ADC Resolution vs Sensor Less Motor Control Performance

Cristian Ornelas

Tyler Hawkins

Tamara Hussam A Basfar

John Adam King

**Subsystem Reports**

REVISION – Original

2 December 2024

Subsystem Reports

for

ADC Resolution vs Sensor Less Motor Control Performance

Team <1>

Approved by:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Project Leader Date

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof. Kalafatis Date

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

T/A Date

**Change Record**

| **Rev.** | **Date** | **Originator** | **Approvals** | **Description** |
| --- | --- | --- | --- | --- |
| **1** | [12/02/2024] | [ADC Resolution vs Sensor Less Motor Control Performance] |  | Original Release |

**Table of Contents**

[**Table of Contents**](#_30j0zll) **4**

[**List of Figures**](https://docs.google.com/document/d/1VWHH3bZeuPOXZgesPccBDEj311N5xgtd/edit#heading=h.tyjcwt) **5**

[**1.** **Introduction**](https://docs.google.com/document/d/1VWHH3bZeuPOXZgesPccBDEj311N5xgtd/edit#heading=h.1t3h5sf) **7**

[**2.** **PWM subsystem Report**](https://docs.google.com/document/d/1VWHH3bZeuPOXZgesPccBDEj311N5xgtd/edit#heading=h.4d34og8)  8

[2.1. Subsystem Introduction](https://docs.google.com/document/d/1VWHH3bZeuPOXZgesPccBDEj311N5xgtd/edit#heading=h.2s8eyo1)8

[2.2.](https://docs.google.com/document/d/1VWHH3bZeuPOXZgesPccBDEj311N5xgtd/edit#heading=h.17dp8vu) [Subsystem](https://docs.google.com/document/d/1VWHH3bZeuPOXZgesPccBDEj311N5xgtd/edit#heading=h.2s8eyo1) [Design](https://docs.google.com/document/d/1VWHH3bZeuPOXZgesPccBDEj311N5xgtd/edit#heading=h.17dp8vu) 6

2.2.1 Software Overview 8

2.2.2 submodule Description 9

2.2.3 submodule Interactions 9

[2.3.](https://docs.google.com/document/d/1VWHH3bZeuPOXZgesPccBDEj311N5xgtd/edit#heading=h.3rdcrjn) [Subsystem](https://docs.google.com/document/d/1VWHH3bZeuPOXZgesPccBDEj311N5xgtd/edit#heading=h.2s8eyo1) Validation10

2.3.1 Angle Generator 10

2.3.2 Inverse Park11

2.3.3 Space Vector Modulation12

2.3.4 PWM Driver12

2.4 [Subsystem](https://docs.google.com/document/d/1VWHH3bZeuPOXZgesPccBDEj311N5xgtd/edit#heading=h.2s8eyo1) Conclusion13

[**3.** **ADC**](https://docs.google.com/document/d/1VWHH3bZeuPOXZgesPccBDEj311N5xgtd/edit#heading=h.lnxbz9) **S**[**ubsystem Report**](https://docs.google.com/document/d/1VWHH3bZeuPOXZgesPccBDEj311N5xgtd/edit#heading=h.4d34og8) **13**

[3.1. Subsystem Introduction](https://docs.google.com/document/d/1VWHH3bZeuPOXZgesPccBDEj311N5xgtd/edit#heading=h.35nkun2) 13

[3.2.](https://docs.google.com/document/d/1VWHH3bZeuPOXZgesPccBDEj311N5xgtd/edit#heading=h.1ksv4uv) Subsystem Design14

3.2.1 Software Overview 14

3.2.2 Submodule Description 14

3.2.3 Submodule Interactions 15

[3.3.](https://docs.google.com/document/d/1VWHH3bZeuPOXZgesPccBDEj311N5xgtd/edit#heading=h.44sinio) Subsystem Validation16

3.3.1 Memory Buffer 16

3.3.2 ADC Buffer PWM Waveform 17

3.3.3 LED Behavior 17

3.4 Subsystem Conclusion 17

[**4.** **Estimator**](https://docs.google.com/document/d/1VWHH3bZeuPOXZgesPccBDEj311N5xgtd/edit#heading=h.1y810tw) [**subsystem Report**](https://docs.google.com/document/d/1VWHH3bZeuPOXZgesPccBDEj311N5xgtd/edit#heading=h.4d34og8)**18**

4.1. Subsystem Introduction 18

4.2. Subsystem Design 18

4.2.1 Software Overview 19

4.2.2 submodule Interactions 19

4.3. Subsystem Validation 19

4.3.1 Mathematics Validation 19

4.3.2 Ideal Condition Tests 19

4.3.3 Non-Ideal Condition Tests 21

4.3.4 PWM Driver 21

4.4 Subsystem Conclusion 22

[**5.** **Porting**](https://docs.google.com/document/d/1VWHH3bZeuPOXZgesPccBDEj311N5xgtd/edit#heading=h.2xcytpi) [**subsystem Report**](https://docs.google.com/document/d/1VWHH3bZeuPOXZgesPccBDEj311N5xgtd/edit#heading=h.4d34og8)**22**

[5.1.](https://docs.google.com/document/d/1VWHH3bZeuPOXZgesPccBDEj311N5xgtd/edit#heading=h.1ci93xb) Subsystem Introduction22

[5.2. Subsystem Details](https://docs.google.com/document/d/1VWHH3bZeuPOXZgesPccBDEj311N5xgtd/edit#heading=h.3whwml4) 23

[5.3. Subsystem Validation](https://docs.google.com/document/d/1VWHH3bZeuPOXZgesPccBDEj311N5xgtd/edit#heading=h.2bn6wsx) 23

[5.4. Subsystem Conclusion](https://docs.google.com/document/d/1VWHH3bZeuPOXZgesPccBDEj311N5xgtd/edit#heading=h.2bn6wsx) 24

**List of Figures**

Fig 1[. Subsystem Diagram](https://docs.google.com/document/d/1VWHH3bZeuPOXZgesPccBDEj311N5xgtd/edit#heading=h.2s8eyo1) 7

Fig 2[.](https://docs.google.com/document/d/1VWHH3bZeuPOXZgesPccBDEj311N5xgtd/edit#heading=h.17dp8vu) PWM subsystem software flow8

Fig 3[.](https://docs.google.com/document/d/1VWHH3bZeuPOXZgesPccBDEj311N5xgtd/edit#heading=h.3rdcrjn) Output for angle generator at 10Hz10

Fig 4[.](https://docs.google.com/document/d/1VWHH3bZeuPOXZgesPccBDEj311N5xgtd/edit#heading=h.3rdcrjn) Output for angle generator at 50Hz10

Fig 5[.](https://docs.google.com/document/d/1VWHH3bZeuPOXZgesPccBDEj311N5xgtd/edit#heading=h.3rdcrjn) Output for angle generator at 100Hz11

Fig 6[.](https://docs.google.com/document/d/1VWHH3bZeuPOXZgesPccBDEj311N5xgtd/edit#heading=h.3rdcrjn) Inverse park input and output waveforms11

Fig 7[.](https://docs.google.com/document/d/1VWHH3bZeuPOXZgesPccBDEj311N5xgtd/edit#heading=h.3rdcrjn) SVM input and output waveforms12

