CAL POLY
College of Engineering

DC to AC Voltage Source Converter Control System

by Brayden Young, Mohummad Elgassier, and Jordan Reichhardt
Advisor: William L. Ahlgren

Senior Project
California Polytechnic University
San Luis Obispo
2025

Table of Contents

Acknowledgements

Thank you to our advisor Professor Ahlgren for giving us the opportunity to work on this project, for providing clarity, and directing us to be able to complete the project.

Thank you to Professor Taufik for providing technical knowledge and providing perspective on issues we faced in development.

Thank you to all the groups and persons that were referenced throughout the report and bibliography for providing relevance reports that helped guide our design and increased the breadth of knowledge in the world.

Abstract

The Voltage Source Converter Control System integrates a control system for a 2-phase half-bridge voltage source converter (VSC) utilizing a TMS320F2837xD Dual-Core Real-Time Microcontroller. VSCs are critical for the integration of power production into the power grid. They allow for accurate control over DC-AC energy conversion. Additionally with the improvement of renewable energies VSCs play a critical role in controlling the flow of these environmentally controlled and more unpredictable sources of energy. The control system regulates the output current and voltage, regulates switching patterns, and maintains power stability under varying loads. Within the control system Digital control systems, PWM generation, and feedback controls are implemented in order to further system stability and reliability. Ultimately this project implements applications for a VSC control system in a scalable practical setting allowing for advanced power conversion and a more efficient and environmental power grid.

I. Introduction & BackGround

Power electronics are becoming more critical as the average power demand per household increases as well as the complexity of power integration into the grid grows. This complexity in integration is the consequence of new and renewable sources of energy. The push to become a more environmentally conscious grid has made it necessary to tie more varied types of energy and loads to the grid. This includes things such as solar power, hydro, and wind. Voltage source converters are particularly useful for High Voltage Direct Current (HVDC) Transmission, flexible AC transmission systems and grid stabilization[1]. VSCs allow for power to flow in both directions, they improve power quality in terms of active and reactive components, and they support the operation and integration of AC and DC power grids. As more renewable energy sources are integrated into the power grid, the need for VSCs is becoming critical for ensuring synchronization. With a properly designed control system issues such as overcurrent protection, control optimization, and operational efficiency can be addressed ensuring reliable performance [2].

In order to address design restrictions for the controller PCB we have based this project off a VSC that utilizes GaN_FETs as a component within the 2-level half bridges. The microcontroller here is designed specifically for use in power electronics as it includes high processing power, integrated Pulse Width Modulation (PWM), and real time controls abilities [3]. This control system will guide both active and reactive power, optimize switching states, and control overcurrent and overvoltage protection measures [4].

Many factors contribute to developing a stable and efficient VSC. This includes optimized digital control algorithms, precise PWM generation, and feedback-based controls allowing for stability within power balancing systems. Excessive current created by shorts or ground faults could lead to catastrophic failures. Thus these current issues must be identified and stopped nearly instantly within the system. If currents were to increase to a level that may not be necessary to immediately address this current can still lead to component failures and instability and must also be addressed quickly. The controller implements various methods to address these overcurrent issues such as digital Hall effect overcurrent detectors, and software based current limiting [5][6]. Adequate overcurrent protection requires an integration of both hardware and software measures as outlined by multiple studies [7].

In contrast to Line-commutated converter HVDC (LCC_HVDC) systems, VSC systems allow for independent active and reactive power control, black-start capability, and ease of integration with weaker grids. Thus VSCs are particularly valuable for renewable energy applications where power production varies greatly and decentralized power networks [8]. This allows for more power solutions in remote communities. The practical ramifications of VSCs are further explored within Lindblom's study on lab-scale VSCs. This gives insight into the hardware selection, performance evaluation, and control processes for a lab VSC [9]. Additionally historical analysis of DC power transmission highlights the

importance of VSC systems within modern power grids. Andrade and Leão track the development of DC systems from where they began as contemporary high voltage systems. THis underscores the importance of VSCs for enabling stable and efficient power transmission [10].

Ultimately this project is motivated by a need to develop a strong and scalable VSC control system which enhances power transmission and grid stability. Through microcontroller technology and control techniques the project aims to aid in the evolution of power electronics within renewable energy technologies.In order to verify the control system's performance, testing and verification will be conducted using hardware-in-the-loop (HIL) simulations. This allows for testing of the system under "world" like conditions ensuring its application in advanced power conversion systems.

II. Requirements and Specifications

TABLE I

CUSTOMER REQUIREMENTS AND ENGINEERING SPECIFICATIONS

Customer Requirements	Engineering Specification	Justification (ADD REFERENCES)
E. G. H.	#1 THD 5% with the nominal frequency being 60Hz.	Grid Compliance IEEE standard 519-2022 is 8.0% for systems operating at <= 1.0kV. [11],[14]
B. G. H.	#2 Utilizes Internal Switching Frequency of 500kHz $\pm$ $1kHz$.	GanFETs can have switching speeds as high as 10MHz which is too high for some commercial MCUs. 500kHz gives us high switching speed while avoiding some of the losses from fast switching. [1], [7]
B. G.	#3 Maximum Operating Temperature ranging between -10 $^{\circ}C$ $-$ $100^{\circ}C$.	The MCU has a maximum operating temperature of $125^{\circ}C$ and this would give us a good margin for proper operation. [1]
G. D. A. C.	#4 Minor overcurrent 110%-120% over current protection addressed within 1ms	In order to comply with grid current levels the current output of the VSC must make slight adjustments to the PWM to account for slight overcurrent protection values.[2], [5]
G. D. A. C.	#5 Sustained overcurrent 120% - 150% over current protection addressed within 500µs	To Comply with grid current levels the current must be significantly limited or completely shutoff at this level depending on how long it is sustained.[3], [4], [5]
G. D. A. C.	#6 Severe overcurrent 150%-200% overcurrent protection addressed within 50µs	To Comply with grid current levels fast shutdown of PWM needed to stop ramifications of short circuit or ground fault.[3], [5]

A. E. G.	#7 Sustained voltage DC input should be 60 V at <=10 A.	Cal Poly power Laboratory DC power supplies can provide up to 60V and 50A. 50A is too much power and would be unsafe. [9]
A. C. E. G.	#8 Sustained voltage of AC Output should be 120V <= 10A.	Cal Poly high power laboratory is capable of providing 120V and 10A. [9]
F. B	#9 Approx. 114mm x 64mm PCB	Ample size to accommodate all components (MCU, Sensors, etc.) and remain compact and portable. [7]
H.	#10 η = Pout/Pint = 93%	Grid tied systems require high efficiency. [8], [10]

List of Customer Requirements:

A: DC-AC (3-phase) VSC Control System

B: Commercially available MCU

C: Automation

D: Powered by Cal Poly Lab Equipment

E: Work with a simulated grid

F: Compact in size

G: Safety

H: Efficiency

TABLE II

ENGINEERING SPECIFICATIONS

Spec. #	**Parameter**	**Target (units)**	**Tolerance**	**Risk** (H, M, L)	**Compliance** (A, T, S, I)	**Test Equipment Needed**
1	THD (total harmonic distortion)	5% of 60Hz.	Max	M	T	Spectrum analyzer
2	Switching Frequency	500kHz	$\pm$ $1kHz$	L	A, S	Oscilloscope

3	Operating Temperature	-10°C < T < 100°C	Min, Max	L	A, S	On-board temperature monitoring
4	Minor overcurrent 110%-120% over current protection addressed.	$10ms$	Max	L	T	Current source, Oscilloscope
5	Sustained overcurrent 120% - 150% over current protection	500 µs	Max	H	T	Current source, Oscilloscope
6	Severe overcurrent 150%- 200% overcurrent protection addressed.	$50\mu s$	Max	H	T	Current source, Oscilloscope
7	Sustained DC input voltage.	$60V$ @ $\leq 10A$	Max	M	A	DC, power Supply, Multimeter
8	Sustained Output AC voltage	$120V$ @ $\leq 10A$	Max	M	A	AC power Source, Multimeter
9	PCB Size	114mm x 64mm	$\pm$ 5mm	M	I	Altium
10	Power Efficiency	93%	Min	H	T, S	Multimeter

III.a System Design Draft (Initial Design)

Section	Equation
Orthogonal Signal Generation	$v\alpha = va$
	$v\beta = \hat{v}a$
	$i\alpha = ia$
	$i\beta = \hat{i}a$
PLL Angle Dynamics	$\theta(t) = \int \omega(t)\, dt$
	$vq \approx 0, \quad vd \approx$
αβ → dq Transformation	$vd = v\alpha \cos\theta + v\beta \sin\theta$
	$vq = -v\alpha \sin\theta + v\beta \cos\theta$
	$id = i\alpha \cos\theta + i\beta \sin\theta$
	$iq = -i\alpha \sin\theta + i\beta \cos\theta$
Instantaneous Power (αβ Frame)	$p = v\alpha\, i\alpha + v\beta\, i\beta$
	$q = v\alpha\, i\beta - v\beta\, i\alpha$
Power in dq Frame	$P = (3/2)\, vd\, id$
	$Q = (3/2)\, vd\, iq$
DC-Link Voltage Controller	$evdc = Vdc^* - Vdc$
	$id^* = Kpd\, evdc + Kid \int evdc\, dt$
Reactive Power Controller	$eQ = Q^* - Q$
	$iq^* = Kpq\, eQ + Kiq \int eQ\, dt$
dq Current Controllers	$vd^* = vd + Kpd(id^* - id) + Kid\int(id^* - id)dt - \omega L\, iq$
	$vq^* = vq + Kpq(iq^* - iq) + Kiq\int(iq^* - iq)dt + \omega L\, id$
Inverse dq → αβ Transform	$v\alpha^* = vd^* \cos\theta - vq^* \sin\theta$
	$v\beta^* = vd^* \sin\theta + vq^* \cos\theta$

Modulation Voltage Reference vref = vα*

The above equations are used to determine the system blocks values of the control systems elements only. Included is the DQ/alpha beta transform normally done in matrix from but written in linear equation form for readability and use. Since the system is a single phase, b and c components of a normal three phase system are excluded in the transform and the rest of the calculations.

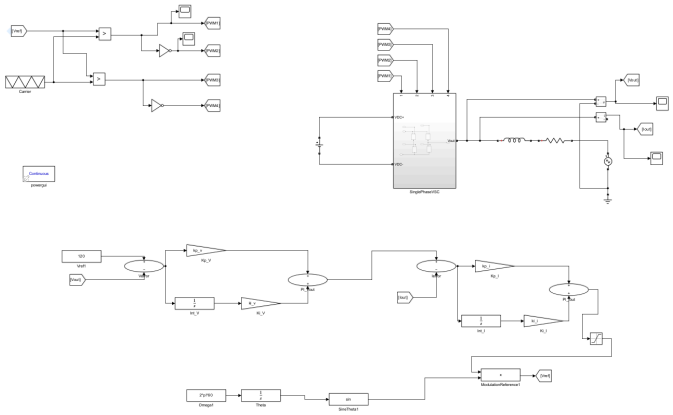


Image [1]: Overall Control Scheme w/ Single Phase VSC

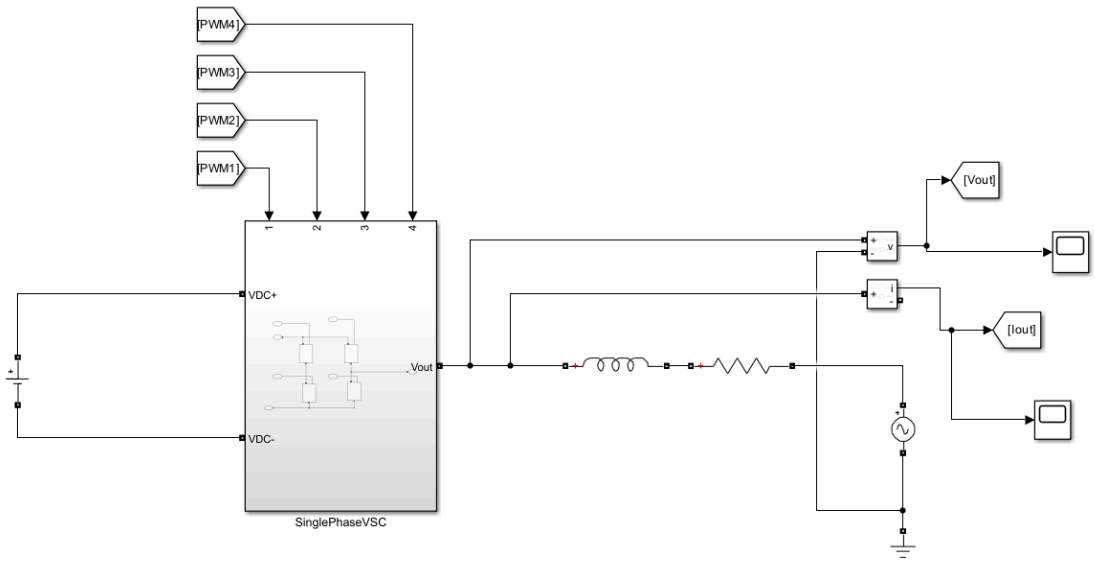


Image [2]: Single Phase VSC w/ Grid Connection

All of the circuitry besides the control system was to be developed by the Power team. Essentially the physical characteristics of the circuitry were values that were predetermined. Due to the harmonics that

are caused by the switching characteristics of a full bridge inverter we needed to introduce a LC filter that was not implemented in the initial design seen above.

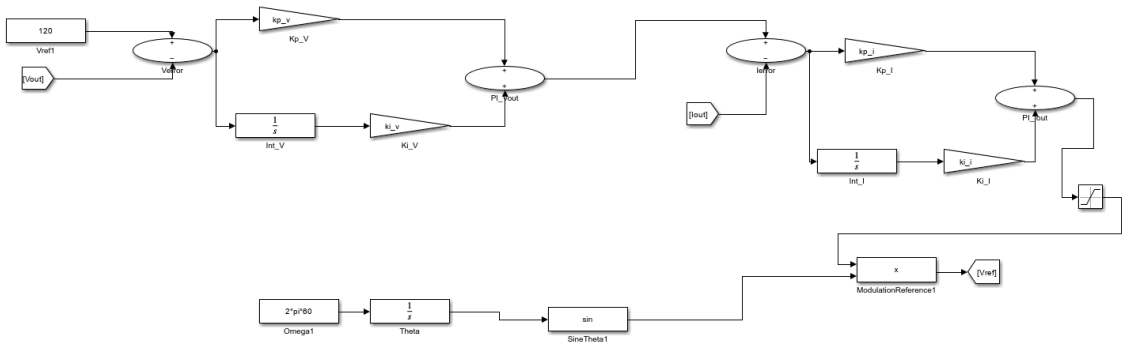


Image [3]:Initial Control Scheme for the Current Control, Voltage Control, & PLL

This control system was a good introduction into learning how to use the different Simulink blocks as well as trying to develop a control system that did not implement reactive power control. The control system was able to successfully produce a reference signal that could be used to compare to a triangular carrier signal and therefore produce a PWM signal. However when implemented the resulting output voltage was now what we desired.

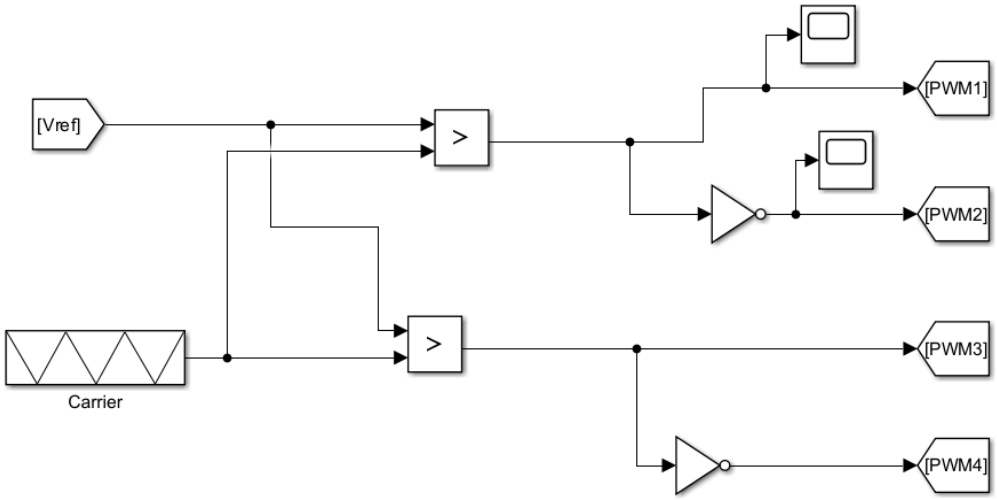


Image [4]: Custom Made PWM Generator

This is a standard method of producing a PWM for any kind of inverter system. The reference voltage is compared to a triangular carrier wave. This will produce two sets of PWMs, one which is high and an inverted signal. These signals are fed to our gate drivers which turn the IGBT's on or off. This initial draft design will also be implemented in our finalized version nearly identically.

III.b Initial System Draft Simulation

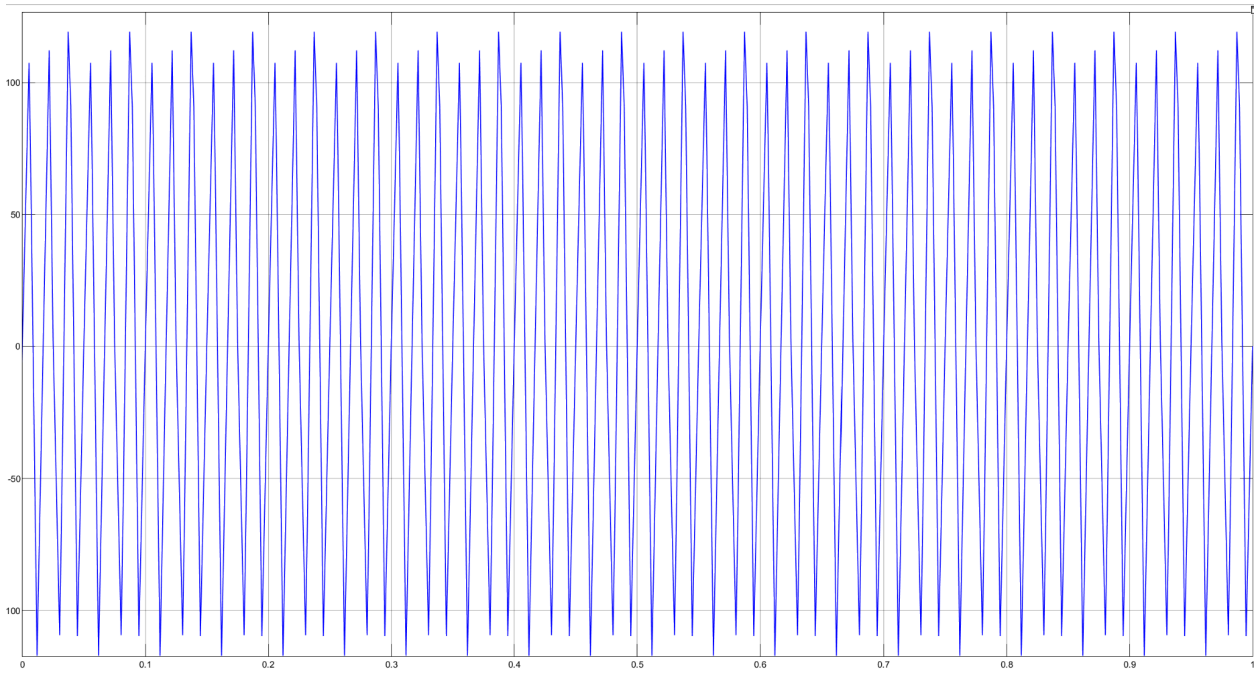


Image [5]: Simulation Vout 1 Sec Elapsed (Pre PI Tuning)

Expected output voltage: We would expect an output voltage at grid frequency of 60hz which we have, and a clean sine wave with amplitude nearly identical to that of the grid. While our amplitude is close to 120V the sine wave could be cleaner and would benefit from using a Phase lock loop which measures the grid voltage angle in order to match the current angle with the voltage angle. We also see that the signal values are not consistent, varying over time. This is due to the incorrect production of pwm signals along with incorrect filtering at the output.

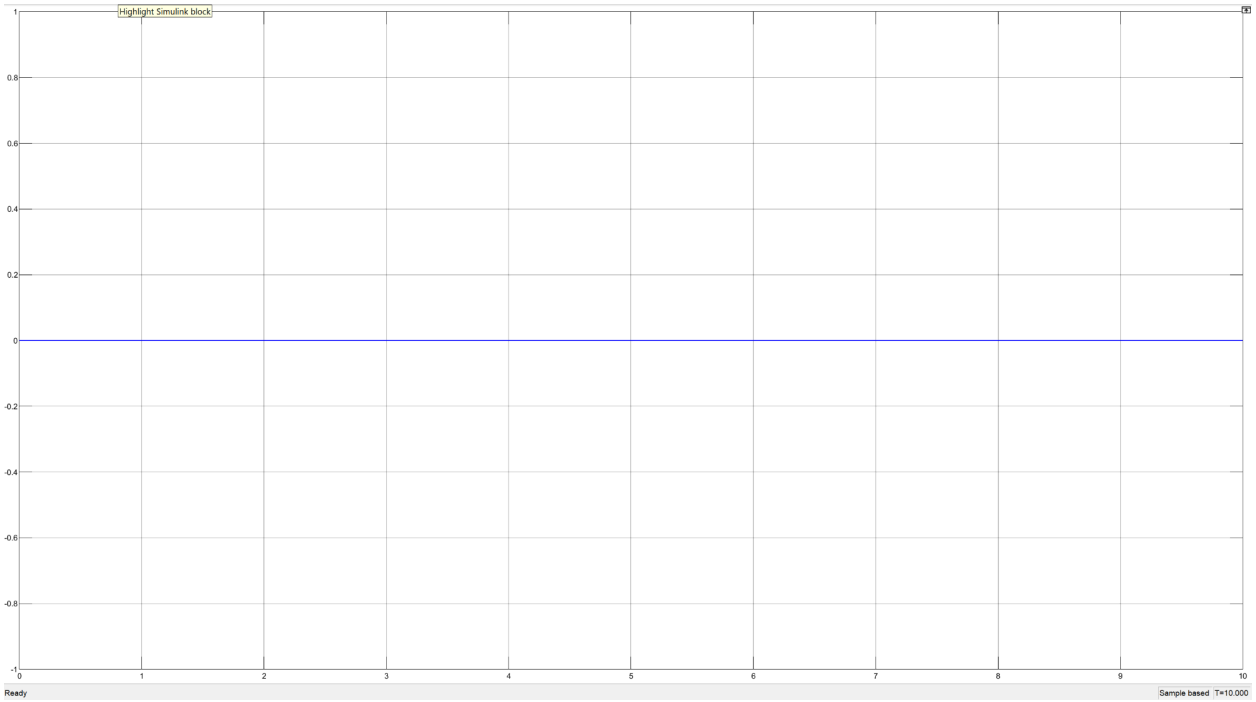


Image [6]: Simulation Iout 1 Sec Elapsed (Post PI Tuning)

Reasoning for 0 current at output: Within the current control loop of the simulation there is no significant voltage difference across the inductor, hence no current can be measured based on the change in voltage. This is why we see no current at the output. Here we would expect a current that is sinusoidal and in phase with the voltage produced in the previous voltage loop phase. To fix the lack of output current we must replace the sinusoidal input with a phase lock loop.

IV. Finalized design

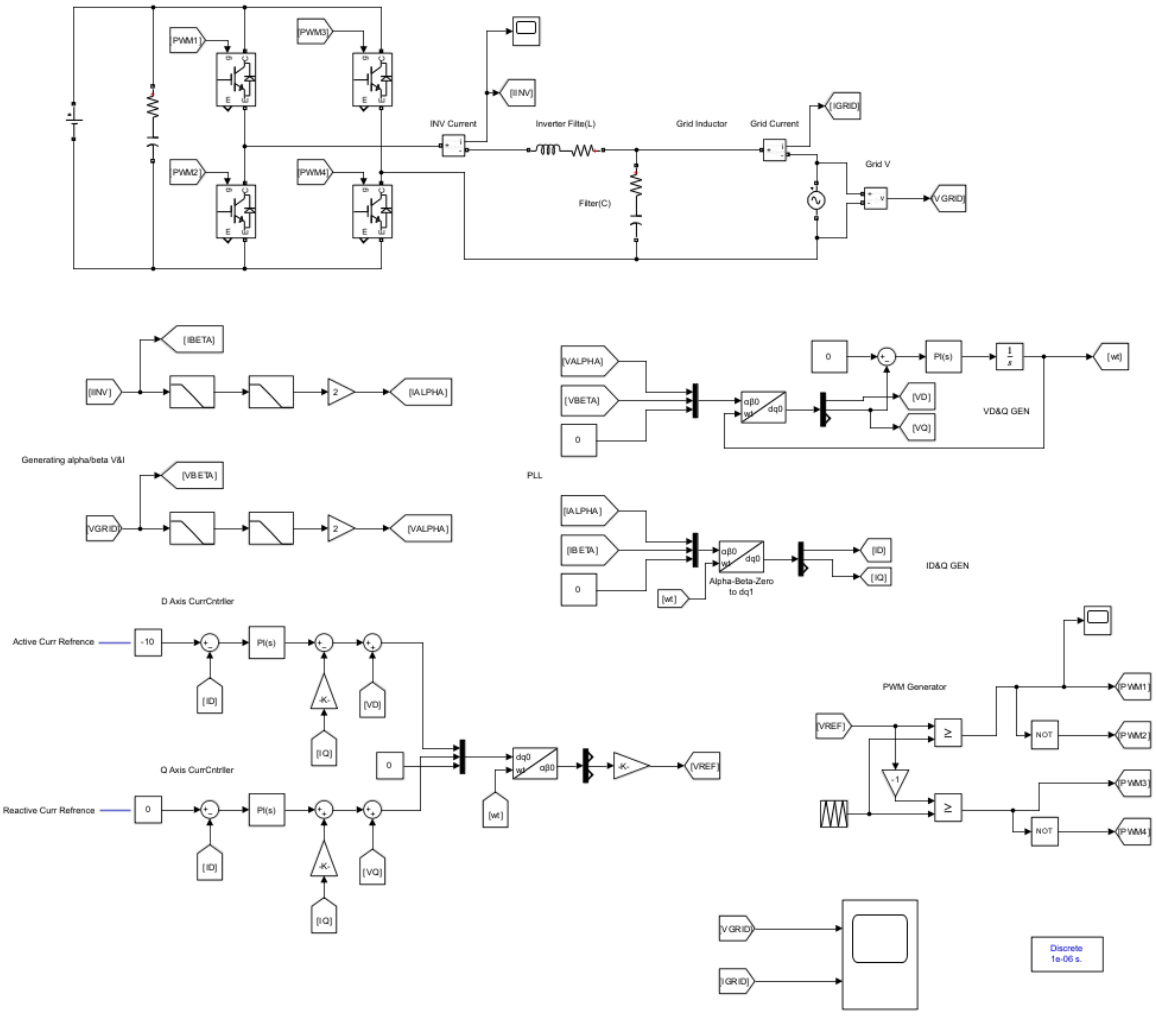


Image [7]: Finalized VSC Setup

The image above details the complete simulink setup that includes the inverter with IGBTs, filters for grid voltage, and DC source. These items are to be provided and created by the power team while our role was to create the control system to produce the PWMs that would switch the IGBTs as well as introduce a method for Active and Reactive power control. The power control was done by separating the signals reactive and active components into an alpha beta / dq reference frame respectively. This would allow for the signals to be treated as DC where a PI controller could be used to create a reference that would be compared to a carrier signal. This comparison is used to create the PWMs.

