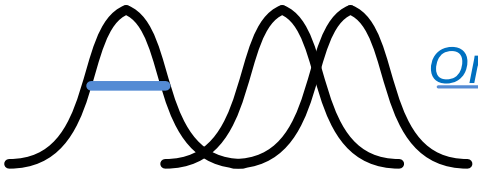


Photogrammetry for Non-Invasive Terrestrial Position/Velocity Measurement of High-Flying Aircraft

Part 7: Basic Telescope Mount Functionality

James A Crawford

Synopsis

This installment marks completion of the mechanical design and fabrication of the mount. From this point on, attention will be focused entirely on software development, both for the C# - based GUI as well as the TMS320F28379D software in C.

Regarding the mechanical motion, the original cogging torque associated with the azimuth axis caused me sufficient concern that I decided to redesign and refabricate the thrust bearing portion of that axis. The associated details are discussed herein.

While the original elevation axis servo design proved fully functional, the small 3-phase motor overheated (in my opinion) under continuous operation so I embedded a belt-drive stage with a 5:1 reduction ratio immediately following the motor to lessen the work-load presented to the motor. With the 80:1 reduction ratio of the harmonic drive, this resulted in an overall "gear ratio" of 400:1 for this axis. The associated details are also described herein.

The balance of this memo recaps the electronic interfaces involved and provides an update as to the software development with future short-term plans.

1 Wrapping Up the Mechanical

1.1 Azimuth

In hindsight, the original azimuth axis design left a lot to be desired. The main concern I had looking forward was the variable/non-uniform viscosity of this axis when it was rotated, and the associated control system difficulties this could present later. I consequently redesigned the thrust bearing portion of this axis using a high-quality thrust bearing. The end-results were remarkably improved.

The first step in updating the azimuth axis hardware was fabrication of a new top-plate for the large motor housing as shown in Figure 1. With this, a new mount for a high-quality thrust bearing had to be made as shown in Figure 2, with the new large-diameter thrust bearing added in as shown in Figure 3. It is difficult to see the thrust bearing in Figure 3 because it is a purposely close-fit and it snuggles down into the roughly half-inch groove unobtrusively.

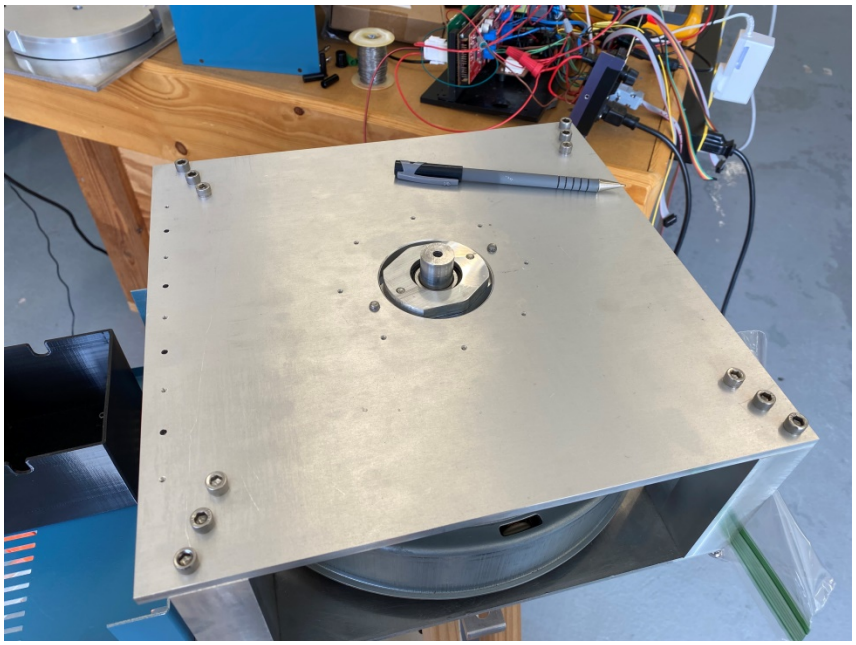


Figure 1 Redesign and fabrication of the azimuth axis top-plate. Extra holes (and length) were included along the left-side edge to support the electronics chassis.

The circular disk which supports the elevation yoke also had to be redesigned and fabricated. Although it looks almost identical to the original disk, the design features on its bottom side had to be completely redone in order to precisely match up with the new thrust bearing arrangement. This new disk is shown in Figure 4 in its final mounted position.

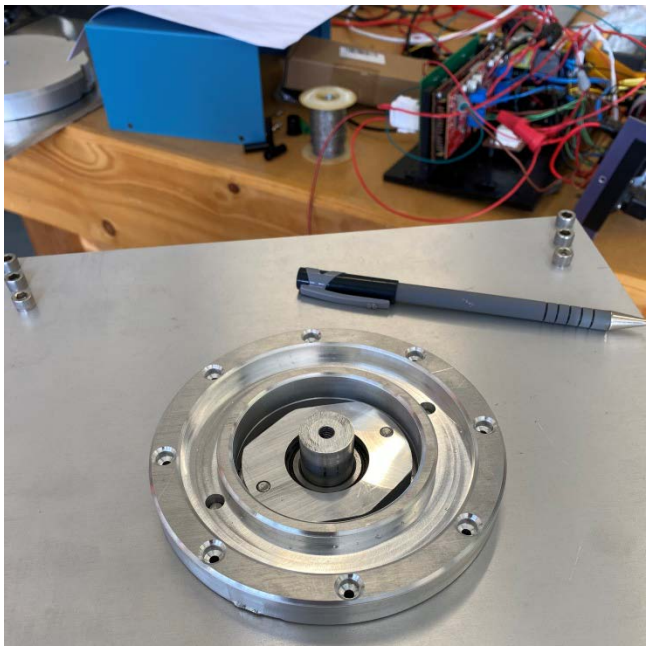


Figure 2 Azimuth plate top-side thrust bearing housing. The new thrust bearing reside is the large groove.

Figure 3 Repeat of Figure 2 but with the large diameter thrust bearing now dropped into the large groove

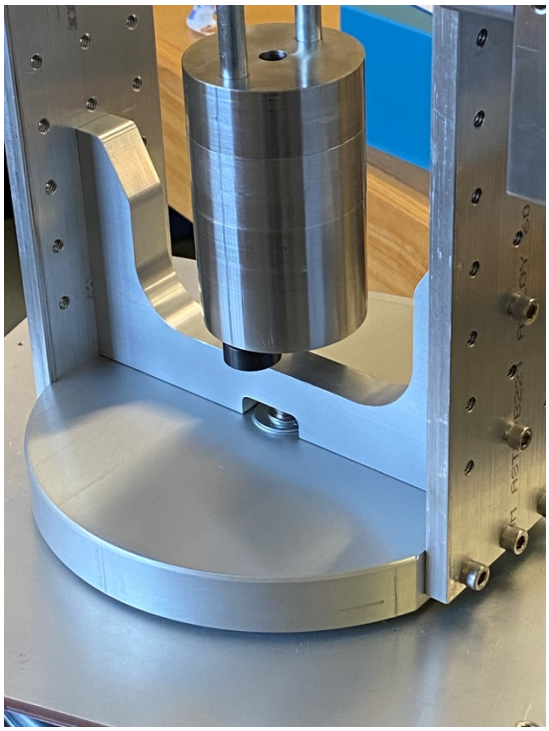


Figure 4 New circular elevation axis disk with original yoke elements attached

2 Elevation Axis Revisited

The elevation motor axis at the conclusion of Part VI is shown here in Figure 5. The heart of this drive is the 80:1 harmonic drive mounted inside the square aluminum housing. As already mentioned, the drive performed its duties as intended, but became fairly hot with continuous operation. This presented me some concerns for operational situations where I might want to continuously scan the open sky.

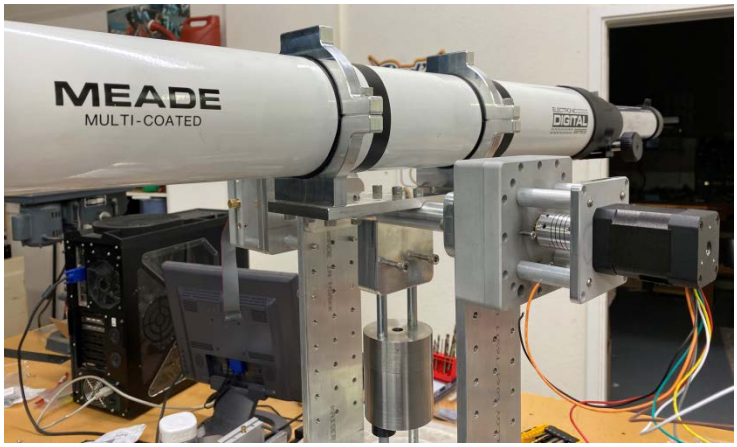


Figure 5 Elevation motor drive at the conclusion of Part VI of the project

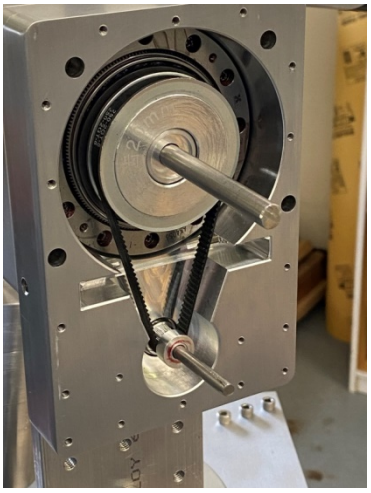


Figure 6 New 5:1 pulley drive added into the elevation axis drive immediately following the motor (shown partially completed here). The large pulley has 80 “teeth” and the small pulley has 16.

In order to circumvent this problem, I redesigned the elevation axis to incorporate an additional 5:1 pulley-drive step-down as shown in Figure 6. A number of new precision parts had to be fabricated in order to add in this pulley-drive stage. A pulley-drive approach was chosen in order to introduce as little backlash as possible.

First of all, a new mating-hub had to be fabricated for the harmonic drive as shown in Figure 7. The driving factor here was that the axle had to be considerably longer than in the original design, but the new axle was also fabricated from a steel rod for greater toughness as compared to the original aluminum axle. The new hub is shown attached to the wave generator portion of the harmonic drive in Figure 8.

The steel axle is shown secured into the wave generator hub in Figure 9 with the bottom portion of a thrust bearing added in Figure 10, and a bored-out GT2 pulley added as shown in Figure 11. The entire wave generator, new axle, and new pulley were then assembled into the flex-spline of the harmonic drive as shown in Figure 12. Assembly of the elevation drive is continued in Figure 13, Figure 6, Figure 14, and Figure 15.

Figure 7 New hub for the harmonic drive. Set-screws clearly shown.

Figure 8 New mating hub for the harmonic drive attached to the harmonic drive wave generator

Figure 9 Harmonic drive hub with new (removable) longer axle

Figure 10 Same as Figure 9 but the lower half of a new thrust bearing has also been slipped into place near the bottom of the axle

Figure 11 A GT2 pulley was milled down and center-bore increased to precisely mate with the harmonic drive hub. Note the set-screw access situated directly in the pulley grooves.

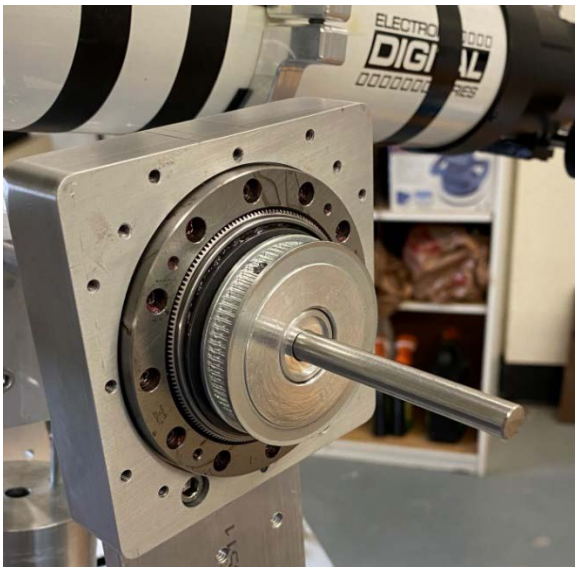


Figure 12 Pulley assembly of Figure 11 reassembled into the original harmonic drive

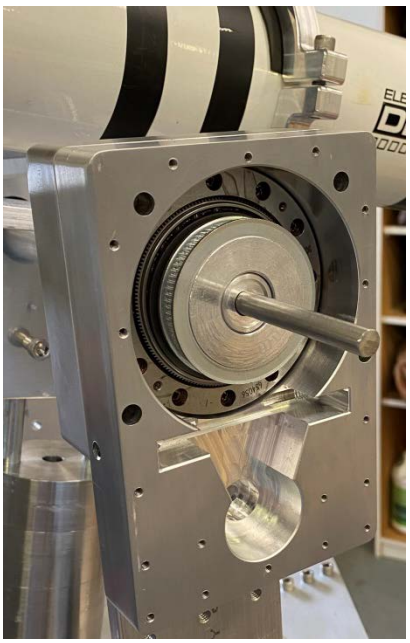


Figure 13 New pulley drive chassis case secured into the assembly of Figure 12

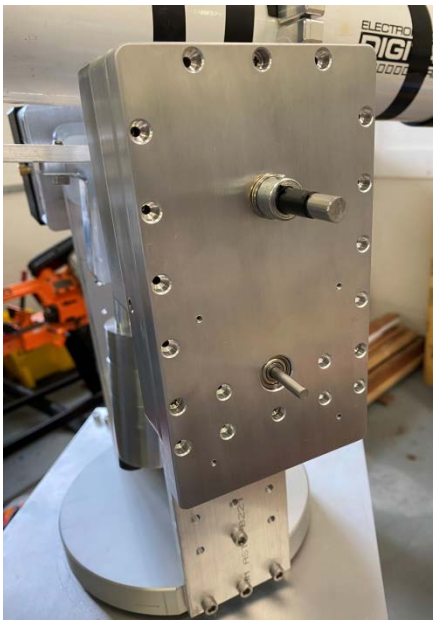


Figure 14 New harmonic drive cover plate plus bearings added for the two axes (harmonic drive and motor)

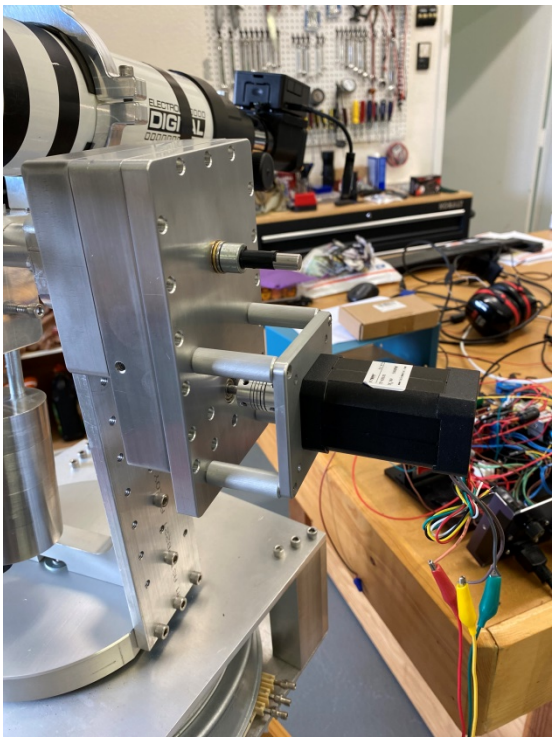


Figure 15 Fully assembled elevation axis drive motor. With the harmonic drive additions, the completed harmonic drive chassis is about 1” thicker than the original chassis shown in Figure 5.

3 Big Picture

A high-level overview of the electronics was provided earlier in [1] and is repeated here in Figure 16. This diagram does not, however, provide any clues about how the signal processing is to be partitioned between the TMS320F28379D's two CPU/CLA[1] combos.

In order to incur the smallest amount of 3-phase motor control jitter possible, it is crucial that the ADC and ePWM activities for each axis be completely synchronous and without any execution-related time-jitter. To this end, the following functional allocation is being adopted as given in Table 1.