Fig 8[. PWM Signals](https://docs.google.com/document/d/1VWHH3bZeuPOXZgesPccBDEj311N5xgtd/edit#heading=h.3rdcrjn) 13

Fig 9[.](https://docs.google.com/document/d/1VWHH3bZeuPOXZgesPccBDEj311N5xgtd/edit#heading=h.3rdcrjn) Analog-to-Digital Converter Block Diagram 14

Fig 10[.](https://docs.google.com/document/d/1VWHH3bZeuPOXZgesPccBDEj311N5xgtd/edit#heading=h.3rdcrjn) ADCA High/Low Toggle Conversion Behavior in Memory Buffer 16

Fig 11[.](https://docs.google.com/document/d/1VWHH3bZeuPOXZgesPccBDEj311N5xgtd/edit#heading=h.3rdcrjn) ADCB High Only Value Conversion Behavior in Memory Buffer16

Fig 12[.](https://docs.google.com/document/d/1VWHH3bZeuPOXZgesPccBDEj311N5xgtd/edit#heading=h.3rdcrjn) ADCC Low Only Value Conversion Behavior in Memory Buffer16

Fig 13[.](https://docs.google.com/document/d/1VWHH3bZeuPOXZgesPccBDEj311N5xgtd/edit#heading=h.3rdcrjn) ADCA PWM High/Low Toggle Conversion Behavior Sawtooth Waveform17

Fig 14[.](https://docs.google.com/document/d/1VWHH3bZeuPOXZgesPccBDEj311N5xgtd/edit#heading=h.3rdcrjn) Rotor Estimator Subsystem Flow Diagram18

Fig 15[.](https://docs.google.com/document/d/1VWHH3bZeuPOXZgesPccBDEj311N5xgtd/edit#heading=h.3rdcrjn) Linear\_2 Test Data Input and Output20

Fig 16[.](https://docs.google.com/document/d/1VWHH3bZeuPOXZgesPccBDEj311N5xgtd/edit#heading=h.3rdcrjn) Rising\_2 Test Data Input and Output20

Fig 17[.](https://docs.google.com/document/d/1VWHH3bZeuPOXZgesPccBDEj311N5xgtd/edit#heading=h.3rdcrjn) Noise\_1 Test Data Input and Output21

Fig 18[.](https://docs.google.com/document/d/1VWHH3bZeuPOXZgesPccBDEj311N5xgtd/edit#heading=h.3rdcrjn) Variation\_1 Test Data Input and Output22

# 

# 

# 

# 

# 

# 

# **1.0 Introduction**

The ADC Resolution vs Sensorless Motor Control Performance project involves the development and integration of key subsystems required to achieve precise and efficient motor control without relying on physical position sensors. This approach enhances system reliability, reduces costs, and enables compact design by replacing hardware sensors with advanced computational algorithms. The project is structured around four critical subsystems: the PWM subsystem, the ADC subsystem, the estimator subsystem, and the overall porting subsystem. Each subsystem plays a pivotal role in achieving the project's goals and ensuring seamless operation of the motor.

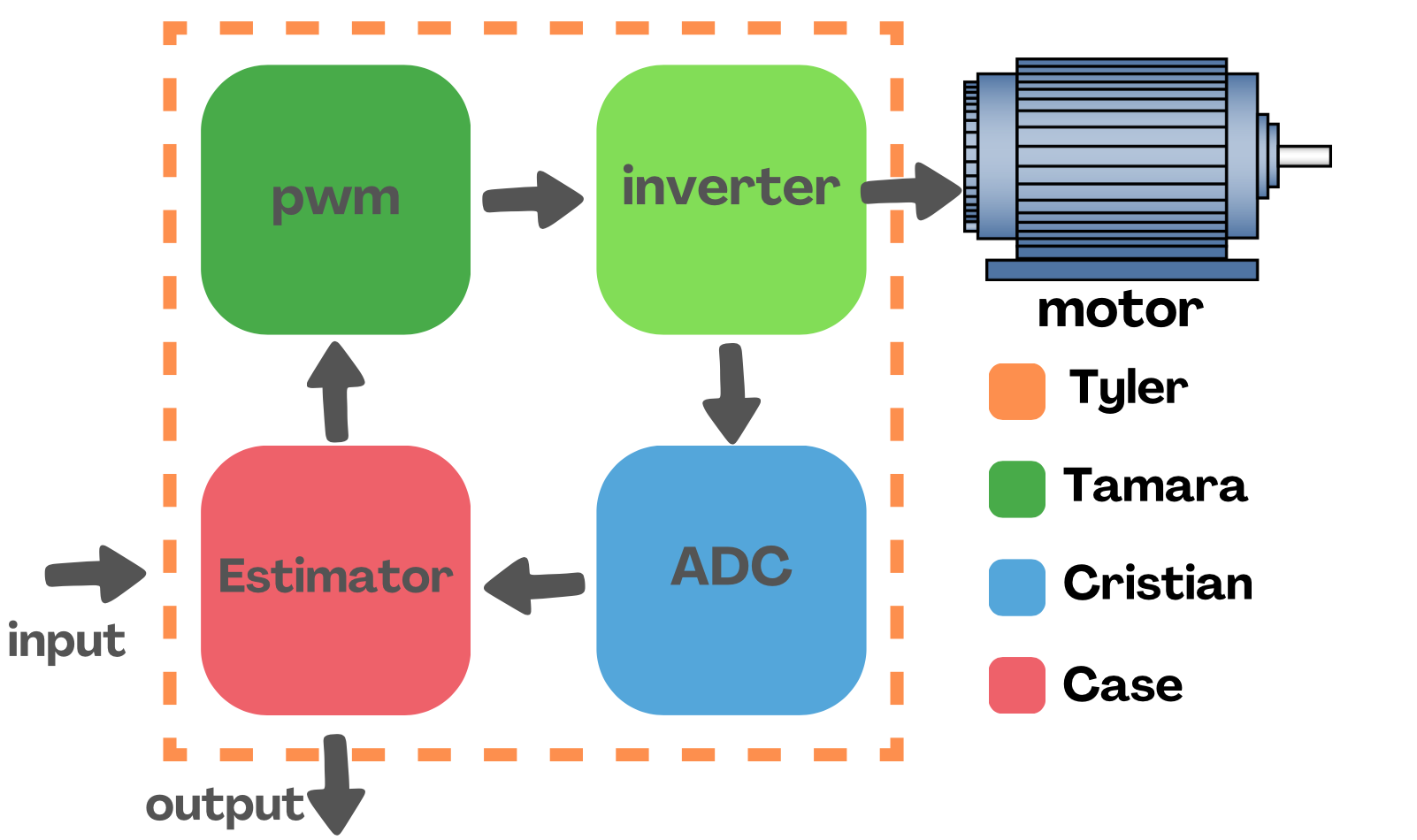


Figure 1: Subsystem Diagram

# **2.0 PWM Subsystem Report - Tamara Basfar**

## **2.1 Subsystem Introduction**

Pulse Width Modulation (PWM) is a powerful technique used in various applications to control the amount of power delivered to an electrical load without dissipating energy in the form of heat. This subsystem explores the principles and practical implementations of PWM, focusing on its applications in sensorless open-loop motor control visualized by LED dimming.