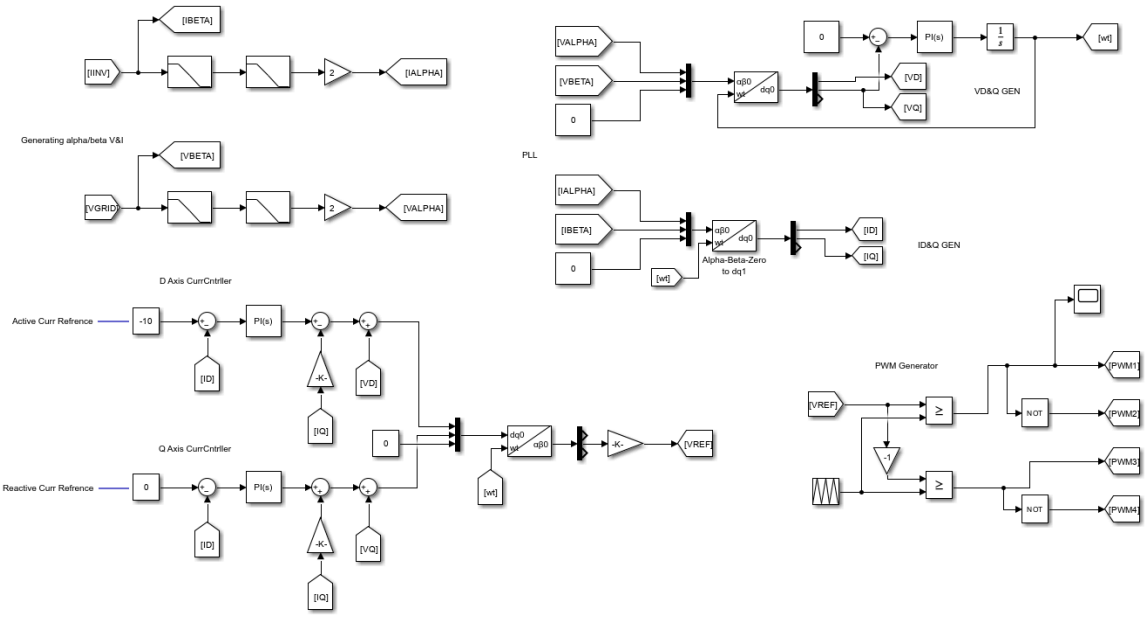


Image [8]: Finalized Control System Elements

TABLE III

CONTROL SYSTEM BLOCK ELEMENTS VALUES

First Order Filter	1/(2*pi*60)
PID Controller (PLL)	P=10 I=50000
Active Current Reference	-10
Reactive Curr Reference	0
PID Controller (Current Controllers)	P=40 I=6.7
Gain of Current Controllers (K)	2*pi*60*8.41e-3
Gain of Gain Block Before Vref	1/400
Carrier Signal Values	Time Values: [0 0.00005 0.0001] Output Values: [-1 1 -1]

PID Equation: P+I(1/s)

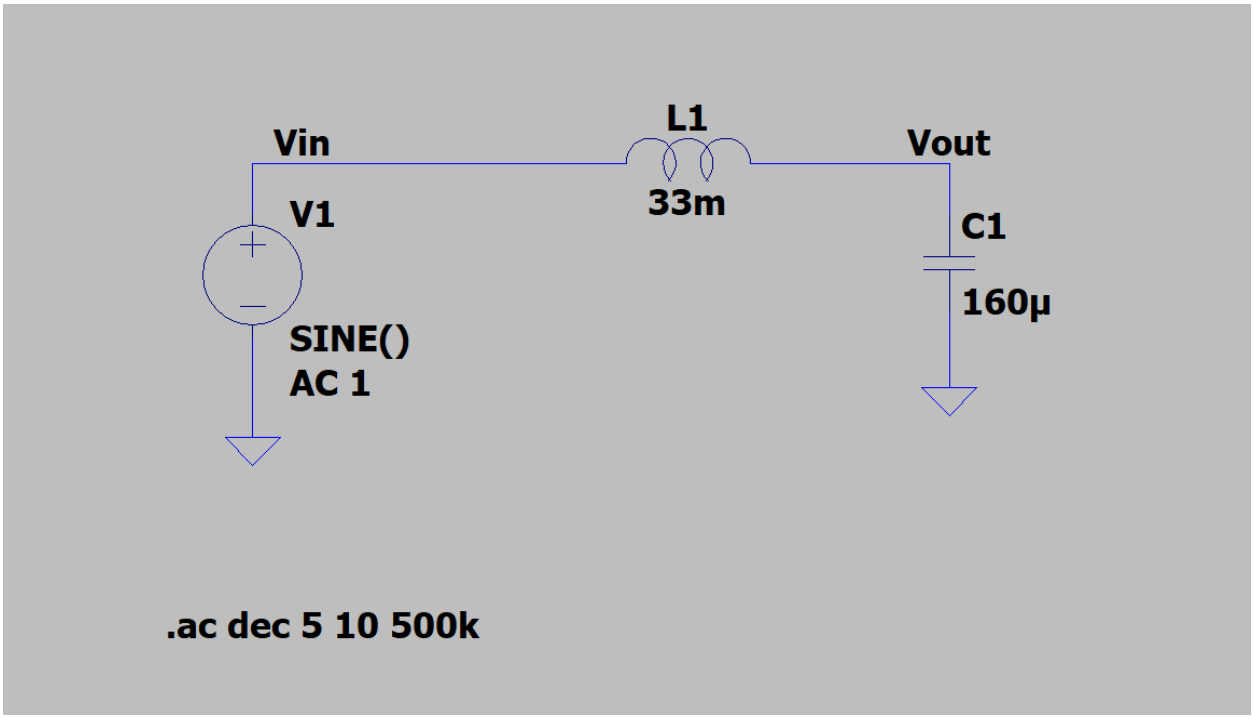


Image [9]: Finalized Grid Filter Design

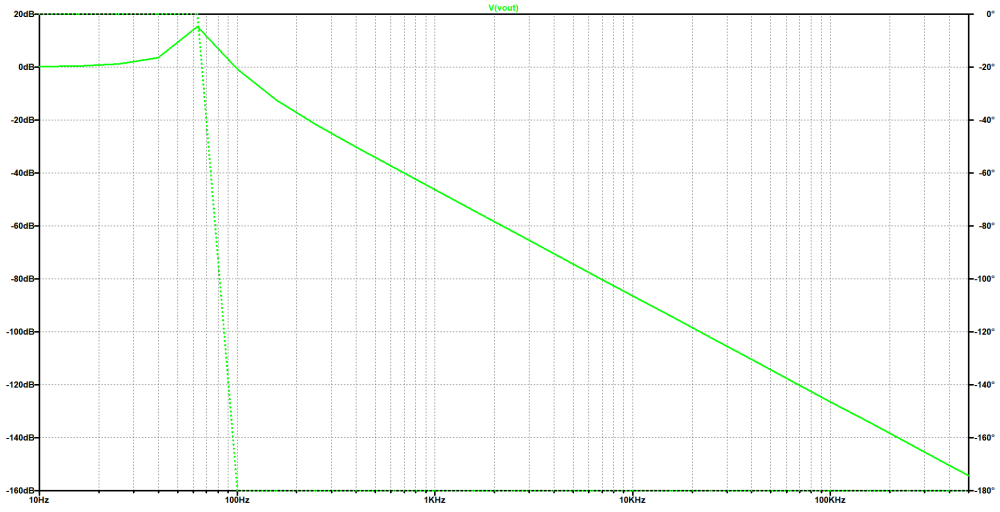


Image [10]: Finalized Grid Filter Results

On the output with incorrect filtering we were seeing very large harmonic distortion that would not be compliant with IEEE standards. Therefore a redesigned low pass LC filter was needed at the output to ensure that we were only seeing the desired 60 Hz at the output. Using LTspice and sweeping across the frequencies we came up with the above filter design. Exact specs L=33mH and C=160uF. This gave a realistic value for Inductor and capacitor that may be used in real high voltage systems.

V. Simulation

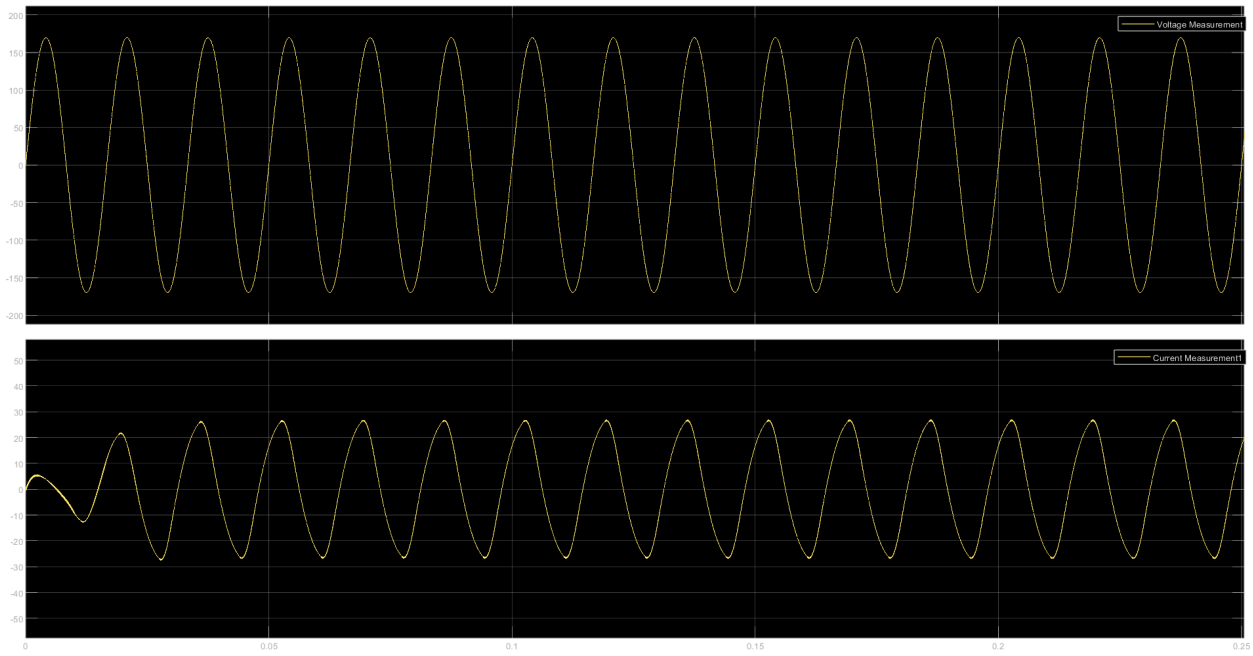


Image [11]: Simulink Simulation Results Voltage and Current

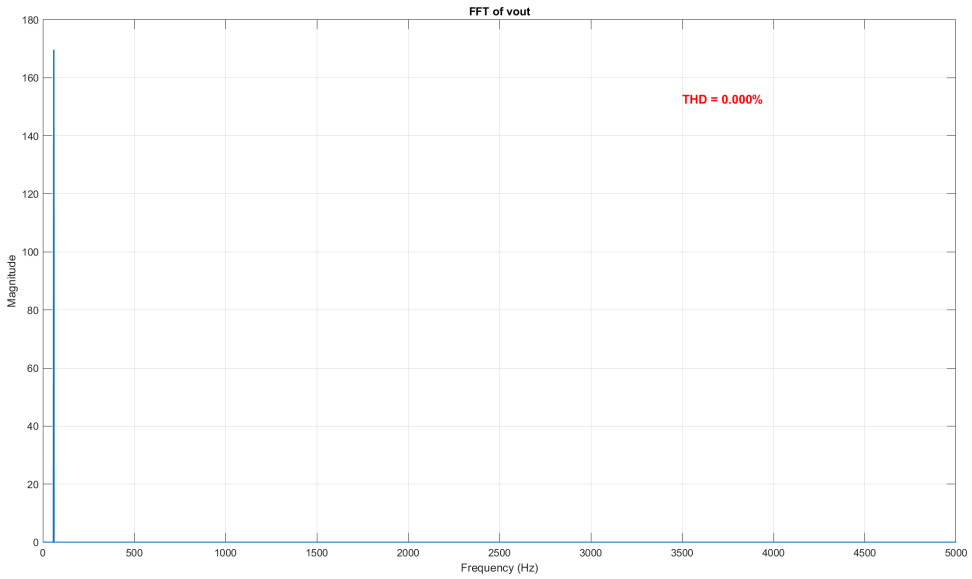


Image [12]: FFT Voltage Signal and THD

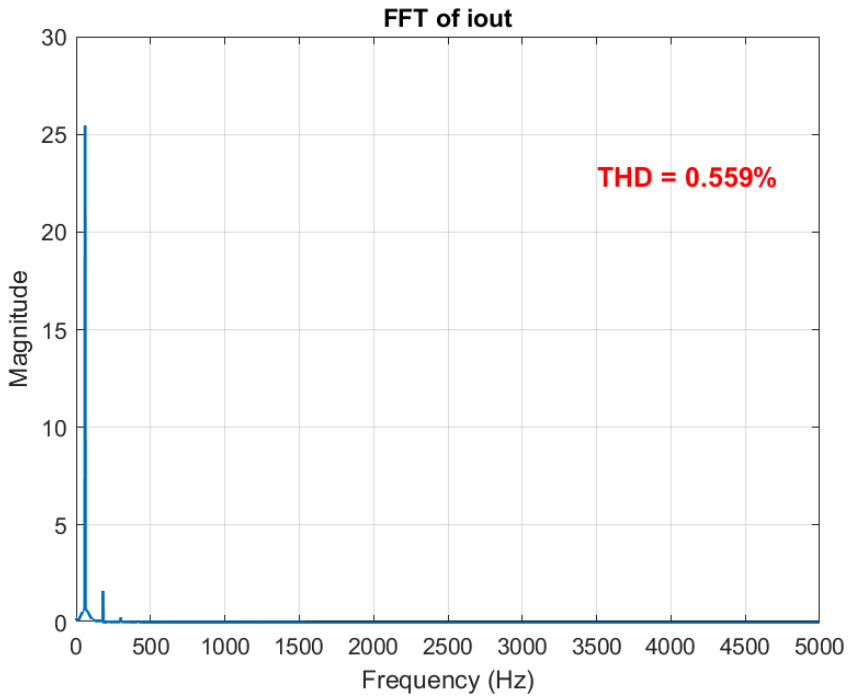


Image [13]: FFT Current Signal and THD

From the results we can see that the simulation yielded a voltage value of 120 Vrms or 170 Vpeak with a current value of 26 Amps peak. The current value was lower than the desired power requirements. However, this was intentionally stepped down due to the current limit of our Hall Effect sensor. In theory, with the correct sensor capability this current can be scaled higher by increasing the voltage of the DC source.

Taking the FFT of the voltage signal shows that we have a correctly filtered output that has no distortion adhering to the IEEE standard. The FFT of the current signal also shows that our filtering was sufficient as the THD was less than the 5% limit set by IEEE. Future work would require that a new current sensor be found in order to be able to correctly sense the higher current values required to hit our power requirement.

Another requirement of our project was the ability to have active and reactive power control. Looking at the output voltage and current we can see that the two signals are in phase. This indicates that the current is neither lagging nor leading the voltage signal. Meaning that we were successfully able to create a control system that delivers fully active power to the output. If there were to be an unexpected change in the output's power demand the solution is as simple as changing the value of the reference current in the current control to compensate for this change. This would correct the control system and cause the Power Factor to be corrected to one.

VI. Software Development

The firmware running on the C2000 microcontroller was developed using a model based design workflow in Simulink. Embedded Coder automatically generates C-code targeted specifically for the C2000 board. During code generation, all control modules, peripherals, and PWM generators are configured into reusable C functions. The firmware for the inverter controller is organized around a real-time execution loop functioning at a switching frequency of 10kHz. All control activity is driven from this loop, which is initiated by a hardware timer interrupt and executed through a generated function. This design ensures that signal acquisition, control processing, and PWM updates occur with consistent timing.

At system startup, the board initialization routine configures clocks, enables peripheral modules, and prepares both the ADC and PWM subsystems for operation. Once initialization is complete, global interrupts are enabled and the controller begins executing its periodic control cycle.

Sinusoidal Pulse Width Modulation (SPWM): In order to generate this pulse width modulation the sinusoidal reference Vref is compared with the triangular carrier waveform. This produces PWM1, PWM2, PWM3,and PWM4. An alternative method to produce PWM signals is by using the onboard ePWM (Enhanced Pulse Width Modulation) modules included in the C2000 microcontroller library. The method begins with configuring the ePWM time-based unit, which establishes a switching period using internal clocks to provide a counter that drives the modules. The counter runs up and down creating a symmetrical waveform and reduces harmonic distortion. Once this structure is established upon setup, the Compare units (CMPA) are used to map the duty cycles generated from the control scheme into hardware switching. These transitions are sent to digital IO pins to be used as inputs to the gate drivers.

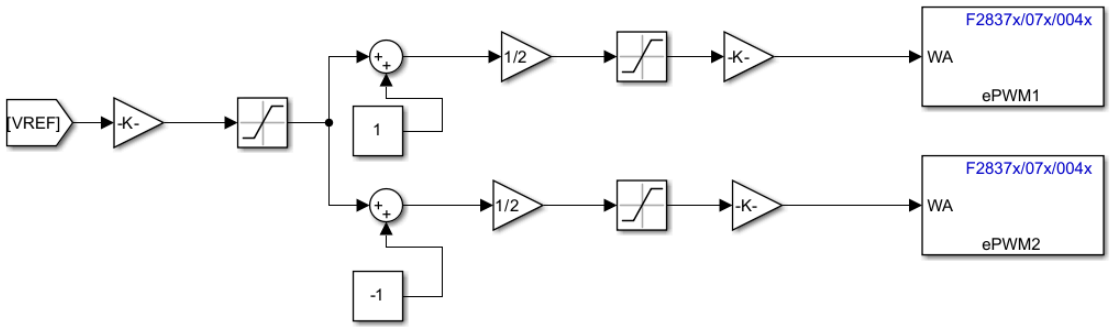


Image [14]: ePWM generation

To protect the power stage, the configuration also includes a dead-band generation. The Dead-Band unit programs a small delay between turning off one transistor and the turning on of its complementary device. This delay is automatically generated and applied, preventing cross conduction.

Synchronization between ePWM modules is handled by the ePWM synchronization network. This is initiated upon startup and ensures that all four (4) pwm signals are always synchronized. This alignment ensures constant sampling and also serves as a trigger source for the ADC units. This creates accurate and well timed control signals. Each ePWM module produces a complementary signal allowing for two (2) modules to produce four (4) signals.

Unfortunately this configuration was not able to be fully tested as it requires constant feedback from the ADC's as well as a live connection to the inverter hardware. For this reason, the PWM generator below was used to simulate the effectiveness of the control algorithm.

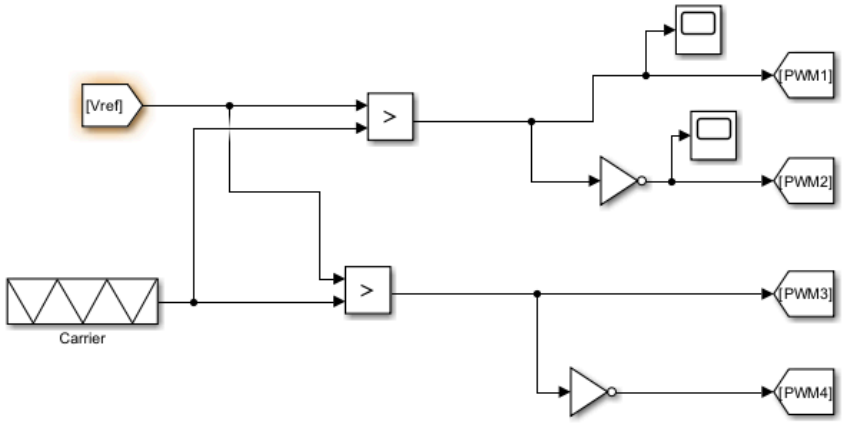


Image [15]: SPWM generation

CMSIS-DSP integration: the C2000 Ti microcontrollers have their own C28xDSP cores that are similar to CMSIS-DSP in that they perform math and filtering operations. The C2000s use both Control Law accelerators which execute control code separately from the main processing unit. Thus the CLA deals with PWM updates and ADC reads which are very time dependent. Typically a CPU will run tasks sequentially but with the Control Law Accelerator the ADC starts CLA tasks, the CLA computes the new duty cycle using the PI loop and updates the PWM registers.

ADC usage: The F28379D has four separate 12-bit ADCs. Each of these ADC modules have multiple Start of Conversion (SOC) channels which indicate which channel to sample, when to sample the channel, and how long to sample the channel before converting. We synchronize the ADCs with the PWM carrier waveform so that during each cycle current and voltage are sampled at consistent points which are

typically in the middle of a PWM period. Once the raw ADC values are captured, the firmware applies the appropriate scaling and offset corrections to translate the digital results into physically meaningful quantities. Voltage measurements are determined by scaling by the front end voltage divider and the internal gain of the isolated voltage sensor. The same procedure is applied to the current measurement, utilizing the zero current offset value found for the Hall Effect sensor. This is then scaled by sensor transfer gain to convert the voltage measurement into the desired current value. The diagram below illustrates this processing sequence, where each ADC result is first scaled down to match the expected bit rate, then scaled through the voltage-divider and sensor-gain factors, and finally offset-corrected. These conditioned signals form the inputs to the Alpha Beta transforms used by the control algorithm.

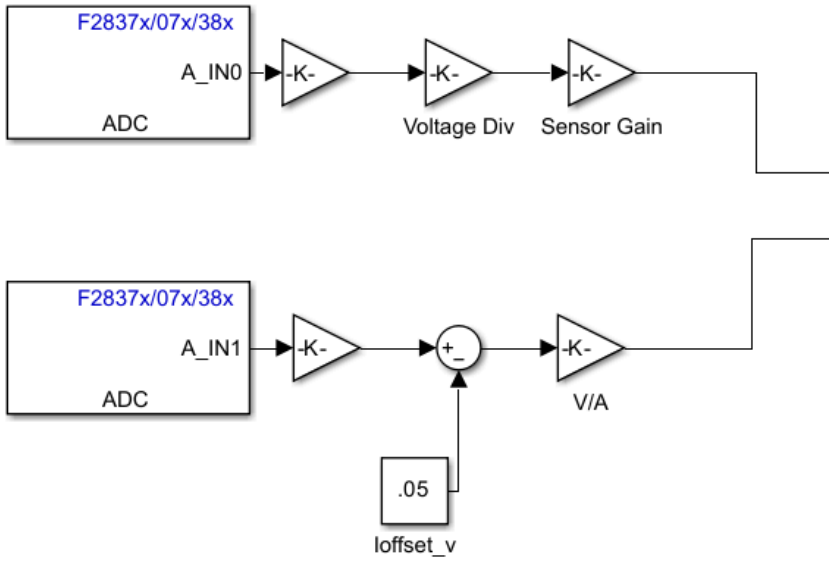


Image [16]: ADC signal Processing

Direct Memory Access (DMA) controller: The controller sends ADC results from the result register to a register in RAM without involving the CPU. The DMA is running in parallel with the CLA which is separately executing control logic.