Table 1 Function Partitioning Across TMS320F28379D

Function	CPU1	CLA1	CPU2	CLA2
PC-DSP Communications	X			
Azimuth Non-Critical Time Calculations	X			
Azimuth Encoder and ADC Operations		X		
Azimuth ePWM Operations		X		
Azimuth Control Law Real-Time Calculations		X		
Elevation Non-Critical Time Calculations	X			
Elevation Encoder and ADC Operations				X
Elevation ePWM Operations				X
Elevation Control Law Real-Time Calculations				X

The planned update rate for the sampled control systems is to be 16 ksps which provides about 12,500 CPU cycles (at 200 MHz) between control loop samples. Performing real-time synchronous functions (e.g., reading encoders, launching ePWM cycles) through the CLAs guarantees synchronous operation, a guarantee which could be tedious to implement in the CPUs.

Since one SPI port must be shared by the two optical encoders, both Azimuth and Elevation encoders need to be read using the same CLA in order to avoid timing conflicts. Two possible resource allocations come to mind as sketched out in Figure 17.

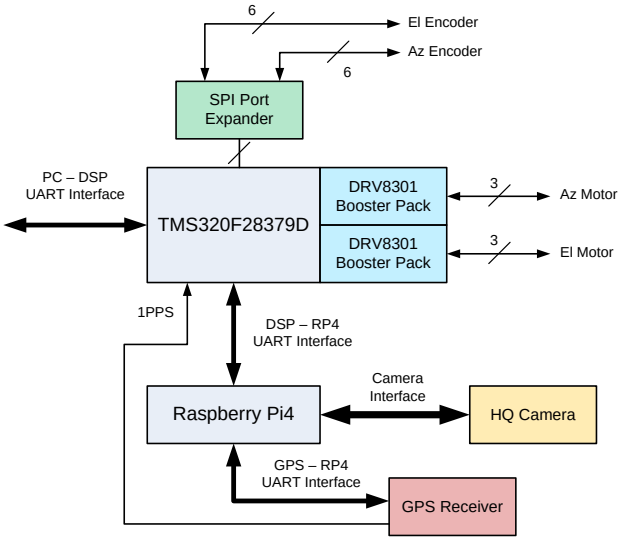


Figure 16 Multi-processor approach going forward to accommodate (i) Az & El 3-phase motor control, (ii) Az & El optical encoders, and (iii) Raspberry Pi4 to host an HQ image sensor and GPS receiver[2]

[1] CLA is an acronym for the built-in *control law accelerator* hardware module. Each CPU has one dedicated CLA available.

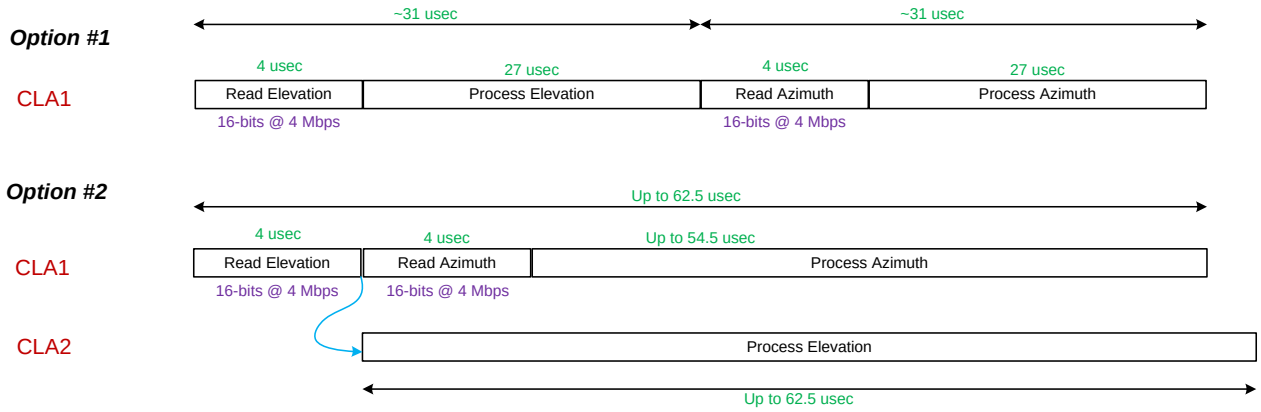


Figure 17 Single CPU/CLA combo versus dual CPU/CLA combo alternatives[3]

In some respects, there is no real reason not to use the full computational horsepower of the F28379D DSP in Figure 17 as it provides up to 117 μs of total CLA processing time versus about 54 μs if only one CLA is used, a ratio of 2.17 : 1. Going with Option #2 all but guarantees enough computational horse power will be available while also doubling the total number of CLA Tasks which can be partitioned out. CPU1 will be used to handle all PC/DSP communications whereas CPU2 remains available for future growth if needed. This partitioning keeps inter-processor communications at a minimum because Azimuth and Elevation are handled in separate CLAs. The 117 μs represents about 23,400 available machine cycles (at 200 MHz) which should provide more than adequate computational capability.

- Regardless, using CPU1/CLA1 *and* CPU2/CLA2 would entail having to develop CPU1↔CPU2 communications thereby involving several additional software development efforts: (i) CPU1 ↔ CPU2, (ii) CPU2 code, and (iii) CLA2 code.
- Since the nominal rotation rate of either axis uses rotation rates very near zero, using a full implementation of Figure 27 is not necessary.
- Given this previous point, phase-only control loops should be completely adequate. This does not minimize supply current magnitudes, but this is not a driving factor at this factor time.
- Note- monitoring i_d and i_q would likely shed light on torque *stickiness* if it is present. This capability will no doubt be revisited.

The GUI has undergone a complete make-over for this phase of the project as shown in Table 2.

[2] U27771_Part6_Figures.vsd.
[3] U28405 Photogrammetry Part 7 Figures.vsd.

Table 2 Screenshots from TeleTwo C# GUI. The TeleTwo software package is my first entrance into real telescope motion control

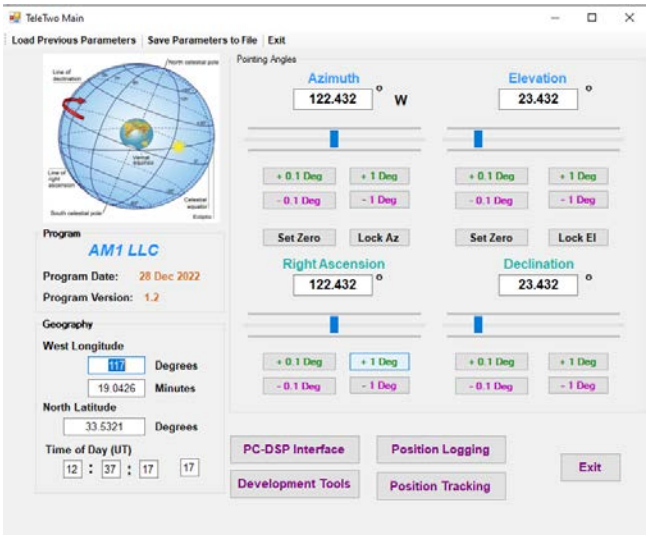


Figure 18 Main screen

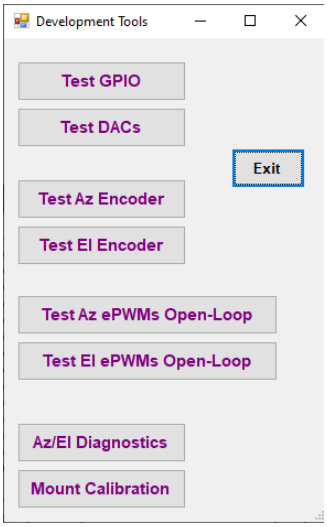


Figure 19 Development tools menu

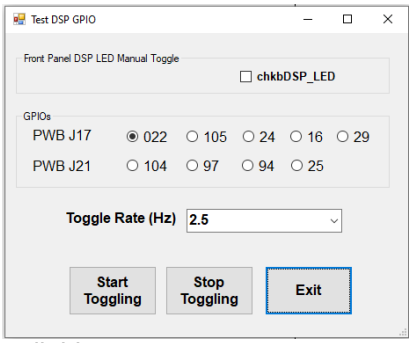


Figure 20 Test available DSP GPIO

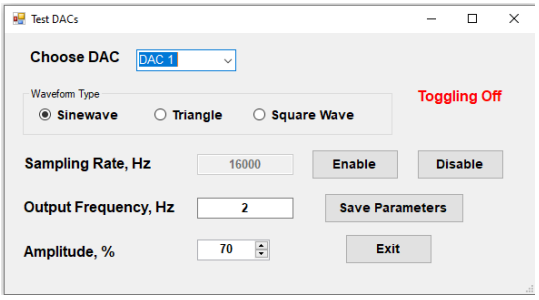


Figure 21 Test available DAC outputs

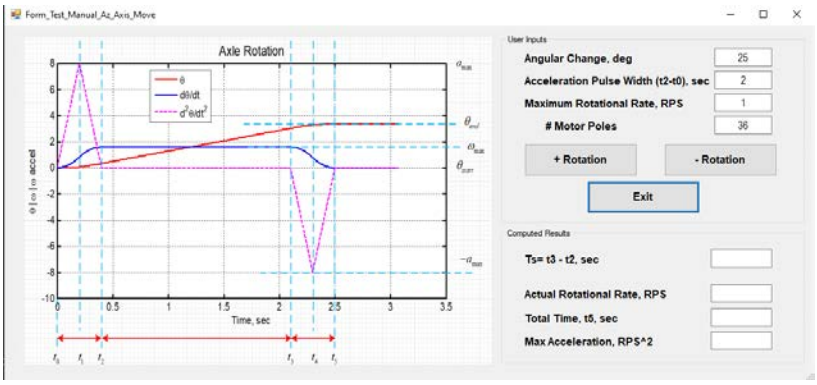


Figure 22 Set Azimuth axis acceleration/deceleration/radian velocity limits. Identical screen for Elevation axis.

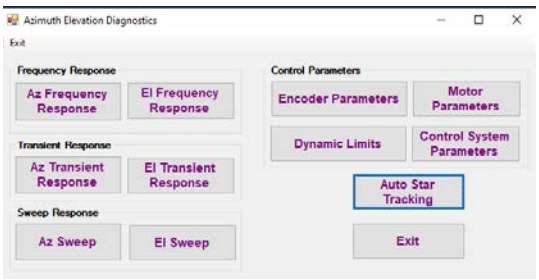


Figure 23 Closed-loop Azimuth/Elevation assessment

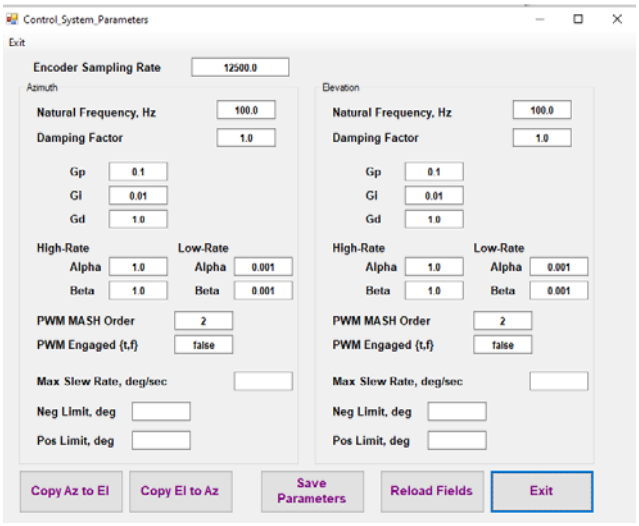


Figure 24 Detailed control system parameters

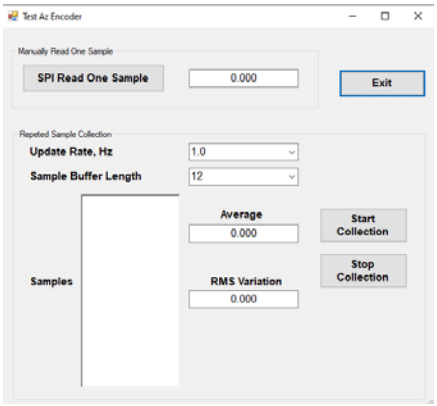


Figure 25 Simplified Azimuth angle read-back. Identical screen for Elevation.

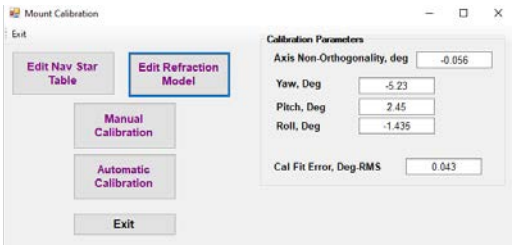


Figure 26 Sub-menu for overall telescope mount calibration with star field

4 DSP Software Details

A preliminary block diagram of the original algorithm approach[4] is shown in Figure 27. As the project development has matured, a number of significant changes to this original concept have evolved, more specifically:

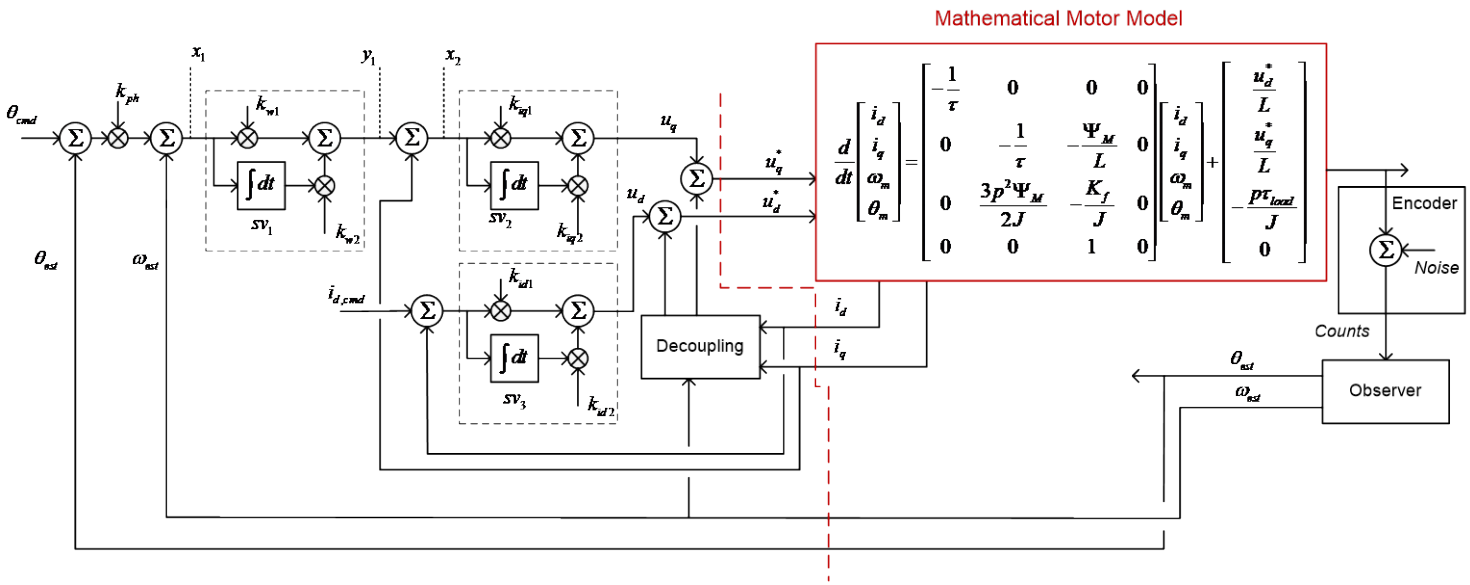


Figure 27 Block diagram of originally envisioned algorithm approach[5]

- Although the TMS320F28379D digital controller has an available high-resolution ePWM capability, this capability is only[6] provided for CPU1/CLA1.
  - a $\Delta-\Sigma$ algorithm can be used to effectively increase the ePWM's resolution
  - Scheme can be implemented for both CPU1/CLA1 and CPU2/CLA1.
  - The high reduction rate for the elevation axis admittedly mitigates this concern, however, and azimuth control could be carried out by the CPU1/CLA1 pair.
- It is desirable to use the same motor control algorithm for the azimuth and elevation axes in order to avoid two separate development efforts.
- Precise motor control is only required at very slow rotational rates thereby making the frequency feedback portion of Figure 27 all but unnecessary.
- The greatest project benefit for the i_d / i_q formulation is to reduce power supply current by applying *optimal* torque levels to each axis.
  - Reading of the 3-phase currents would be required adding more complexity
  - Power supply current savings for the azimuth axis would be quite limited because the maximum deliverable torque does not exceed the minimum required torque by very much.
  - Therefore, it makes sense to operate the control loops based upon only phase errors thereby affording design simplification for now.
- The control loop concept must accommodate several different modes with specific attention given to creation of the external excitation θ_{cmd} in Figure 27.