The primary objective of this subsystem was to design and implement a PWM controller that can efficiently manage power delivery in different scenarios. The design is based on the block diagram in figure 2 that includes an angle generator, an inverse Park transformation, Space Vector Modulation (SVM), and a PWM driver. By varying the duty cycle of the PWM signal, the speed, torque, and flux of the three phase motor can be controlled precisely, thereby achieving desired performance characteristics.

This report will cover the design process of the PWM controller, the submodule descriptions, and the validation of each submodule as well as the whole subsystem.

## **2.2 Subsystem Design**

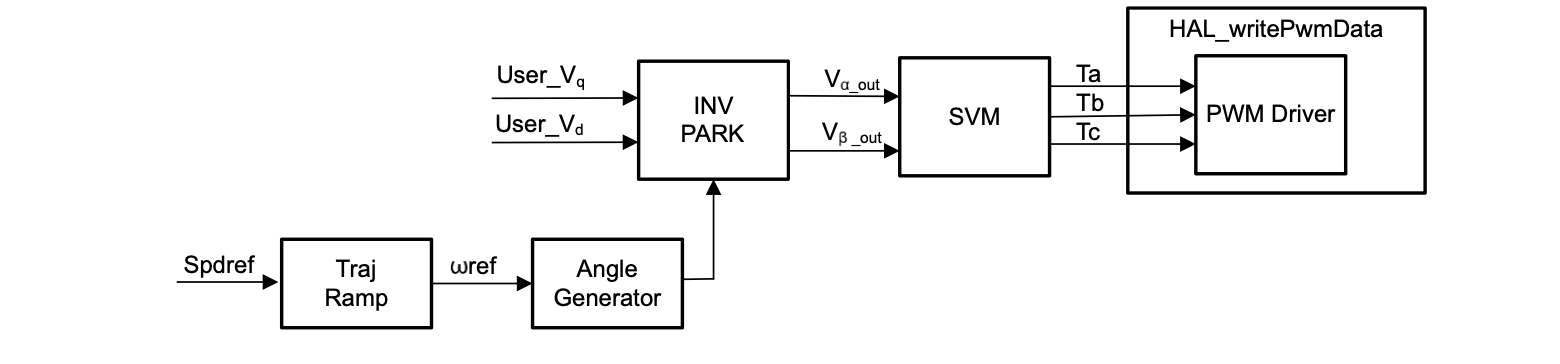


Figure 2: PWM subsystem software flow

## **2.2.1 Software Flow Overview**

The software flow for the PWM subsystem of the sensorless motor control project ensures seamless generation of control signals to drive the motor. It begins with the Angle Generator, which calculates the rotor angle (θ) based on a reference speed (ω\_ref) and updates this angle dynamically over time. This angle is passed to the Inverse Park Transformation (INV PARK) block, which converts control signals (Vq, Vd) from the rotating reference frame (d-q) into stationary reference frame signals (Vα, Vβ). These signals are then processed by the Space Vector Modulation (SVM) block, which calculates the precise duty cycles (Ta, Tb, Tc) required for the three motor phases. Finally, the PWM Driver uses these duty cycles to generate actual PWM signals that control the motor’s three-phase inverter.

## **2.2.2 Submodule Description**

The Angle Generator is responsible for computing the rotor angle (θ) based on a user-defined reference frequency. It first calculates the angle delta (angle\_delta\_rad) by multiplying the reference frequency with a time step factor. This delta is then added to the current angle (angle\_rad) to simulate the rotor's rotation. To ensure smooth operation, the angle is wrapped around to stay within the range of [-π, π] using a masking condition. This continuous and dynamic angle generation allows the control system to maintain synchronization with the rotor’s position.

The Inverse Park Transformation translates the d-q frame control voltages (Vq, Vd) into the stationary reference frame (Vα, Vβ) using trigonometric relationships. The formulas

Vα=Vdcos⁡(θ)−Vqsin⁡(θ)Vα​=Vd​cos(θ)−Vq​sin(θ), Vβ=Vdsin⁡(θ)+Vqcos⁡(θ)Vβ​=Vd​sin(θ)+Vq​cos(θ) are employed, where the angle θ is supplied by the Angle Generator. This transformation ensures that control signals designed in the d-q reference frame are compatible with the motor's stationary frame operation.

The Space Vector Modulation (SVM) block takes the stationary frame voltages (Vα, Vβ) and calculates the appropriate duty cycles (Ta, Tb, Tc) for the PWM signals. It identifies the sector of the hexagonal voltage space where the input voltages lie and determines the durations for the active and zero vectors. These durations are then used to compute the duty cycles for each phase, ensuring optimal inverter switching to produce sinusoidal currents with minimal harmonic distortion.

The PWM Driver takes the duty cycles (Ta, Tb, Tc) generated by the SVM block and applies them to the motor's three-phase inverter. This subsystem directly controls the switching of the inverter’s transistors, translating the calculated duty cycles into physical PWM signals. These signals dictate the current flow in the motor phases, effectively controlling the motor’s speed and torque.

## **2.2.2 Submodule Interactions**

The four submodules work together in a modular and synchronized manner to achieve precise PWM signal control. The Angle Generator produces a continuously updated rotor angle (θ), which is a critical input to the Inverse Park Transformation (INV PARK) block. This transformation adapts the control voltages (Vq, Vd) into stationary frame signals (Vα, Vβ) to match the motor's electrical requirements. The SVM block then processes these voltages to compute the exact duty cycles (Ta, Tb, Tc) required to generate sinusoidal phase currents. Finally, the PWM Driver converts these duty cycles into PWM signals that operate the inverter, providing efficient and accurate control of the motor.

This modular architecture offers flexibility and scalability. Each submodule is focused on a specific function, enabling ease of development and debugging. Additionally, any submodule can be modified or replaced without impacting the others, making the design robust and adaptable to future enhancements.

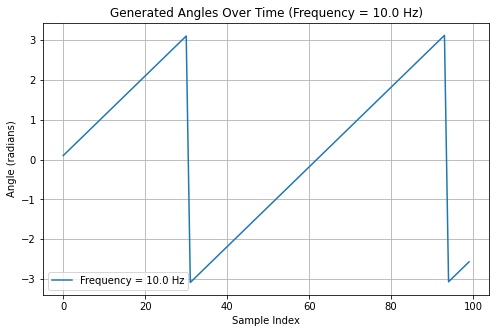
## **2.3 Subsystem Validation**

The first step of validation was to test all the functions individually to ensure that the logic was correct and the outputs behaved as expected. This approach ensured that if any errors occurred during integration, the debugging process would be more straightforward and efficient. Each function was validated against its intended behavior with controlled inputs and visualized outputs to confirm its correctness.

## **2.3.1 Angle Generator**

The first function to be developed and validated was the Angle Generator, as explained in the design section. After finalizing the necessary logic, the function was tested with various input frequencies to observe its response. The output angle was graphed to ensure it followed a sawtooth waveform, wrapping around correctly within the range of [−π,π][−π,π].

Figures 2, 3, and 4 show the outputs of the angle generator at frequencies of 10 Hz, 50 Hz, and 100 Hz, respectively. As seen in the graphs, the angle generator produces a tighter sawtooth waveform as the frequency increases, confirming that the angle delta calculation and wrap-around logic are functioning as expected.