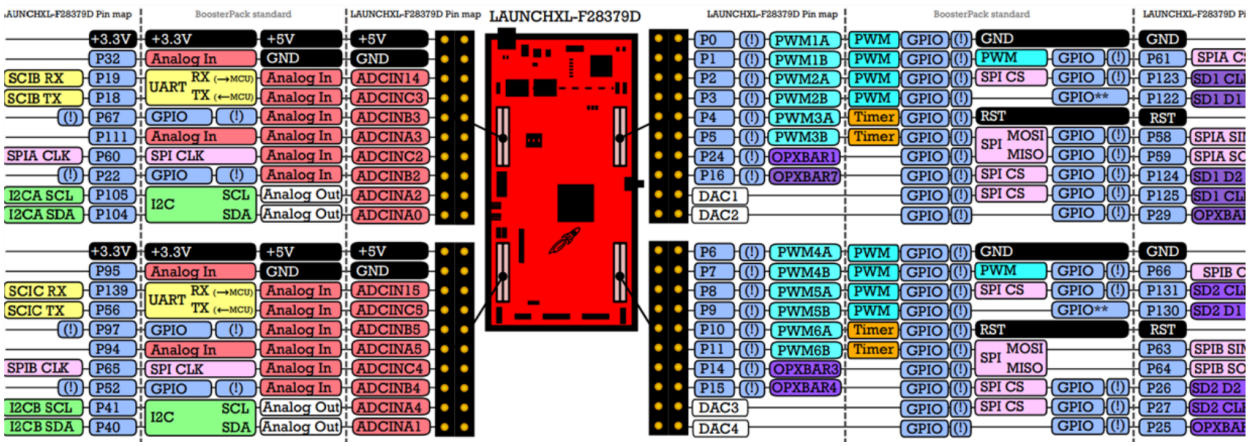


Image [17]: LAUNCHXL-F28379D Pin Diagram

VII. Integration and Testing

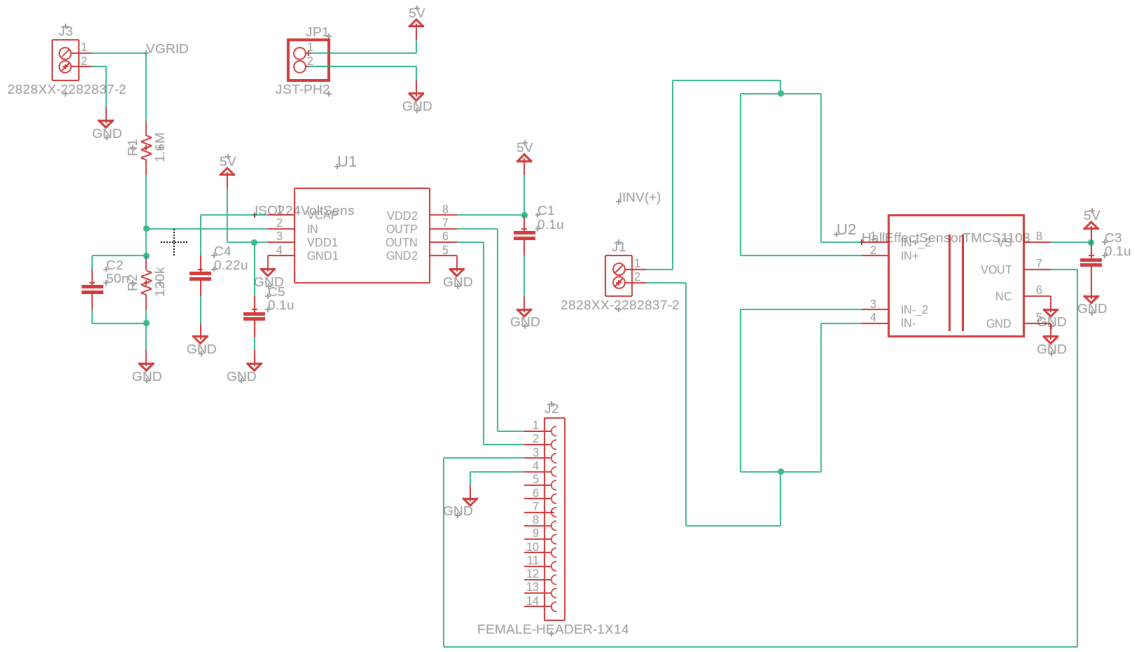


Image [18]:PCB schematic

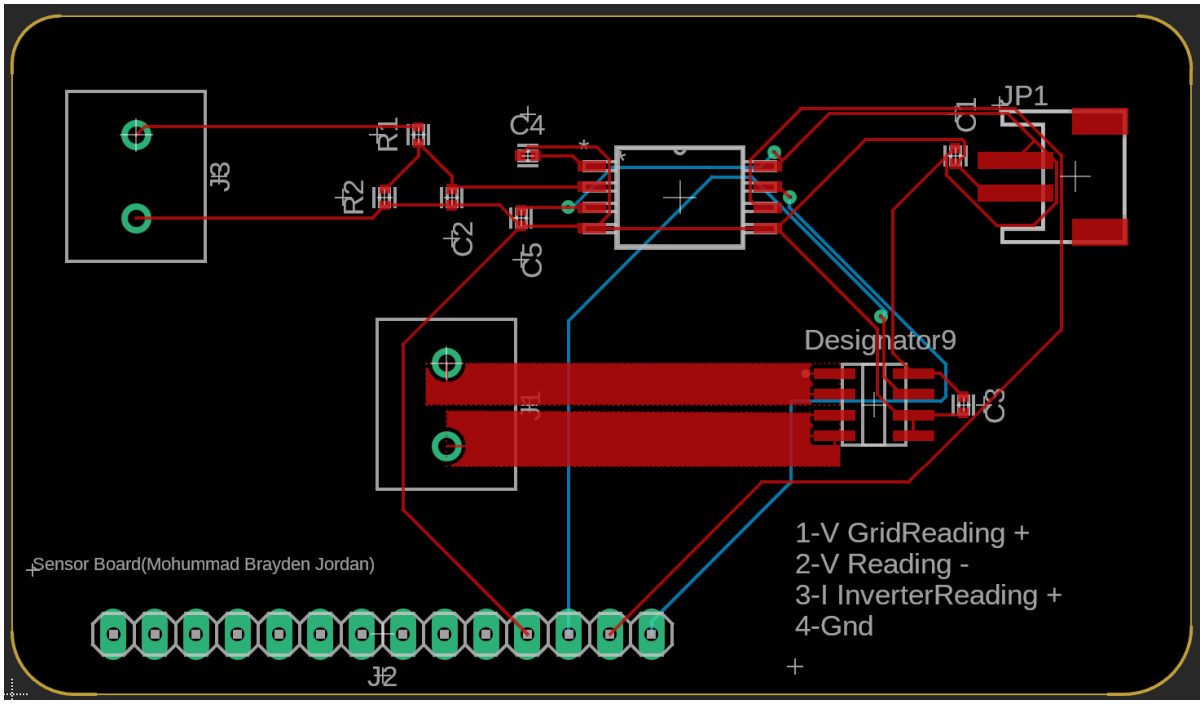


Image [19]:PCB layout

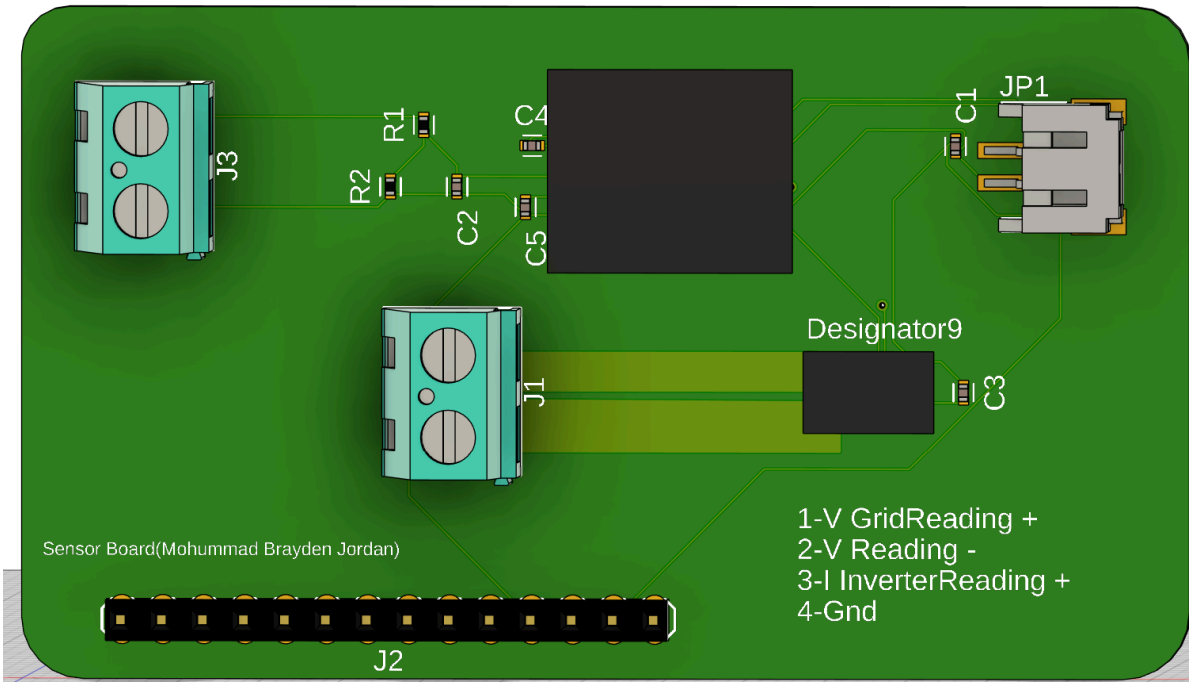


Image [20]: 3D PCB Layout

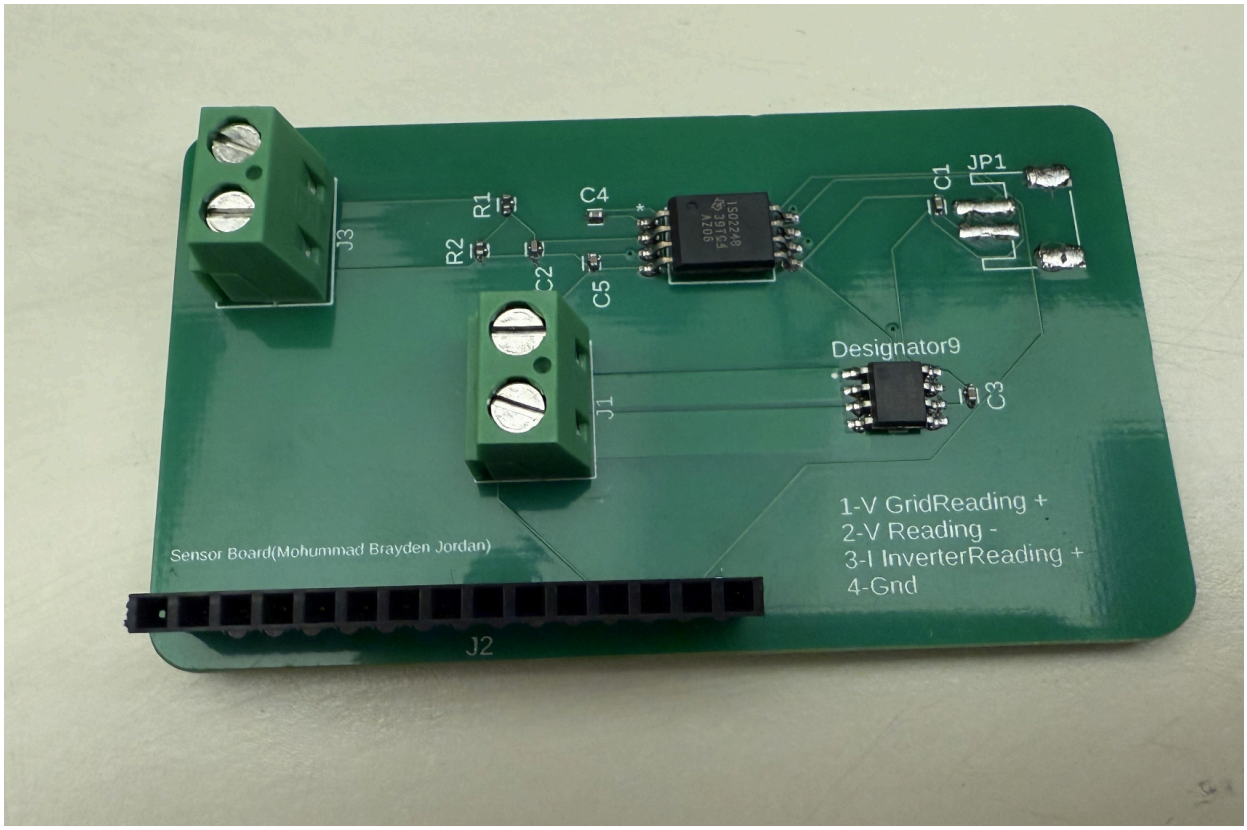


Image [21]: Populated PCB

PCB Functionality: In the figure above the ISO224 uses an isolated voltage sensor so that it can receive the grid voltage that is stepped down through a voltage divider and then outputs an isolated low voltage that can be fed to the ADC's and processed by the MCU. The hall effect sensor is used to measure the grid current via an output voltage proportional to the current.

Profiling the Current Sensor (TI TMCS1108 Hall-Effect Current Sensor)

Sensitivity error: This metric indicates how the sensor output current changes proportionally to the change in input conductor current. With varied inputs our sensitivity values hovers around .02. This makes sense because the Hall effect sensor is meant to measure large current and output small voltages proportionally that our MCU can process.

$$V_{out} = S\,(I_{in}) + V_{out,0A}$$

V_{out}: analog output voltage

S: ideal sensitivity of the device

I_{in}: isolated input current.

$V_{out,0A}$: zero current output voltage for the device variant

TABLE IV

SENSITIVITY ERROR CALCULATIONS Hall Effect Sensor

V_{out}	S	I_{in}	$V_{out,0A}$
111.0mV	0.02	3A	51mV
89.375mV	0.021	2A	51mV
70.625mV	0.022	1A	51mV
79.6mV	0.02	1.43A	51mV

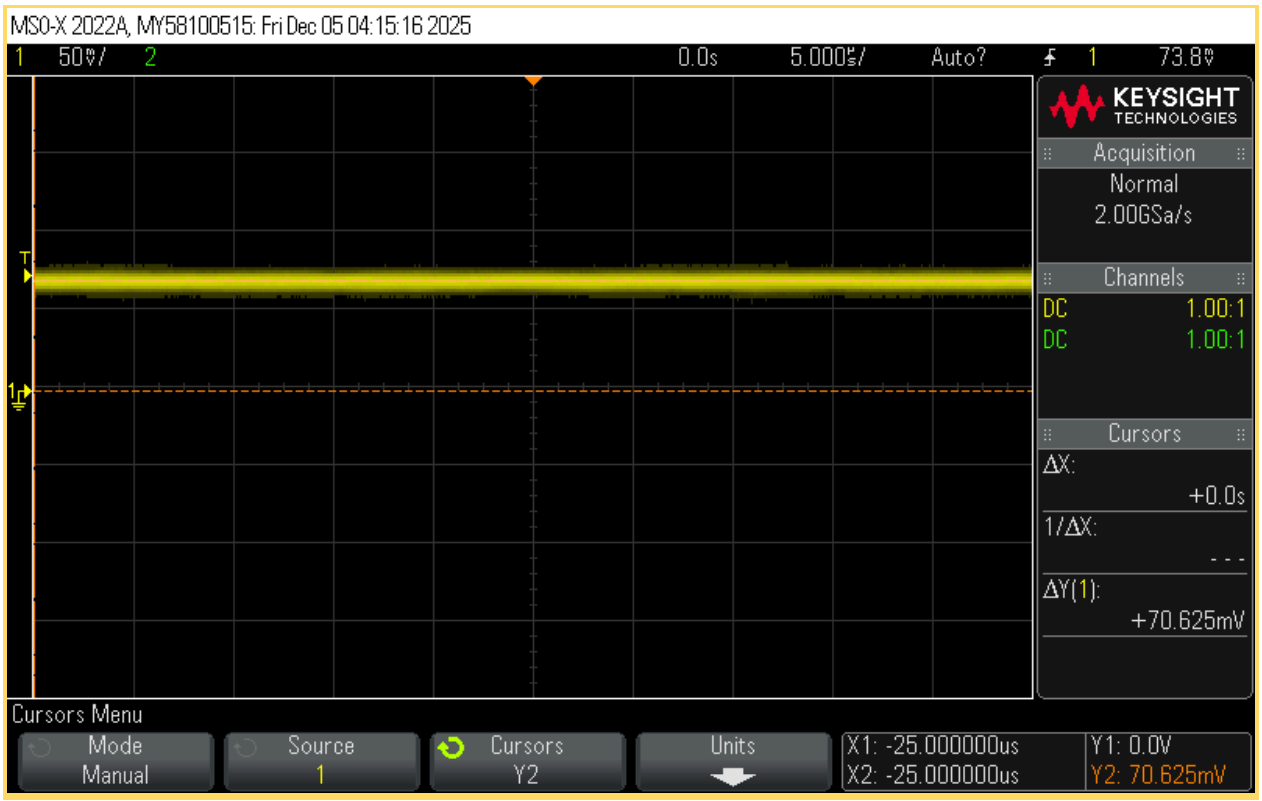


Image [22]: Hall Effect 1

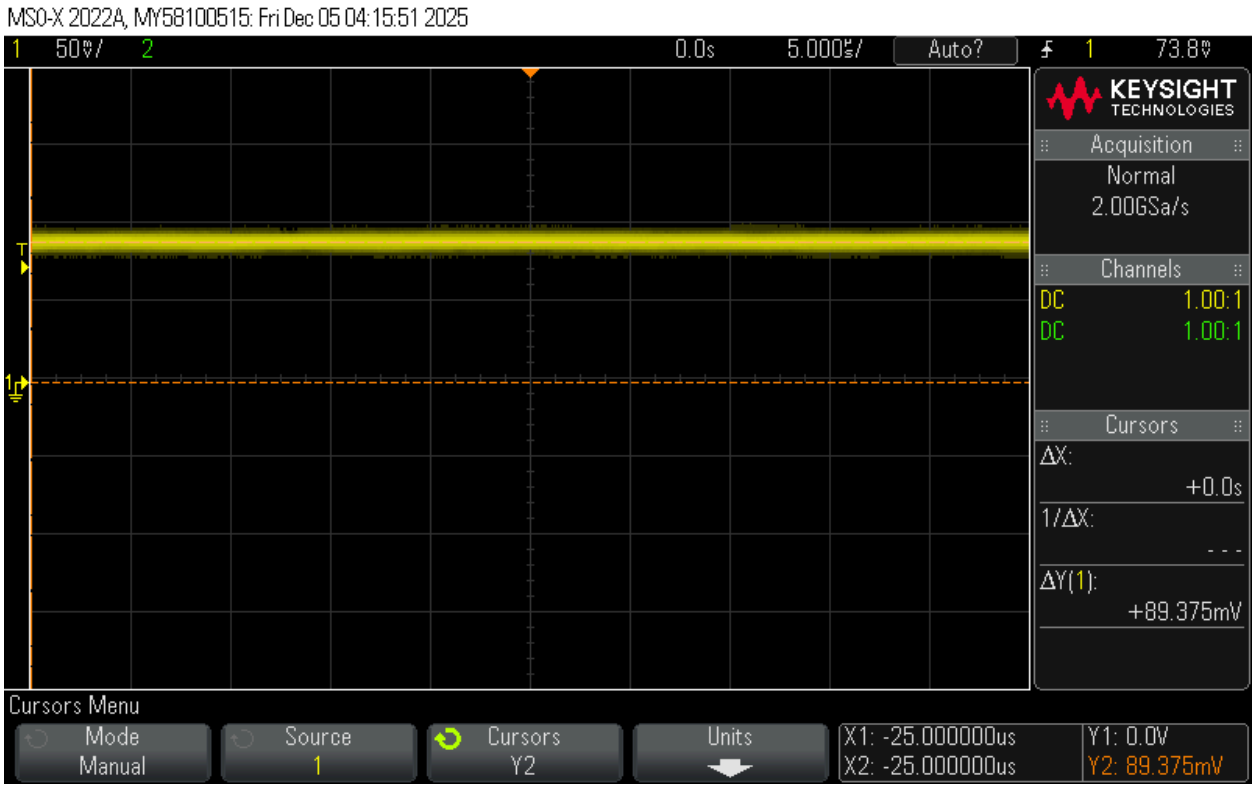


Image [23]: Hall Effect 2

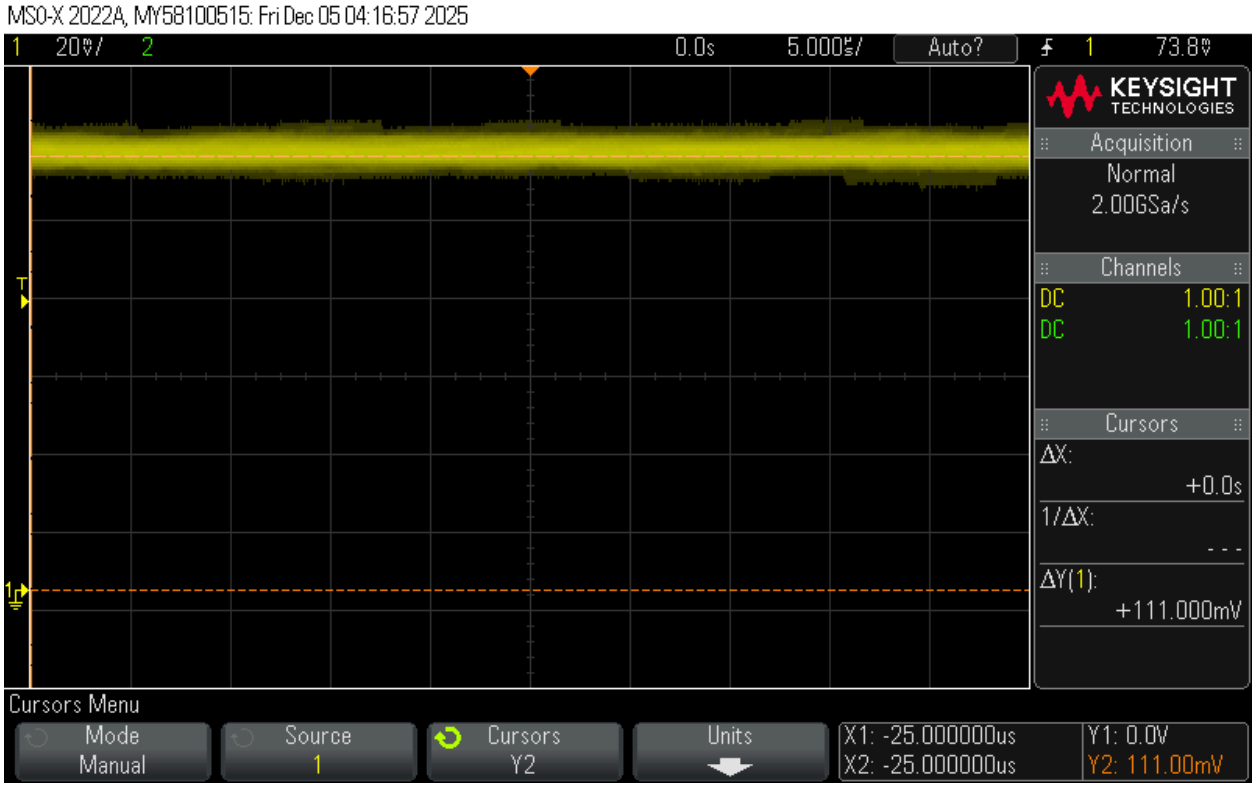


Image [24]: Hall Effect 3

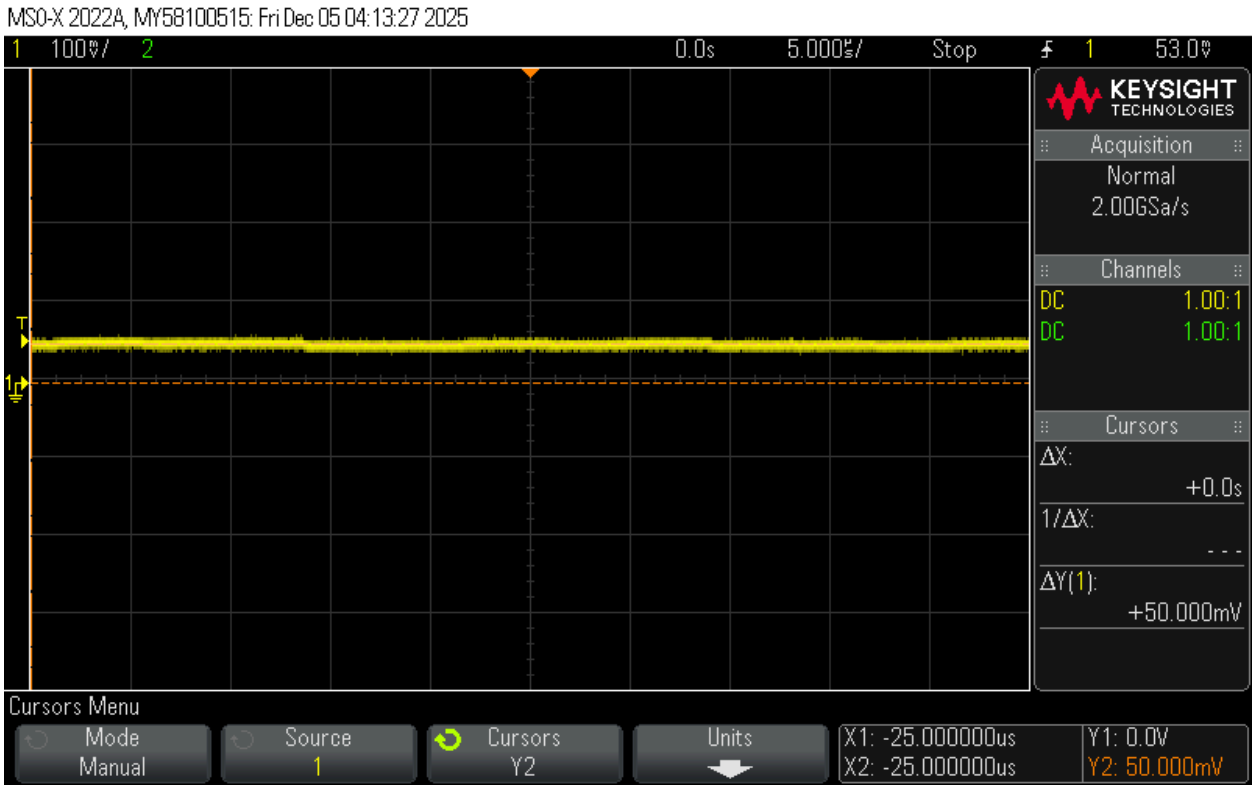


Image [25]: Voltage at 0 Amps Input Hall Effect

The Hall Effect sensor was intended to measure the inverter current which was a sinusoidal signal, however due to the available lab equipment the signal generator is only able to produce currents up to 100mA. Due to the high power nature of the sensor and current expectations in the design we were unable to profile the Hall Effect Senor under AC conditions.