[4] Figure 26 & Figure 27 of [7]
[5] Figure 26 & Figure 27 of [7]
[6] High-Resolution ePWM only available on CPU1 and ePWM1-8 per Figure 1 of U26360.

4.1 Simplified Control Algorithm

Three-phase motor control can be described using complex space vectors which only involve two orthogonal axes. Utilization of the 2-phase motor model reduces the number of equations and simplifies the control design [23]. Most of the underlying theory was covered in [7] and will not be repeated here.

4.2 $\Delta-\Sigma$ Approach to Higher Effective ePWM Resolution

The minimum dead-time for the DRV8301 is 50 nsec per §7.4.2 of [21].

The DRV8301 BoosterPack uses a 1Ω resistor to ground thereby delivering the minimum dead-time of 50 nsec per [22]. Since the dead time behavior is taken care of within the DRV8301, it seems unlikely that RED and FED available provisions within the ePWMs will have to be used.

The system clock used by the DSP is 100 MHz. The EPWM_TIMER_TBPRD parameter sets the number of system clock periods used for the time base when the clock dividers are all set to unity. Setting EPWM_TIMER_TBPRD to 2048 results in an ePWM time resolution of 10 nsec and a ePWM pulse repetition rate of

$$f_{prr} = \frac{100MHz}{2048} = 48.828125kHz \tag{1}$$

Even though the associated interrupt rate is roughly 50 kHz, updating the control loop elements every 2nd or even 4th interrupt should be more than sufficient. The extra sampling rate can still be advantageously used to implement a simple first-order Δ-Σ modulator for each motor phase rather than having to resort to the more complicated high-resolution ePWM mode (which is only available with CPU1 anyway).

Using EPWM_TIMER_TBPRD = 2048 makes the rounding operation associated with the Δ-Σ modulator shown in Figure 28 a simple binary-shift operation (11 bits with sign extension) which is attractive.

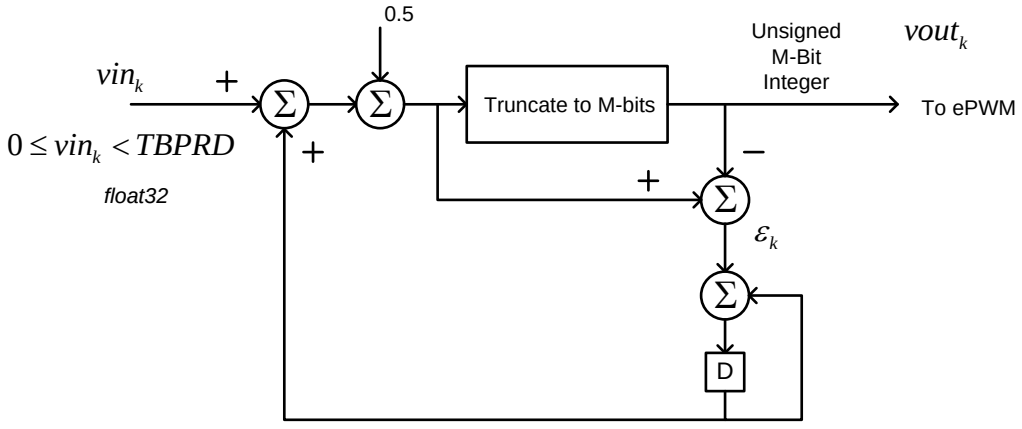


Figure 28 Simple 1st-order Δ-Σ modulator for improved suppression of low-frequency quantization-related output noise[7]

[7] From U27771_Part7_Figure.vsd.

In the context of voltages V_β and V_β as discussed in §10 of [7] for a phase argument θ, the 3-phase voltages can be found by effectively inverting (44) as

$$\begin{bmatrix} V_\alpha \\ V_\beta \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{2}{3} & \frac{-1}{3} & \frac{-1}{3} \\ 0 & \frac{1}{\sqrt{3}} & \frac{-1}{\sqrt{3}} \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} V_a \\ V_b \\ V_c \end{bmatrix} \tag{2}$$

from which

$$\begin{bmatrix} V_a \\ V_b \\ V_c \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \frac{-1}{2} & \frac{\sqrt{3}}{2} \\ \frac{-1}{2} & \frac{-\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} V_\alpha \\ V_\beta \end{bmatrix} \tag{3}$$

where

$$\begin{aligned} V_\alpha &= 0.50 \times TBPRD\left[1 + A_o \cos(\theta)\right] \\ V_\beta &= 0.50 \times TBPRD\left[1 + A_o \sin(\theta)\right] \end{aligned} \tag{4}$$

with $0 \le A_o \le TBPRD$.

4.3 Smooth Telescope Move Commanding

4.3.1 Method 1

Smooth motion of the telescope mount in azimuth and elevation is important, particularly for large angular steps. One method was sketched out in §8 of [7] which will serve as the starting point for a later discussion in this section.

I originally considered using a lowpass-filtered frequency-step function for positioning the telescope (in azimuth and elevation) but the resultant movement was not at all smooth at the beginning of the associated move as shown in Figure 29, but rather abrupt. The associated Laplace transfer function for the 2nd-order all-pole lowpass filter is given by

$$H(s) = \frac{\omega_n^2}{s^2 + 2\varsigma\omega_n s + \omega_n^2} \tag{5}$$

This can be converted to sampled-digital form using the bilinear transform leading to

$$\theta_o(z) = \theta_i(z)\frac{\omega_n^2\left(1+z^{-1}\right)^2}{a + bz^{-1} + cz^{-2}} \tag{6}$$

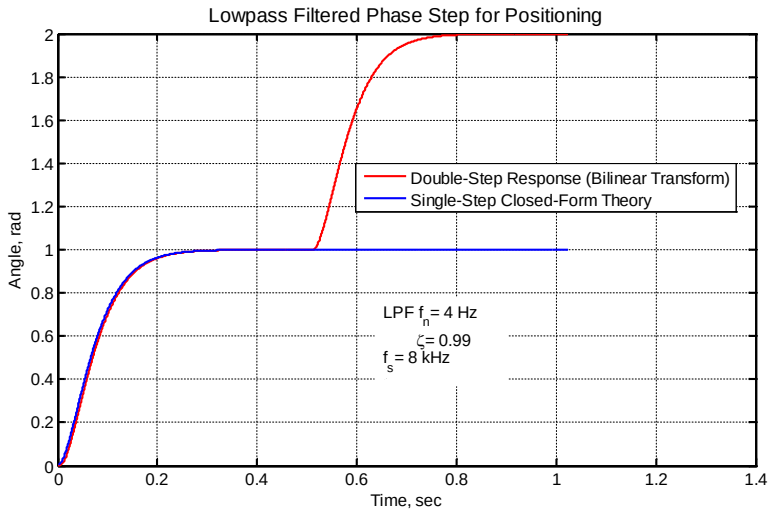


Figure 29 Lowpass-filtered phase step response[8]

with

$$\begin{aligned} a &= \left(\frac{2}{T}\right)^2 + \frac{4\varsigma\omega_n}{T} + \omega_n^2 \\ b &= 2\omega_n^2 - \frac{8}{T^2} \\ c &= \left(\frac{2}{T}\right)^2 - \frac{4\varsigma\omega_n}{T} + \omega_n^2 \end{aligned} \tag{7}$$

[8] u28404_smooth_positioner.m.

4.3.2 Method 2 (Originally from §8 of [7])

If an axle is to be rotated θ_{cmd} radians, this needs to be done (i) within a prescribed amount of time, (ii) with a specified degree of smoothness, and (iii) with a radian velocity no greater than a specified limit. These requirements are not mutually independent, however. For structural safety, it is better to think in terms of (iv) maximum angular velocity allowed and (v) maximum angular acceleration allowed since (v) limits the applied current and consequently the applied torque and (vi) limits the total rotational inertia allowed in the system.

With items (v) and (vi) in mind for large angular changes, the methodology outlined in Figure 30 achieves both objectives while being relatively simple. In this figure, the (positive) angular acceleration is limited to $a_{\theta\max} = 2\omega_{\max} / T_r$ rad/sec^2 and the radian velocity is limited to ω_{max} rad/sec. A computed example[9] is shown in Figure 31.

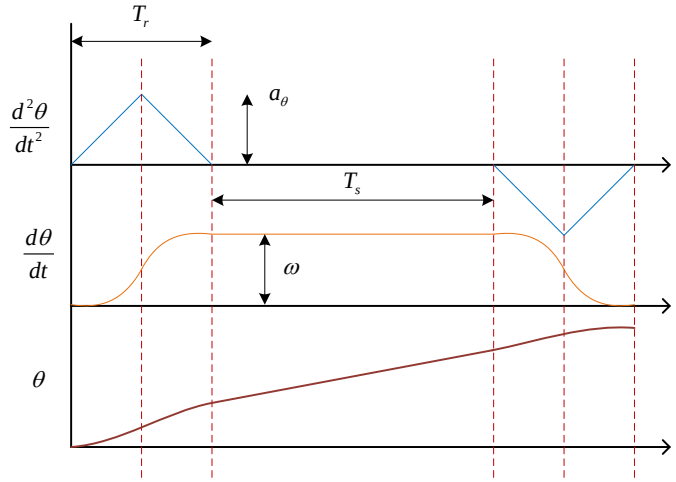


Figure 30 Angular acceleration and angular velocity should be constrained in order to limit the torque required and rotational speed used

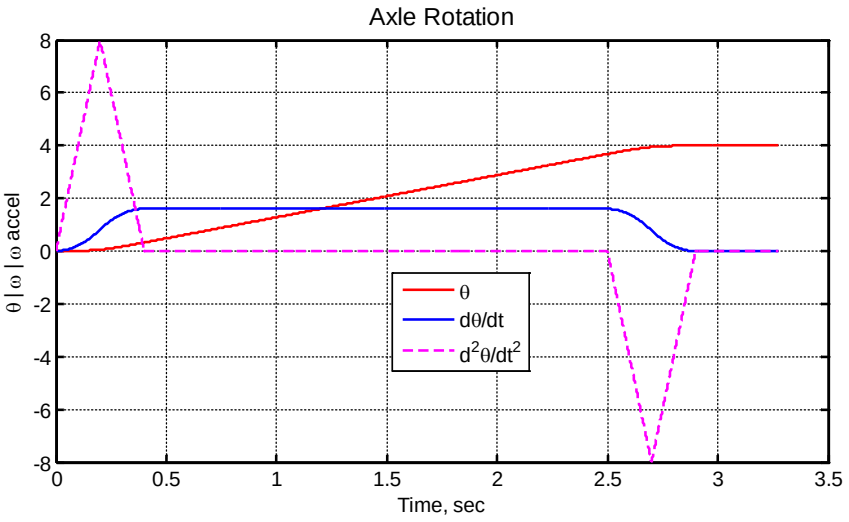


Figure 31 Computed example using the concepts of Figure 30. Maximum radian acceleration limited to 8 rad/sec^2 and maximum radian velocity limited to 1.6 rad/sec. Time required to traverse 4 radian change is about 2.5 sec.

[9] u25930_rotation_trajectory.m.

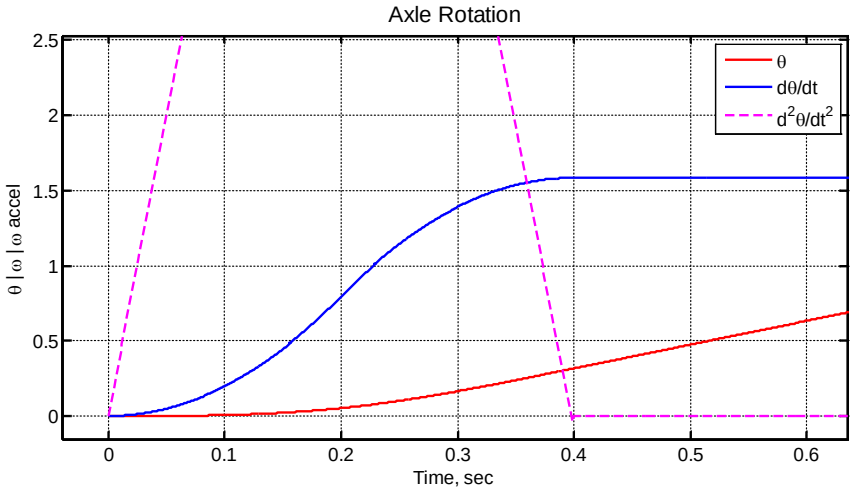


Figure 32 Close-up of Figure 31

More concisely in the context of Figure 30 and Figure 31, denote the maximum angular acceleration as $a_{\theta\max}$. Assuming that the angular rotation is in the positive sense, denote the maximum allowed angular velocity as $\omega_{\max}$. From Figure 30,

$$T_r = \frac{2\omega_{\max}}{a_{\theta\max}} \tag{8}$$

For simplicity, the same value of T_r can be used for all cases. Denote the maximum possible phase change impressed during the rotational acceleration as θ_{pmax} where

$$\theta_{pmax} = \frac{1}{2}\omega_{max}T_r \tag{9}$$

Denote the commanded rotational phase change by θ_{cmd}.

$$\begin{aligned}
&if\ \left(\theta_{cmd} > 2\omega_{\max}T_r\right)\\
&\quad T_s = \frac{\theta_{cmd} - 2\theta_{pmax}}{\omega_{max}} = \frac{\theta_{cmd} - \omega_{max}T_r}{\omega_{max}} = \frac{\theta_{cmd}}{\omega_{max}} - T_r\\
&else\\
&\quad T_s = 0\\
&\quad \theta_p = a_\theta \frac{T_r^2}{4}\\
&\quad a_\theta = \frac{2\theta_{cmd}}{T_r^2}\\
&end
\end{aligned} \tag{10}$$

where θ_{cmd} is the desired rotation angle change in radians. From the previous two figures, in the case where $T_s \equiv 0$,

$$a(t) = \begin{Bmatrix} a_\theta & 0 < t \le \frac{T_r}{2} \\ -a_\theta & \frac{T_r}{2} < t \le \frac{3}{2}T_r \\ a_\theta & \frac{3}{2}T_r < t \le 2T_r \end{Bmatrix} \tag{11}$$

and otherwise zero. In the case where $T_s \neq 0$,

$$a(t) = \begin{Bmatrix} a_\theta & 0 < t \le \frac{T_r}{2} \\ -a_\theta & \frac{T_r}{2} < t \le T_r \\ -a_\theta & T_r + T_s < t \le T_s + \frac{3}{2}T_r \\ a_\theta & T_s + \frac{3}{2}T_r < t \le T_s + 2T_r \end{Bmatrix} \tag{12}$$

but otherwise equal to zero. Given $a(t)$, the radian velocity and angle are given by

$$\begin{aligned} \omega(t) &= \int_0^t a(u)\,du \\ \theta(t) &= \int_0^t \omega(u)\,du \end{aligned} \tag{13}$$

4.3.3 Additional Smoothing

The phase trajectory shown in Figure 31 and Figure 32 is very smooth as desired, but it can be made even more smooth by replacing the triangular acceleration pulses shown in Figure 31 with raised-cosine pulses as shown in Figure 33. In this approach, the triangular acceleration pulse is replaced by a raised-cosine pulse where the angular acceleration is given by

$$a(t) = a_\theta \frac{1 - \cos(\gamma t)}{2} \quad \textit{for } 0 \le t \le T_r \tag{14}$$

and the maximum angular acceleration is given by a_θ.