Figure 3: output for angle generator at 10Hz Figure 4: output for angle generator at 50Hz

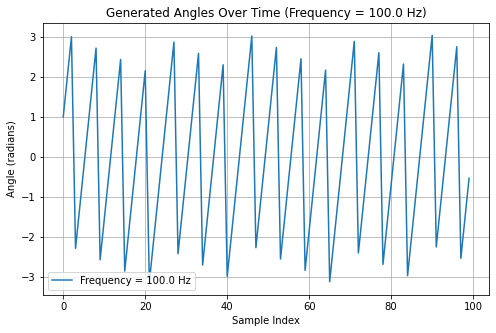


Figure 5: output for angle generator at 100Hz

## **2.3.2 Inverse Park**

The Inverse Park Transformation was validated next. This function was tested by providing a set of known d-q inputs and a controlled rotor angle (θ) from the Angle Generator. The corresponding α-β outputs were calculated and compared with theoretical values derived from the transformation equations.

Figure 5 shows the input and output waveforms for the Inverse Park Transformation. The results confirmed that the function correctly maps d-q voltages into the stationary α-β frame. The waveforms of Vα and Vβ followed the expected sinusoidal patterns, demonstrating that the trigonometric relationships were implemented correctly.

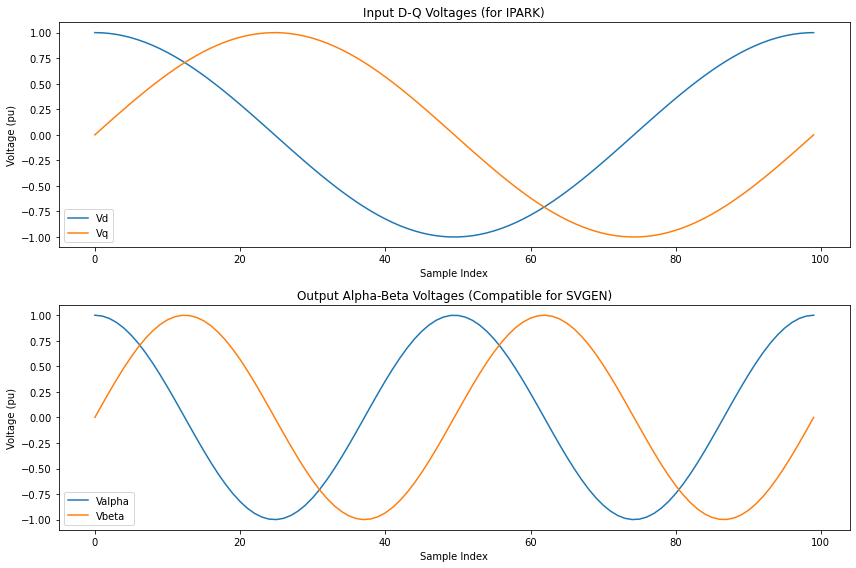


Figure 6: Inverse park input and output waveforms

## **2.3.3 Space Vector Modulation**

The Space Vector Modulation (SVM) block was validated by supplying it with the outputs from the Inverse Park Transformation and analyzing the resulting duty cycles (Ta, Tb, Tc). The validation involved ensuring that the calculated duty cycles were consistent with the input voltage space vectors and adhered to the expected PWM switching patterns.

Figure 6 illustrates the input waveforms to the SVM block, while Figure 7 shows the corresponding duty cycle outputs. The results indicate that the SVM block correctly identified the active sectors, calculated the appropriate vector durations, and generated the expected duty cycles for the three phases.

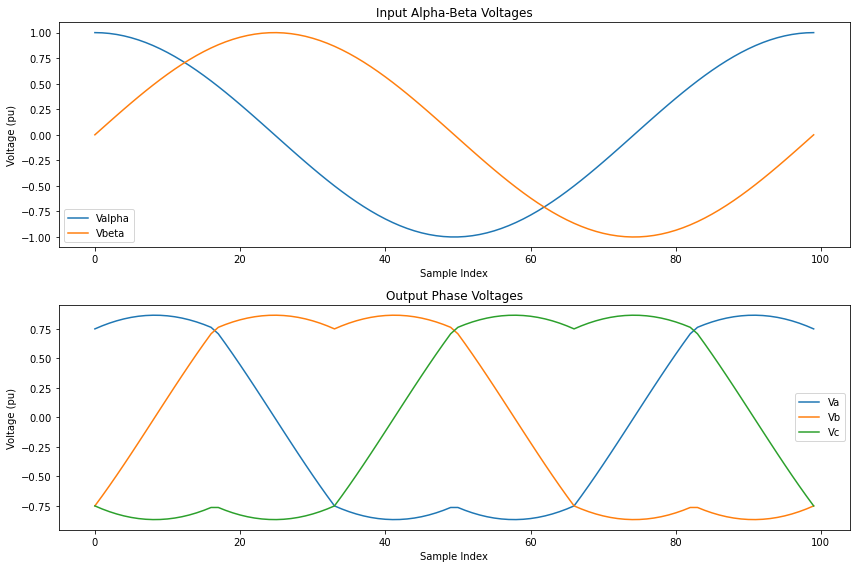


Figure 7: SVM input and output waveforms

## **2.3.3 PWM Driver**

Finally, the PWM Driver was validated by analyzing the PWM signals it generated based on the SVM duty cycles. The validation process involved capturing the PWM signals for each phase and ensuring they matched the calculated duty cycles. Oscilloscope waveforms were used to verify that the switching signals aligned correctly with the expected timing, providing smooth and balanced phase currents.

The PWM signals shown in Figure 7 demonstrate clean transitions and proper switching sequences. This confirms that the integration of the PWM Driver with the SVM block is working correctly and that the PWM signals are suitable for controlling the three-phase inverter.