Profiling the Voltage Sensor (TI ISO224 Reinforced Isolated Amplifier)

Calacultating Gain of the voltage sensor: To calculate the gain from the voltage divider we calculated the voltage divider value which gives us each input value to the voltage sensor. We then calculated gain using: $Gain = \frac{V_{in}}{V_{out}}$.

Equations: Voltage divider into Voltage sensor

$V_{in} = V^{+}(\frac{120k}{1.6M+120k})$

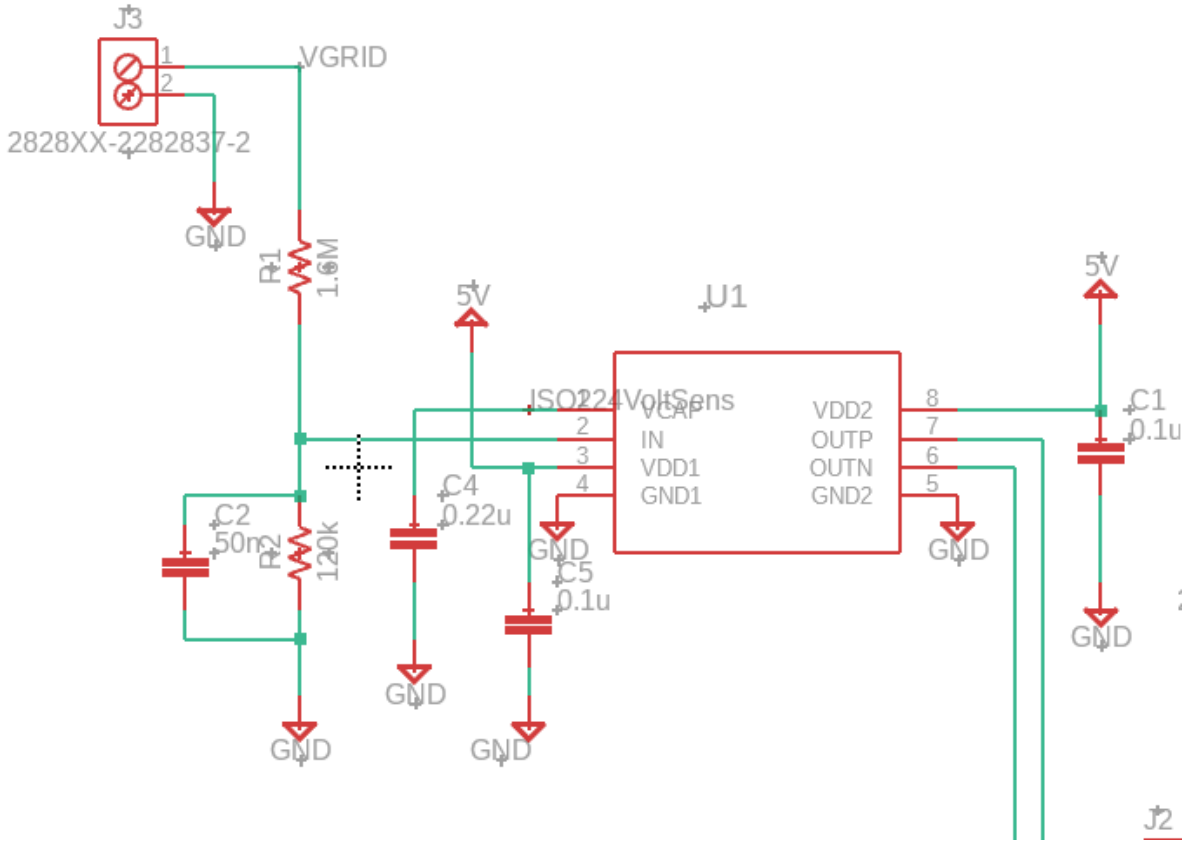


Image [26]: voltage sensor schematic used to calculate gain

TABLE V

GAIN VALUE CALCULATIONS VIA THE VOLTAGE SENSOR GIVEN VARYING PSEUDO GRID VOLTAGES

V^{+}(V)	V_{in}(V)	V_{out}(mV)	Gain
32	2.23	65.00	0.029
30	2.09	62.75	0.030
25	1.74	50.00	0.029
20	1.395	42.75	0.031
10	1.05	31.75	0.030

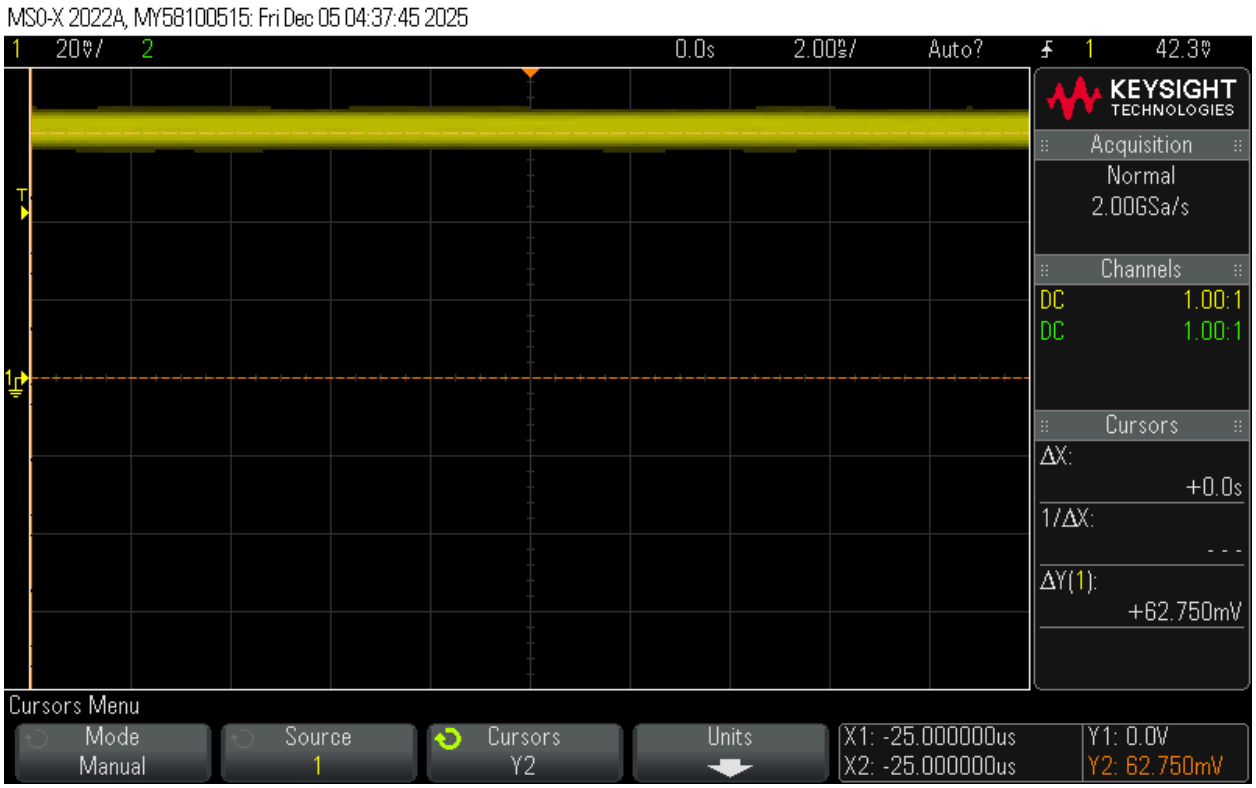


Image [27]: Voltage Sensor Measurement at DC

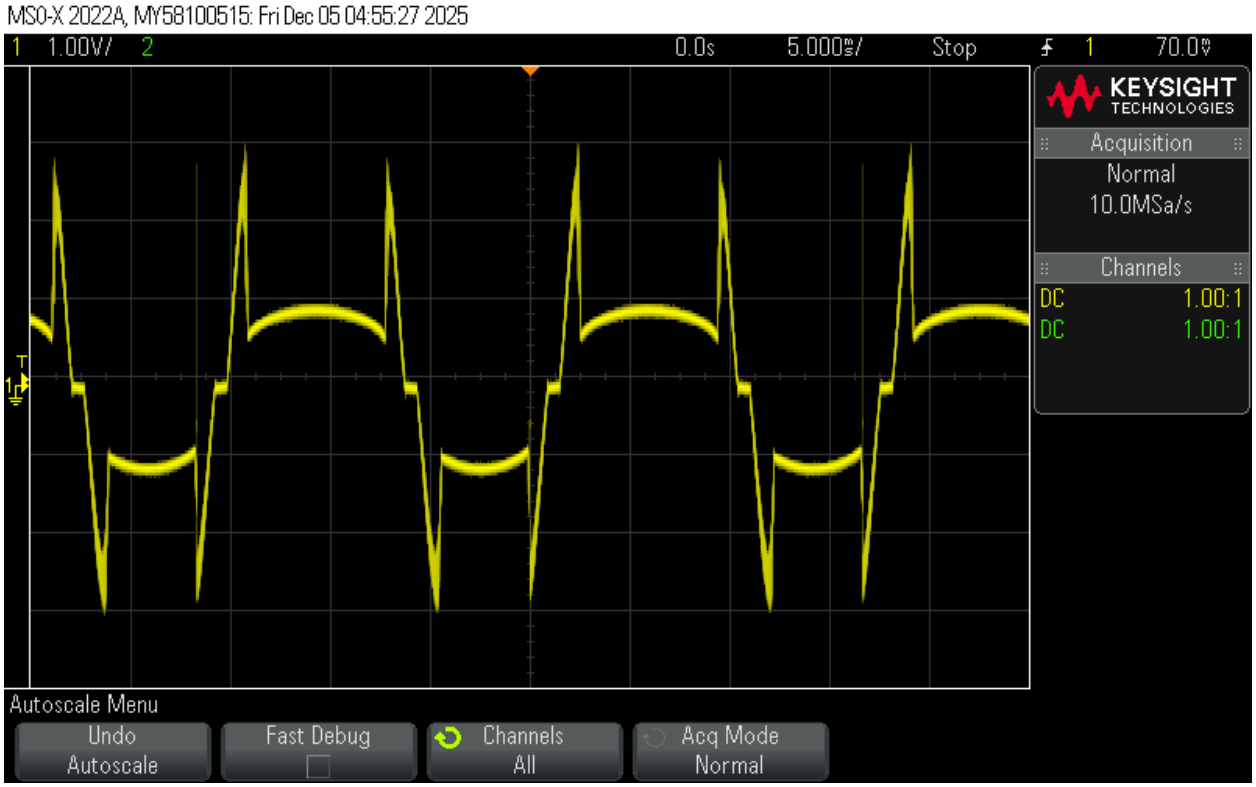
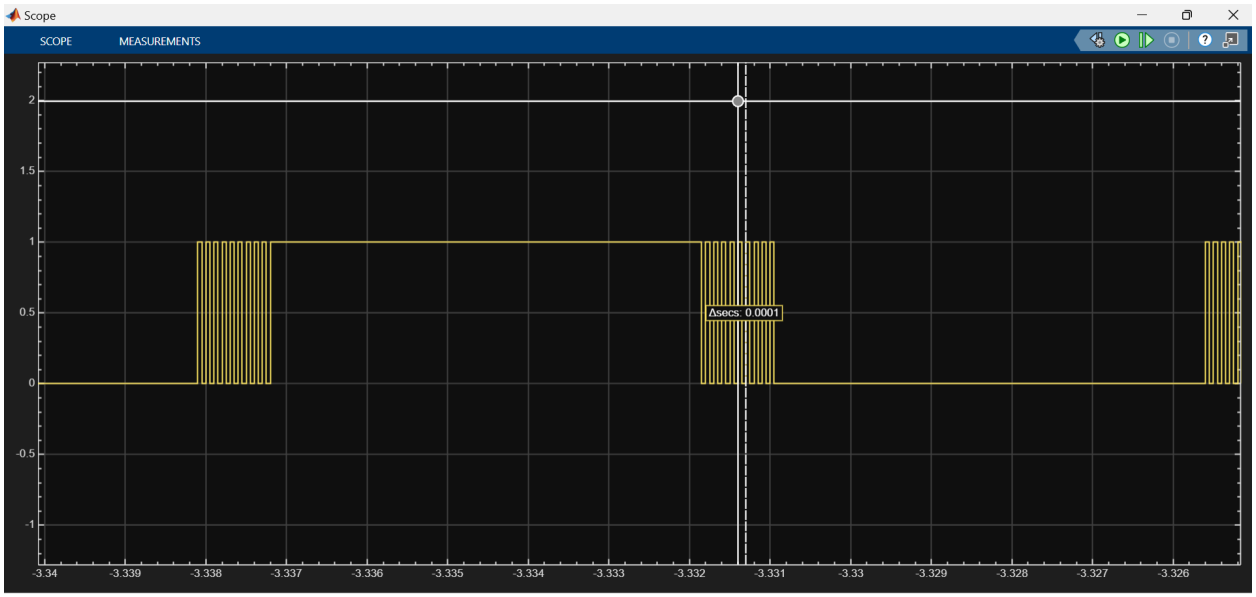


Image [28]: Voltage Sensor Measurement at $22V_{pp}$ AC

We can see that the voltage sensor is able to sense voltage at both DC and AC indicating that the sensor is working in the correct manner. However when sensing AC voltage we can see various spikes that if sampled using the ADC would produce inaccurate readings. This spiking occurs due to the voltage being sensed approaching an internal reference voltage. This transition from above or below the reference voltage to below or above causes the voltage being read to spike to an incorrect value. To avoid this spike the output of the sensor needed to be filtered using a low pass filter. This is a design consideration to be used in future projects and future implementations of any kind of amplifying voltage sensor.



Statistics

Controller/1	Max	Min	Peak to Peak	Mean	Standard Deviation	Median	RMS
Time (s)	-3.3381	-3.3400					
Value	1	0	1.0000	0.4362	0.4968	0.0000	0.6605

Image [29]: PWM Testing at 50 Hz (Simulink)

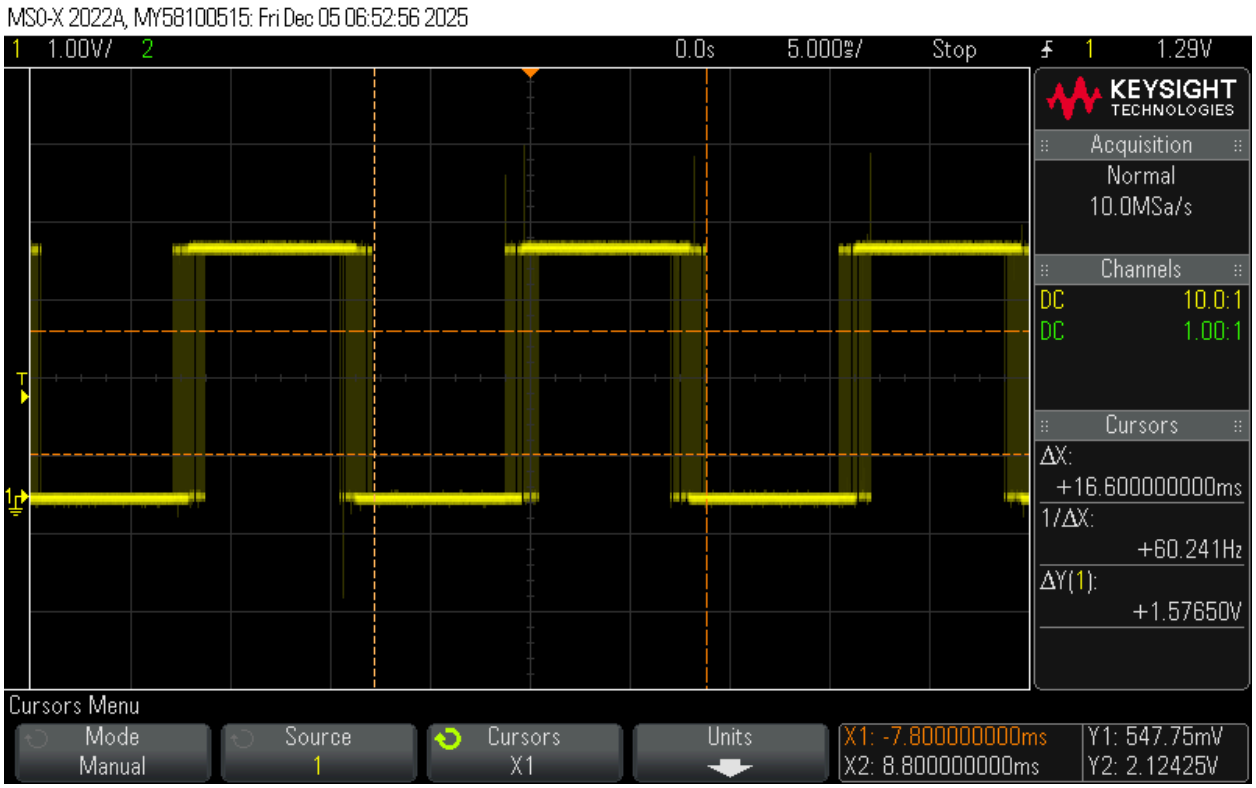


Image [30]: PWM scope shot at 60Hz (Oscilloscope)

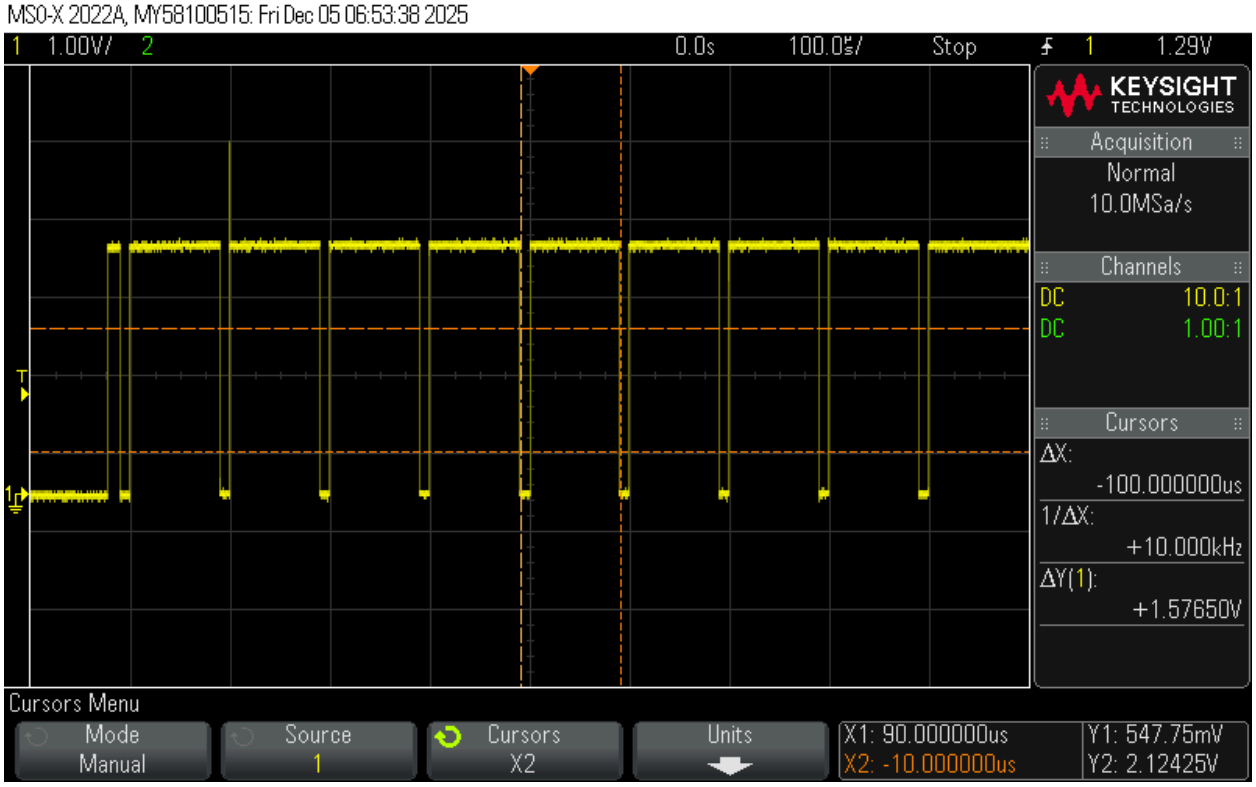


Image [31]: PWM scope Shot Showing Fsw

In order to test the PWM being produced from our control system we tied the PWM output to a GPIO Output pin and probed it using an oscilloscope. We were expecting a repeated sequence of larger 60Hz pulses that had varying pulses within whose frequency would be near that of our carrier signal. This carrier signal that exists within the control system has a frequency of 10kHz and in the reading of the PWM transition regions we were able to measure a matching frequency. This demonstrates a successful attempt at creating a control system using our microcontroller platform. Theoretically if this PWM were to be applied to a physical full bridge inverter, with the DC and AC sides being defined within the scope of our project, we would get a 60Hz 120V sinusoidal signal on the AC grid side.

VIII. Conclusions and Recommendations

The simulation of the VSC control system demonstrated the full scale plausibility of a control system for a single phase voltage source converter. The system measures grid voltage and current waveforms and then uses a phase lock loop to process and derive the phase angle. This phase angle can then be used to ensure that the output of the VSC is in sync with the grid.

The control system was measured using simulink scope blocks which probed both voltage and current nodes showing output voltages of about 120 Vrms or 170Vpeak at the 60hz standard grid frequency in the United States. This simulation demonstrated the precise synchronization of the converter to the standard voltage grid. The Fast Fourier Transform (FFT) shows that the total harmonic distortion of the output is less than 5% for the given simulated system.

After developing the control system the C code was extracted from simulink based on the simulation model, and sent to the LAUNCHXL-F28379D microcontroller. This script created a PWM output from the MCU equivalent to that of the simulink simulation. Through the implementation of the PWM output onto the MCU the project demonstrated the ability to control physical real time control systems.

Finally we tested our sensor board which relied on a Hall-Effect current sensor and an isolated amplifier voltage sensor to measure a scaled down version of the voltage and current measurements coming from the grid. The sensor board testing showed the capability of using these kinds of power isolating converters in order to sample high power systems. While we were able to read voltages and currents using the sensors the high power implementation was not reached.

Further works would see the sensor outputs be filtered so they can be successfully sampled by ADCs. Along with filtered outputs we would find a lab or power source that can provide the kinds of voltages and currents that our control system is expecting at the input. The final step for full integration would be to connect the PWMs to a full bridge IGBT single phase inverter and read the output voltage and current. This final step is the simplest as the PWMs only need to be connected to the gate drivers of the power circuit. However, this is also the most dangerous and requires the most caution of all the steps.

IX. Bibliography

[1] Texas Instruments, *TMS320F28379D 32-bit Floating-Point Microcontroller*, 2024. [Online]. Available: https://www.ti.com/product/TMS320F28379D. [Accessed: Feb. 5, 2025].

[2] IEEE PSRC Working Group, *Impact of Voltage Source Converter (VSC) Based HVDC Systems on AC System Protection*, IEEE Power System Relaying and Control Committee (PSRC) Report, 2017. [Online]. Available: https://www.pes-psrc.org/kb/report/107.pdf.

[3] E-T-A Elektrotechnische Apparate GmbH, *Fast Acting Over Current Power Circuit Protection Scheme*, Application Note, 2024. [Online]. Available: https://www.e-t-a.com/fileadmin/user_upload/USA/PDF-files/Application_Notes/12109_Fast_Acting_Over_Current_Power_Circuit_Protection.pdf. [Accessed: Feb. 5, 2025].

[4] Keysight Technologies, *How Does Over Current Protection (OCP) Work?*, 2024. [Online]. Available: https://docs.keysight.com/kkbopen/how-does-over-current-protection-ocp-work-620692888.html. [Accessed: Feb. 5, 2025].

[5] M. E. Baran and N. R. Mahajan, "Overcurrent Protection on Voltage-Source-Converter-Based Multiterminal DC Systems," *IEEE Transactions on Power Delivery*, vol. 22, no. 1, pp. 406–412, Jan. 2007. [Online]. Available: https://ieeexplore.ieee.org/document/4039403. [Accessed: Feb. 5, 2025].

[6] N. Flourentzou, V. G. Agelidis, and G. D. Demetriades, "VSC-Based HVDC Power Transmission Systems: An Overview," *IEEE Transactions on Power Electronics*, vol. 24, no. 3, pp. 592–602, Mar. 2009, doi: 10.1109/TPEL.2008.2008441. [Accessed: Feb. 5, 2025].

[7] Texas Instruments, *Voltage Source Inverter Reference Design*, 2025. [Online]. Available: https://www.ti.com/lit/ug/tiduay6c/tiduay6c.pdf. [Accessed: Feb. 5, 2025].

[8] O. E. Oni, I. E. Davidson, and K. N. I. Mbangula, "A Review of LCC-HVDC and VSC-HVDC Technologies and Applications," in *Proc. 2016 IEEE 16th Int. Conf. Environment and Electrical Engineering (EEEIC)*, Jun. 2016, pp. 1–7, doi: 10.1109/EEEIC.2016.7555677.

[9] H. Lindblom, *Design and Construction of a Voltage Source Converter for a Laboratory Setup*, Uppsala Univ., 2019. [Online]. Available: https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1313281&dswid=-9080.

[10] L. de Andrade and T. P. de Leão, "A Brief History of Direct Current in Electrical Power Systems," in *Proc. 2012 3rd IEEE HISTory of ELectro-technology CONference (HISTELCON)*, Sep. 2012, pp. 1–6, doi: 10.1109/HISTELCON.2012.6487566.

[11] *IEEE Standard Conformance Test Procedures for Equipment Interconnecting Distributed Energy Resources with Electric Power Systems and Associated Interfaces*, IEEE Std 1547.1-2020, pp. 1–282, May 21, 2020, doi: 10.1109/IEEESTD.2020.9097534.

[12] *IEEE Guide for Self-Commutated Converters*, ANSI/IEEE Std 936-1987, 1987.

[13] *IEEE Guide for Harmonic Control and Reactive Compensation of Static Power Converters*, ANSI/IEEE Std 519-1981, pp. 1–54, Apr. 27, 1981.