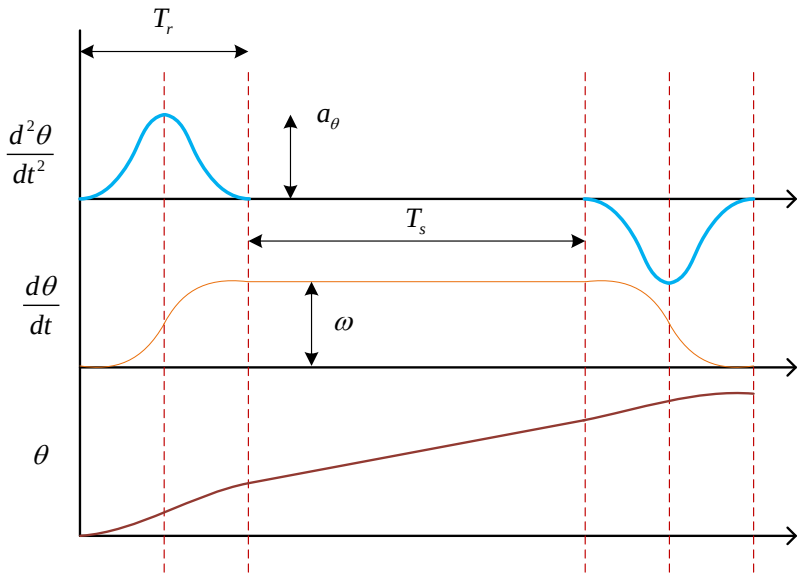


Figure 33 Smooth control waveform making use of raised-cosine acceleration pulses[10]

The radian frequency for $0 \le t \le T_r$ is the integral of (14) and given by

$$\omega(t) = \frac{a_\theta}{2}\left[t - \frac{\sin(\gamma t)}{\gamma}\right] \quad for\ 0 \le t \le T_r \tag{15}$$

with

$$\omega_{\max} = \frac{a_\theta T_r}{2} \tag{16}$$

This result makes it possible to rewrite (15) in a more convenient form as

$$\omega(t) = \frac{\omega_{\max}}{T_r}\left[t - \frac{\sin(\gamma t)}{\gamma}\right] \quad for\ 0 \le t \le T_r \tag{17}$$

The instantaneous phase during the ramping interval follows as given by

$$\begin{aligned}\theta(t) &= \frac{\omega_{\max}}{T_r}\left[\frac{t^2}{2} - \frac{\cos(\gamma t)}{\gamma^2} + \frac{1}{\gamma^2}\right] \quad for\ 0 \le t \le T_r \\ &= \frac{\omega_{\max}}{2T_r}\left[t^2 - \left(\frac{2}{\gamma}\right)^2 \sin^2\left(\frac{\gamma t}{2}\right)\right] \quad for\ 0 \le t \le T_r \\ &= \frac{\omega_{\max}}{2T_r}\left\{t^2 - \left[\frac{2}{\gamma}\sin\left(\frac{\lambda t}{2}\right)\right]^2\right\} \quad for\ 0 \le t \le T_r\end{aligned} \tag{18}$$

[10] U28405 Photogrammetry Part 7 Figures.vsd.

In this context,

$$\gamma = \frac{2\pi}{T_r} \tag{19}$$

and

$$\theta_{pulse} = \theta\left(T_r\right) = \frac{\omega_{\max} T_r}{2} \tag{20}$$

In order to make use of these results, assume that the pointing direction is to be changed by $\Delta\theta$ (assumed to be positive).

If $\Delta\theta \le 2\theta_{pulse}$, T_s in Figure 33 can be made zero by choosing

$$\omega = \frac{\Delta\theta}{T_r} \le \omega_{\max} \tag{21}$$

and using this value for $\omega_{\max}$ in the previous equations. If, on the other hand, $\Delta\theta > 2\theta_{pulse}$, use (20) along with

$$T_s = \frac{\Delta\theta - 2\theta_{pulse}}{\omega_{\max}} \tag{22}$$

An example start-up segment using these equations is shown in Figure 34 where the acceleration interval T_r = 3 seconds and the maximum rotational rate is limited to 36° per second which is quite fast! Adopting T_r = 5 seconds with a maximum rotational rate of 15° per second is shown in

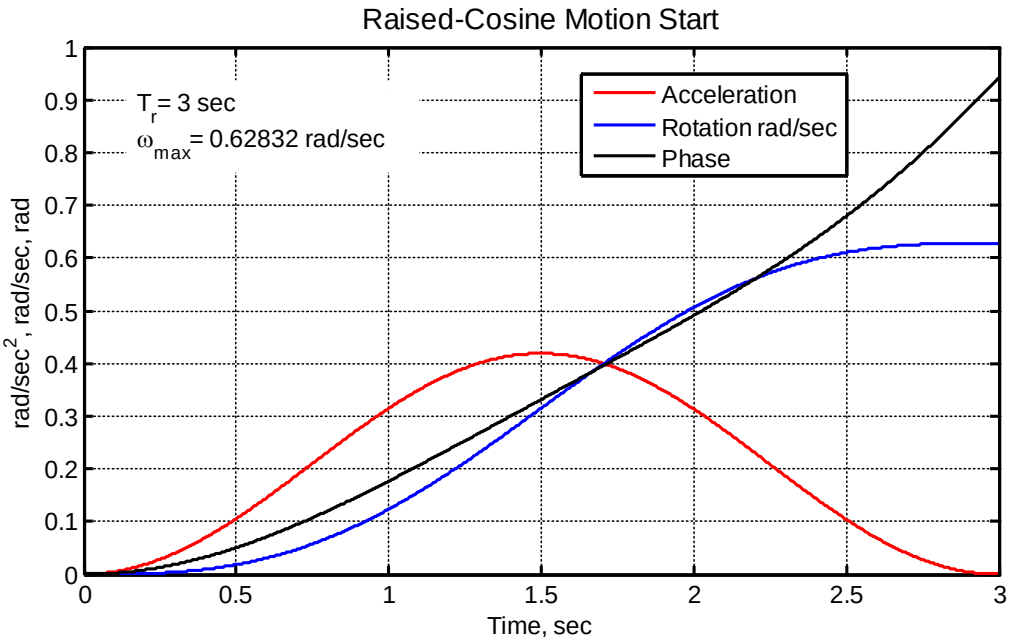


Figure 34 Start-up motion using raised-cosine trajectory shaping[11]

[11] u25930_rotation_trajectory.m

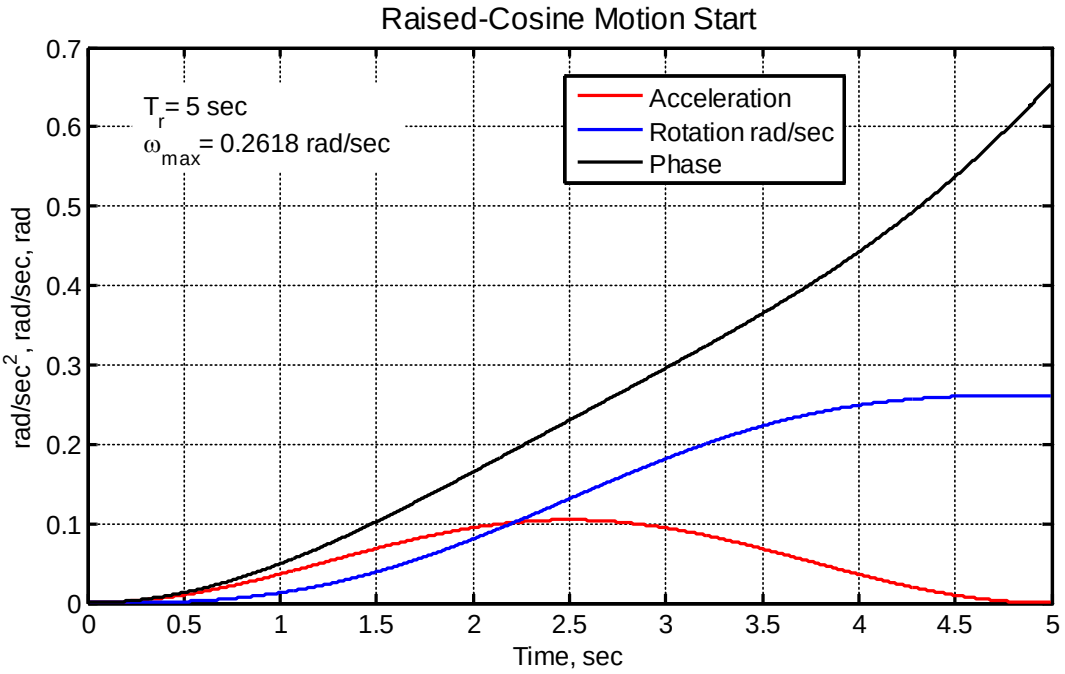


Figure 35 Start-up motion using raised-cosine trajectory shaping[12] with lower acceleration than in Figure 34

5 Wrap-Up

This installment wraps-up the mechanical design and fabrication stages as already mentioned. Most of the mathematics behind the in-work software programming have already been developed in this and earlier installments.

Moving forward, I have some hard decisions to make. I originally set out to build only one/two telescope mounts but this has changed to a minimum of 5 or 6. The cost associated with using the 3-phase motors is appreciable (on the order of $300+). The worst cost-offender, however, is the harmonic drive which has ballooned from roughly $140 to upwards of $500 or more. I originally intended to host the GUI on a laptop-class PC, but going this route would add even more cost to each instantiation of the project. Neither did I factor in any weight constraints with this original design, counting some additional weight as a positive because it can add physical stability to the mount if properly used. The present mount is coming in heavier than I would like, however, given the additional use-cases on my drawing board.
Regardless, this has been a wonderful project for me, and it will continue on in one form or another to completion!

[12] u25930_rotation_trajectory.m

6 References[13]

1. J.A. Crawford, “Photogrammetry for Non-Invasive Terrestrial Position/Velocity Measurement of High-Flying Aircraft- Part 6AB,” October 2022.
2. _____, “Photogrammetry for Non-Invasive Terrestrial Position/Velocity Measurement of High-Flying Aircraft- Part 6A,” January 2022.
3. _____, “Photogrammetry for Non-Invasive Terrestrial Position/Velocity Measurement of High-Flying Aircraft- Part V: Optical Encoder Completion and Transition to TI DSP,” January 2021, U24933.
4. _____, “Photogrammetry for Non-Invasive Terrestrial Position/Velocity Measurement of High-Flying Aircraft- Part V,” U24933, January 2021.
5. _____, “Photogrammetry for Non-Invasive Terrestrial Position/Velocity Measurement of High-Flying Aircraft- Part IV: Optical Encoder and Elevation Drive,” U24933, August 2020.
6. _____, “Photogrammetry for Non-Invasive Terrestrial Position/Velocity Measurement of High-Flying Aircraft- Part III,” U24933, November 2019.
7. _____, “Photogrammetry for Non-Invasive Terrestrial Position/Velocity Measurement of High-Flying Aircraft- Part II, Direct-Drive Motors,” U24933, April 2019.
8. _____, “Photogrammetry for Non-Invasive Terrestrial Position/Velocity Measurement of High-Flying Aircraft- Part I,” 2017, U24866.
9. Texas Instruments, “BOOSTAXL-DRV8301 Hardware User’s Guide,” SLVU974, Oct. 2013, U27331.
10. _____, “The TMS320F2837xD Architecture: Achieving a New Level of High Performance,” SPRT720 Feb 2016, U26360.
11. _____, “TMS320F2837xD Dual-Core Delfino Microcontrollers, Technical Reference Manual,” Jan. 2019, SPRUHM8H, U26369.
12. _____, “My (Aluminum) Anodizing Procedure,” 15 Oct 2022, U28255.
13. Elliott D. Kaplan and Christopher J. Hegarty, *Understanding GPS, Principles and Applications*, 2nd ed., Artech House, 2006.
14. Harmonic Drive LLC, “HarmonicDrive® Reducer Catalong”, U27619.
15. Oliver Montenbruck and Thomas Pfleger, *Astronomy on the Personal Computer*, 1999, Springer-Verlag.
16. Eric Burgess, *Celestial Basic, Astronomy on Your Computer*, 1985, Sybex.
17. Jean Meeus, *Astronomical Algorithms*, 2nd ed., 1998, William-Bell.
18. Roger R. Bate, Donald D. Mueller, and Jerry E. White, *Fundamentals of Astrodynamics*, Dover, 1971.
19. Pratap Misra and Per Enge, *Global Positioning System, Signals, Measurements, and Performance*, 2nd ed., Ganga-Jamuna Press, 2012.
20. Richard H. Battin, *An Introduction to the Mathematics and Methods of Astrodynamics*, Revised Ed., American Institute of Aeronautics and Astronautics, 1999.
21. U25070.
22. U28401
23. Freescale Semiconductor, “PM Sinusoidal Motor Vector Control with Quadrature Encoder,”DRM105, Rev. 0, 09/2008, U24984.
24. Kvetoslav Belda, “Mathematical Modelling and Predictive Control of Permanent Magnet Synchronous Motor Drives,”*Trans. Electrical Engineering, 2013, U25081.*
25. Matthew Piccoli and Mark Yim, “Cogging Torque Ripple Minimization via Position-Based Characterization,” U25085.
26. Freescale Semiconductor, “3-Phase PM Synchronous Motor Vector Control Using a 56F80x, 56F8100, or 56F8300 Device,” Application Note AN1931, Jan. 2005.

[13] Expanded section from Part A.

7 Appendix: Electrical Interfaces DSP & Custom PWB

A number of updates have been made to the I/O map as summarized in this section.

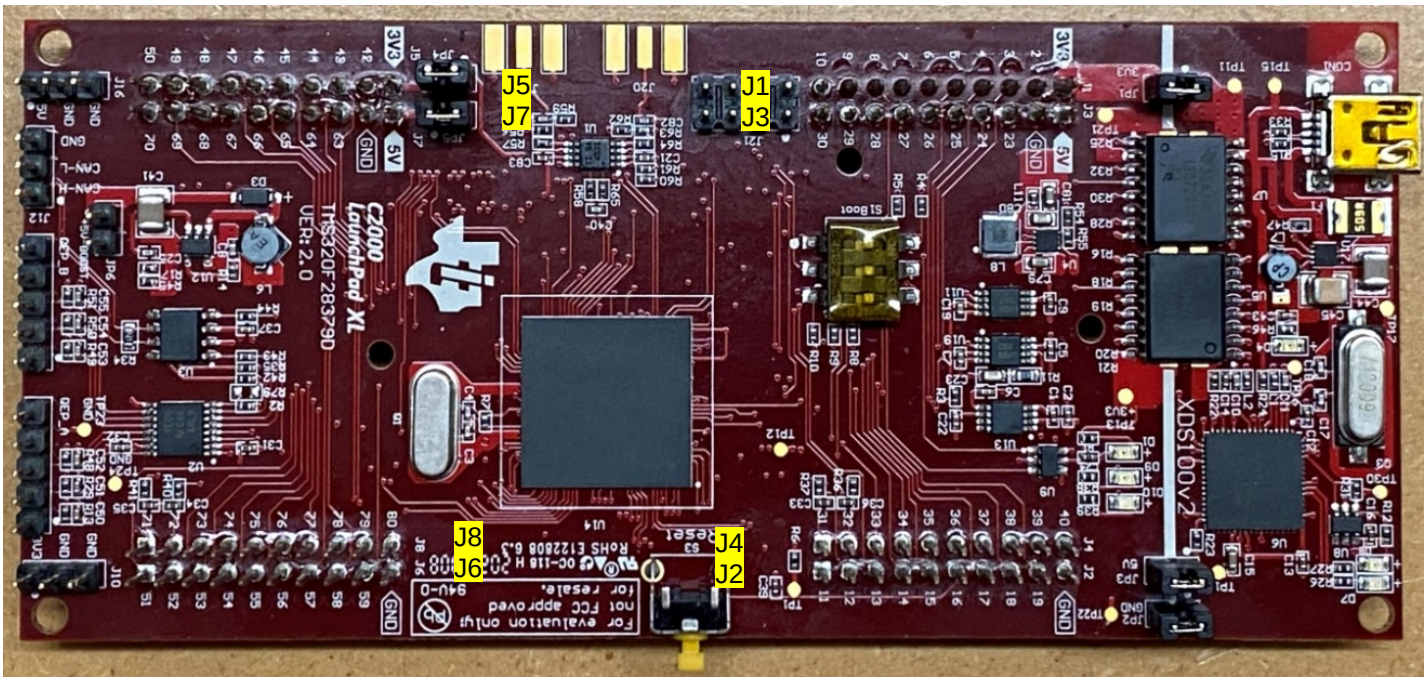


Figure 36 Top-side of the TMS320F28379D Launchpad board. Note the yellow reset button at the bottom.