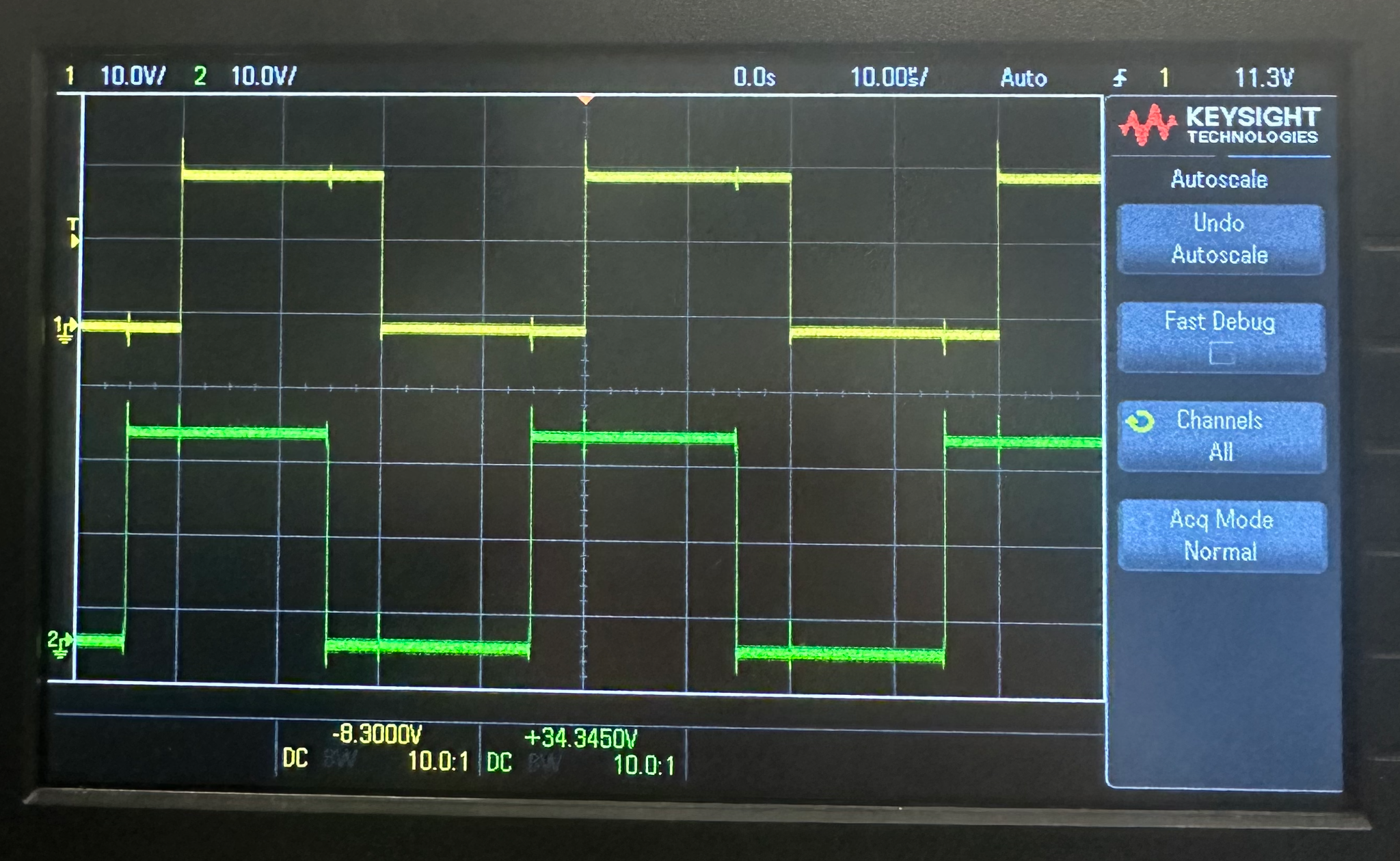


Figure 8: PWM signals

## **2.4 Subsystem Conclusion**

By testing each function individually and then integrating them step-by-step, it was ensured that each subsystem functioned as intended before moving to the next. The results from the validation process confirmed that the PWM subsystem is capable of generating accurate and efficient control signals for sensorless motor operation. The outputs of each subsystem closely matched the theoretical expectations, validating the design and implementation of the overall system.

# **3.0 ADC Subsystem Report - Cristian Ornelas**

## **3.1 – Subsystem Introduction**

Analog-to-digital conversion is an important technique used in the modern industry to convert analog signals into digital data that digital systems can process. This introduction explores the principles and practical implementations of the ADC subsystem.

The primary objective of an analog-to-digital converter in sensorless motor control is to take analog voltage and current values of the PWM/ inverter and convert them into digital values for the estimator to estimate the position of the rotor accurately. This subsystem includes the start of conversion signals/ control, post-processing blocks, input channel mux, and a reference voltage generator as shown in the figure below. When the ADC module is sampled at a certain rate, we can see and verify the conversion to a digital format that can be seen and validated in Texas Instrument’s Code Composer Studio.

## **3.2 – Subsystem Design**



Figure 9: Analog-to-Digital Converter Block Diagram

## **3.2.1 Software Flow Overview**

To start the process of the ADC conversion, the input channel MUX selects the input signal to convert. Next, I configured an EPWM module in TI’s Sysconfig. that interacts with the start-of-conversion interrupt and triggers the conversion process. The SOC arbitration makes sure that there is an orderly fashion of scheduling for the several SOC requests. The ADC conversion priority determines the order in which the conversion happens. The submodule contains 3 ADCs that operate in a round-robin scheme where no SOC has more priority than another. In other words, it happens circularly. Next, the ADC core performs the actual conversion. The post-processing block refines the raw output and reduces the error. A baseline of accurate voltage measurements is created by the reference voltage generator. Finally, the repeat block is utilized for a continuous conversion in real time.

## **3.2.2 Submodule Description**

GPIO pins on the board are utilized to show the different behaviors of the 3 ADC modules. Each ADC has a different behavior to show an interesting output and to add more validation for the ADC conversion. For the high/low toggle behavior of converted values, GPIO 3 is configured with ADCA to show the values toggle from 4095 to 0. The reason for the ceiling of 4095 is because of the 12-bit format. There are ((2^12) - 1) values (4095) in the 12-bit range. GPIO 16 is configured with ADCB to show the behavior of only high values being updated continuously (4095). Finally, GPIO 32 on the board is configured with ADCC to show only low values being updated continuously (0).

The input channel multiplexer is used to select one input signal from several analog signals and route it to the ADC core for conversion. In this case, this block allows the program to handle the multiple input channels that were configured in the code. The round-robin method previously mentioned makes the conversion process efficient by setting up the conversion in a circular scheme.

The start-of-conversion arbitration block is utilized as it manages and prioritizes the many SOC requests when input channels simultaneously call for conversion. The EPWM module triggers the SOC interrupt and sets up the different configurations such as sampling time.

The post-processing block handles the raw ADC conversion results and corrects the errors to help with scaling. The PPB makes sure the output values are efficient and usable for further processing in the system.

The ADC core is the heartbeat of the whole subsystem. The core is responsible for the resolution, speed, and accuracy of the conversions. As mentioned before, the 12-bit ADCs in this subsystem produce digital values ranging from 0 to 4095 for the input voltage range. The reference voltage generator interacts with the core as it provides a stable reference voltage that acts as a basis for determining input signal levels.

**3.2.2 Submodule Interactions**

To help with efficiency, the graphical user interface “Sysconfig.” is implemented in Code Composer Studio to help with PinMux configuration, GPIO setup, ADC setup, etc. in a working environment. Using this tool automatically generates the files needed to help the ADC subsystem run properly.

The submodules in the ADC work jointly to ensure efficient analog-to-digital conversion. Before sending the required input signal to the ADC core, the input channel MUX first chooses from the 3 different input channels configured in Sysconfig (A0, B0, C0). The SOC arbitration then ensures a fair scheduling of conversion using a circular buffer scheme. It settles disputes if several start-of-conversion interrupts trigger at the same time. Moreover, the SOC block is triggered by the EPWM and starts the conversion process. For precise signal quantization, the reference voltage generator uses its steady voltage as a baseline for the conversion. The board has a max input voltage of 3.3 V so that the board does not get damaged. The post-processing block then processes the raw converted values and makes any necessary adjustments. After conversion, the voltage values are compiled into the buffer which can be later viewed in the CCS memory buffer. Finally, the end-of-conversion interrupt is utilized to show the completed conversion. Finally, the repeat block shows that the ADC conversion runs continuously in real time if desired.

## **3.3 – Subsystem Validation**

The purpose of subsystem validation is to ensure all design-specified requirements from the submodule are met. This idea helps reduce the risk of issues in the later stages of the integration process. By validating the subsystems, the integration process will lead to a higher-quality product.