[14] *IEEE Recommended Practices and Requirements for Harmonic Control in Electrical Power Systems*, IEEE Std 519-1992, pp. 1–112, Apr. 9, 1993.

Appendix A – Senior Project Analysis

Project Title: DC to Ac Voltage Source Converter Control System

Student's Name: Mohummad Elgassier Student's Signature:

Student's Name: Brayden Young Student's Signature:

Student's Name: Jordan Reichhardt Student's Signature:

Advisor's Name: William Ahlgren Advisor's Initials:

Date: 12/11/2025

1. **Summary of Functional Requirements**

The Voltage Source Converter Control System is designed to regulate the DC-AC power conversion of a half bridge Voltage Source Converter for grid integration. This device facilitates the efficient conversion of DC power to AC while maintaining stability in order to tie into the larger scale American power grid. The control system enables control with precision in order to control the power flow, voltage, and current. This control allows for efficient and reliable HVDC power transmission. Measuring and managing the power quality, efficiency, and grid compliance the control system ensures that the output is grid compliant and as efficient/stable as possible. With the addition of added capabilities to do real-time monitoring for overcurrent, thermal, and fault events. The system is not only power efficient but also adds elements of safety.

2. Primary Constraints

While developing the Voltage Source Converter Control System, several challenges presented themselves. Voltage source converters require very fast and accurate switching times in order to keep precise values of active and reactive power. This led to a constraint on the microcontroller to be used in the design. High speed microcontrollers often lead to a higher price point which limits the speed available to stay within the project budget. This led to the decision to use the TMS320F28379D as the main controller for the project as it meets the required speed and PWM capability while also staying within the project budget. Another limiting factor within the project is the lack of direct access to a VSC. Without the ability to test the control system on a VSC, a decision was made to use simulations or acquire and use a HIL system (Hardware-in-the-Loop). A final constraint that became apparent was code generation from simulation. Using a board such as the STM-32L4 requires converting Simulink simulations to working C code to be implemented on the board. This is not an easy conversion and often leads to buggy code. To overcome this, the TMS320F28379D was chosen as it comes with software

that is specifically made to convert from Simulink to properly formatted C code.

3. Economic

The VSC Control team, composed of three members, can be split into three main tasks. Control system simulation, microcontroller programming, and low power PCB design. This allows for a relatively low total cost as simulation and programming only take on the price of hourly wages for time allotted to them respectively. Most of the costs for this project will be accrued near the end of the timeline when PCB manufacturing begins, requiring components to be purchased. Major costs will include the MCU at a price of $46.80, PCB design costs ranging from $10-$35, and various sensors and components totalling to approximately $210. Simulations, test equipment, and HIL simulator can be provided by Cal Poly at no cost to the project. The project timing is estimated at 33 weeks with 12 hours of work per week per member. Using average hourly wages for an entry level Electrical Engineer in California, $43/hour was used to compute a total project cost of $51,084. This product has a long expected lifetime with no to little maintenance cost as the PCB should be able to have a lifespan of 15-20 years of regular use.

4. If manufactured on a commercial basis

With recent developments in HVDC technology and the increased amount of HVDC links being commissioned in Europe the necessity for Voltage Source Converters has steadily increased over the years. With smaller scale applications as well there is an increase in the need for Voltage Source Converter Control Systems that are flexible and easily managed. A rough estimate in the amount of new installations per year is around 300. In order to manufacture this device the cost is typically around $300 depending on the type of PCB materials used as well as the method in which the PCB is populated. Typical market cost of Voltage Source Converters vary widely depending on the levels of Voltages being transmitted. At our levels the market purchase price of our device would be around $2,000 . With this being the cost the Average profit would be around $510,00. If considering the area then in San Luis Obispo the average utility rate is 0.42$/kWh(according to the California Public Utilities Commission). This is highly dependent upon what time of day the device is used and how much energy is used in the time operated however we can expect around 1% power loss using this device so the typical cost of operation for the VSC is .0042$/kWh.

5. Environmental

A significant portion of a voltage source converter's environmental impact occurs in its manufacturing phase and in its end lifecycle. In its use phase the VSC control system is there to regulate and increase efficiency having a positive effect. In that cycle of its life its environmental impact comes from powering the microcontroller and sensors. The manufacturing of the Voltage Source Converter (VSC) control system involves the extraction and processing of materials such as silicon, gallium nitride (GaN), copper, and various polymers. Mining these raw materials have significant impacts on the environment along with the manufacturing of semiconductor chips consuming significant amounts of energy. During the products end lifecycle many of the materials that are used in the manufacturing process and are necessary components of the electronics can be harmful e-waste if not disposed properly. The

VSC control system improves the efficiency of HVDC transmission seeing less losses in the conversion to AC. This improvement in efficiency leads to lower carbon emissions and greater sustainability by delivering more efficient power. It is also a crucial piece in integrating many renewable sources to the grid providing more alternative and cleaner sources of energy to power our grid.

6. Manufacturability

Manufacturing a VSC Control System comes with several constraints including circuit design, component availability, and system reliability. The system must be able to handle both analog and digital signals with high precision to manage switching effectively. Choosing to use a printed circuit board (PCB) is essential for handling these signals while also eliminating electromagnetic interference (EMI). The board provides stable performance at higher frequencies ,which is essential for VSC systems, while also providing easy manufacturing. PCBs use readily available components with high accuracy ratings and provide precise connections, eliminating the risk of shorting or electrical shock. When considering ease of construction, the control system will only require that components be soldered to the board correctly. This entails using a stencil to place solder on desired locations, attaching components, and placing in a reflow oven to connect all components to the board. This is all fairly simple and can lead to quick manufacturing times with low risk of mistakes.

7. Sustainability

Multiple factors pose challenges to the maintenance of Voltage Source Converters (VSC) because of the complexity of power electronics, system reliability, and thermal management. Additionally, VSC systems require fault detection to prevent large and dangerous levels of excess current and voltage, this requires regular maintenance and trained personnel. Further, the integration of VSC technology into power grids facilitates the transmission of energy over long distances, reducing the global dependency on fossil fuels to power our world. Ultimately this contributes to global sustainability goals. Unfortunately, improving these systems often includes developing more advanced configurations such as modular Multi-level Converter (MMCs) which are more relevant to high-power applications as they provide better harmonics, scalability, and efficiency. While they are useful in high-power applications, MMCs are more complicated than traditional VSC as they use cascaded half-bridge or full-bridge submodules, each containing capacitors and semiconductor switches (IGBTs or MOSFETs). This separation of components allows for reactive and active power to be managed individually.

8. Ethical Considerations

The ethical ramifications of VSC systems can be seen from design and manufacturing to their application and potential misuse. Ethical design requires that systems are produced in a way that is safe to install and operate, is environmentally responsible, and an efficient use of finite resources. The IEEE code of ethics emphasizes that it is the responsibility of engineers to ensure public safety including minimizing risks associated with high-voltage systems. This responsibility requires that extensive testing is done before the system is deployed for public use. Additionally for VSC control systems that are interconnected cyber security attacks may pose a potential risk of system manipulation. Implementing extensive security measures is

crucial in mitigating this risk. Ethical constructs such as utilitarianism suggest that the benefits of improving environmental sustainability and the connectedness of energy systems have short-term challenges and costs associated with them.

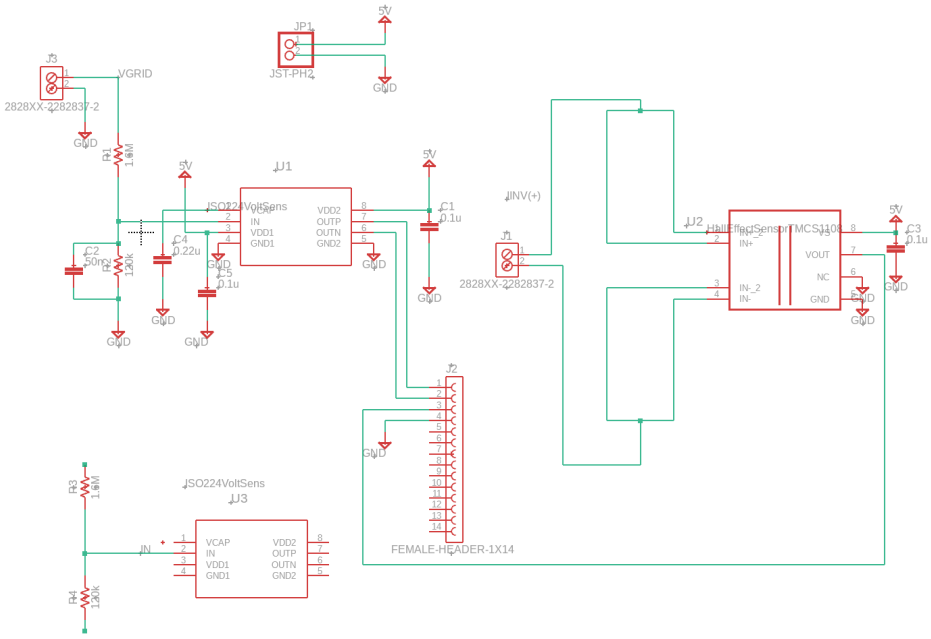
9. Health and Safety

The engineering, manufacturing, and operation of VSC systems involve health and safety risks both to the designers and to the public that must be mitigated. Within the controls sector of VSC systems currents and voltages above the system allowance must be mitigated using protection techniques. Fault protection techniques must be implemented in the control design for the VSC and additional safety measures such as insulation and protective enclosures must be constructed around the physical system so that there are both software and physical layers of protection. During manufacturing of large-scale VSCs manufacturers must wear PPE and comply with electrical safety standards. Additional safety measures must be implemented both on the control system of the VSC and physically to protect maintenance personnel. These personnel will require specific training for the system so they are equipped to manage emergency shutdown procedures, and system fault diagnostics.

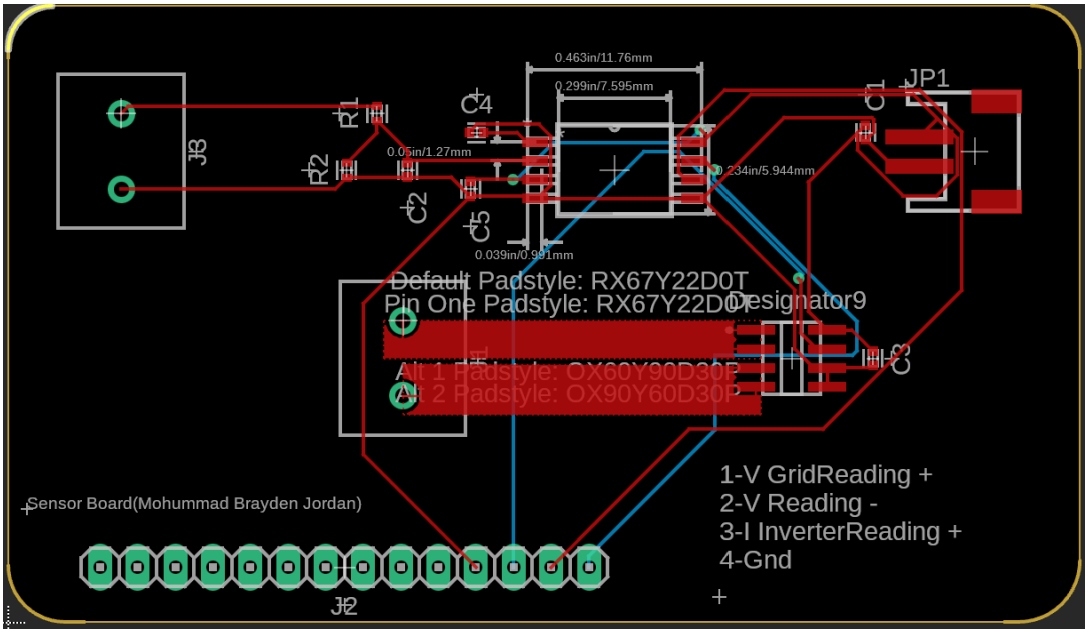
10. Social and Political Considerations

VSCs affect an array of stakeholders including but not limited to utility companies, regulatory bodies, local communities, and energy consumers both individual and large scale. The main advantage of VSC-based HVDC systems is their ability to transmit electricity over long distances efficiently. This is particularly useful in the integration of remote renewable energies into power grids which will ultimately foster economic growth founded on environmental sustainability and efficiency of energy integration (by minimizing power loss). The increase in efficiency of integration is beneficial to energy companies as it reduces their losses, and individuals in remote communities because if the energy company is losing less power the cost of energy will decrease for consumers. However, there is some potential for disruption of surrounding environments when installing VSCs into power systems. This disruption, however, is minimal as it simply requires modifications of current energy systems. Another social and political factor to consider is the disparities in the distribution of benefits that VSCs provide. If urban areas gain improved energy access while rural and underdeveloped areas remain isolated in the power grid the project could increase power disparities. Instead, regulators must focus on first installing VSCs in remote locations where the efficiency of power integration will be most improved. In a geopolitical landscape, VSCs allow international energy exchange between adjacent countries within the same continent. This would impact both energy security and international relations. As nations integrate more renewable energy sources into their grids more planning is necessary to balance national energy independence and collaborative international energy agreements.

Appendix B – Schematics



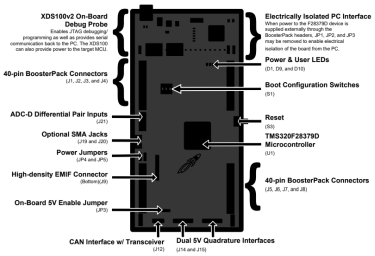
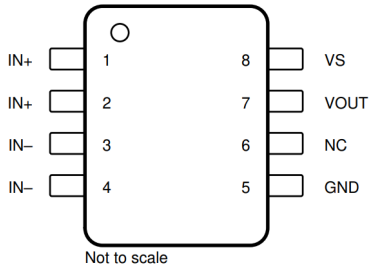
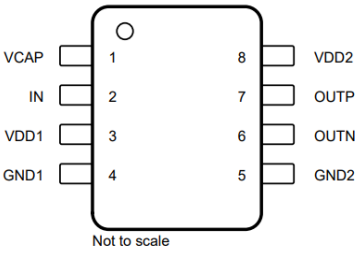
Image[32]: PCB schematic



Image[33]: Routed PCB

TABLE VI

COMPONENT PIN DIAGRAMS

Part	Description	Pin Diagram
LAUNCHXL-F28379D	MCU	 Figure 1. LAUNCHXL-F28379D Board Overview
TMCS1108A3UQDR	Hall Effect Current sensor	 Image[34]: Hall Pin Diagram
ABLS-16.384MHZ-B4-T	Crystal Oscillator	 Image[35]: Oscillator Pin Diagram

Appendix C – Parts List and Gantt Charts

Cost Estimates

Cost of Materials:

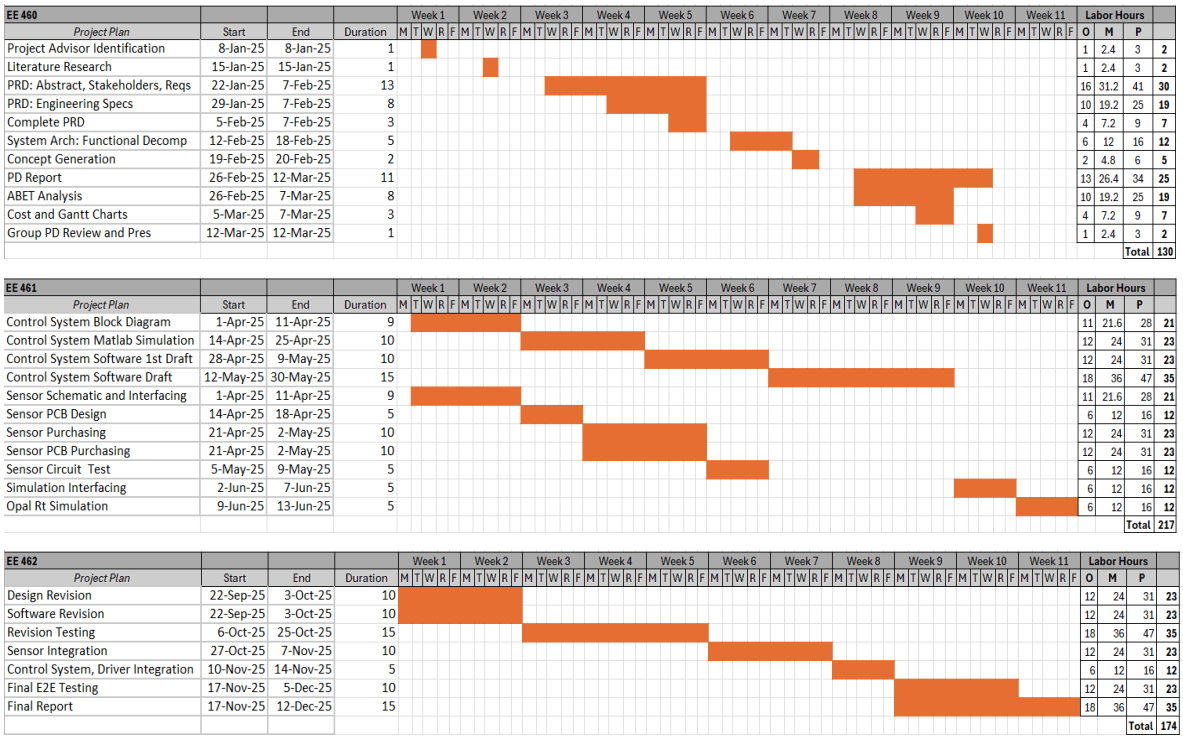
Part	Description	Unit Cost	Total Cost ($)	Justification
MCU (LAUNCHXL-F28379D)	Microcontroller for control and processing	46.8	93.6	Used in the Control module for PWM generation and system processing. Necessary to have 2 units for redundancy/testing. Unit cost is reliable as it is a fixed cost. No variation in price range.
Hall Effect Current Sensor (VTMCS1108A2UQDR)	Measures current for feedback control	1.53-2.50	15.30-25.00	Used in Current Measurement module. 10 sensors are needed for three-phase current monitoring and redundancy. Unit price varies between $1.53-$2.50, so total cost varies. Estimate is based on typical market prices.
PCB Design Costs	Custom PCB for circuit integration	Oct-35	30-105	Used across all modules for circuit integration. Cost varies based on board complexity but will likely bea two layer board. Estimate is from $10-$35 per PCB, and 3 are planned to be made, making total cost between $30-$105.
Crystal Oscillator (ABLS-16.384MHZ-B4-	Provides stable clock for MCU	0.25	0.75	Used in the Control module for MCU clock

T)	timing			cycles. 3 units purchased for redundancy. Unit cost is reliable ($0.25 each) as it's a standard component with minimal deviations in price.
Resistor Kit	Provides necessary resistances for PCB circuits	5	5	Used in Voltage Measurement and Control modules for signal conditioning. Only 1 kit needed, and unit cost is fixed at $5.Minimal to no variation in price.
Capacitors	Stabilizes voltage and current signals	0.30-1.50	9-45	Used in the Control and Voltage Measurement modules. 30 capacitors purchased to cover all circuit needs. Price range varies between $0.30-$1.50 per capacitor, leading to a total cost variation of $9-$45.
3-Phase Current & Voltage Sensor IC (ATM90E32AS-AU-Y)	Monitors voltage and current for processing	5.6	16.8	Used in Voltage and Current Measurement modules. 3 units needed for three-phase sensing. Unit price is fixed at $5.60 each, leading to a total cost of $16.80.
Rogowski Coil	Non-intrusive current sensing	4.05	12.15	Used in the Current Measurement module for AC current monitoring. 3 units needed for a three-phase system. Unit cost is fixed at $4.05, making total cost $12.15.
Banana Female Side Socket	Connection interface for external probes	8.99	8.99	Used in system interface for easy external connections. Only 1 unit needed. Price is fixed at $8.99, no variation.
3D Printed Enclosure	Housing for system components	15-50	15-50	Used to enclose the electronics and provide

				structural protection. Price varies from $15-$50 depending on material and complexity. Only 1 enclosure was needed.
Total	Overall cost estimation	-	206.59 - 362.29	Summation of all component costs, including variations where applicable.

Labor Cost

For this project, we assume an hourly wage of about $43 based on the average salary for recent electrical engineering (EE) graduates in California working in controls and embedded systems design for power systems. The estimated work per person is 12 hours/ week, with 3 team members contributing to the project. Over our 33-week development period, the total labor cost will be $17,028. This estimate aligns with comparable industry jobs and the compensation for those given jobs. Additionally it reflects the effort required to design, prototype, and test the system as outlined in our Gantt Chart Schedule (Section 3). The timeline includes phases for research, hardware development, software implementation, integration, and final testing, ensuring a realistic assessment of labor costs for project completion.



EE 460				Labor Hours			
Project Plan	Start	End	Duration	O	M	P	
Project Advisor Identification	8-Jan-25	8-Jan-25	1	1	2.4	3	2
Literature Research	15-Jan-25	15-Jan-25	1	1	2.4	3	2
PRD: Abstract, Stakeholders, Reqs	22-Jan-25	7-Feb-25	13	16	31.2	41	30
PRD: Engineering Specs	29-Jan-25	7-Feb-25	8	10	19.2	25	19
Complete PRD	5-Feb-25	7-Feb-25	3	4	7.2	9	7
System Arch: Functional Decomp	12-Feb-25	18-Feb-25	5	6	12	16	12
Concept Generation	19-Feb-25	20-Feb-25	2	2	4.8	6	5
PD Report	26-Feb-25	12-Mar-25	11	13	26.4	34	25
ABET Analysis	26-Feb-25	7-Mar-25	8	10	19.2	25	19
Cost and Gantt Charts	5-Mar-25	7-Mar-25	3	4	7.2	9	7
Group PD Review and Pres	12-Mar-25	12-Mar-25	1	1	2.4	3	2
						Total	130

EE 461				Labor Hours			
Project Plan	Start	End	Duration	O	M	P	
Control System Block Diagram	1-Apr-25	11-Apr-25	9	11	21.6	28	21
Control System Matlab Simulation	14-Apr-25	25-Apr-25	10	12	24	31	23
Control System Software 1st Draft	28-Apr-25	9-May-25	10	12	24	31	23
Control System Software Draft	12-May-25	30-May-25	15	18	36	47	35
Sensor Schematic and Interfacing	1-Apr-25	11-Apr-25	9	11	21.6	28	21
Sensor PCB Design	14-Apr-25	18-Apr-25	5	6	12	16	12
Sensor Purchasing	21-Apr-25	2-May-25	10	12	24	31	23
Sensor PCB Purchasing	21-Apr-25	2-May-25	10	12	24	31	23
Sensor Circuit Test	5-May-25	9-May-25	5	6	12	16	12
Simulation Interfacing	2-Jun-25	7-Jun-25	5	6	12	16	12
Opal Rt Simulation	9-Jun-25	13-Jun-25	5	6	12	16	12
						Total	217

EE 462				Labor Hours			
Project Plan	Start	End	Duration	O	M	P	
Design Revision	22-Sep-25	3-Oct-25	10	12	24	31	23
Software Revision	22-Sep-25	3-Oct-25	10	12	24	31	23
Revision Testing	6-Oct-25	25-Oct-25	15	18	36	47	35
Sensor Integration	27-Oct-25	7-Nov-25	10	12	24	31	23
Control System, Driver Integration	10-Nov-25	14-Nov-25	5	6	12	16	12
Final E2E Testing	17-Nov-25	5-Dec-25	10	12	24	31	23
Final Report	17-Nov-25	12-Dec-25	15	18	36	47	35
						Total	174

Image [36]: Gantt Charts

Appendix D – Functional Decomposition

Decomposition Diagrams and Module Descriptions

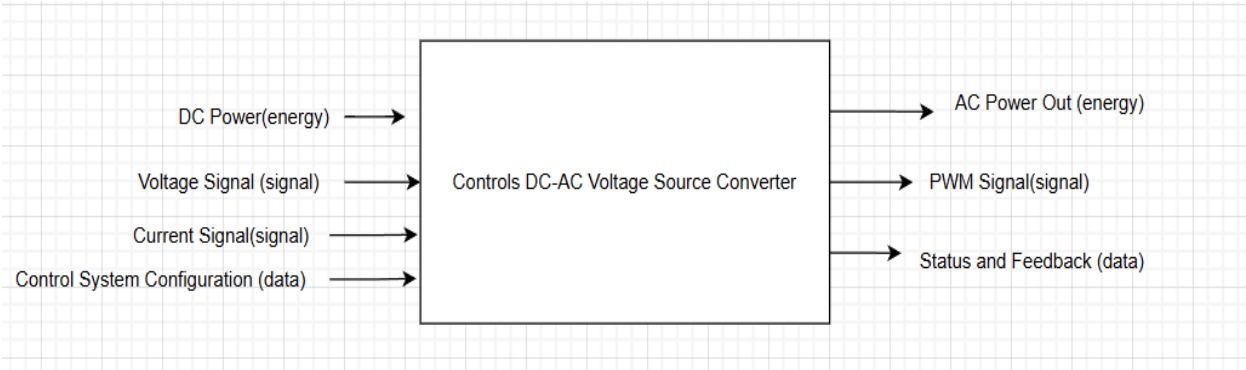


Image [37]: VSC Controller Level-0 Block Diagram

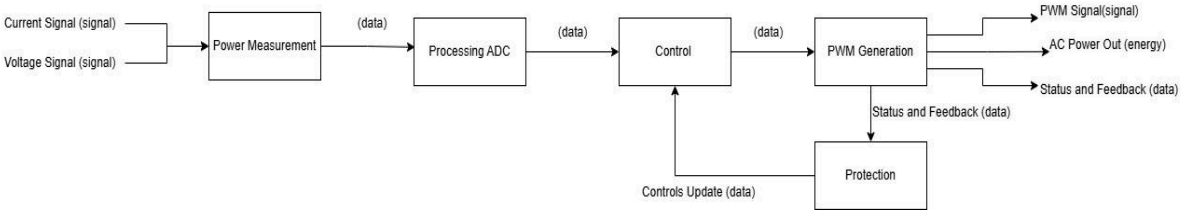


Image [38]: VSC Controller Level-1 Block Diagram

TABLE VII
LEVEL-0 Block Diagram Description

Module	Description	Inputs	Outputs	Target Value and Range	Units
DC-AC Voltage Source Converter Controller	Converts DC to AC power while maintaining efficiency, safety, and grid compliance.	1)DC Input Voltage (60V, ≤10A) 2)Current and Voltage Signals 3) Control System Configuratio	1) PWM Signal 2)AC Power Out 3)Status and Feedback	1)Voltage: 60V (DC), 120V (AC) 2)Current: ≤10A 3)93% efficient power conversion	V, A, %, kW

		n		4) THD≤5%	

TABLE VIII
Description of the Power Measurement Module

Module	Description	Inputs	Outputs	Target Values/Ranges	Units
Power Measurement	Measures the voltage, current, and power on the AC and DC side to monitor system performance.	- Voltage & Current Sensors	-Measured power data	- Voltage: 60V (DC), 120V (AC) - Current: ≤10A - Accuracy: ±TBD%	V, A, kW

TABLE IX
Description of the ADC Module (Second)

Module	Description	Inputs	Outputs	Target Values/Ranges	Units
Processing and ADC	Convert the analog sensor data to digital for digital control use.	-Analog voltage and current data signals.	-Digital data for the control system.	-Resolution 12 bits (TBD). -Sampling Rate: TBD	Bits, Hz

TABLE X
Description of the Control module that has feedback included.