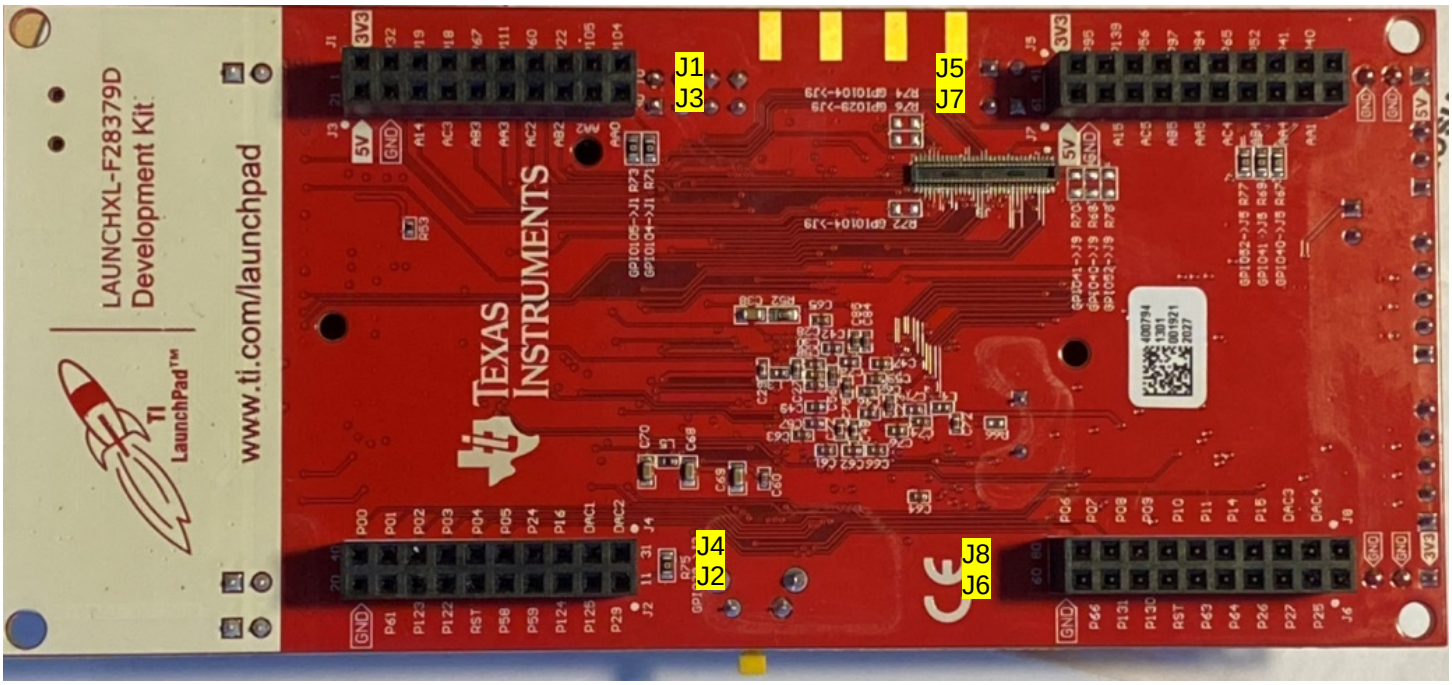


Figure 37 Back-side of the TMS320F28379D Launchpad board. Note the yellow reset button at the top.

Table 7-1 DRV8301 I/O Mapping[14] to F28379D Configured for Booster Pack Situated Closest to USB Connector for Elevation Axis (CPU2 in most cases, SPI-A). All of the green signal definitions were confirmed by way of pin_map.h.

DRV8301 Signal	F28379D Pins				DSP GPIO	PCB J#	J-Pin	F28379D Signal (Not Exhaustive)	Non-DRV8301 Usage
	J1	J3	J4	J2					
3.3V	1					J3andJ1	19	3.3V	
x	2				32	J3andJ1	17	GPIO32 / SDAA	LED1 Power
FAULT	3				19	J3andJ1	15	GPIO19 / SCIRXDB / SPISTEA*	
OCTW	4				18	J3andJ1	13	GPIO18 / SCITXDB / SPICLKA	
x	5				67	J3andJ1	11	GPIO67	nRST_AZ
x	6				111	J3andJ1	9	GPIO111	nRST_EL
SCLK	7				60	J3andJ1	7	GPIO60 / SPICLKA / SPISIMOB	
x	8				22	J3andJ1	5	GPIO22 / SCITXDB / SPICLKB	Available- brought out to J17-1
x	9				105	J3andJ1	3	GPIO105 / SCLA / SCIRXDD	Available- brought out to J17-2
x	10				104	J3andJ1	1	GPIO104 / SDAA / SCITXDD	Available- brought out to J21-1
x		21				J3andJ1	20	5V	5V_ENCODER_B
GND		22				J3andJ1	18	GND	GND_ENCODER_B
DC-V-FB		23				J3andJ1	16	ADCIN14 / CMPIN4P	
VA-FB		24				J3andJ1	14	ADCINC3 / CMPIN6N	
VB-FB		25				J3andJ1	12	ADCINB3 / CMPIN3N	
VC-FB		26				J3andJ1	10	ADCINA3 / CMPIN1N	
IA-FB		27				J3andJ1	8	ADCINC2 / CMPIN6P	
IB-FB		28				J3andJ1	6	ADCINB2 / CMPIN3P	
IC-FB		29				J3andJ1	4	ADCINA2 / CMP1N1P	
x		30				J3andJ1	2	ADCINA0 / DACOUTA	
PWM-AH			40		0	J2andJ4	19	EPWM1A / GPIO0/SDAA	
PWM-AL			39		1	J2andJ4	17	EPWM1B / GPIO1/SCLA	
PWM-BH			38		2	J2andJ4	15	EPWM2A / GPIO2	
PWM-BL			37		3	J2andJ4	13	EPWM2B / GPIO3/SCLB	
PWM-CH			36		4	J2andJ4	11	EPWM3A / GPIO4	
PWM-CL			35		5	J2andJ4	9	EPWM3B / GPIO5	
x			34		24	J2andJ4	7	OUTPUTXBAR1 / GPIO24 / SPISIMOB	Available- brought out to J17-3
x			33		16	J2andJ4	5	OUTPUTXBAR7 / GPIO16 / SPISIMOA	Available- brought out to J17-4
x			32			J2andJ4	3	DAC1	*New*: CPU2 DAC1 to J20-2
x			31			J2andJ4	1	DAC2	*New:* CPU2 DAC2 to J20-1
GND				20		J2andJ4	20	GND	
SCS				19	61	J2andJ4	18	GPIO61 / SPISOMIB / SPISTEA*	
x				18	123	J2andJ4	16	GPIO123 / SD1_C1 / SPISOMIC	*Change:* Optical Encoders
x				17	122	J2andJ4	14	GPIO122 / SD1_D1 / SPISIMOC	*Change:* Optical Encoders
x				16		J2andJ4	12	RST	
SDI				15	58	J2andJ4	10	GPIO58 / SPISIMOA /SPICLKB	
SDO				14	59	J2andJ4	8	GPIO59 / SPISOMIA/SPISTEB*	
EN-GATE				13	124	J2andJ4	6	GPIO124 / SD1_D2	
DC-CAL				12	125	J2andJ4	4	GPIO125 / SD1_C2	
x				11	29	J2andJ4	2	GPIO29 / OUTPUTXBAR6 / SCITXDA	Available- brought out to J17-5

[14] From U26355 LAUNCHXL-F28379D Overview, SPRUI77C, August 2016, March 2019. Not all GPIO etc pins are mapped to J-connectors. See U27334 F28379D Datasheet sprs880k for details.

Table 7-2 DRV8301 I/O Mapping to F28379D Configured for Booster Pack situated furthest from USB connector (CPU1 in most cases SPI-B). All of the green signal definitions were confirmed by way of pin_map.h.

DRV8301 Signal	F28379D Pins				DSP GPIO	PCB J#	J-Pin	F28379D Signal (Not Exhaustive)	Non-DRV8301 Usage
	J5	J7	J8	J6					
3.3V	41					J7andJ5	19	3.3V	
x	42				95	J7andJ5	17	GPIO95	1PPS from GPS Receiver
FAULT	43				139	J7andJ5	15	GPIO139 / SCIRXDC	
OCTW	44				56	J7andJ5	13	GPIO56 / SCITXDC/SPICLKA	
x	45				97	J7andJ5	11	GPIO97	*Change:* Available, brought out to J21-2
x	46				94	J7andJ5	9	GPIO94	*Change:* Available, brought out to J21-3
SCLK	47				65	J7andJ5	7	GPIO65 / SPICLKB/SCITXDA	
x	48				52	J7andJ5	5	GPIO52 / SPICLKC	SPICLKC for Optical Encoders
x	49				41	J7andJ5	3	GPIO41 / SCLB	ZERO_RESET_AZ
x	50				40	J7andJ5	1	GPIO40 / SDAB	ZERO_RESET_EL
x		61				J7andJ5	20	5V	5V_ENCODER_A
GND		62				J7andJ5	18	GND	GND_ENCODER_A
DC-V-FB		63				J7andJ5	16	ADCIN15 / CMPIN4N	
VA-FB		64				J7andJ5	14	ADCINC5 / CMPIN5N	
VB-FB		65				J7andJ5	12	ADCINB5	
VC-FB		66				J7andJ5	10	ADCINA5 / CMPIN2N	
IA-FB		67				J7andJ5	8	ADCINC4 / CMPIN5P	
IB-FB		68				J7andJ5	6	ADCINB4	
IC-FB		69				J7andJ5	4	ADCINA4 / CMPIN2P	
x		70				J7andJ5	2	ADCINA1 / DACOUTB	
PWM-AH			80		6	J6andJ8	19	EPWM4A / GPIO6	
PWM-AL			79		7	J6andJ8	17	EPWM4B / GPIO7	
PWM-BH			78		8	J6andJ8	15	EPWM5A / GPIO8	
PWM-BL			77		9	J6andJ8	13	EPWM5B / GPIO9	
PWM-CH			76		10	J6andJ8	11	EPWM6A / GPIO10	
PWM-CL			75		11	J6andJ8	9	EPWM6B / GPIO11	
x			74		14	J6andJ8	7	GPIO14 / OUTPUTXBAR3 / SCITXDB	SCITXDB for RP4 UART
x			73		15	J6andJ8	5	GPIO15 / OUTPUTXBAR4 / SCIRXDB	SCIRXDB for RP4 UART
x			72			J6andJ8	3	DAC3	*New:* CPU1 DAC3 to J20-3
x			71			J6andJ8	1	DAC4	*New:* CPU1 DAC4 to J20-4
GND				60		J6andJ8	20	GND	
SCS				59	66	J6andJ8	18	GPIO66 / SPISTEB*	
x				58	131	J6andJ8	16	GPIO131 / SD2_C1	READAZ*
x				57	130	J6andJ8	14	GPIO130 / SD2_D1	READEL*
x				56		J6andJ8	12	RST	
SDI				55	63	J6andJ8	10	SPISIMOB / GPIO63/SCITXDC	
SDO				54	64	J6andJ8	8	SPISOMIB / GPIO64	
EN-GATE				53	26	J6andJ8	6	GPIO26 / SD2_D2/SPICLKB	
DC-CAL				52	27	J6andJ8	4	GPIO27 / SD2_C2/SPISTEB*	
x				51	25	J6andJ8	2	GPIO25 / OUTPUTXBAR2 / SPISOMIB	Available, brought out to J21-4

*7.1 Using GPIO to Emulate SPISTE**

The F28379D has only 3 SPI ports, two of which are dedicated to the DRV8301 Booster Packs as mentioned earlier. Fortunately, individual GPIO pins can be configured to mimic SPISTE* signals to as many slave devices as desired with all slaves sharing the SPICLK, SPISIMO, and SPISOMI lines[15]. GPIO131 will perform the SPISTE* function for the azimuth encoder read operation, and GPIO130 will perform the same function for the elevation encoder read function.

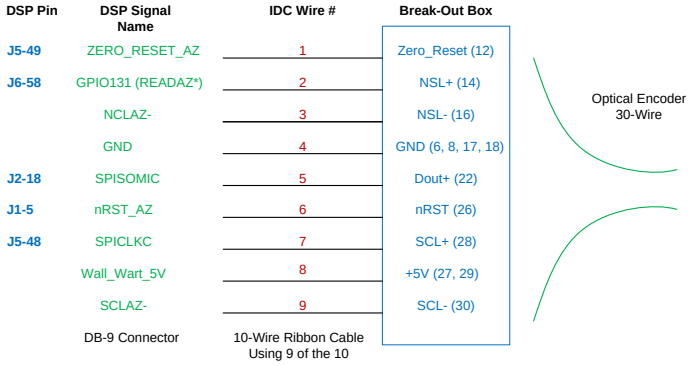


Figure 38 DSP / ribbon cable / encoder wiring[16] for Az optical encoder. Duplicate wiring diagram for El optical encoder except GPIO130 is used for READEL*, ZERO_RESET_AZ and nRST_AZ similarly modified as shown in Figure 39. The red ribbon cable wire is taken to be #1.

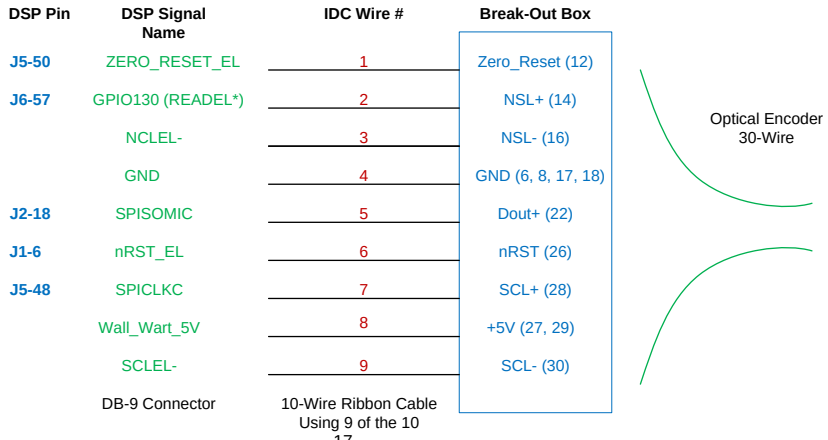
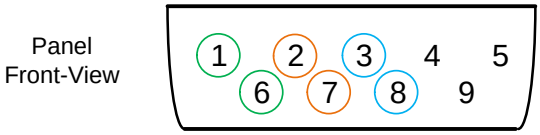


Figure 39 DSP / ribbon cable / encoder wiring[17] for El optical encoder

[15] Described in §18.2.1 of [11].
[16] From U27771_Part7_Figures.vsd. Revised here from Part 6.
[17] From U27771_Part7_Figures.vsd. Revised here from Part 6.



Pins 1,6 Green Phase C
Pins 2,7 Brown Phase B
Pins 3,8 Blue Phase A
Pins 4, 5, 9 Not Used

Figure 40 Front-Panel view of 3-phase motor connector[18]. Phase nomenclature pertains to DRV8301 pinout whereas colors are for actual wire colors.

[18] U27771_Part7_Figures.vsd.

8 Appendix: Sin() and Cos() Approximations

Three-phase motor control requires highly efficient calculation of sin() and cos() functions suitable for CLA implementation. The method adopted here is based upon a combination of table look-ups with some minor additional calculations.

