implements the motor control algorithm and
2. Algorithm configuration parameters defined as constants in header files, which help the algorithm source code to adapt according to the specific system conditions set by the user.

For instance, source code that implements the motor startup algorithm needs to know the following parameters:

1. Number of poles of the motor
2. Motor lock time
3. Startup end velocity
4. Motor startup nominal current

These parameters are typically initialized at software reset using constants from a header file. These parameters constants may potentially be derived from a set of calculations. In this case, there are two options:

1. Perform parameter calculations ahead of time and directly assign the final integer numbers to the corresponding parameters constants, or
    Ex: `#define KIP 5584 // Q15(0.17041)`
2. Use the C preprocessor to perform the parameter calculations at compile-time
    Ex: `#define KIP Q15(0.17041) // Q15() is another pre-processor macro`

1.2.11.1 How does the legacy AN1292 software handle parameter calculations?

Legacy AN1292 software uses the C preprocessor to perform most of the parameter calculations in the ***userparams.h*** file.

1.2.11.2 How do the MC Application Framework / motorBench™ Development Suite handle parameter calculations?

The motorBench™ Development Suite calculates the parameter values and uses a templating engine to render parameter constants with final integer values into various header files within the ***parameters*** folder in the MC Application Framework. In addition to this, it also includes the real-world values as line-comments.

1.2.11.3 Advantages of taking the motorBench™ Development Suite / MC Application Framework approach

Since valid values of the parameters are directly rendered into various parameter header files by motorBench™ Development Suite, there is no risk of integer overflow or improper cast errors from occurring at compile time.

1.2.11.4 Disadvantages of taking the motorBench™ Development Suite / MC Application Framework approach

In some cases, it may be hard for the end-users to determine how the final numbers for the parameters constants were derived within the motorBench™ Development Suite. The line-comments included in the template header file with actual calculations should help alleviate this concern to a large extent.

1.2.12 Unified source code base

A majority of the source code that we currently have on the web for the Field Oriented Control (FOC) based application notes use source code components that are common to each other. Even the non-FOC based application notes share some of their source code components with the FOC based application notes. This commonality allows us, in principle, to have a *unified source code base* from which each variant of an application note can derive its source code.

1.2.12.1 Legacy application note software

Our legacy application note software has one copy of source code project for each hardware combination of PIM and motor control board, as shown in Figure 10.

AN1292

Title: **Sensorless Field Oriented Control (FOC) for a Permanent Magnet Synchronous Motor (PMSM)Using a PLL Estimator and Field Weakening (FW)**

Name: AN1292

Date: 06/30/2011

Author: Mihai Cheles

Description: This application note describes a sensorless FOC algorithm for PMSM motors using a Phase-Locked Loop (PLL) position and speed estimator.

Keywords: Sensorless, FOC, dsPIC, DSC, Digital Signal Control, DSP, 16-bit, Sensorless, BLDC, PMSM, Brushless, PLL, Estimator, Vector Control, FW, Field Weakening, Flux Weakening

Application Notes & Source Code	Last Updated	Size	
AN1292	06/30/2011	440KB	
AN1292 Source Code for DV330100 using dsPIC33EP512GM710	12/22/2014	2376KB	
AN1292 Source Code for dsPIC33EP512MU810 and dsPICDEM MCHV	11/07/2012	1978KB	
AN1292 Source Code for dsPIC33EP512MU810 and dsPICDEM MCLV	11/07/2012	1584KB	
AN1292 Source Code for MCLV-2 using dsPIC33EP256MC506 External OpAmp PIM	08/29/2012	1479KB	
AN1292 Source Code for MCLV-2 using dsPIC33EP256MC506 Internal OpAmp PIM	08/29/2012	1569KB	
AN1292 Source Code for MCHV-2 using dsPIC33EP256MC506 External OpAmp PIM	08/29/2012	1496KB	
AN1292 Source Code for MCHV-2 using dsPIC33EP256MC506 Internal OpAmp PIM	08/29/2012	1684KB	
AN1292 Source Code for DV330100 using dsPIC33EV256GM106	02/12/2015	2298KB	
AN1292 Source Code for dsPIC33FJ32MC204 and dsPICDEM MCLV	12/15/2009	835KB	
AN1292 Source Code for dsPIC33EP64MC504 and dsPICDEM MCLV	11/10/2011	1505KB	