Module	Description	Inputs	Outputs	Target Values/Ranges	Units
Control	Digital control system for power factor manipulation and power regulation.	-Digital Sensor Data -Circuit Feedback	-Control Signals to produce the PWM signals.	-Response time≤1ms. -PI Controlled	ms

TABLE XI

PWM Generation Module which interfaces with the Power Electronics.

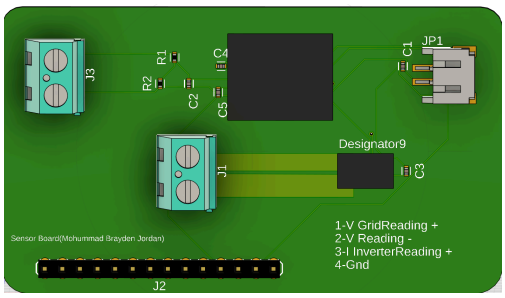
Module	Description	Inputs	Outputs	Target Values/Ranges	Units
PWM Generation	Produces the pulse width modulation signals to switch the power electronics (IGBTs).	-Control Signals	-PWM signals that control IGBT gates.	- Frequency: 500kHz ± 1kHz - Duty Cycle: Variable	Hz, %

TABLE XII

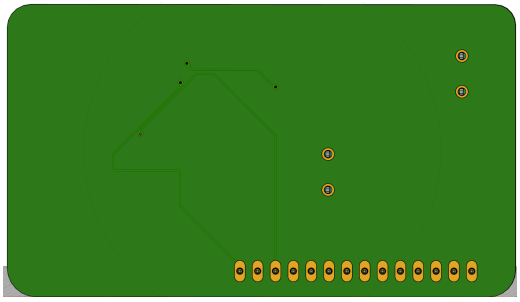
Safety Module to ensure a safe product especially with working with such high power.

Module	Description	Inputs	Outputs	Target Values/Ranges	Units
Protection	Detects overcurrent, voltage faults, and shuts down the system if needed.	Voltage and Current Feedback signals	-Fault indicator -Emergency Stop signal	- Minor Overcurrent: 1ms - Sustained Overcurrent: 500µs - Severe Overcurrent: 50µs	ms, µs

Appendix E – PCB Artwork



Image[39]: PCB top view



Image[40]: PCB bottom view

Appendix F – Matlab Scripts and Generated C Code (Control System)

MatLab Code for Initial Simulink Setup

```
%% Grid and Inverter Parameters for Single-Phase System
% Grid frequency
f = 60;                      % Hz
omega = 2 * pi * f;          % rad/s
% Power and voltage ratings
Pg = 1000;                   % Rated power in watts (1 kW)
Vg_rms = 120;                % Grid RMS voltage
Vg_pk = Vg_rms * sqrt(2);    % Grid peak voltage
% Base impedance
Zb = Vg_rms^2 / Pg;          % Base grid impedance
Lgrid = 0.1 * Zb / omega;    % Grid inductance (10% of Zb)
Rgrid = 0.1 * Zb;            % Grid resistance (10% of Zb)
%% Sampling and Switching
fs = 10e3;                   % Sampling frequency (Hz)
t_s = 1 / fs;                % Sampling period (s)
f_sw = 500;                  % Switching frequency (Hz)
%% Current Control Gains (PI)
tau = 1e-3;                  % Desired current loop time constant
kp_i = Lgrid / tau;          % Proportional gain for current loop
ki_i = Rgrid / tau;          % Integral gain for current loop
%% Voltage Control Gains (PI)
% Use slower response for voltage loop
tau_v = 5e-3;
kp_v = 1.0;                  % need to tune
ki_v = 100.0;                % need to tune
```

MatLab Code for FFT

```
% fft_vout_iout.m
% Computes FFT and THD for out.vout and out.iout
% Calculates and annotates the THD on fft Plots
%% Settings
f0        = 60;      % fundamental frequency (Hz)
maxHarm   = 10;      % number of harmonics to include in THD
maxFreqPlt = 5000;   % x-axis limit for FFT plots (Hz)
%% --- List of signals to process ---
signalNames = {'vout', 'iout'};
for n = 1:numel(signalNames)
   sigName = signalNames{n};

   % 1) Extract structure with time from SimulationOutput
   %    e.g., out.vout or out.iout
   data = out.(sigName);     % this is your 1x1 struct (with .time,
.signals.values)

   t = data.time;            % time vector
   x = data.signals.values;  % signal values
```

```
    % If there are multiple columns, pick one:
    % x = data.signals.values(:,1);

    % 2) Compute sampling frequency
    dt = mean(diff(t));
    Fs = 1/dt;
    L  = length(x);

    % 3) Compute FFT
    X  = fft(x);
    P2 = abs(X / L);                    % two-sided spectrum
    P1 = P2(1:floor(L/2)+1);            % single-sided spectrum
    if numel(P1) > 2
        P1(2:end-1) = 2 * P1(2:end-1);  % double non-DC components
    end

    f = Fs * (0:floor(L/2)) / L;        % frequency axis

    % 4) Manual THD calculation for this signal
    [~, k0] = min(abs(f - f0));   % index of fundamental
    Afund = P1(k0);

    harmIdx = [];
    for h = 2:maxHarm
        k = round(h * k0);
        if k <= length(P1)
            harmIdx(end+1) = k; %#ok<AGROW>
        end
    end

    Aharm      = P1(harmIdx);
    THD_linear = norm(Aharm) / Afund;
    THD_percent = THD_linear * 100;

    fprintf('THD (manual FFT) for %s = %.3f %%\n', sigName, THD_percent);

    % 5) Plot FFT magnitude
    figure;
    plot(f, P1, 'LineWidth', 1.3);
    grid on;
    xlabel('Frequency (Hz)');
    ylabel('Magnitude');
    title(sprintf('FFT of %s', sigName));
    xlim([0, maxFreqPlt]);

    % 6) Print THD on the figure
    text(0.70 * maxFreqPlt, ...
         0.90 * max(P1), ...
         sprintf('THD = %.3f%%', THD_percent), ...
         'FontSize', 12, 'FontWeight', 'bold', 'Color', 'red');
end
```

Auto-Generated Real-Time Main Program

```
/*
 * Academic License - for use in teaching, academic research, and meeting
 * course requirements at degree granting institutions only.  Not for
 * government, commercial, or other organizational use.
 *
 * File: ert_main.c
 *
 * Code generated for Simulink model 'Final_Code'.
 *
 * Model version                  : 1.11
 * Simulink Coder version         : 25.1 (R2025a) 21-Nov-2024
 * C/C++ source code generated on : Thu Dec  4 15:20:24 2025
 *
 * Target selection: ert.tlc
 * Embedded hardware selection: Texas Instruments->C2000
 * Code generation objectives: Unspecified
 * Validation result: Not run
 */

#include "Final_Code.h"
#include "rtwtypes.h"
#include "MW_target_hardware_resources.h"

volatile int IsrOverrun = 0;
static boolean_T OverrunFlag = 0;
void rt_OneStep(void)
{
  /* Check for overrun. Protect OverrunFlag against preemption */
  if (OverrunFlag++) {
    IsrOverrun = 1;
    OverrunFlag--;
    return;
  }

  enableTimer0Interrupt();
  Final_Code_step();

  /* Get model outputs here */
  disableTimer0Interrupt();
  OverrunFlag--;
}
```

```
volatile boolean_T stopRequested;
volatile boolean_T runModel;
int main(void)
{
  float modelBaseRate = 5.0E-5;
  float systemClock = 200;
 /* Initialize variables */
  stopRequested = false;
  runModel = false;
  c2000_flash_init();
  init_board();

#if defined(MW_EXEC_PROFILER_ON) || (defined(MW_EXTMODE_RUNNING) &&
!defined(XCP_TIMESTAMP_BASED_ON_SIMULATION_TIME))

  hardwareTimer1Init();

#endif

  ;
  rtmSetErrorStatus(Final_Code_M, 0);
  Final_Code_initialize();
  globalInterruptDisable();
  configureTimer0(modelBaseRate, systemClock);
  runModel =
    rtmGetErrorStatus(Final_Code_M) == (NULL)&& !rtmGetStopRequested
    (Final_Code_M);
  enableTimer0Interrupt();
  initBoardEnd();
  globalInterruptEnable();
  while (runModel) {
    stopRequested = !(
                      rtmGetErrorStatus(Final_Code_M) == (NULL)&&
                      !rtmGetStopRequested(Final_Code_M));
  }

  /* Terminate model */
  Final_Code_terminate();
  globalInterruptDisable();
  return 0;
}
```