Assume that the angle of interest is θ and that this value is optimally close to the k^{th} angular entry in the tables such that

$$\theta = \varphi_k + \delta \tag{23}$$

As such,

$$\begin{aligned} \exp(j\theta) &= \exp(j\varphi_k)\exp(j\delta) \\ &= (I_k + jQ_k)\left[\cos(\delta) + j\sin(\delta)\right] \\ &\cong (I_k + jQ_k)\left(1 - \frac{\delta^2}{2} + j\delta\right) \\ I + jQ \quad &\cong I_k\left(1 - \frac{\delta^2}{2}\right) - \delta Q_k + j\left[Q_k\left(1 - \frac{\delta^2}{2}\right) + \delta I_k\right] \end{aligned} \tag{24}$$

which is equivalently described by Figure 41

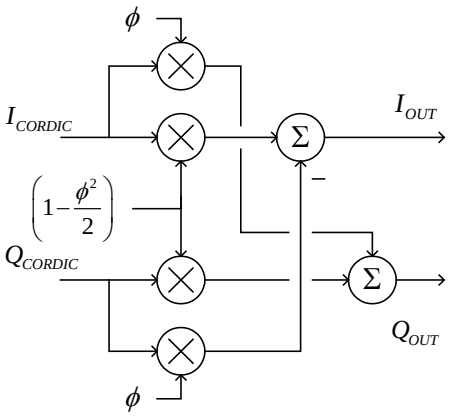


Figure 41 Second-order CORDIC correction method[19] given by (24). I_{CORDIC} and Q_{CORDIC} as provided by the table look-up steps.

The approximation precision can be further improved with some additional computation as given by

$$\begin{bmatrix} I \\ Q \end{bmatrix} \cong \begin{bmatrix} \alpha & -\beta \\ \beta & \alpha \end{bmatrix} \begin{bmatrix} I_k \\ Q_k \end{bmatrix} \tag{25}$$

with

[19] Figure 5-35 from *Advanced Phase-Lock Applications- Synthesis*, J.A. Crawford.

$$\alpha = 1 - \frac{\delta^2}{2}$$
$$\beta = \delta\left(1 - \frac{\delta^2}{6}\right) \tag{26}$$

8.1 Approximation Performance

The CLA implementation will use IEEE single-precision floating-point calculations. As such, the mantissa precision is limited to 24-bits or equivalently about ±1.192e-7. MATLAB-based precision estimates[20] for different look-up table sizes are shown in Figure 42 through Figure 45. Consquently, a Sin and Cos table look-up size of 32 samples (in a single quadrant) is completely adequate to eliminate table size as a significant contributor to the net overall precision.

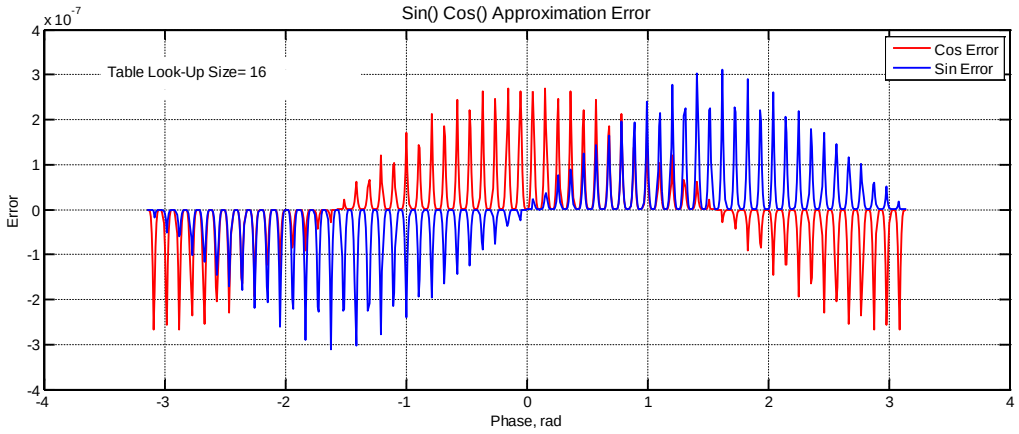


Figure 42 Approximation error using sin and cos table look-up sizes of 16 (for one quadrant)

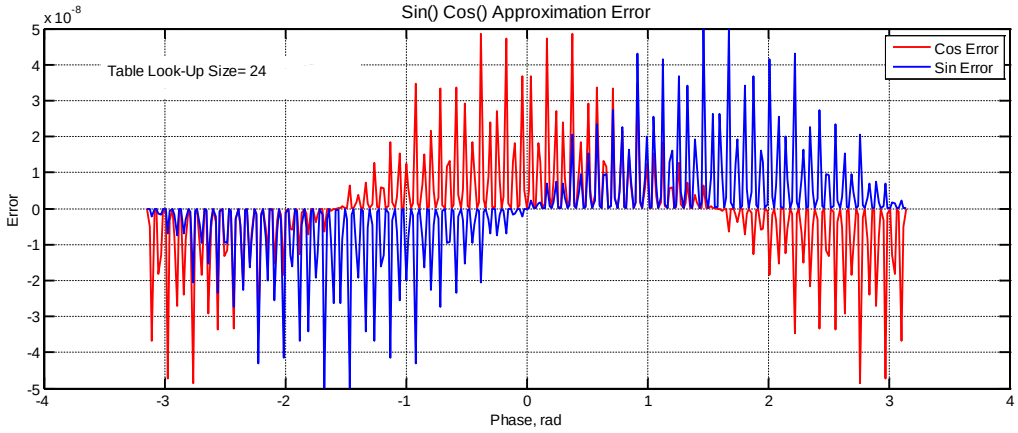


Figure 43 Approximation error using sin and cos table look-up sizes of 24 (for one quadrant)

[20] u28446_trial_sin_cos.m.

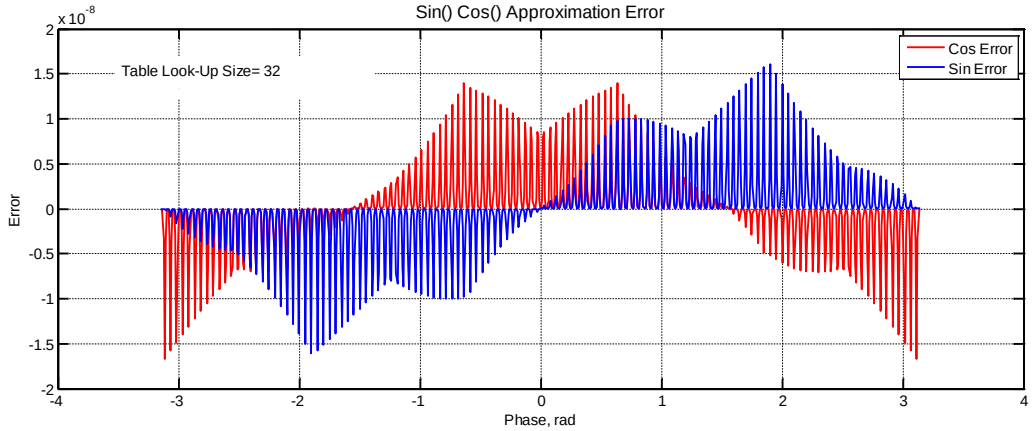


Figure 44 Approximation error using sin and cos table look-up sizes of 32 (for one quadrant)

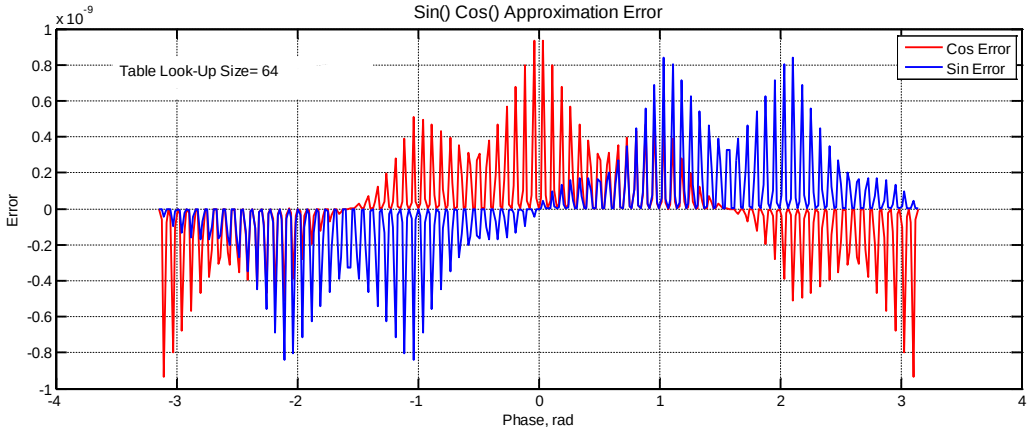


Figure 45 Approximation error using sin and cos table look-up sizes of 64 (for one quadrant)

8.2 MATLAB Script

The first script takes advantage of sin() and cos() table symmetry so that only the first quadrant of values are stored in the tables. Although this keeps the tables small, the additional computational overhead is painful to code for the CLA because subroutines and or function calls are not permitted with the CLA code since the CLAs do not have a stack feature.

```
%================ u28446_trial_sin_cos.m ================================
%
%   MATLAB trial for sin() and cos() approximations
%
%   J.A. Crawford
%   19 Feb 2023
%
%
function u28446_trial_sin_cos( table_points, theta_min, theta_max, npts )
%
global sin_table;
global cos_table;
global pi_over_2;
global two_pi;
global dtheta;
global thetas;

disp( '==================');
disp( ' ' );
```

```
pi_over_2= pi/2;
two_pi= 2*pi;
dtheta= pi_over_2 / ( table_points - 1 );
disp( ['dtheta=', ' ', num2str(dtheta) ] );

%
%   Build sine and cosine first-quadrant tables
%
sin_table= zeros(1, table_points);
cos_table= sin_table;
thetas= cos_table;

for ii=1:table_points
    thetas(ii)= (ii-1)*dtheta;
    sin_table(ii)= sin( thetas(ii) );
    cos_table(ii)= cos( thetas(ii) );
end

theta_plot= theta_min + (theta_max - theta_min)*(0:npts-1)/(npts-1);

err_cos= zeros(1,npts);
err_sin= zeros(1,npts);
for ii=1:npts
    ideal_cos= cos(theta_plot(ii));
    ideal_sin= sin(theta_plot(ii));
    [Ix,Qx]= my_cos_sin( theta_plot(ii), table_points );
    %
    %   Compute approx values for sine and cosine
    %
    err_cos(ii)= ideal_cos - Ix;
    err_sin(ii)= ideal_sin - Qx;
end

fig1= figure(1);
clf;
axes( 'FontName', 'Arial', 'FontSize', 12 );
p1= plot( theta_plot, err_cos, 'r' );
set( p1, 'LineWidth', 2 );
hold on
p1= plot( theta_plot, err_sin, 'b' );
set( p1, 'LineWidth', 2 );
xlabel( 'Phase, rad', 'FontName', 'Arial', 'FontSize', 12 );
ylabel( 'Error', 'FontName', 'Arial', 'FontSize', 12 );
title( 'Sin() Cos() Approximation Error', 'FontName', 'Arial', 'FontSize', 14 );
grid on
h= gca;
set( h, 'LineWidth', 2 );

txt1= strcat( ['Table Look-Up Size=', ' ', num2str(table_points) ] );
annotation(fig1,'textbox','String',{txt1},'FontSize',12,...
    'FontName','Arial',...
    'FitBoxToText','off',...
    'LineStyle','none',...
    'BackgroundColor',[1 1 1],...
    'Position',[0.1757 0.7655 0.2093 0.09595]);

legend( 'Cos Error', 'Sin Error' );

end
%=========================================================================

function [Ix, Qx]= my_cos_sin( theta, table_points )
%
%   Approximation function for cos() and sin()
%
%   theta is the angle of interest, rad
%   table_points is the number of table look-up points
%
global sin_table;       % Table look-up for 1st quadrant
global cos_table;       % Table look-up for 1st quadrant
global thetas;          % Angles, rad, associated with table look-up

global pi_over_2;
global two_pi;
global dtheta;
```

```
%
%   Constrain angle to within -2pi to +2pi
%
theta= theta - (theta >  0)*two_pi*floor(theta/two_pi) - ...
               (theta <= 0)*two_pi*floor(theta/two_pi);

%
%   Contain angle within +/-pi
%
if( theta < -pi )
    theta= theta + 2*pi;
elseif( theta > pi )
    theta= theta - 2*pi;
end

%
%   Figure out angle quadrant to apply symmetry later
%
if( theta > 0 )
    quad= 1 + (theta > pi_over_2);
else
    quad= 3 + (theta > -pi_over_2);
end

epu= 0.5;
switch( quad )
    case 1
        ang= theta;
        table_index= 1 + floor( ang/dtheta + epu );
        Ixx= cos_table( table_index );
        Qxx= sin_table( table_index );

        dphi= ang - thetas( table_index );
    case 2
        ang= pi - theta;
        table_index= 1 + floor( ang/dtheta + epu ) ;
        Ixx= -cos_table( table_index );
        Qxx=  sin_table(  table_index );

        dphi= -ang + thetas(table_index);
    case 3
        ang= theta + pi;
        table_index= 1 + floor( ang/dtheta + epu );
        Ixx= -cos_table( table_index );
        Qxx= -sin_table( table_index );

        dphi= ang - thetas(table_index);
    case 4
        ang= -theta;
        table_index= 1 + floor( ang/dtheta + epu );
        Ixx=  cos_table( table_index );
        Qxx= -sin_table( table_index );

        dphi= -ang + thetas(table_index);
    otherwise
        Ixx= 0;
        Qxx= 0;
end
%
%   Apply third-order correction
%
alpha= 1 - 0.50*dphi^2;
beta= dphi*( 1 - dphi^2/6 );        % Approx to sin(dphi)
%
Ix= Ixx*alpha - Qxx*beta;
Qx= Qxx*alpha + Ixx*beta;

end
```