Figure 10: Application note webpage for AN1292 with multiple copies of source code projects

1.2.12.2 Unified code base approach in MC Application Framework / motorBench™ Development Suite

The motorBench™ Development Suite maintains a single copy of all of the source code modules that it needs to support a set of application notes and software features. It then renders the final system-specific code based on a specific selection of modules from the unified code base, combined with the Hardware Abstraction Layer (HAL) files and a parameter header file that is generated by templating process. Figure 11 shows two examples of this rendering process.

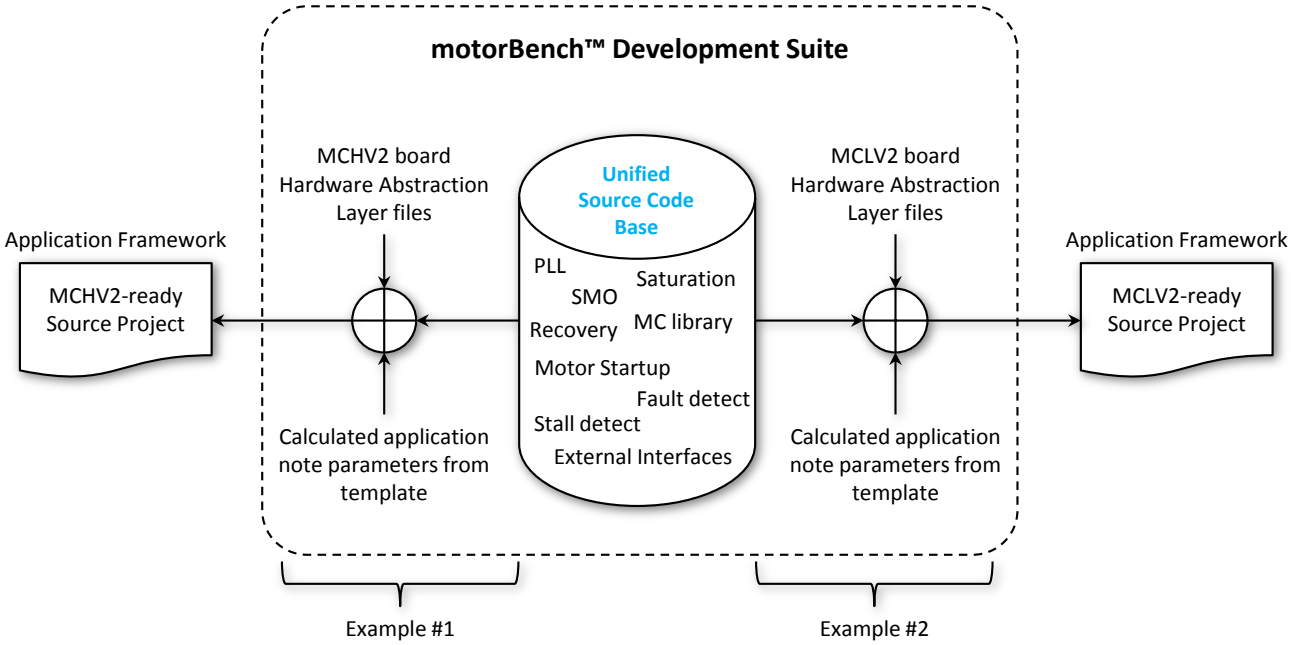

Figure 11: Examples of system-specific code rendering by motorBench™ Development Suite

1.2.12.3 Perceivable advantages of unified code base approach

The unified code base approach allows for efficient maintenance of the source code. Bug fixes to the unified code base automatically propagate to all variants of the rendered output.

1.2.12.4 Perceivable disadvantages of unified code base approach

End-users will have to use the motorBench™ Development Suite (or an alternate templating script) to work with the unified code base approach.

1.2.13 Diagnostics and testing interfaces

Ideally, application note software must provide debug interfaces for the application developer to probe/write variables, check the system status and log variables in real-time without requiring time and resource intensive debug tools.