Control Algorithm and PWM Duty Computation

```
/*
 * Academic License - for use in teaching, academic research, and meeting
 * course requirements at degree granting institutions only.  Not for
 * government, commercial, or other organizational use.
 *
 * File: Final_Code.c
 *
 * Code generated for Simulink model 'Final_Code'.
 *
 * Model version                  : 1.11
 * Simulink Coder version         : 25.1 (R2025a) 21-Nov-2024
 * C/C++ source code generated on : Thu Dec  4 15:20:24 2025
 *
 * Target selection: ert.tlc
 * Embedded hardware selection: Texas Instruments->C2000
 * Code generation objectives: Unspecified
 * Validation result: Not run
 */

#include "Final_Code.h"
#include "rtwtypes.h"
#include "Final_Code_private.h"
#include <math.h>
#include "multiword_types.h"
#include <string.h>
#include "rt_nonfinite.h"

/* Block signals (default storage) */
B_Final_Code_T Final_Code_B;

/* Continuous states */
X_Final_Code_T Final_Code_X;

/* Disabled State Vector */
XDis_Final_Code_T Final_Code_XDis;

/* Block states (default storage) */
DW_Final_Code_T Final_Code_DW;
```

```
/* Real-time model */
static RT_MODEL_Final_Code_T Final_Code_M_;
RT_MODEL_Final_Code_T *const Final_Code_M = &Final_Code_M_;

#ifndef __TMS320C28XX_CLA__

uint16_T MW_adcAInitFlag = 0;

#endif

real_T uMultiWord2Double(const uint64_T u1[], int16_T n1, int16_T e1)
{
  real_T y;
  int16_T exp_0;
  int16_T i;
  y = 0.0;
  exp_0 = e1;
  for (i = 0; i < n1; i++) {
    y += ldexp((real_T)u1[i], exp_0);
    exp_0 += 64;
  }

  return y;
}

void uMultiWordMul(const uint64_T u1[], int16_T n1, const uint64_T u2[],
int16_T
                   n2, uint64_T y[], int16_T n)
{
  uint64_T a0;
  uint64_T a1;
  uint64_T b1;
  uint64_T cb;
  uint64_T u1i;
  uint64_T w01;
  uint64_T w10;
  uint64_T yk;
  int16_T i;
  int16_T j;
  int16_T k;
  int16_T ni;
```

```
  /* Initialize output to zero */
  for (k = 0; k < n; k++) {
    y[k] = 0ULL;
  }

  for (i = 0; i < n1; i++) {
    cb = 0ULL;
    u1i = u1[i];
    a1 = u1i >> 32U;
    a0 = u1i & 4294967295ULL;
    ni = n - i;
    ni = n2 <= ni ? n2 : ni;
    k = i;
    for (j = 0; j < ni; j++) {
      u1i = u2[j];
      b1 = u1i >> 32U;
      u1i &= 4294967295ULL;
      w10 = a1 * u1i;
      w01 = a0 * b1;
      yk = y[k] + cb;
      cb = (uint64_T)(yk < cb);
      u1i *= a0;
      yk += u1i;
      cb += (uint64_T)(yk < u1i);
      u1i = w10 << 32U;
      yk += u1i;
      cb += (uint64_T)(yk < u1i);
      u1i = w01 << 32U;
      yk += u1i;
      cb += (uint64_T)(yk < u1i);
      y[k] = yk;
      cb += w10 >> 32U;
      cb += w01 >> 32U;
      cb += a1 * b1;
      k++;
    }

    if (k < n) {
      y[k] = cb;
    }
  }
}
```

```
/*
 * This function updates continuous states using the ODE3 fixed-step
 * solver algorithm
 */
static void rt_ertODEUpdateContinuousStates(RTWSolverInfo *si )
{
  /* Solver Matrices */
  static const real_T rt_ODE3_A[3] = {
    1.0/2.0, 3.0/4.0, 1.0
  };

  static const real_T rt_ODE3_B[3][3] = {
    { 1.0/2.0, 0.0, 0.0 },

    { 0.0, 3.0/4.0, 0.0 },

    { 2.0/9.0, 1.0/3.0, 4.0/9.0 }
  };

  time_T t = rtsiGetT(si);
  time_T tnew = rtsiGetSolverStopTime(si);
  time_T h = rtsiGetStepSize(si);
  real_T *x = rtsiGetContStates(si);
  ODE3_IntgData *id = (ODE3_IntgData *)rtsiGetSolverData(si);
  real_T *y = id->y;
  real_T *f0 = id->f[0];
  real_T *f1 = id->f[1];
  real_T *f2 = id->f[2];
  real_T hB[3];
  int_T i;
  int_T nXc = 8;
  rtsiSetSimTimeStep(si,MINOR_TIME_STEP);

  /* Save the state values at time t in y, we'll use x as ynew. */
  (void) memcpy(y, x,
                (uint_T)nXc*sizeof(real_T));

  /* Assumes that rtsiSetT and ModelOutputs are up-to-date */
  /* f0 = f(t,y) */
  rtsiSetdX(si, f0);
  Final_Code_derivatives();
```

```
  /* f(:,2) = feval(odefile, t + hA(1), y + f*hB(:,1), args(:)(*)); */
  hB[0] = h * rt_ODE3_B[0][0];
  for (i = 0; i < nXc; i++) {
    x[i] = y[i] + (f0[i]*hB[0]);
  }

  rtsiSetT(si, t + h*rt_ODE3_A[0]);
  rtsiSetdX(si, f1);
  Final_Code_step();
  Final_Code_derivatives();

  /* f(:,3) = feval(odefile, t + hA(2), y + f*hB(:,2), args(:)(*)); */
  for (i = 0; i <= 1; i++) {
    hB[i] = h * rt_ODE3_B[1][i];
  }

  for (i = 0; i < nXc; i++) {
    x[i] = y[i] + (f0[i]*hB[0] + f1[i]*hB[1]);
  }

  rtsiSetT(si, t + h*rt_ODE3_A[1]);
  rtsiSetdX(si, f2);
  Final_Code_step();
  Final_Code_derivatives();

  /* tnew = t + hA(3);
     ynew = y + f*hB(:,3); */
  for (i = 0; i <= 2; i++) {
    hB[i] = h * rt_ODE3_B[2][i];
  }

  for (i = 0; i < nXc; i++) {
    x[i] = y[i] + (f0[i]*hB[0] + f1[i]*hB[1] + f2[i]*hB[2]);
  }

  rtsiSetT(si, tnew);
  rtsiSetSimTimeStep(si,MAJOR_TIME_STEP);
}

/*
 * Disable for enable system:
```

```
 *    '<S3>/Subsystem - pi//2 delay'
 *    '<S4>/Subsystem - pi//2 delay'
 */
void Final_Subsystempi2delay_Disable(DW_Subsystempi2delay_Final_Co_T
*localDW)
{
  localDW->Subsystempi2delay_MODE = false;
}

/*
 * Output and update for enable system:
 *    '<S3>/Subsystem - pi//2 delay'
 *    '<S4>/Subsystem - pi//2 delay'
 */
void Final_Code_Subsystempi2delay(RT_MODEL_Final_Code_T * const
Final_Code_M,
  uint16_T rtu_Enable, real_T rtu_alpha_beta, real_T rtu_alpha_beta_d,
real_T
  rtu_wt, real_T *rty_dq, real_T *rty_dq_a, DW_Subsystempi2delay_Final_Co_T
  *localDW)
{
  real_T tmp;
  real_T tmp_0;

  /* Outputs for Enabled SubSystem: '<S3>/Subsystem - pi//2 delay'
incorporates:
   *  EnablePort: '<S16>/Enable'
   */
  if (rtmIsMajorTimeStep(Final_Code_M) && rtsiIsModeUpdateTimeStep
      (&Final_Code_M->solverInfo)) {
    if (rtu_Enable > 0U) {
      localDW->Subsystempi2delay_MODE = true;
    } else if (localDW->Subsystempi2delay_MODE) {
      Final_Subsystempi2delay_Disable(localDW);
    }
  }

  if (localDW->Subsystempi2delay_MODE) {
    /* Fcn: '<S16>/Fcn' incorporates:
     *  Fcn: '<S16>/Fcn1'
     */
    tmp = cos(rtu_wt);
```

```
    tmp_0 = sin(rtu_wt);
    *rty_dq = rtu_alpha_beta * tmp_0 - rtu_alpha_beta_d * tmp;

    /* Fcn: '<S16>/Fcn1' */
    *rty_dq_a = rtu_alpha_beta * tmp + rtu_alpha_beta_d * tmp_0;
  }

  /* End of Outputs for SubSystem: '<S3>/Subsystem - pi//2 delay' */
}

/*
 * Disable for enable system:
 *    '<S3>/Subsystem1'
 *    '<S4>/Subsystem1'
 */
void Final_Code_Subsystem1_Disable(DW_Subsystem1_Final_Code_T *localDW)
{
  localDW->Subsystem1_MODE = false;
}

/*
 * Output and update for enable system:
 *    '<S3>/Subsystem1'
 *    '<S4>/Subsystem1'
 */
void Final_Code_Subsystem1(RT_MODEL_Final_Code_T * const Final_Code_M,
uint16_T
  rtu_Enable, real_T rtu_alpha_beta, real_T rtu_alpha_beta_f, real_T
rtu_wt,
  real_T *rty_dq, real_T *rty_dq_n, DW_Subsystem1_Final_Code_T *localDW)
{
  real_T tmp;
  real_T tmp_0;

  /* Outputs for Enabled SubSystem: '<S3>/Subsystem1' incorporates:
   *  EnablePort: '<S17>/Enable'
   */
  if (rtmIsMajorTimeStep(Final_Code_M) && rtsiIsModeUpdateTimeStep
      (&Final_Code_M->solverInfo)) {
    if (rtu_Enable > 0U) {
      localDW->Subsystem1_MODE = true;
    } else if (localDW->Subsystem1_MODE) {
```

```
      Final_Code_Subsystem1_Disable(localDW);
    }
  }

  if (localDW->Subsystem1_MODE) {
    /* Fcn: '<S17>/Fcn' incorporates:
     *  Fcn: '<S17>/Fcn1'
     */
    tmp = sin(rtu_wt);
    tmp_0 = cos(rtu_wt);
    *rty_dq = rtu_alpha_beta * tmp_0 + rtu_alpha_beta_f * tmp;

    /* Fcn: '<S17>/Fcn1' */
    *rty_dq_n = -rtu_alpha_beta * tmp + rtu_alpha_beta_f * tmp_0;
  }

  /* End of Outputs for SubSystem: '<S3>/Subsystem1' */
}

/* Model step function */
void Final_Code_step(void)
{
  uint128m_T tmp;
  real_T rtb_Gain6;
  real_T rtb_Gain7;
  real_T rtb_Sum1_kr;
  real_T rtb_Switch_i_idx_0;
  real_T rtb_Switch_i_idx_1;
  real_T rtb_Switch_idx_0;
  real_T rtb_VA;
  uint64_T tmp_0;
  uint64_T tmp_1;
  if (rtmIsMajorTimeStep(Final_Code_M)) {
    /* set solver stop time */
    rtsiSetSolverStopTime(&Final_Code_M->solverInfo,
                          ((Final_Code_M->Timing.clockTick0+1)*
      Final_Code_M->Timing.stepSize0));
  }                                    /* end MajorTimeStep */

  /* Update absolute time of base rate at minor time step */
  if (rtmIsMinorTimeStep(Final_Code_M)) {
    Final_Code_M->Timing.t[0] = rtsiGetT(&Final_Code_M->solverInfo);
```

```
  }

  /* S-Function (c2802xadc): '<Root>/ADC1' */
  {
    /*  Internal Reference Voltage : Fixed scale 0 to 3.3 V range.  */
    /*  External Reference Voltage : Allowable ranges of VREFHI(ADCINA0) =
3.3 and VREFLO(tied to ground) = 0  */
    Final_Code_B.ADC1 = (AdcaResultRegs.ADCRESULT1);
  }

  /* Gain: '<Root>/V//A' incorporates:
   *  Constant: '<Root>/Ioffset_v'
   *  Gain: '<Root>/Gain1'
   *  Sum: '<Root>/Sum'
   */
  rtb_VA = (54080.0 * (real_T)Final_Code_B.ADC1 * 1.4901161193847656E-8 -
0.05) *
    50.0;

  /* Gain: '<S1>/Gain6' incorporates:
   *  Gain: '<S23>/D'
   *  Integrator: '<S22>/Integrator'
   *  Integrator: '<S23>/Integrator'
   *  Sum: '<S23>/Sum1'
   */
  rtb_Gain6 = (0.0 * Final_Code_X.Integrator_CSTATE_n +
               Final_Code_X.Integrator_CSTATE) * 2.0;

  /* Outputs for Enabled SubSystem: '<S4>/Subsystem1' */
  /* Switch: '<S2>/Sw_Igrid' incorporates:
   *  Integrator: '<S1>/Integrator1'
   */
  Final_Code_Subsystem1(Final_Code_M, Final_Code_ConstB.Compare, rtb_Gain6,
                        rtb_VA, Final_Code_X.Integrator1_CSTATE,
                        &Final_Code_B.Fcn_a, &Final_Code_B.Fcn1_f,
                        &Final_Code_DW.Subsystem1_e);

  /* End of Outputs for SubSystem: '<S4>/Subsystem1' */

  /* Outputs for Enabled SubSystem: '<S4>/Subsystem - pi//2 delay' */
  Final_Code_Subsystempi2delay(Final_Code_M, Final_Code_ConstB.Compare_n,
    rtb_Gain6, rtb_VA, Final_Code_X.Integrator1_CSTATE,
```

```
&Final_Code_B.Fcn_d,
    &Final_Code_B.Fcn1_a, &Final_Code_DW.Subsystempi2delay_c);

  /* End of Outputs for SubSystem: '<S4>/Subsystem - pi//2 delay' */

  /* Switch: '<S4>/Switch' incorporates:
   *  RelationalOperator: '<S18>/Compare'
   */
  if (Final_Code_ConstB.Compare != 0U) {
    rtb_Switch_i_idx_0 = Final_Code_B.Fcn_a;
    rtb_Switch_i_idx_1 = Final_Code_B.Fcn1_f;
  } else {
    rtb_Switch_i_idx_0 = Final_Code_B.Fcn_d;
    rtb_Switch_i_idx_1 = Final_Code_B.Fcn1_a;
  }

  /* End of Switch: '<S4>/Switch' */

  /* S-Function (c2802xadc): '<Root>/ADC' */
  {
    /*  Internal Reference Voltage : Fixed scale 0 to 3.3 V range.  */
    /*  External Reference Voltage : Allowable ranges of VREFHI(ADCINA0) =
3.3 and VREFLO(tied to ground) = 0  */
    Final_Code_B.ADC = (AdcaResultRegs.ADCRESULT0);
  }

  /* Gain: '<Root>/Sensor Gain' incorporates:
   *  Gain: '<Root>/Gain'
   *  Gain: '<Root>/Voltage Div'
   *  Switch: '<S2>/Sw_Vgrid'
   */
  tmp_0 = 9607679205057058816ULL;
  tmp_1 = 54080UL * Final_Code_B.ADC * 3851297791ULL;
  uMultiWordMul(&tmp_0, 1, &tmp_1, 1, &tmp.chunks[0U], 2);

  /* Switch: '<S2>/Sw_Vgrid' */
  rtb_Gain6 = uMultiWord2Double(&tmp.chunks[0U], 2, 0) *
1.92592994438723592E-34;

  /* Sum: '<S24>/Sum1' incorporates:
   *  Gain: '<S24>/D'
   *  Integrator: '<S24>/Integrator'
```

```
   */
  rtb_Sum1_kr = 0.0 * rtb_Gain6 + Final_Code_X.Integrator_CSTATE_g;

  /* Gain: '<S1>/Gain7' incorporates:
   *  Gain: '<S25>/D'
   *  Integrator: '<S25>/Integrator'
   *  Sum: '<S25>/Sum1'
   */
  rtb_Gain7 = (0.0 * rtb_Sum1_kr + Final_Code_X.Integrator_CSTATE_l) * 2.0;

  /* Outputs for Enabled SubSystem: '<S3>/Subsystem1' */
  /* Integrator: '<S1>/Integrator1' */
  Final_Code_Subsystem1(Final_Code_M, Final_Code_ConstB.Compare_m,
rtb_Gain7,
                        rtb_Gain6, Final_Code_X.Integrator1_CSTATE,
                        &Final_Code_B.Fcn_j, &Final_Code_B.Fcn1_fx,
                        &Final_Code_DW.Subsystem1);

  /* End of Outputs for SubSystem: '<S3>/Subsystem1' */

  /* Outputs for Enabled SubSystem: '<S3>/Subsystem - pi//2 delay' */
  Final_Code_Subsystempi2delay(Final_Code_M, Final_Code_ConstB.Compare_l,
    rtb_Gain7, rtb_Gain6, Final_Code_X.Integrator1_CSTATE,
&Final_Code_B.Fcn_e,
    &Final_Code_B.Fcn1_j, &Final_Code_DW.Subsystempi2delay);

  /* End of Outputs for SubSystem: '<S3>/Subsystem - pi//2 delay' */

  /* Switch: '<S3>/Switch' incorporates:
   *  RelationalOperator: '<S14>/Compare'
   */
  if (Final_Code_ConstB.Compare_m != 0U) {
    rtb_Switch_idx_0 = Final_Code_B.Fcn_j;
    rtb_Gain7 = Final_Code_B.Fcn1_fx;
  } else {
    rtb_Switch_idx_0 = Final_Code_B.Fcn_e;
    rtb_Gain7 = Final_Code_B.Fcn1_j;
  }

  /* End of Switch: '<S3>/Switch' */

  /* Gain: '<S1>/Gain8' incorporates:
```

```
   *  Gain: '<S1>/Gain9'
   */
  rtb_Switch_i_idx_1 *= 2.6420794216690164;

  /* Sum: '<S1>/Sum10' incorporates:
   *  Constant: '<S1>/Constant9'
   *  Gain: '<S117>/Proportional Gain'
   *  Gain: '<S1>/Gain8'
   *  Integrator: '<S112>/Integrator'
   *  Sum: '<S121>/Sum'
   *  Sum: '<S1>/Sum8'
   *  Sum: '<S1>/Sum9'
   */
  rtb_Switch_idx_0 += ((-10.0 - rtb_Switch_i_idx_0) * 40.0 +
                       Final_Code_X.Integrator_CSTATE_ly) -
rtb_Switch_i_idx_1;

  /* Sum: '<S1>/Sum13' incorporates:
   *  Constant: '<S1>/Constant10'
   *  Gain: '<S169>/Proportional Gain'
   *  Integrator: '<S164>/Integrator'
   *  Sum: '<S173>/Sum'
   *  Sum: '<S1>/Sum11'
   *  Sum: '<S1>/Sum12'
   */
  rtb_Switch_i_idx_1 = (((0.0 - rtb_Switch_i_idx_0) * 40.0 +
    Final_Code_X.Integrator_CSTATE_j) + rtb_Switch_i_idx_1) + rtb_Gain7;

  /* Outputs for Enabled SubSystem: '<S12>/Subsystem - pi//2 delay'
incorporates:
   *  EnablePort: '<S184>/Enable'
   */
  /* Outputs for Enabled SubSystem: '<S12>/Subsystem1' incorporates:
   *  EnablePort: '<S185>/Enable'
   */
  if (rtmIsMajorTimeStep(Final_Code_M) && rtsiIsModeUpdateTimeStep
      (&Final_Code_M->solverInfo)) {
    /* RelationalOperator: '<S182>/Compare' */
    Final_Code_DW.Subsystem1_MODE = (Final_Code_ConstB.Compare_k > 0U);

    /* RelationalOperator: '<S183>/Compare' */
    Final_Code_DW.Subsystempi2delay_MODE = (Final_Code_ConstB.Compare_lw >
```

```
0U);
  }

  /* End of Outputs for SubSystem: '<S12>/Subsystem - pi//2 delay' */
  if (Final_Code_DW.Subsystem1_MODE) {
    /* Fcn: '<S185>/Fcn1' incorporates:
     *  Integrator: '<S1>/Integrator1'
     */
    Final_Code_B.Fcn1 = rtb_Switch_idx_0 *
sin(Final_Code_X.Integrator1_CSTATE)
      + rtb_Switch_i_idx_1 * cos(Final_Code_X.Integrator1_CSTATE);
  }

  /* End of Outputs for SubSystem: '<S12>/Subsystem1' */

  /* Outputs for Enabled SubSystem: '<S12>/Subsystem - pi//2 delay'
incorporates:
   *  EnablePort: '<S184>/Enable'
   */
  if (Final_Code_DW.Subsystempi2delay_MODE) {
    /* Fcn: '<S184>/Fcn1' incorporates:
     *  Integrator: '<S1>/Integrator1'
     */
    Final_Code_B.Fcn1_p = -rtb_Switch_idx_0 * cos
      (Final_Code_X.Integrator1_CSTATE) + rtb_Switch_i_idx_1 * sin
      (Final_Code_X.Integrator1_CSTATE);
  }

  /* End of Outputs for SubSystem: '<S12>/Subsystem - pi//2 delay' */

  /* Switch: '<S12>/Switch' incorporates:
   *  RelationalOperator: '<S182>/Compare'
   */
  if (Final_Code_ConstB.Compare_k != 0U) {
    rtb_Switch_idx_0 = Final_Code_B.Fcn1;
  } else {
    rtb_Switch_idx_0 = Final_Code_B.Fcn1_p;
  }

  /* Gain: '<S1>/Gain' incorporates:
   *  Gain: '<S1>/Gain10'
   *  Switch: '<S12>/Switch'
```

```
 */
rtb_Switch_i_idx_1 = 0.0025 * rtb_Switch_idx_0 * 0.005;

/* Saturate: '<S1>/Saturation' */
if (rtb_Switch_i_idx_1 > 1.0) {
  rtb_Switch_i_idx_1 = 1.0;
} else if (rtb_Switch_i_idx_1 < -1.0) {
  rtb_Switch_i_idx_1 = -1.0;
}

/* End of Saturate: '<S1>/Saturation' */

/* Saturate: '<S1>/Saturation2' incorporates:
 *  Gain: '<S1>/Gain2'
 *  Sum: '<S1>/Sum1'
 */
if ((rtb_Switch_i_idx_1 - 1.0) * 0.5 < 0.2) {
  rtb_Switch_idx_0 = 0.2;
} else {
  rtb_Switch_idx_0 = (rtNaN);
}

/* Gain: '<S1>/Gain4' incorporates:
 *  Saturate: '<S1>/Saturation2'
 */
rtb_Switch_idx_0 *= 2500.0;

/* S-Function (c2802xpwm): '<S1>/ePWM' */
uint16_T tbprdValue2Outputs = EPwm2Regs.TBPRD;

/*-- Update CMPA value for ePWM2 --*/
{
  EPwm2Regs.CMPA.bit.CMPA = (uint16_T)(rtb_Switch_idx_0);
}

/* Gain: '<S1>/Gain1' incorporates:
 *  Constant: '<S1>/Constant'
 *  Sum: '<S1>/Sum'
 */
rtb_Switch_i_idx_1 = (rtb_Switch_i_idx_1 + 1.0) * 0.5;

/* Saturate: '<S1>/Saturation1' */
```

```
if (rtb_Switch_i_idx_1 > 0.98) {
  rtb_Switch_i_idx_1 = 0.98;
} else if (rtb_Switch_i_idx_1 < 0.2) {
  rtb_Switch_i_idx_1 = 0.2;
}

/* Gain: '<S1>/Gain3' incorporates:
 *  Saturate: '<S1>/Saturation1'
 */
rtb_Switch_idx_0 = 2500.0 * rtb_Switch_i_idx_1;

/* S-Function (c2802xpwm): '<S1>/ePWM1' */
uint16_T tbprdValue1Outputs = EPwm1Regs.TBPRD;

/*-- Update CMPA value for ePWM1 --*/
{
  EPwm1Regs.CMPA.bit.CMPA = (uint16_T)(rtb_Switch_idx_0);
}

/* Gain: '<S161>/Integral Gain' incorporates:
 *  Constant: '<S1>/Constant10'
 *  Sum: '<S1>/Sum11'
 */
Final_Code_B.IntegralGain = (0.0 - rtb_Switch_i_idx_0) * 6.7;

/* Gain: '<S109>/Integral Gain' incorporates:
 *  Constant: '<S1>/Constant9'
 *  Sum: '<S1>/Sum8'
 */
Final_Code_B.IntegralGain_f = (-10.0 - rtb_Switch_i_idx_0) * 6.7;

/* Gain: '<S57>/Integral Gain' incorporates:
 *  Constant: '<S1>/Constant7'
 *  Sum: '<S1>/Sum7'
 */
Final_Code_B.IntegralGain_h = (0.0 - rtb_Gain7) * 50000.0;

/* Sum: '<S69>/Sum' incorporates:
 *  Constant: '<S1>/Constant7'
 *  Gain: '<S65>/Proportional Gain'
 *  Integrator: '<S60>/Integrator'
 *  Sum: '<S1>/Sum7'
```

```
   */
  Final_Code_B.Sum = (0.0 - rtb_Gain7) * 10.0 +
Final_Code_X.Integrator_CSTATE_c;

  /* Sum: '<S25>/Sum' incorporates:
   *  Gain: '<S25>/A'
   *  Gain: '<S25>/B'
   *  Integrator: '<S25>/Integrator'
   */
  Final_Code_B.x = 314.15926535897933 * rtb_Sum1_kr + -314.15926535897933 *
    Final_Code_X.Integrator_CSTATE_l;

  /* Sum: '<S24>/Sum' incorporates:
   *  Gain: '<S24>/A'
   *  Gain: '<S24>/B'
   *  Integrator: '<S24>/Integrator'
   */
  Final_Code_B.x_p = 314.15926535897933 * rtb_Gain6 + -314.15926535897933 *
    Final_Code_X.Integrator_CSTATE_g;

  /* Sum: '<S23>/Sum' incorporates:
   *  Gain: '<S23>/A'
   *  Gain: '<S23>/B'
   *  Integrator: '<S22>/Integrator'
   *  Integrator: '<S23>/Integrator'
   */
  Final_Code_B.x_g = 314.15926535897933 * Final_Code_X.Integrator_CSTATE_n
+
    -314.15926535897933 * Final_Code_X.Integrator_CSTATE;

  /* Sum: '<S22>/Sum' incorporates:
   *  Gain: '<S22>/A'
   *  Gain: '<S22>/B'
   *  Integrator: '<S22>/Integrator'
   *  Switch: '<S2>/Sw_Igrid'
   */
  Final_Code_B.x_pd = 314.15926535897933 * rtb_VA + -314.15926535897933 *
    Final_Code_X.Integrator_CSTATE_n;
  if (rtmIsMajorTimeStep(Final_Code_M)) {
    rt_ertODEUpdateContinuousStates(&Final_Code_M->solverInfo);

    /* Update absolute time for base rate */
```

```
    /* The "clockTick0" counts the number of times the code of this task
has
     * been executed. The absolute time is the multiplication of
"clockTick0"
     * and "Timing.stepSize0". Size of "clockTick0" ensures timer will not
     * overflow during the application lifespan selected.
     */
    ++Final_Code_M->Timing.clockTick0;
    Final_Code_M->Timing.t[0] =
rtsiGetSolverStopTime(&Final_Code_M->solverInfo);

    {
      /* Update absolute timer for sample time: [5.0E-5s, 0.0s] */
      /* The "clockTick1" counts the number of times the code of this task
has
       * been executed. The resolution of this integer timer is 5.0E-5,
which is the step size
       * of the task. Size of "clockTick1" ensures timer will not overflow
during the
       * application lifespan selected.
       */
      Final_Code_M->Timing.clockTick1++;
    }
  }                                    /* end MajorTimeStep */
}

/* Derivatives for root system: '<Root>' */
void Final_Code_derivatives(void)
{
  XDot_Final_Code_T *_rtXdot;
  _rtXdot = ((XDot_Final_Code_T *) Final_Code_M->derivs);

  /* Derivatives for Integrator: '<S23>/Integrator' */
  _rtXdot->Integrator_CSTATE = Final_Code_B.x_g;

  /* Derivatives for Integrator: '<S22>/Integrator' */
  _rtXdot->Integrator_CSTATE_n = Final_Code_B.x_pd;

  /* Derivatives for Integrator: '<S1>/Integrator1' */
  _rtXdot->Integrator1_CSTATE = Final_Code_B.Sum;

  /* Derivatives for Integrator: '<S25>/Integrator' */
```

```
  _rtXdot->Integrator_CSTATE_l = Final_Code_B.x;

  /* Derivatives for Integrator: '<S24>/Integrator' */
  _rtXdot->Integrator_CSTATE_g = Final_Code_B.x_p;

  /* Derivatives for Integrator: '<S112>/Integrator' */
  _rtXdot->Integrator_CSTATE_ly = Final_Code_B.IntegralGain_f;

  /* Derivatives for Integrator: '<S164>/Integrator' */
  _rtXdot->Integrator_CSTATE_j = Final_Code_B.IntegralGain;

  /* Derivatives for Integrator: '<S60>/Integrator' */
  _rtXdot->Integrator_CSTATE_c = Final_Code_B.IntegralGain_h;
}

/* Model initialize function */
void Final_Code_initialize(void)
{
  /* Registration code */

  /* initialize non-finites */
  rt_InitInfAndNaN(sizeof(real_T));

  /* initialize real-time model */
  (void) memset((void *)Final_Code_M, 0,
                sizeof(RT_MODEL_Final_Code_T));

  {
    /* Setup solver object */
    rtsiSetSimTimeStepPtr(&Final_Code_M->solverInfo,
                          &Final_Code_M->Timing.simTimeStep);
    rtsiSetTPtr(&Final_Code_M->solverInfo, &rtmGetTPtr(Final_Code_M));
    rtsiSetStepSizePtr(&Final_Code_M->solverInfo,
                       &Final_Code_M->Timing.stepSize0);
    rtsiSetdXPtr(&Final_Code_M->solverInfo, &Final_Code_M->derivs);
    rtsiSetContStatesPtr(&Final_Code_M->solverInfo, (real_T **)
                         &Final_Code_M->contStates);
    rtsiSetNumContStatesPtr(&Final_Code_M->solverInfo,
      &Final_Code_M->Sizes.numContStates);
    rtsiSetNumPeriodicContStatesPtr(&Final_Code_M->solverInfo,
      &Final_Code_M->Sizes.numPeriodicContStates);
    rtsiSetPeriodicContStateIndicesPtr(&Final_Code_M->solverInfo,
```

```
      &Final_Code_M->periodicContStateIndices);
    rtsiSetPeriodicContStateRangesPtr(&Final_Code_M->solverInfo,
      &Final_Code_M->periodicContStateRanges);
    rtsiSetContStateDisabledPtr(&Final_Code_M->solverInfo, (boolean_T**)
      &Final_Code_M->contStateDisabled);
    rtsiSetErrorStatusPtr(&Final_Code_M->solverInfo, (&rtmGetErrorStatus
      (Final_Code_M)));
    rtsiSetRTModelPtr(&Final_Code_M->solverInfo, Final_Code_M);
  }

  rtsiSetSimTimeStep(&Final_Code_M->solverInfo, MAJOR_TIME_STEP);
  rtsiSetIsMinorTimeStepWithModeChange(&Final_Code_M->solverInfo, false);
  rtsiSetIsContModeFrozen(&Final_Code_M->solverInfo, false);
  Final_Code_M->intgData.y = Final_Code_M->odeY;
  Final_Code_M->intgData.f[0] = Final_Code_M->odeF[0];
  Final_Code_M->intgData.f[1] = Final_Code_M->odeF[1];
  Final_Code_M->intgData.f[2] = Final_Code_M->odeF[2];
  Final_Code_M->contStates = ((X_Final_Code_T *) &Final_Code_X);
  Final_Code_M->contStateDisabled = ((XDis_Final_Code_T *)
&Final_Code_XDis);
  Final_Code_M->Timing.tStart = (0.0);
  rtsiSetSolverData(&Final_Code_M->solverInfo, (void
*)&Final_Code_M->intgData);
  rtsiSetSolverName(&Final_Code_M->solverInfo,"ode3");
  rtmSetTPtr(Final_Code_M, &Final_Code_M->Timing.tArray[0]);
  Final_Code_M->Timing.stepSize0 = 5.0E-5;

  /* block I/O */
  (void) memset(((void *) &Final_Code_B), 0,
                sizeof(B_Final_Code_T));

  /* states (continuous) */
  {
    (void) memset((void *)&Final_Code_X, 0,
                  sizeof(X_Final_Code_T));
  }

  /* disabled states */
  {
    (void) memset((void *)&Final_Code_XDis, 0,
                  sizeof(XDis_Final_Code_T));
  }
```

```
  /* states (dwork) */
  (void) memset((void *)&Final_Code_DW, 0,
                sizeof(DW_Final_Code_T));

  /* Start for S-Function (c2802xpwm): '<S1>/ePWM' */
  real32_T tbprdValue2 = (real32_T)EPwm2Regs.TBPRD;

  /*** Initialize ePWM2 modules ***/
  {
    /*  -- Time Base Control Register
       EPwm2Regs.TBCTL.bit.CTRMODE              = 2U;          -- Counter
Mode
       EPwm2Regs.TBCTL.bit.SYNCOSEL             = 3U;          -- Sync
Output Select
       EPwm2Regs.TBCTL2.bit.SYNCOSELX           = 0U;          -- Sync
Output Select - additional options

       EPwm2Regs.TBCTL.bit.PRDLD                = 0U;          -- Shadow
select

       EPwm2Regs.TBCTL2.bit.PRDLDSYNC           = 0U;          -- Shadow
select

       EPwm2Regs.TBCTL.bit.PHSEN                = 1U;          -- Phase
Load Enable
       EPwm2Regs.TBCTL.bit.PHSDIR               = 0U;          -- Phase
Direction Bit
       EPwm2Regs.TBCTL.bit.HSPCLKDIV            = 1U;          -- High
Speed TBCLK Pre-scaler
       EPwm2Regs.TBCTL.bit.CLKDIV               = 0U;          -- Time Base
Clock Pre-scaler
     */
    EPwm2Regs.TBCTL.all = (EPwm2Regs.TBCTL.all & ~0x3FFFU) | 0xB6U;
    EPwm2Regs.TBCTL2.all = (EPwm2Regs.TBCTL2.all & ~0xF000U) | 0x0U;

    /*-- Setup Time-Base (TB) Submodule --*/
    EPwm2Regs.TBPRD = 2500U;           // Time Base Period Register

    /* -- Time-Base Phase Register
       EPwm2Regs.TBPHS.bit.TBPHS               = 0U;          -- Phase
offset register
```

```
     */
    EPwm2Regs.TBPHS.all = (EPwm2Regs.TBPHS.all & ~0xFFFF0000U) | 0x0U;

    // Time Base Counter Register
    EPwm2Regs.TBCTR = 0x0000U;         /* Clear counter*/

    /*-- Setup Counter_Compare (CC) Submodule --*/
    /*      -- Counter Compare Control Register