The second MATLAB script which follows uses one full-length table of sin() values covering [-π, π) with index manipulation to provide both sin() and cos() values. Owing to the full-length table, quadrant-related computations are completely avoided, making this script better suited for CLA-based computations.


```
%================ u28483_trial_sin_cos.m ================================
%
%   MATLAB trial for sin() and cos() approximations
%   Uses one 4-quadrant sin() table
%   Table quadrant-symmetries not exploited in order to keep CLA code
%   more simple
%
%   J.A. Crawford
%   19 Feb 2023
%
%
function u28483_trial_sin_cos( table_points, theta_min, theta_max, npts )
%
global sin_table;
global cos_table;
global pi_over_2;
global two_pi;
global dtheta;
global thetas;

disp( '==================');
disp( ' ' );

pi_over_2= pi/2;
two_pi= 2*pi;

dtheta= two_pi / ( table_points - 0  );
disp( ['dtheta=', ' ', num2str(dtheta) ] );

%
%   Build sine table
%
sin_table= zeros(1, table_points);
thetas= sin_table;

for ii=1:table_points
    thetas(ii)= -pi + (ii-1)*dtheta;
    sin_table(ii)= sin( thetas(ii) );
    cos_table(ii)= cos( thetas(ii) );
end

prt_case= 4;
for ii=1:table_points
    switch( prt_case )
        case 1
            %
            %   Print thetas
            %
            disp( strcat( num2str( thetas(ii), '%10.8f\n' ) ) );
        case 2
            %
            %   Print sin()
            %
            disp( strcat( num2str( sin_table(ii), '%10.8f\n' ) ) );
        case 3
            %
            %   Print theta, sin()
            %
            disp( strcat( [num2str( thetas(ii), '%10.8f' ), '      ', num2str(
sin_table(ii), '%10.8f\n' ) ] ) );
        case 4
            %
            %   Print theta sin(), cos()
            %
            disp( strcat( [num2str(ii), '   ',num2str( thetas(ii), '%10.8f' ), '      ',
num2str( sin_table(ii), '%10.8f' ), ...
                                                                    '      ',
num2str( cos_table(ii), '%10.8f\n' )] ) );
```



```
    end
end

theta_plot= theta_min + (theta_max - theta_min)*(0:npts-1)/(npts-1);

err_cos= zeros(1,npts);
err_sin= zeros(1,npts);
for ii=1:npts
    ideal_cos= cos(theta_plot(ii));
    ideal_sin= sin(theta_plot(ii));
    [Ix,Qx]= my_cos_sin( theta_plot(ii), table_points );
    %
    %   Compute approx values for sine and cosine
    %
    err_cos(ii)= ideal_cos - Ix;
    err_sin(ii)= ideal_sin - Qx;
end

fig1= figure(1);
clf;
axes( 'FontName', 'Arial', 'FontSize', 12 );
p1= plot( theta_plot, err_cos, 'r' );
set( p1, 'LineWidth', 2 );
hold on
p1= plot( theta_plot, err_sin, 'b' );
set( p1, 'LineWidth', 2 );
xlabel( 'Phase, rad', 'FontName', 'Arial', 'FontSize', 12 );
ylabel( 'Error', 'FontName', 'Arial', 'FontSize', 12 );
title( 'Sin() Cos() Approximation Error', 'FontName', 'Arial', 'FontSize', 14 );
grid on
h= gca;
set( h, 'LineWidth', 2 );

txt1= strcat( ['Table Look-Up Size=', ' ', num2str(table_points) ] );
annotation(fig1,'textbox','String',{txt1},'FontSize',12,...
    'FontName','Arial',...
    'FitBoxToText','off',...
    'LineStyle','none',...
    'BackgroundColor',[1 1 1],...
    'Position',[0.1757 0.7655 0.2093 0.09595]);

legend( 'Cos Error', 'Sin Error' );

end
%=========================================================================


function [Ix, Qx]= my_cos_sin( theta, table_points )
%
%   Approximation function for cos() and sin()
%
%   theta is the angle of interest, rad
%   table_points is the number of table look-up points
%
global sin_table;       % Table look-up for 1st quadrant
global thetas;          % Angles, rad, associated with table look-up

global two_pi;
global dtheta;

prt_on= false;

%
%   Constrain angle to within -pi to +pi
%
if( prt_on )
    disp( ['theta in = ', num2str(theta) ] );
```

```
end

if( abs(theta) > pi )
    theta= theta - two_pi*floor(theta/two_pi);
end

if( prt_on )
    disp( ['theta out=', ' ', num2str(theta) ] );
end

table_index= floor( theta/dtheta + 0.5 ) + table_points/2 + 1;
if( table_index > table_points )
    table_index= table_index - table_points;
end
if( table_index < 1 )
    table_index= table_index + table_points;
end
Qxx= sin_table( table_index );

dphi= theta - thetas( table_index );
if( dphi > pi/4 )                    % dphi should be small
    dphi= dphi - 2*pi;
end
if( dphi < -pi/4 )                   % dphi should be small
    dphi= dphi + 2*pi;
end

table_index2= table_index + table_points/4;  % Exploit table symmetry
if( table_index2 > table_points )
    table_index2= table_index2 - table_points;
end
Ixx= sin_table( table_index2 );

if( prt_on )
    disp( ['theta=', ' ', num2str(theta), ' ','table index=', ' ', num2str(table_index),
' ', ...
          'theta table=', ' ', num2str(thetas(table_index)), ' ', 'sin=', ' ',
num2str(Qxx), ' ', 'cos=', ' ', num2str(Ixx) ] );
end

%
%   Apply third-order correction
%
alpha= 1 - 0.50*dphi^2;
beta= dphi*( 1 - dphi^2/6 );        % Approx to sin(dphi)
%
Ix= Ixx*alpha - Qxx*beta;
Qx= Qxx*alpha + Ixx*beta;
if( prt_on )
    disp( ['                          Polished sin=', ' ', num2str(Qx), ' ', 'Polished
cos=', ' ', num2str(Ix) ] );
    disp( ['                          Exact sin=', ' ', num2str(sin(theta)), ' ', 'Exact
cos=', ' ', num2str(cos(theta)) ] );
    disp( ['                           sin err=', ' ', num2str(Qx-sin(theta)), ' ', 'cos
err=', ' ', num2str(Ix-cos(theta)) ] );
    disp( ['                           alpha=', ' ', num2str(alpha), '  ', 'beta=', ' ',
num2str(beta) ] );
    disp( ' ' );
    pause
end

end
```

9 Appendix: ePWM Programming

Getting on track with ePWM programming can be rather challenging. Prior to about 2019, the C2000 software from Texas Instruments (TI) was organized differently than it is now and the older literature can consequently be difficult to follow. C2000Ware is the successor to controlSUITE as the centralized, interactive, software repository for everything C2000[21]. In addition, not all motor control application notes make use of controlSUITE or C2000Ware. And with this change, C2000Ware, unlike controlSUITE, is versioned at the packed level thereby resulting in a separate directory installation for each revision! Consequently, this naturally leads to multiple versions of C2000Ware being installed at the same time.

> A potentially helpful full delineation of the ePWM bit-level control parameters can be found in F2837xD_epwm.h and F2837xD_epwm_xbar.h.

TI also provides their *SysConfig* utility for configuring the ePWM module along with many other modules within the F2837xD device. Picking which design path to adopt again requires some *measuring of the cost before starting the journey* because each option presents positives as well as negatives.

It is easy to get a bit overwhelmed with the enhanced pulse width modulator (ePWM) programming of the TMS320F28379D as there are 7,398 lines of code in the epwm.h file, and 1,650 lines of code in the hrpwm.h file. Example files focusing on the ePWM blocks are fairly minimal as well, and one must dive into the full-control code examples (e.g., IDDK_PM_Servo_F28379x) to see the full scope of the ePWM blocks in use.

The technical reference manual [11] devotes 89 pages to the ePWM function plus an additional 25 pages to discuss the high-resolution pulse width modulator function. The register map discussion is 132 pages long itself.

The F28379D's ePWM modules provide far more features, however, than required for my motor control application. This leads to a lot of programming simplification.

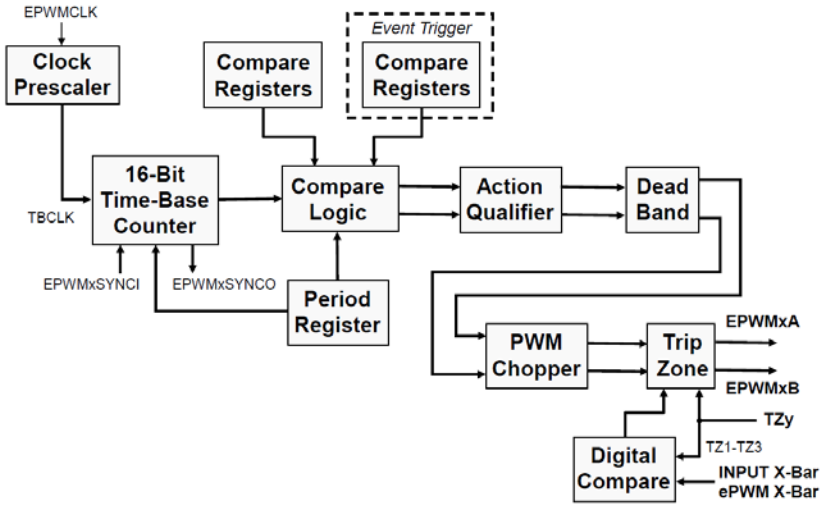


Figure 46 ePWM module block diagram[22]

[21] "controlSUITE to C2000Ware Transition Guide," SPRUI45C, Revised Dec. 2019, U28389.
[22] Figure 11 of [10].

A 3-phase inverter arrangement suitable for permanent magnet synchronous motors is shown[23] in Figure 47. Each leg must switch at the same frequency and all legs must be synchronized for proper operation. A master plus two slaves can fulfill this need. The associated timing diagram is shown in Figure 48.

A 3-phase interleaved configuration can be constructed by using the phase registers (TBPHS) of each PWM. A PWM module can be configured (see Figure 49) to allow a SyncIn pulse to cause the TBPHS register to be loaded into the TBCTR register thereby maintaining the desired phase relationships.

> One subtle point worth mentioning is the use of the Figure 48 timing. Since the ePWM pulse widths are all tied to their respective CA triggers, the distance from the *center* of each pulse remains constant. This means the effective time delay between pulse centers remains constant as well. This is of vital importance in most applications because the effective time delay through the block would otherwise vary. Using the ePWM in up/down mode ensures the group-delay remains constant.

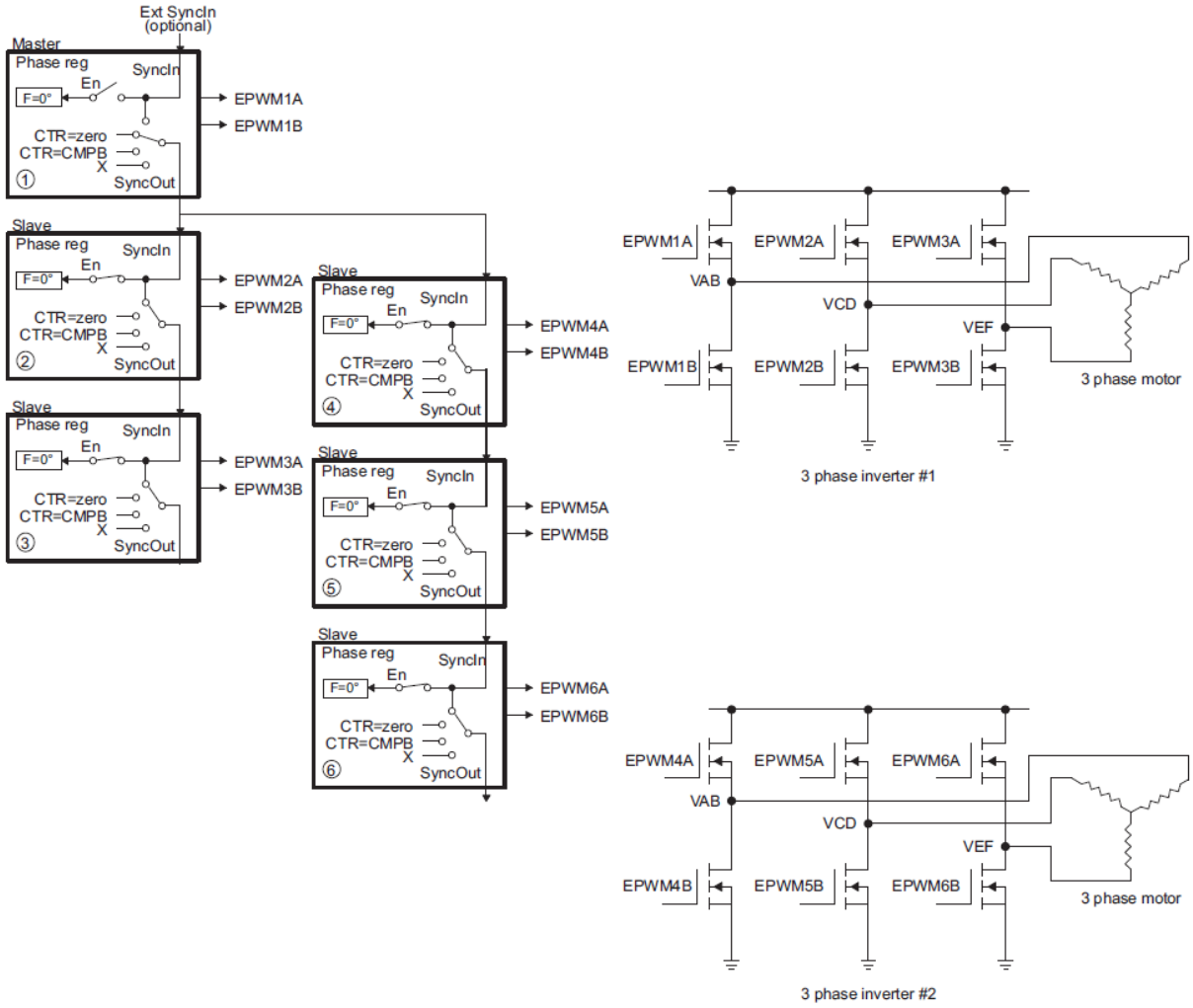


Figure 47 Control of dual 3-phase inverter stages as is commonly used in motor control[24]

[23] Figure 15-66 from [11].
[24] Figure 15-66 from [11].

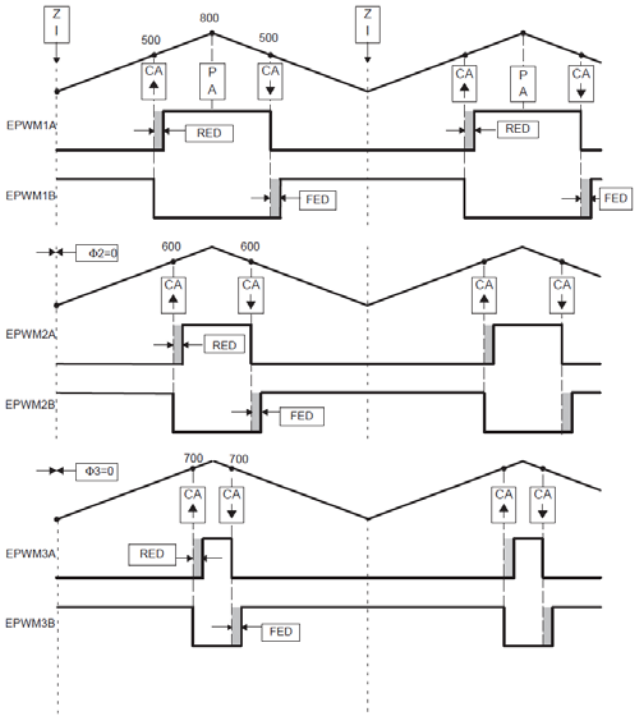


Figure 48 Phase inverter diagrams associated with Figure 47 illustrating only one of the 3-phase inverters shown operating at different output pulse widths

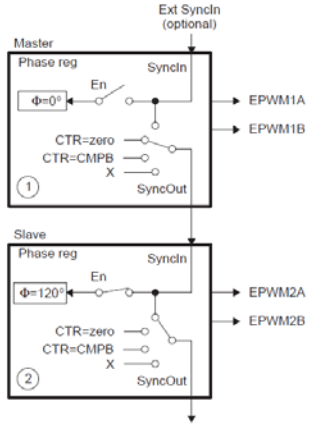


Figure 49

9.1 Direct Copy from [10]

The time-base submodule consists of a dedicated 16-bit counter, along with built-in synchronization logic to allow multiple ePWM modules to work together as a single system. A clock pre-scaler divides the EPWM clock to the counter and a period register is used to control the frequency and period of the generated waveform. The period register has a shadow register, which acts like a buffer to allow the register updates to be synchronized with the counter, thus avoiding corruption or spurious operation from the register being modified asynchronously by the software. The time-base counter operates in three modes: up-count, down-count, and up-down-count. In up-count mode the time-base counter starts counting from zero and increments until it reaches the period register value, then the time-base counter

resets to zero and the count sequence starts again. Likewise, in down-count mode the time-base counter starts counting from the period register value and decrements until it reaches zero, then the time-base counter is loaded with the period value and the count sequence starts again. In up-down-count mode the time-base counter starts counting from zero and increments until it reaches the period register value, then the time-base counter decrements until it reaches zero and the count sequence repeats. The up-count and down-count modes are used to generate asymmetrical waveforms, and the up-down-count mode is used to generate symmetrical waveforms.

The counter-compare submodule continuously compares the time-base count value to four Counter Compare Registers (CMPA, CMPB, CMPC, and CMPD) and generates four independent compare events (that is, time-base counter equals a compare register value) which are fed to the action-qualifier and event-trigger submodules. The counter compare registers are shadowed to prevent corruption or glitches during the active PWM cycle. Typically CMPA and CMPB are used to control the duty cycle of the generated PWM waveform, and all four compare registers can be used to start an ADC conversion or generate an ePWM interrupt. For the up-count and down-count modes, a counter match occurs only once per cycle, however for the up-down-count mode a counter match occurs twice per cycle since there is a match on the up count and down count.