## **3.3.1 Memory Buffer**

The first validation test is to see the ADC result memory buffer. This idea ensures that the ADC is correctly converting analog signals to digital values. If there is any error in the ADC function, this is the first place to examine the problem. Concerning this subsystem module, 3 different ADCs simultaneously convert shadowing different behaviors. First, ADCA converts values that follow a high/ low toggle behavior. Second, ADCB follows a high output conversion behavior. Third, ADCC follows a low output conversion. In the figures below, one should see the values behaving in these configurations.



Figure 10: ADCA High/Low Toggle Conversion Behavior in Memory Buffer

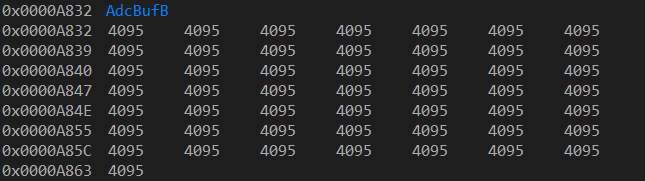


Figure 11: ADCB High Only Value Conversion Behavior in Memory Buffer

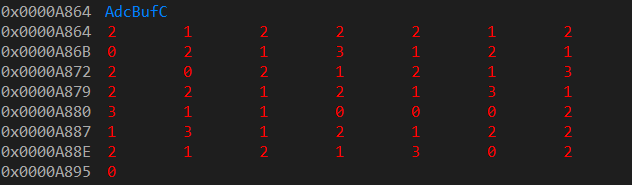


Figure 12: ADCC Low Only Value Conversion Behavior in Memory Buffer

## **3.3.2 ADC Buffer PWM Waveform**

Another form of validation for this module is to see the waveforms of the ADC converting in real-time. This waveform is validation to show the ADC is correctly sampling the PWM signal mentioned in the design section above. The waveform shows the data is being processed and displayed correctly along with the timing and synchronization of the subsystem. Below is the ADCA Buffer PWM sawtooth waveform for the high/low GPIO toggle pin behavior.



Figure 13: ADCA PWM High/Low Toggle Conversion Behavior Sawtooth Waveform

## **3.3.3 LED Behavior**

LEDs 4 and 5 are utilized on the f28p65x board to show further validation that the ADC module is correctly converting the analog signal to digital values. LED 4 is configured with ADCA which follows the high/low GPIO 3 pin and blinks continuously to show the behavior. LED 5 is configured with ADCB that only converts high values continuously. In this case, the LED is on at all times during the program run to show the proper conversion. Finally, it does not seem necessary to configure an LED with ADCC that only converts low values. This ADC follows the behavior of the GPIO 32 pin and would be off the whole time running.

## **3.3 – Subsystem Conclusion**

By testing each component of the ADC module, it can be seen that the ADC subsystem properly converts the varying voltage into a digital value that the sensorless motor can use for modules such as the estimator. The results of these validation tests show the theoretical behaviors of the ADC which were deemed successful. This allows the subsystem module to be passed into the integration phase.

# 

# **4.0 Estimator Subsystem Report - Case**

## **4.1 Subsystem Introduction**

Rotor Position Estimation is the means by which digital calculations are used in order to estimate the position of a rotor in a motor. Position Estimators arose as a solution for creating less-expensive, more robust motors that can be built much easier by removing the need for position sensors.

The objective of this subsystem was to design and implement a Rotor Position Estimator code that is able to accurately estimate the position of a rotor, as well as its RPM, for use in other subsystems, as well as to be seen by the user. The estimator functions using the BackEMF generated across the stator windings as a result of the interactions between the stator and the rotor. The calculations utilized include the Clarke Transform, BackEMF Calculation, a trigonometric function, and speed calculator, as shown in Figure 4-1.

This report will cover the design process of the Estimator, the purposes of each component, and the validation process used for this system.

## **4.2 Subsystem Design**

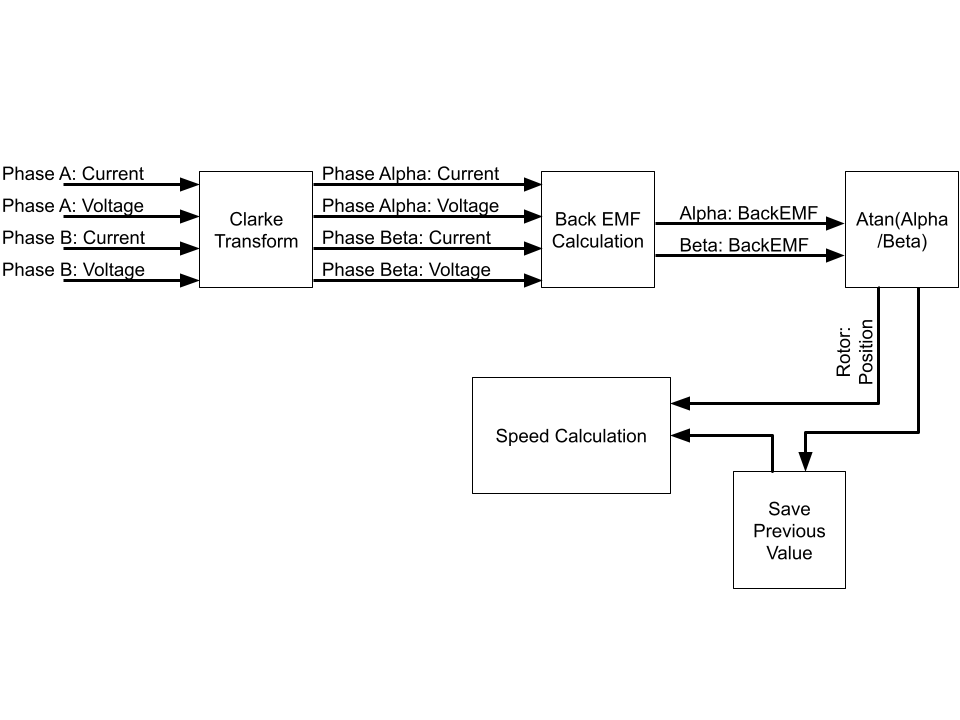


Figure 14: Rotor Estimator Subsystem Flow Diagram

## **4.2.1 Software Flow Overview**

The Estimator software ensures that other portions of the motor know the rough position of the rotor at all times. The motor in use is a three-phase AC motor. The process of determining the rotor position begins by taking the current of voltage of two phases (Ia, Va, Ib, Vb), though this can be done with all three, and performing a Clarke Transform. This takes the two phases, which were originally offset by 120𝆩, and creates Phase Alpha Current and Voltage, and Phase Beta Current and Voltage (Iα, Vα, Iβ, Vβ), which represent the three motor as two phases offset by 90𝆩 of each other. The current and voltage of Phases Alpha and Beta, as well as the motor design data, are used to calculate the BackEMF through these specific phases (BackEMFα, BackEMFβ). Using an Atan function, the angular position of the motor can be obtained. When combined with the previous position, and time between estimations, the rotor's angular velocity can be found as well.

## **4.2.2 Submodule Code**

The code that contains the rotor estimation code is written in a way such that they can be easily modified for different three-phase AC motors. Additionally, the functions themselves are linear in nature, and thus the code can be optimized by reducing the number of functions necessary to complete each iteration.

## **4.3 Subsystem Validation**

The validation plan for this subsystem was designed on the idea that complexity within the system would slowly increase to allow for as much resolution when looking for problems within the code as possible.