1.2.13.1 Diagnostics and testing interfaces in legacy AN1292 software

Legacy AN1292 software provides the following interface features to help the application developer with testing, debugging and diagnostics:

1. RTDM/DMCI for reading and writing variables, plotting up to four variables with limited data throughput
2. Basic test modes to help with the tuning process:
    a. Software ramp – use macro `TUNING` in ***userparams.h*** file
    b. Open loop mode – use macro `OPEN_LOOP_FUNCTIONING` in ***userparams.h*** file
    c. Torque control mode – use macro `TORQUE_MODE` in ***userparams.h*** file

1.2.13.2 Diagnostics and testing interfaces in MC Application Framework

MC Application Framework provides the following interface features to help the application developer with testing, debugging and diagnostics:

1. User Interfaces with LEDs and buttons to convey information about the status and faults

2. X2C-Scope from the Linz Center of Mechatronics GmbH
3. Software test modes with test guard protection; supports
    a. Multiple operating modes – can be used for accretive testing one stage at a time
    b. Manual commutation frequency override
    c. Square-ware perturbation to either velocity, current or voltage commands
    d. Input command override with a fixed velocity command

Refer to the MC Application Framework documentation - External Interfaces section for more details.

1.2.13.3 Perceivable advantages

MC Application Framework provides more features to the application developer for diagnostics and testing purposes.

1.2.13.4 Perceivable disadvantages

Some of the diagnostics and testing features supported by MC Application Framework requires a dedicated PC software tool that has not been published to the public yet.

1.2.14 Saturation handling

All real-world motor control systems have limitations on the maximum/minimum voltage and current that they can sustain. Due to this, application notes have limits on the maximum/minimum output values of the velocity and current controllers. Current saturation occurs in cases where the reference and measured values of velocity cause an output current that is beyond the valid range for a given system. Similarly, voltage saturation occurs in cases where the reference and measured values of current cause an output voltage that is beyond the valid range for a given system.

1.2.14.1 Saturation handling in legacy AN1292 software

In legacy AN1292 software, the velocity controller is set to saturate at 0x5000 (see macro SPEEDCNTR_OUTMAX in ***userparams.h*** file), the current reference for I_Q is limited to I_{Qsat} as calculated by the Equation (1).

$$I_{Qsat} = 0x5000 \times \frac{I_{peak}}{2^{15}} \quad \text{..... (2)}$$

Where,
I_{peak} is the peak current that can be measured by the ADC

For the MCLV2 board with a peak current capability of 4.4A, maximum value of motor current, I_{Qsat}, based on Equation (2) comes out to be about $2.75A_{peak} = 1.94A_{RMS}$.

The current controller is set to saturate at the maximum Q15 value, i.e. 32767. This corresponds to the maximum voltage available on the inverter.

Once in saturation, both the velocity and current controllers have their excess gains (Kc) set to the maximum. Thus, the integral term will be immediately zeroed out while coming out of saturation.

Additionally, there is no communication between the two controllers (i.e. velocity and current) to behave differently if one or both of them enter into saturation.

1.2.14.2 Saturation handling in MC Application Framework

MC Application Framework has a more sophisticated saturation handling system with the support for following features:

1. Current and velocity controllers communicate with each other and react differently when one or both of them are in saturation
2. Current and velocity controllers use a saturation window rather than a single threshold value
3. Saturation window limits used in the current and velocity controllers are derived based on the motor, board and power supply specifications.

Refer to the documentation Section 1.0 – Saturation for more details.

1.2.14.3 Perceivable advantages of MC Application Framework approach

Using the cooperative approach with the PI controllers for velocity and current control loops helps in maintaining the motor angle estimate with a safe operating boundary, which in turn results in better efficiency.

Correlating the saturation window limits to the motor, board and power supply specifications prevents hardware damage due to overload/overheating.

1.2.14.4 Perceivable disadvantages

Updating the PI controller saturation limits for a new motor/board requires the use of motorBench™ Development Suite.

1.3 Revision History

Version	Date	Author	Description
1	24/Feb/2017	Srikar Deshmukh - C14317	First revision of the document