The action-qualifier submodule is the key element in the ePWM module which is responsible for constructing and generating the switched PWM waveforms. It utilizes match events from the time-base and counter-compare submodules for performing actions on the EPWMxA and EPWMxB output pins. These actions are setting the pin high, clearing the pin low, toggling the pin, or do nothing to the pin, based independently on count-up and count-down time-base match event. The match events are when the time-base counter equals the period register value, the time-base counter is zero, the time-base counter equals CMPA, the time-base counter equals CMPB, or a Trigger event (T1 and T2) based on a comparator, trip, or sync signal. Note that zero and period actions are fixed in time, whereas CMPA and CMPB actions are moveable in time by programming their respective registers. Actions are configured independently for each output using shadowed registers, and any or all events can be configured to generate actions on either output.

The dead-band submodule provides a classical approach for delaying the switching action of a power device. Since power switching devices turn on faster than they turn off, a delay is needed to prevent having a momentary short circuit path from the supply rail to ground. This submodule supports independently programmable rising-edge and falling-edge delays with various options for generating the appropriate signal outputs on EPWMxA and EPWMxB.

The trip-zone submodule utilizes a fast clock independent logic mechanism to quickly handle fault conditions by forcing the EPWMxA and EPWMxB outputs to a safe state, such as high, low, or high impedance, thus avoiding any interrupt latency that may not protect the hardware when responding to over current conditions or short circuits through ISR software. It supports one-shot trips for major short circuits or over current conditions, and cycle-by-cycle trips for current limiting operation. The trip-zone signals can be generated externally from any GPIO pin which is mapped through the Input X-Bar (TZ1 – TZ3), internally from an inverted eQEP error signal (TZ4), system clock failure (TZ5), or from an emulation stop output from the CPU (TZ6). Additionally, numerous trip-zone source signals can be generated from the digital-compare subsystem.

The digital-compare subsystem compares signals external to the ePWM module, such as a signal from the CMPSS analog comparators, to directly generate PWM events or actions which are then used by the trip-zone, time-base, and event-trigger submodules. These 'compare' events can trip the ePWM module, generate a trip interrupt, sync the ePWM module, or generate an ADC start of conversion. A compare event is generated when one or more of its selected inputs are either high or low. The signals can originate from any external GPIO pin which is mapped through the Input X-Bar and from various internal peripherals which are mapped through the ePWM X-Bar. Additionally, an optional 'blanking' function can be used to temporarily disable the compare action in alignment with PWM switching to eliminate noise effects.

The event-trigger submodule manages the events generated by the time-base, counter-compare, and digital-compare submodules for generating an interrupt to the CPU and/or a start of conversion pulse to the ADC when a selected event occurs. These event triggers can occur when the time-base counter equals zero, period, zero or period, the up or down count match of a compare register (that is, CMPA, CMPB, CMPC, or CMPD). Recall that digital-compare subsystem can also generate an ADC start of conversion based on one or more compare events. The event-trigger submodule incorporates pre-scaling logic to issue an interrupt request or ADC start of conversion at every event or up to every fifteenth event. The ePWM module is capable of significantly increase its time resolution capabilities over the standard conventionally derived digital PWM. This is accomplished by adding 8-bit extensions to the High-Resolution Compare Register (CMPxHR), Time Base Period High Resolution Register (TBPRDHR), and High-Resolution Phase Register (TBPHSHR), providing a finer time granularity for edge positioning control. This is known as high-resolution PWM (HRPWM) and it is based on micro edge positioner (MEP) technology.

The MEP logic is capable of positioning an edge very finely by sub-dividing one coarse system clock of the conventional PWM generator with time step accuracy on the order of 150 ps. A self-checking software diagnostics mode is used to determine if the MEP logic is running optimally, under all operating conditions such as for variations caused by temperature, voltage, and process. HRPWM is typically used when the PWM resolution falls below approximately 9 or 10 bits which occurs at frequencies greater than approximately 200 kHz with an EPWMCLK of 100 MHz.

If the ePWM update rate is taken to be 32 kHz, the time available between updates would be 31.25 μs with the number of 200 MHz clock ticks equal to 6,250. With the instruction execution efficiency of the TMS320F28379D, this is quite a few clocks. And this number can effectively be doubled if both DSP cores are used.

9.1.1 Control Law Accelerator (CLA)

The CLA is an independent 32-bit floating-point math hardware accelerator which executes real-time control algorithms in parallel with the main C28x CPU, effectively doubling the computational performance.

Each CPU subsystem has its own CLA that responds directly to peripheral triggers, which can free up the C28x CPU for other tasks, such as communications and diagnostics. With direct access to the various control and communication peripherals, the CLA minimizes latency, enables a fast trigger response, and avoids CPU overhead. Also, with direct access to the ADC results registers, the CLA is able to read the result on the same cycle that the ADC sample conversion is completed, providing “just-in-time” reading, which reduces the sample to output delay.

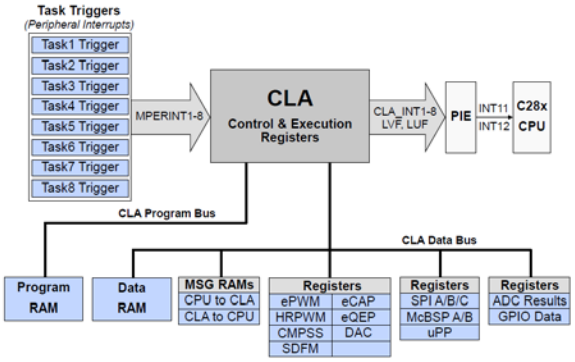


Figure 50 Control Law Accelerator (CLA) block diagram

The CLA has access to the LSx RAM blocks and each memory block can be configured to be either

dedicated to the CPU or shared between the CPU and CLA. After reset the memory block is mapped to the CPU, where it can be initialized by the CPU before being shared with the CLA. Once it is shared between the CPU and CLA it then can be configured to be either program memory or data memory. When configured as program memory it contains the CLA program code, and when configured as data memory it contains the variable and coefficients that are used by the CLA program code. Additionally, dedicated message RAMs are used to pass data between the CPU and CLA, and CLA and CPU.

Programming the CLA consists of initialization code, which is performed by the CPU, and tasks. A task is similar to an interrupt service routine, and once started it runs to completion. Tasks can be written in C or assembly code, where typically the user will use assembly code for high performance time-critical tasks, and C for non-critical tasks. Each task is capable of being triggered by a variety of peripherals without CPU intervention, which makes the CLA very efficient since it does not use interrupts for hardware synchronization, nor must the CLA do any context switching. Unlike the traditional interrupt-based scheme, the CLA approach becomes deterministic. The CLA supports eight independent tasks and each is mapped back to an event trigger. Since the CLA is a software programmable accelerator, it is very flexible and can be modified for different applications.

9.1.2 Inter-Processor Communications (IPC)

The IPC module facilitates communications between the two CPU subsystems, and all IPC features are independent of each other. As discussed in the Memory section, there are two dedicated 1Kx16 blocks of Message RAM that are used to transfer messages or data between CPU1 and CPU2. One block configuration is fixed for “CPU1 to CPU2”, and the other block configuration is fixed for “CPU2 to CPU1”.

Messaging can be accomplished using IPC flags and interrupts. There are 32 IPC event signals from CPU1 to CPU2, and vice-versa. These signals can be used for flag-based event polling and four of them (IPC0 – IPC3) can be configured to generate IPC interrupts on the remote CPU. IPC Command registers provide a simple and flexible means for CPU1 and CPU2 to exchange more complex messages. Each CPU has eight dedicated registers, four for sending messages and four for receiving messages. On the local CPU, three are writeable registers and one is a read-only register. These same registers are accessible on the remote CPU as three read-only registers and one writeable register. The given register names were chosen to support a simple command/response protocol, but they can be used for any purpose to suit the applications software.

A variety of options exist for supporting IPC. The basic option does not require any software drivers and uses only the IPC registers for simple message passing. An IPC-Lite software API driver uses only the IPC registers (that is, no memory used), but is limited to one IPC interrupt or one IPC command/message at a time. The full IPC software API driver uses circular buffers for message RAMs, and can queue up to four messages prior to processing. It can also be used with multiple IPC ISRs at a time, but it requires additional setup in the application code prior to use. Each option has tradeoffs between complexity, processing overhead, and messaging capabilities.

10 Appendix: Additional Reference Materials

Table 10-1 Reference Materials

Title	#	Comments
TMS320C28x Optimizing C/C++ Compiler v20.8.0.STS, U27392.	U27392	Compiling, linking, optimization, pragma directives
TMS320F2837xD Dual-Core Delfino Microcontrollers, Technical Reference Manual, U26369.	U26369	Interrupts, timers, memory, DMA, CLA overview & building application**
Sensorless Field Oriented Control of 3-Phase Induction Motors Using Control Law Accelerator (CLA)	U27328	Digital motor control on CLA**
C2000 CLA Software Development Guide	U27590	CLA introduction
Programming TMS320x28xx and TMS320x28xxx Peripherals in C/C++, U26359.	U26359	
F2837xD Firmware Development Package, U27306.	U27306	
Sensored Field Oriented Control of 3-Phase Permanent Magnet Synchronous Motors Using F2837x, U27432.	U27432	
Sensored Field Oriented Control of 3-Phase Permanent Magnetic Synchronous Motors Using TMS320F2837x, U27302, U27322.	U27302 U27322	
Dual-Axis Motor Control Using FCL and SFRA on a Single C2000 MCU	U27427	

11 Appendix: Parting Shots

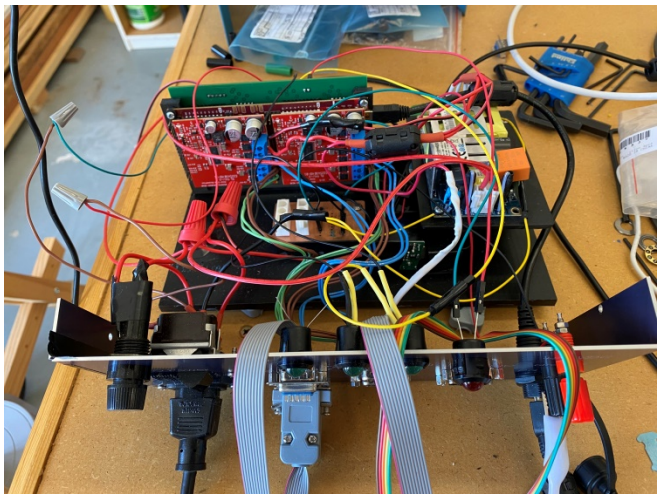


Figure 51 Electronics chassis opened up, showing the TMS320F28379D Launchpad board, piggy-backed with two 3-phase motor controllers, backed by a custom-designed break-out PCB, with an AC-to-24VDC switching power supply

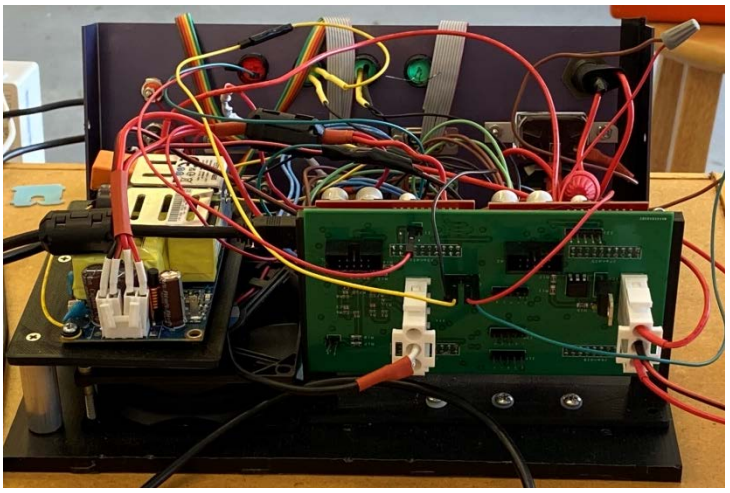


Figure 52 Back-side view of the chassis electronics showing the custom-designed break-out PCB I designed to direct signals from the DSP Launchpad to DB-9 connectors and other. In total, there are over 100 I/O lines associated with the DSP processor and another 50+ associated with the DB-9 connectors, LEDs, etc.

Figure 53 Front-side of the electronics chassis which I milled, engraved, and anodized. Front panel anodizing was the subject of a separate memo [12]. This picture was taken during development in which the chassis has not been closed up yet.

Figure 54 Bottom of electronics chassis at the bottom secured to the mount plate (see Figure 1), the full mount, open electronics chassis on the left, Haas mill in the background

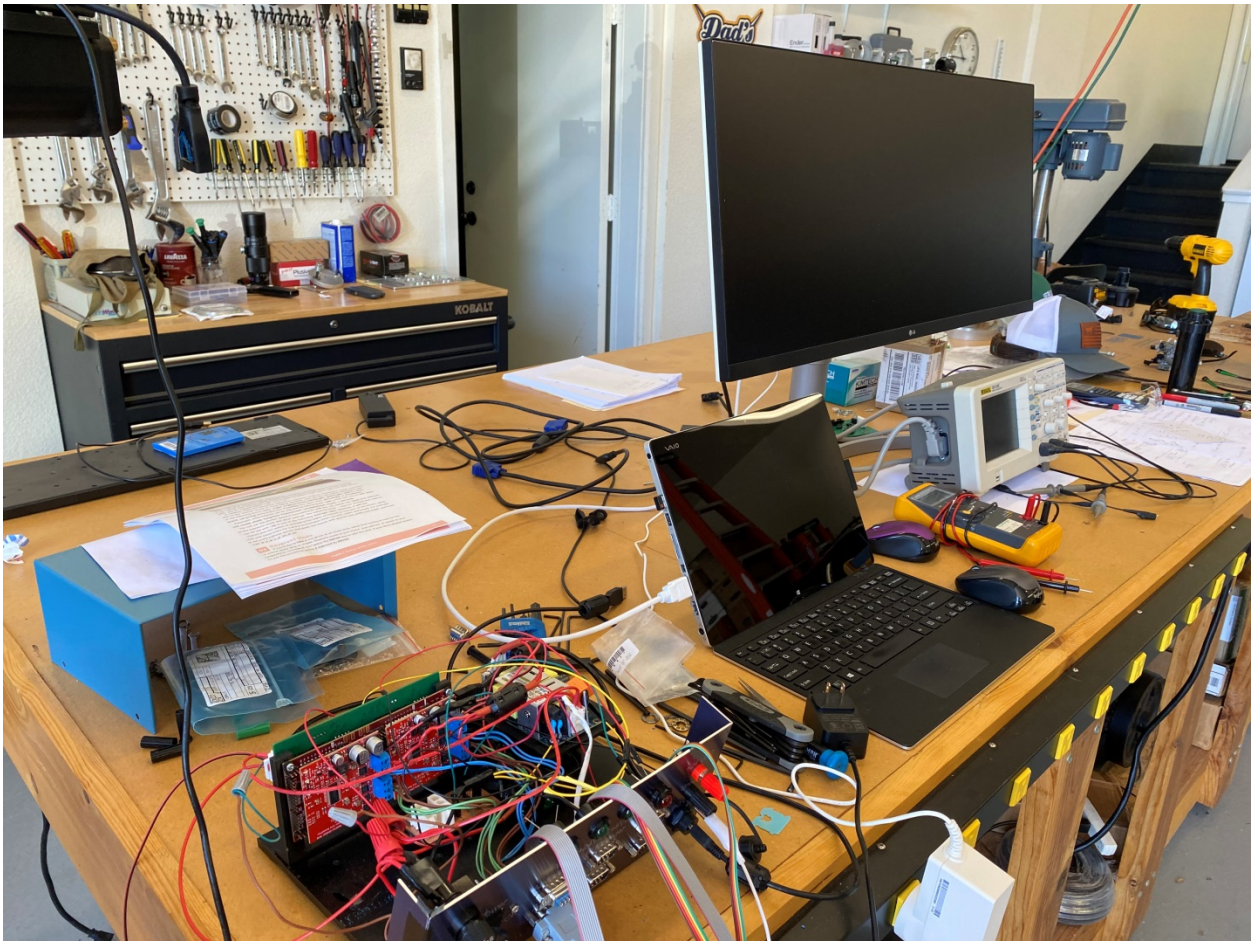


Figure 55 A great deal of the software has been written in my office, but full-up/real-time/hands-on code adjustments on the hardware is done in my shop using a Vaio laptop and large 35” diagonal monitor. The Vaio ties into a network appliance which hosts all of the code.