## **4.3.1 Testing Environment Validation**

The first validation step was to ensure that a proper testing environment was created for code. This was written in Visual Studio Code in C++, as this allowed for writing and testing of the code to begin much sooner, as well as streamline the testing process. Validation was achieved when the testing environment was able to input and output information to and from a CSV file, using the earliest version of the estimator code.

## **4.3.2 Ideal Condition Tests**

Ideal Test Conditions refer to a constant, steady input of information that follows a sin and cosine wave. There is no or regular variation in the amplitude or frequency, and thus a constant, or regularly changing rotor velocity was expected. The earliest version of the code however would see a constant output of null or nonsensical values, leading to a rewriting of how the code handled floating point integers. Additionally, it showed the need for limits to be placed on what the code would consider a valid output, as during start-up, massive values that interfered with the graphing data. These tests were labeled as:

* Linear Data: No change to the frequency or amplitude of the input wave, so output data corresponding to a constant velocity is expected.
* Rising Data: Frequency or amplitude increase over time, thus a rising velocity for the frequency but no significant change with the amplitudes is expected.

This step was marked complete when these tests produced consistent, believable results. Samples of this data are included below.

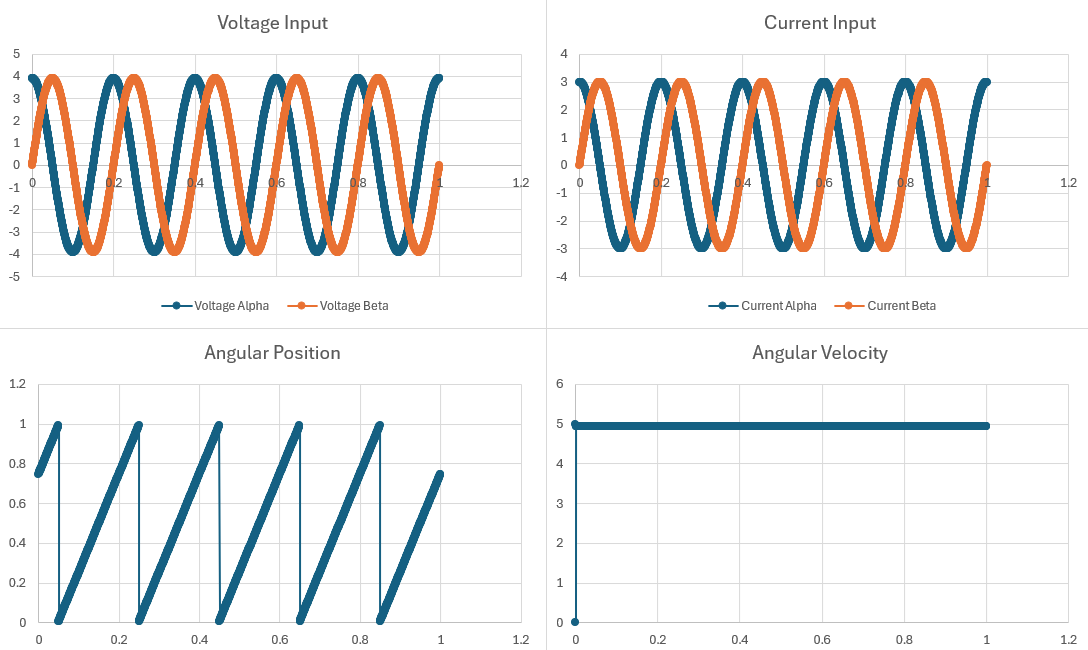


Figure 15: Linear\_2 Test Data Input and Output

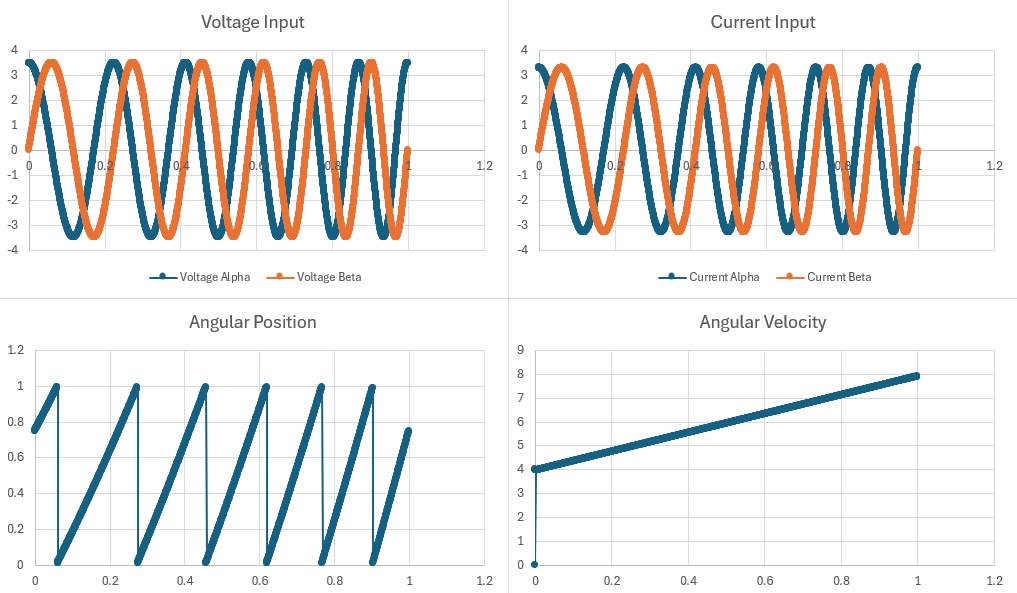


Figure16 : Rising\_2 Test Data Input and Output

## **4.3.3 Non-Ideal Condition Tests**

Non-Ideal Test Conditions refer to a mathematical data with noise injected into it. These tests revealed no major faults within the code and that it could handle problematic inputs. These tests were labeled as:

* Low-Noise Data: Low Noise data refers to the added wave having a small amplitude, and thus distortion is minimal.
* High-Noise Data: High Noise data refers to the added wave having a large amplitude, and thus a larger distortion.
* Variance Data: Input waves have mismatched amplitudes, simulating a stator that is unable to get to full current or voltage levels.

This step was marked complete when these tests produced consistent, believable results. Samples of this data are included below.

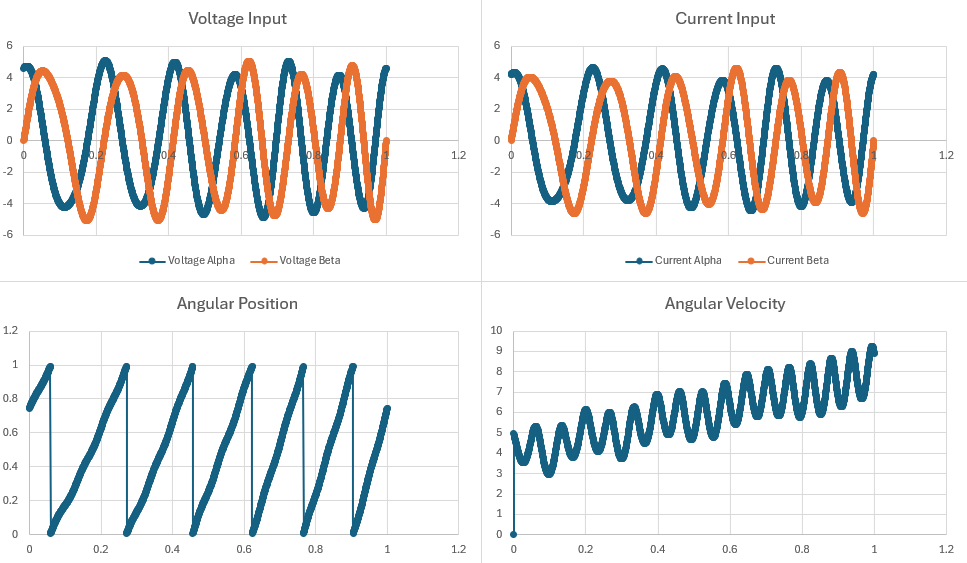


Figure 17: Noise\_1 Test Data Input and Output

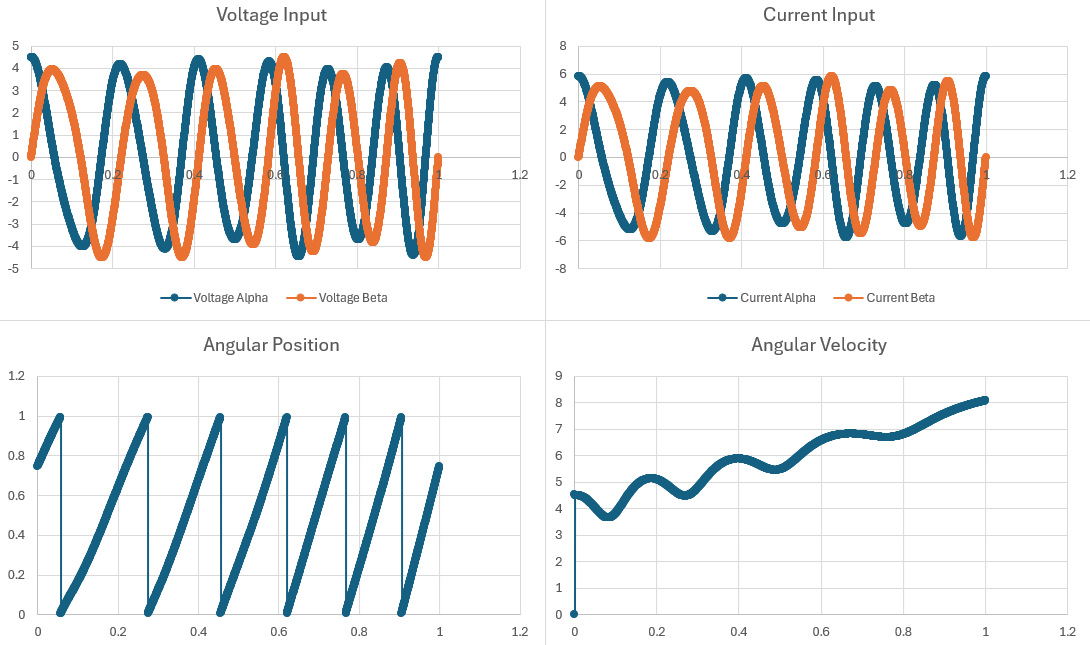


Figure 18: Variance\_1 Test Data Input and Output

## **4.4 Subsystem Conclusion**

The testing procedures shown above ensured that the estimator code was functioning as expected by revealing early problems in an environment where it can be most effectively analyzed. Later tests have also demonstrated that the estimator code was even able to make sense from poor or inconsistent input. Possible tests for the code within a purely mathematical environment have been exhausted, and the estimator code is ready for implementation with the other subsystems.

# 

# 

# 

# 

# **5.0 Porting Subsystem Report - Tyler Hawkins**

## **5.1 Subsystem Introduction**

The Ported Solution allows for simple motor control with the use of the software developed, as well as, the F28p65x launchpad and the DRV8323RS driver board. The ported solution is intended to allow the same operations of TI’s Universal Motor Control Lab solution for the F28002x board while running on the new F28p65x board.

## **5.2 Subsystem Details**

The porting of the Motor Control Solution can be separated into five sections. The Project Spec Cmd Linker, hal.c, hal.h files, and the hardware setup.

The Project Spec file is used to generate the project folder in the user workspace and includes references to device-specific C2000Ware source files. In this file, the diverboard and all included libraries are defined, as well as, their corresponding file locations within the C200 Motor control SDK.

The cmd Linker files handle the memory reservations and locations of the memory for the board. These reservations include the register locations of the RAM as well as the Flash Memory. In order to correctly define the memory map of the F28p65x board TI’s Technical Reference Manual was used to find register locations reserved within the board. The reservations also include the length of the register locations and with the use of CCS the memory allocation can be shown as a percentage of the reserved space used.

The Hal.c and Hal.h files include GPIO, PWM, ADC, and CMPSS modules and defines. These definitions handle the communications between the DRV8323RS driver board and the F28p65x launchpad. By reviewing previously used modules and definitions for the F28002x board as well as studying the differences between TI’s Technical Reference Manuals and board schematics of both the new F28p65x and the old F28002x boards, new modules and defines were written In the Hal.c and Hal.h files. One such update was writing all GPIO definitions needed for communication between the F28p65x launchpad and the DRV8323RS board. Using the F28p65x Board schematic PWM, ADC, and CMPSS modules were written in accordance with the connected modules that correspond to the GPIO connections between the launchpad and driverboard.

The Hardware setup of the F28p65x launchpad and DRV8323RS driverboard largely follow the same setup as the F28002x launchpad. After reviewing TI’s Technical Reference Manuals on both boards the same pins would need to be bent in order for the solution to run. Additionally, the Switch configuration was also largely the same with one exception due to a different GPIO pin being used on the F28p65x.

## **5.3 Subsystem Validation**

Due to the project being largely software-based the Subsyestem was greatly validated using CCS and Clearing all errors that occurred during compiling and testing. The project spec file was validated by clearing all errors that occurred when compiling. The cmd Linker files were verified by clearing all related errors as well as checking that memory allocation was less than 90% of the maximum reserved value. Files Hal.c and Hal.h were validated by clearing all errors within the CCS environment. Resolving these errors and validating these files involves the most amount of time. TI changed several general definitions and commands, and therefore definitions had to be chased down within the files, as well as several hours spent on TI forms trying to find the updated naming conventions that needed to be used.

The Full Ported solution was validated by using TI’s Universal motor control document. The document includes a section on how to validate the solution, and due to the solution being ported from one board to another, the operation should be the same.

The Project Spec, cmd Linker, Hal.c, Hal.h files, and Hardware have all been verified, and no compiling errors or warnings occur. However, the fully ported solution has been unable to be verified. After conversing with our sponsor, I believe the issue arises from a GPIO definition being incorrect. Our sponsor stated if certain definitions are incorrect, it can prevent the solution from running and the ISR clock does not run within the software. The GPIO pins can be defined as several different things that relate to the connection between the launchpad and driver boards. I believe small adjustments must be made in order to fix this problem.

## **5.4 Subsystem Conclusion**

Currently, the Subsystem does not work as intended, and adjustments and further testing must be made to ensure correct operation. I believe that I have identified the problem area and be working to fix the problems as soon as time permits. I plan to have a working solution once the school year returns in January.