       EPwm2Regs.CMPCTL.bit.LOADASYNC           = 0U;          -- Active
Compare A Load SYNC Option
       EPwm2Regs.CMPCTL.bit.LOADBSYNC           = 0U;          -- Active
Compare B Load SYNC Option
       EPwm2Regs.CMPCTL.bit.LOADAMODE           = 0U;          -- Active
Compare A Load
       EPwm2Regs.CMPCTL.bit.LOADBMODE           = 0U;          -- Active
Compare B Load
       EPwm2Regs.CMPCTL.bit.SHDWAMODE           = 0U;          -- Compare A
Register Block Operating Mode
       EPwm2Regs.CMPCTL.bit.SHDWBMODE           = 0U;          -- Compare B
Register Block Operating Mode
     */
    EPwm2Regs.CMPCTL.all = (EPwm2Regs.CMPCTL.all & ~0x3C5FU) | 0x0U;

    /* EPwm2Regs.CMPCTL2.bit.SHDWCMODE           = 0U;          -- Compare
C Register Block Operating Mode
       EPwm2Regs.CMPCTL2.bit.SHDWDMODE           = 0U;          -- Compare
D Register Block Operating Mode
       EPwm2Regs.CMPCTL2.bit.LOADCSYNC           = 0U;          -- Active
Compare C Load SYNC Option
       EPwm2Regs.CMPCTL2.bit.LOADDSYNC           = 0U;          -- Active
Compare D Load SYNC Option
       EPwm2Regs.CMPCTL2.bit.LOADCMODE           = 0U;          -- Active
Compare C Load
       EPwm2Regs.CMPCTL2.bit.LOADDMODE           = 0U;          -- Active
Compare D Load
     */
    EPwm2Regs.CMPCTL2.all = (EPwm2Regs.CMPCTL2.all & ~0x3C5FU) | 0x0U;
    EPwm2Regs.CMPA.bit.CMPA = 1250U;   // Counter Compare A Register
    EPwm2Regs.CMPB.bit.CMPB = 32000U;  // Counter Compare B Register
    EPwm2Regs.CMPC = 32000U;           // Counter Compare C Register
    EPwm2Regs.CMPD = 32000U;           // Counter Compare D Register
```

```
    /*-- Setup Action-Qualifier (AQ) Submodule --*/
    EPwm2Regs.AQCTLA.all = 150U;
                                // Action Qualifier Control Register For
Output A
    EPwm2Regs.AQCTLB.all = 2310U;
                                // Action Qualifier Control Register For
Output B

    /*      -- Action Qualifier Software Force Register
       EPwm2Regs.AQSFRC.bit.RLDCSF              = 0U;          -- Reload
from Shadow Options
     */
    EPwm2Regs.AQSFRC.all = (EPwm2Regs.AQSFRC.all & ~0xC0U) | 0x0U;

    /*      -- Action Qualifier Continuous S/W Force Register
       EPwm2Regs.AQCSFRC.bit.CSFA               = 0U;          --
Continuous Software Force on output A
       EPwm2Regs.AQCSFRC.bit.CSFB               = 0U;          --
Continuous Software Force on output B
     */
    EPwm2Regs.AQCSFRC.all = (EPwm2Regs.AQCSFRC.all & ~0xFU) | 0x0U;

    /*-- Setup Dead-Band Generator (DB) Submodule --*/
    /*      -- Dead-Band Generator Control Register
       EPwm2Regs.DBCTL.bit.OUT_MODE             = 3U;          -- Dead Band
Output Mode Control
       EPwm2Regs.DBCTL.bit.IN_MODE              = 0U;          -- Dead Band
Input Select Mode Control
       EPwm2Regs.DBCTL.bit.POLSEL               = 2U;          -- Polarity
Select Control
       EPwm2Regs.DBCTL.bit.HALFCYCLE            = 0U;          -- Half
Cycle Clocking Enable
       EPwm2Regs.DBCTL.bit.SHDWDBREDMODE        = 0U;          -- DBRED
shadow mode
       EPwm2Regs.DBCTL.bit.SHDWDBFEDMODE        = 0U;          -- DBFED
shadow mode
       EPwm2Regs.DBCTL.bit.LOADREDMODE          = 4U;        -- DBRED load
       EPwm2Regs.DBCTL.bit.LOADFEDMODE          = 4U;        -- DBFED load
     */
    EPwm2Regs.DBCTL.all = (EPwm2Regs.DBCTL.all & ~0x8FFFU) | 0xBU;
    EPwm2Regs.DBRED.bit.DBRED = (uint16_T)(50.0);
```

```
                        // Dead-Band Generator Rising Edge Delay Count
Register
    EPwm2Regs.DBFED.bit.DBFED = (uint16_T)(50.0);
                       // Dead-Band Generator Falling Edge Delay Count
Register

    /*-- Setup Event-Trigger (ET) Submodule --*/
    /*      -- Event Trigger Selection and Pre-Scale Register
       EPwm2Regs.ETSEL.bit.SOCAEN               = 0U;          -- Start of
Conversion A Enable
       EPwm2Regs.ETSEL.bit.SOCASELCMP           = 0U;
       EPwm2Regs.ETSEL.bit.SOCASEL              = 1U;          -- Start of
Conversion A Select
       EPwm2Regs.ETPS.bit.SOCPSSEL              = 1U;          -- EPWM2SOC
Period Select
       EPwm2Regs.ETSOCPS.bit.SOCAPRD2           = 1U;
       EPwm2Regs.ETSEL.bit.SOCBEN               = 0U;          -- Start of
Conversion B Enable
       EPwm2Regs.ETSEL.bit.SOCBSELCMP           = 0U;
       EPwm2Regs.ETSEL.bit.SOCBSEL              = 1U;          -- Start of
Conversion A Select
       EPwm2Regs.ETPS.bit.SOCPSSEL              = 1;          -- EPWM2SOCB
Period Select
       EPwm2Regs.ETSOCPS.bit.SOCBPRD2           = 1U;
       EPwm2Regs.ETSEL.bit.INTEN                = 0U;          -- EPWM2INTn
Enable
       EPwm2Regs.ETSEL.bit.INTSELCMP            = 0U;
       EPwm2Regs.ETSEL.bit.INTSEL               = 1U;          -- Start of
Conversion A Select
       EPwm2Regs.ETPS.bit.INTPSSEL              = 1U;          // EPWM2INTn
Period Select
       EPwm2Regs.ETINTPS.bit.INTPRD2            = 1U;
     */
    EPwm2Regs.ETSEL.all = (EPwm2Regs.ETSEL.all & ~0xFF7FU) | 0x1101U;
    EPwm2Regs.ETPS.all = (EPwm2Regs.ETPS.all & ~0x30U) | 0x30U;
    EPwm2Regs.ETSOCPS.all = (EPwm2Regs.ETSOCPS.all & ~0xF0FU) | 0x101U;
    EPwm2Regs.ETINTPS.all = (EPwm2Regs.ETINTPS.all & ~0xFU) | 0x1U;

    /*-- Setup PWM-Chopper (PC) Submodule --*/
    /*      -- PWM Chopper Control Register
       EPwm2Regs.PCCTL.bit.CHPEN                = 0U;          -- PWM
chopping enable
```

```
       EPwm2Regs.PCCTL.bit.CHPFREQ              = 0U;          -- Chopping
clock frequency
       EPwm2Regs.PCCTL.bit.OSHTWTH              = 0U;          -- One-shot
pulse width
       EPwm2Regs.PCCTL.bit.CHPDUTY              = 0U;          -- Chopping
clock Duty cycle
     */
    EPwm2Regs.PCCTL.all = (EPwm2Regs.PCCTL.all & ~0x7FFU) | 0x0U;

    /*-- Set up Trip-Zone (TZ) Submodule --*/
    EALLOW;
    EPwm2Regs.TZSEL.all = 0U;          // Trip Zone Select Register

    /*      -- Trip Zone Control Register
       EPwm2Regs.TZCTL.bit.TZA                  = 3U;          -- TZ1 to
TZ6 Trip Action On EPWM2A
       EPwm2Regs.TZCTL.bit.TZB                  = 3U;          -- TZ1 to
TZ6 Trip Action On EPWM2B
       EPwm2Regs.TZCTL.bit.DCAEVT1              = 3U;          -- EPWM2A
action on DCAEVT1
       EPwm2Regs.TZCTL.bit.DCAEVT2              = 3U;          -- EPWM2A
action on DCAEVT2
       EPwm2Regs.TZCTL.bit.DCBEVT1              = 3U;          -- EPWM2B
action on DCBEVT1
       EPwm2Regs.TZCTL.bit.DCBEVT2              = 3U;          -- EPWM2B
action on DCBEVT2
     */
    EPwm2Regs.TZCTL.all = (EPwm2Regs.TZCTL.all & ~0xFFFU) | 0xFFFU;

    /*      -- Trip Zone Enable Interrupt Register
       EPwm2Regs.TZEINT.bit.OST                 = 0U;          -- Trip
Zones One Shot Int Enable
       EPwm2Regs.TZEINT.bit.CBC                 = 0U;          -- Trip
Zones Cycle By Cycle Int Enable
       EPwm2Regs.TZEINT.bit.DCAEVT1             = 0U;          -- Digital
Compare A Event 1 Int Enable
       EPwm2Regs.TZEINT.bit.DCAEVT2             = 0U;          -- Digital
Compare A Event 2 Int Enable
       EPwm2Regs.TZEINT.bit.DCBEVT1             = 0U;          -- Digital
Compare B Event 1 Int Enable
       EPwm2Regs.TZEINT.bit.DCBEVT2             = 0U;          -- Digital
Compare B Event 2 Int Enable
```

```
     */
    EPwm2Regs.TZEINT.all = (EPwm2Regs.TZEINT.all & ~0x7EU) | 0x0U;

    /*      -- Digital Compare A Control Register
       EPwm2Regs.DCACTL.bit.EVT1SYNCE           = 0U;          -- DCAEVT1
SYNC Enable
       EPwm2Regs.DCACTL.bit.EVT1SOCE            = 1U;          -- DCAEVT1
SOC Enable
       EPwm2Regs.DCACTL.bit.EVT1FRCSYNCSEL      = 0U;          -- DCAEVT1
Force Sync Signal
       EPwm2Regs.DCACTL.bit.EVT1SRCSEL          = 0U;          -- DCAEVT1
Source Signal
       EPwm2Regs.DCACTL.bit.EVT2FRCSYNCSEL      = 0U;          -- DCAEVT2
Force Sync Signal
       EPwm2Regs.DCACTL.bit.EVT2SRCSEL          = 0U;          -- DCAEVT2
Source Signal
     */
    EPwm2Regs.DCACTL.all = (EPwm2Regs.DCACTL.all & ~0x30FU) | 0x4U;

    /*      -- Digital Compare B Control Register
       EPwm2Regs.DCBCTL.bit.EVT1SYNCE           = 0U;          -- DCBEVT1
SYNC Enable
       EPwm2Regs.DCBCTL.bit.EVT1SOCE            = 0U;          -- DCBEVT1
SOC Enable
       EPwm2Regs.DCBCTL.bit.EVT1FRCSYNCSEL      = 0U;          -- DCBEVT1
Force Sync Signal
       EPwm2Regs.DCBCTL.bit.EVT1SRCSEL          = 0U;          -- DCBEVT1
Source Signal
       EPwm2Regs.DCBCTL.bit.EVT2FRCSYNCSEL      = 0U;          -- DCBEVT2
Force Sync Signal
       EPwm2Regs.DCBCTL.bit.EVT2SRCSEL          = 0U;          -- DCBEVT2
Source Signal
     */
    EPwm2Regs.DCBCTL.all = (EPwm2Regs.DCBCTL.all & ~0x30FU) | 0x0U;

    /*      -- Digital Compare Trip Select Register
       EPwm2Regs.DCTRIPSEL.bit.DCAHCOMPSEL      = 0U;          -- Digital
Compare A High COMP Input Select

       EPwm2Regs.DCTRIPSEL.bit.DCALCOMPSEL      = 0U;          -- Digital
Compare A Low COMP Input Select
       EPwm2Regs.DCTRIPSEL.bit.DCBHCOMPSEL      = 0U;          -- Digital
```

```
Compare B High COMP Input Select
       EPwm2Regs.DCTRIPSEL.bit.DCBLCOMPSEL      = 0U;          -- Digital
Compare B Low COMP Input Select
     */
    EPwm2Regs.DCTRIPSEL.all = (EPwm2Regs.DCTRIPSEL.all & ~ 0xFFFFU) | 0x0U;

    /*      -- Trip Zone Digital Comparator Select Register
       EPwm2Regs.TZDCSEL.bit.DCAEVT1            = 0U;          -- Digital
Compare Output A Event 1
       EPwm2Regs.TZDCSEL.bit.DCAEVT2            = 0U;          -- Digital
Compare Output A Event 2
       EPwm2Regs.TZDCSEL.bit.DCBEVT1            = 0U;          -- Digital
Compare Output B Event 1
       EPwm2Regs.TZDCSEL.bit.DCBEVT2            = 0U;          -- Digital
Compare Output B Event 2
     */
    EPwm2Regs.TZDCSEL.all = (EPwm2Regs.TZDCSEL.all & ~0xFFFU) | 0x0U;

    /*      -- Digital Compare Filter Control Register
       EPwm2Regs.DCFCTL.bit.BLANKE              = 0U;          -- Blanking
Enable/Disable
       EPwm2Regs.DCFCTL.bit.PULSESEL            = 1U;          -- Pulse
Select for Blanking & Capture Alignment
       EPwm2Regs.DCFCTL.bit.BLANKINV            = 0U;          -- Blanking
Window Inversion
       EPwm2Regs.DCFCTL.bit.SRCSEL              = 0U;          -- Filter
Block Signal Source Select
     */
    EPwm2Regs.DCFCTL.all = (EPwm2Regs.DCFCTL.all & ~0x3FU) | 0x10U;
    EPwm2Regs.DCFOFFSET = 0U;          // Digital Compare Filter Offset
Register
    EPwm2Regs.DCFWINDOW = 0U;          // Digital Compare Filter Window
Register

    /*      -- Digital Compare Capture Control Register
       EPwm2Regs.DCCAPCTL.bit.CAPE              = 0U;          -- Counter
Capture Enable
     */
    EPwm2Regs.DCCAPCTL.all = (EPwm2Regs.DCCAPCTL.all & ~0x1U) | 0x0U;

    /*      -- HRPWM Configuration Register
       EPwm2Regs.HRCNFG.bit.SWAPAB              = 0U;          -- Swap
```

```
EPWMA and EPWMB Outputs Bit
       EPwm2Regs.HRCNFG.bit.SELOUTB             = 0U;          -- EPWMB
Output Selection Bit
     */
    EPwm2Regs.HRCNFG.all = (EPwm2Regs.HRCNFG.all & ~0xA0U) | 0x0U;

    /* Update the Link Registers with the link value for all the Compare
values and TBPRD */
    /* No error is thrown if the ePWM register exists in the model or not
*/
    EPwm2Regs.EPWMXLINK.bit.TBPRDLINK = 1U;
    EPwm2Regs.EPWMXLINK.bit.CMPALINK = 1U;
    EPwm2Regs.EPWMXLINK.bit.CMPBLINK = 1U;
    EPwm2Regs.EPWMXLINK.bit.CMPCLINK = 1U;
    EPwm2Regs.EPWMXLINK.bit.CMPDLINK = 1U;

    /* SYNCPER - Peripheral synchronization output event
       EPwm2Regs.HRPCTL.bit.PWMSYNCSEL            = 0U;           --
EPWMSYNCPER selection
       EPwm2Regs.HRPCTL.bit.PWMSYNCSELX           = 0U;           --
EPWMSYNCPER selection
     */
    EPwm2Regs.HRPCTL.all = (EPwm2Regs.HRPCTL.all & ~0x72U) | 0x0U;
    EDIS;
  }

  /* Start for S-Function (c2802xpwm): '<S1>/ePWM1' */
  real32_T tbprdValue1 = (real32_T)EPwm1Regs.TBPRD;

  /*** Initialize ePWM1 modules ***/
  {
    /*  -- Time Base Control Register
       EPwm1Regs.TBCTL.bit.CTRMODE              = 2U;          -- Counter
Mode
       EPwm1Regs.TBCTL.bit.SYNCOSEL             = 3U;          -- Sync
Output Select
       EPwm1Regs.TBCTL2.bit.SYNCOSELX           = 0U;          -- Sync
Output Select - additional options

       EPwm1Regs.TBCTL.bit.PRDLD                = 0U;          -- Shadow
select
```

```
       EPwm1Regs.TBCTL2.bit.PRDLDSYNC           = 0U;          -- Shadow
select

       EPwm1Regs.TBCTL.bit.PHSEN                = 0U;          -- Phase
Load Enable
       EPwm1Regs.TBCTL.bit.PHSDIR               = 0U;          -- Phase
Direction Bit
       EPwm1Regs.TBCTL.bit.HSPCLKDIV            = 1U;          -- High
Speed TBCLK Pre-scaler
       EPwm1Regs.TBCTL.bit.CLKDIV               = 0U;          -- Time Base
Clock Pre-scaler
     */
    EPwm1Regs.TBCTL.all = (EPwm1Regs.TBCTL.all & ~0x3FFFU) | 0xB2U;
    EPwm1Regs.TBCTL2.all = (EPwm1Regs.TBCTL2.all & ~0xF000U) | 0x0U;

    /*-- Setup Time-Base (TB) Submodule --*/
    EPwm1Regs.TBPRD = 2500U;           // Time Base Period Register

    /* -- Time-Base Phase Register
       EPwm1Regs.TBPHS.bit.TBPHS               = 0U;          -- Phase
offset register
     */
    EPwm1Regs.TBPHS.all = (EPwm1Regs.TBPHS.all & ~0xFFFF0000U) | 0x0U;

    // Time Base Counter Register
    EPwm1Regs.TBCTR = 0x0000U;         /* Clear counter*/

    /*-- Setup Counter_Compare (CC) Submodule --*/
    /*      -- Counter Compare Control Register

       EPwm1Regs.CMPCTL.bit.LOADASYNC           = 0U;          -- Active
Compare A Load SYNC Option
       EPwm1Regs.CMPCTL.bit.LOADBSYNC           = 0U;          -- Active
Compare B Load SYNC Option
       EPwm1Regs.CMPCTL.bit.LOADAMODE           = 0U;          -- Active
Compare A Load
       EPwm1Regs.CMPCTL.bit.LOADBMODE           = 0U;          -- Active
Compare B Load
       EPwm1Regs.CMPCTL.bit.SHDWAMODE           = 0U;          -- Compare A
Register Block Operating Mode
       EPwm1Regs.CMPCTL.bit.SHDWBMODE           = 0U;          -- Compare B
Register Block Operating Mode
```

```
     */
    EPwm1Regs.CMPCTL.all = (EPwm1Regs.CMPCTL.all & ~0x3C5FU) | 0x0U;

    /* EPwm1Regs.CMPCTL2.bit.SHDWCMODE           = 0U;          -- Compare
C Register Block Operating Mode
       EPwm1Regs.CMPCTL2.bit.SHDWDMODE           = 0U;          -- Compare
D Register Block Operating Mode
       EPwm1Regs.CMPCTL2.bit.LOADCSYNC           = 0U;          -- Active
Compare C Load SYNC Option
       EPwm1Regs.CMPCTL2.bit.LOADDSYNC           = 0U;          -- Active
Compare D Load SYNC Option
       EPwm1Regs.CMPCTL2.bit.LOADCMODE           = 0U;          -- Active
Compare C Load
       EPwm1Regs.CMPCTL2.bit.LOADDMODE           = 0U;          -- Active
Compare D Load
     */
    EPwm1Regs.CMPCTL2.all = (EPwm1Regs.CMPCTL2.all & ~0x3C5FU) | 0x0U;
    EPwm1Regs.CMPA.bit.CMPA = 1250U;   // Counter Compare A Register
    EPwm1Regs.CMPB.bit.CMPB = 1250U;   // Counter Compare B Register
    EPwm1Regs.CMPC = 32000U;           // Counter Compare C Register
    EPwm1Regs.CMPD = 32000U;           // Counter Compare D Register

    /*-- Setup Action-Qualifier (AQ) Submodule --*/
    EPwm1Regs.AQCTLA.all = 150U;
                               // Action Qualifier Control Register For
Output A
    EPwm1Regs.AQCTLB.all = 2310U;
                               // Action Qualifier Control Register For
Output B

    /*      -- Action Qualifier Software Force Register
       EPwm1Regs.AQSFRC.bit.RLDCSF               = 0U;          -- Reload
from Shadow Options
     */
    EPwm1Regs.AQSFRC.all = (EPwm1Regs.AQSFRC.all & ~0xC0U) | 0x0U;

    /*      -- Action Qualifier Continuous S/W Force Register
       EPwm1Regs.AQCSFRC.bit.CSFA                = 0U;          --
Continuous Software Force on output A
       EPwm1Regs.AQCSFRC.bit.CSFB                = 0U;          --
Continuous Software Force on output B
     */
```

```
    EPwm1Regs.AQCSFRC.all = (EPwm1Regs.AQCSFRC.all & ~0xFU) | 0x0U;

    /*-- Setup Dead-Band Generator (DB) Submodule --*/
    /*      -- Dead-Band Generator Control Register
       EPwm1Regs.DBCTL.bit.OUT_MODE             = 3U;          -- Dead Band
Output Mode Control
       EPwm1Regs.DBCTL.bit.IN_MODE              = 0U;          -- Dead Band
Input Select Mode Control
       EPwm1Regs.DBCTL.bit.POLSEL               = 2U;          -- Polarity
Select Control
       EPwm1Regs.DBCTL.bit.HALFCYCLE            = 0U;          -- Half
Cycle Clocking Enable
       EPwm1Regs.DBCTL.bit.SHDWDBREDMODE        = 0U;          -- DBRED
shadow mode
       EPwm1Regs.DBCTL.bit.SHDWDBFEDMODE        = 0U;          -- DBFED
shadow mode
       EPwm1Regs.DBCTL.bit.LOADREDMODE          = 4U;        -- DBRED load
       EPwm1Regs.DBCTL.bit.LOADFEDMODE          = 4U;        -- DBFED load
     */
    EPwm1Regs.DBCTL.all = (EPwm1Regs.DBCTL.all & ~0x8FFFU) | 0xBU;
    EPwm1Regs.DBRED.bit.DBRED = (uint16_T)(50.0);
                          // Dead-Band Generator Rising Edge Delay Count
Register
    EPwm1Regs.DBFED.bit.DBFED = (uint16_T)(50.0);
                         // Dead-Band Generator Falling Edge Delay Count
Register

    /*-- Setup Event-Trigger (ET) Submodule --*/
    /*      -- Event Trigger Selection and Pre-Scale Register
       EPwm1Regs.ETSEL.bit.SOCAEN               = 1U;          -- Start of
Conversion A Enable
       EPwm1Regs.ETSEL.bit.SOCASELCMP           = 0U;
       EPwm1Regs.ETSEL.bit.SOCASEL              = 1U;          -- Start of
Conversion A Select
       EPwm1Regs.ETPS.bit.SOCPSSEL              = 1U;          -- EPWM1SOC
Period Select
       EPwm1Regs.ETSOCPS.bit.SOCAPRD2           = 1U;
       EPwm1Regs.ETSEL.bit.SOCBEN               = 0U;          -- Start of
Conversion B Enable
       EPwm1Regs.ETSEL.bit.SOCBSELCMP           = 0U;
       EPwm1Regs.ETSEL.bit.SOCBSEL              = 1U;          -- Start of
Conversion A Select
```

```
       EPwm1Regs.ETPS.bit.SOCPSSEL               = 1;          -- EPWM1SOCB
Period Select
       EPwm1Regs.ETSOCPS.bit.SOCBPRD2            = 1U;
       EPwm1Regs.ETSEL.bit.INTEN                 = 0U;          -- EPWM1INTn
Enable
       EPwm1Regs.ETSEL.bit.INTSELCMP             = 0U;
       EPwm1Regs.ETSEL.bit.INTSEL                = 1U;          -- Start of
Conversion A Select
       EPwm1Regs.ETPS.bit.INTPSSEL               = 1U;          // EPWM1INTn
Period Select
       EPwm1Regs.ETINTPS.bit.INTPRD2             = 1U;
     */
    EPwm1Regs.ETSEL.all = (EPwm1Regs.ETSEL.all & ~0xFF7FU) | 0x1901U;
    EPwm1Regs.ETPS.all = (EPwm1Regs.ETPS.all & ~0x30U) | 0x30U;
    EPwm1Regs.ETSOCPS.all = (EPwm1Regs.ETSOCPS.all & ~0xF0FU) | 0x101U;
    EPwm1Regs.ETINTPS.all = (EPwm1Regs.ETINTPS.all & ~0xFU) | 0x1U;

    /*-- Setup PWM-Chopper (PC) Submodule --*/
    /*      -- PWM Chopper Control Register
       EPwm1Regs.PCCTL.bit.CHPEN                 = 0U;          -- PWM
chopping enable
       EPwm1Regs.PCCTL.bit.CHPFREQ               = 0U;          -- Chopping
clock frequency
       EPwm1Regs.PCCTL.bit.OSHTWTH               = 0U;          -- One-shot
pulse width
       EPwm1Regs.PCCTL.bit.CHPDUTY               = 0U;          -- Chopping
clock Duty cycle
     */
    EPwm1Regs.PCCTL.all = (EPwm1Regs.PCCTL.all & ~0x7FFU) | 0x0U;

    /*-- Set up Trip-Zone (TZ) Submodule --*/
    EALLOW;
    EPwm1Regs.TZSEL.all = 0U;          // Trip Zone Select Register

    /*      -- Trip Zone Control Register
       EPwm1Regs.TZCTL.bit.TZA                   = 3U;          -- TZ1 to
TZ6 Trip Action On EPWM1A
       EPwm1Regs.TZCTL.bit.TZB                   = 3U;          -- TZ1 to
TZ6 Trip Action On EPWM1B
       EPwm1Regs.TZCTL.bit.DCAEVT1               = 3U;          -- EPWM1A
action on DCAEVT1
       EPwm1Regs.TZCTL.bit.DCAEVT2               = 3U;          -- EPWM1A
```

```
action on DCAEVT2
       EPwm1Regs.TZCTL.bit.DCBEVT1              = 3U;          -- EPWM1B
action on DCBEVT1
       EPwm1Regs.TZCTL.bit.DCBEVT2              = 3U;          -- EPWM1B
action on DCBEVT2
     */
    EPwm1Regs.TZCTL.all = (EPwm1Regs.TZCTL.all & ~0xFFFU) | 0xFFFU;

    /*      -- Trip Zone Enable Interrupt Register
       EPwm1Regs.TZEINT.bit.OST                 = 0U;          -- Trip
Zones One Shot Int Enable
       EPwm1Regs.TZEINT.bit.CBC                 = 0U;          -- Trip
Zones Cycle By Cycle Int Enable
       EPwm1Regs.TZEINT.bit.DCAEVT1             = 0U;          -- Digital
Compare A Event 1 Int Enable
       EPwm1Regs.TZEINT.bit.DCAEVT2             = 0U;          -- Digital
Compare A Event 2 Int Enable
       EPwm1Regs.TZEINT.bit.DCBEVT1             = 0U;          -- Digital
Compare B Event 1 Int Enable
       EPwm1Regs.TZEINT.bit.DCBEVT2             = 0U;          -- Digital
Compare B Event 2 Int Enable
     */
    EPwm1Regs.TZEINT.all = (EPwm1Regs.TZEINT.all & ~0x7EU) | 0x0U;

    /*      -- Digital Compare A Control Register
       EPwm1Regs.DCACTL.bit.EVT1SYNCE           = 0U;          -- DCAEVT1
SYNC Enable
       EPwm1Regs.DCACTL.bit.EVT1SOCE            = 1U;          -- DCAEVT1
SOC Enable
       EPwm1Regs.DCACTL.bit.EVT1FRCSYNCSEL      = 0U;          -- DCAEVT1
Force Sync Signal
       EPwm1Regs.DCACTL.bit.EVT1SRCSEL          = 0U;          -- DCAEVT1
Source Signal
       EPwm1Regs.DCACTL.bit.EVT2FRCSYNCSEL      = 0U;          -- DCAEVT2
Force Sync Signal
       EPwm1Regs.DCACTL.bit.EVT2SRCSEL          = 0U;          -- DCAEVT2
Source Signal
     */
    EPwm1Regs.DCACTL.all = (EPwm1Regs.DCACTL.all & ~0x30FU) | 0x4U;

    /*      -- Digital Compare B Control Register
       EPwm1Regs.DCBCTL.bit.EVT1SYNCE           = 0U;          -- DCBEVT1
```

```
SYNC Enable
       EPwm1Regs.DCBCTL.bit.EVT1SOCE            = 0U;          -- DCBEVT1
SOC Enable
       EPwm1Regs.DCBCTL.bit.EVT1FRCSYNCSEL      = 0U;          -- DCBEVT1
Force Sync Signal
       EPwm1Regs.DCBCTL.bit.EVT1SRCSEL          = 0U;          -- DCBEVT1
Source Signal
       EPwm1Regs.DCBCTL.bit.EVT2FRCSYNCSEL      = 0U;          -- DCBEVT2
Force Sync Signal
       EPwm1Regs.DCBCTL.bit.EVT2SRCSEL          = 0U;          -- DCBEVT2
Source Signal
     */
    EPwm1Regs.DCBCTL.all = (EPwm1Regs.DCBCTL.all & ~0x30FU) | 0x0U;

    /*      -- Digital Compare Trip Select Register
       EPwm1Regs.DCTRIPSEL.bit.DCAHCOMPSEL      = 0U;          -- Digital
Compare A High COMP Input Select

       EPwm1Regs.DCTRIPSEL.bit.DCALCOMPSEL      = 0U;          -- Digital
Compare A Low COMP Input Select
       EPwm1Regs.DCTRIPSEL.bit.DCBHCOMPSEL      = 0U;          -- Digital
Compare B High COMP Input Select
       EPwm1Regs.DCTRIPSEL.bit.DCBLCOMPSEL      = 0U;          -- Digital
Compare B Low COMP Input Select
     */
    EPwm1Regs.DCTRIPSEL.all = (EPwm1Regs.DCTRIPSEL.all & ~ 0xFFFFU) | 0x0U;

    /*      -- Trip Zone Digital Comparator Select Register
       EPwm1Regs.TZDCSEL.bit.DCAEVT1            = 0U;          -- Digital
Compare Output A Event 1
       EPwm1Regs.TZDCSEL.bit.DCAEVT2            = 0U;          -- Digital
Compare Output A Event 2
       EPwm1Regs.TZDCSEL.bit.DCBEVT1            = 0U;          -- Digital
Compare Output B Event 1
       EPwm1Regs.TZDCSEL.bit.DCBEVT2            = 0U;          -- Digital
Compare Output B Event 2
     */
    EPwm1Regs.TZDCSEL.all = (EPwm1Regs.TZDCSEL.all & ~0xFFFU) | 0x0U;

    /*      -- Digital Compare Filter Control Register
       EPwm1Regs.DCFCTL.bit.BLANKE              = 0U;          -- Blanking
Enable/Disable
```

```
       EPwm1Regs.DCFCTL.bit.PULSESEL            = 1U;          -- Pulse
Select for Blanking & Capture Alignment
       EPwm1Regs.DCFCTL.bit.BLANKINV            = 0U;          -- Blanking
Window Inversion
       EPwm1Regs.DCFCTL.bit.SRCSEL              = 0U;          -- Filter
Block Signal Source Select
     */
    EPwm1Regs.DCFCTL.all = (EPwm1Regs.DCFCTL.all & ~0x3FU) | 0x10U;
    EPwm1Regs.DCFOFFSET = 0U;          // Digital Compare Filter Offset
Register
    EPwm1Regs.DCFWINDOW = 0U;          // Digital Compare Filter Window
Register

    /*      -- Digital Compare Capture Control Register
       EPwm1Regs.DCCAPCTL.bit.CAPE              = 0U;          -- Counter
Capture Enable
     */
    EPwm1Regs.DCCAPCTL.all = (EPwm1Regs.DCCAPCTL.all & ~0x1U) | 0x0U;

    /*      -- HRPWM Configuration Register
       EPwm1Regs.HRCNFG.bit.SWAPAB              = 0U;          -- Swap
EPWMA and EPWMB Outputs Bit
       EPwm1Regs.HRCNFG.bit.SELOUTB             = 0U;          -- EPWMB
Output Selection Bit
     */
    EPwm1Regs.HRCNFG.all = (EPwm1Regs.HRCNFG.all & ~0xA0U) | 0x0U;

    /* Update the Link Registers with the link value for all the Compare
values and TBPRD */
    /* No error is thrown if the ePWM register exists in the model or not
*/
    EPwm1Regs.EPWMXLINK.bit.TBPRDLINK = 0U;
    EPwm1Regs.EPWMXLINK.bit.CMPALINK = 0U;
    EPwm1Regs.EPWMXLINK.bit.CMPBLINK = 0U;
    EPwm1Regs.EPWMXLINK.bit.CMPCLINK = 0U;
    EPwm1Regs.EPWMXLINK.bit.CMPDLINK = 0U;

    /* SYNCPER - Peripheral synchronization output event
       EPwm1Regs.HRPCTL.bit.PWMSYNCSEL            = 0U;          --
EPWMSYNCPER selection
       EPwm1Regs.HRPCTL.bit.PWMSYNCSELX           = 0U;          --
EPWMSYNCPER selection
```

```
     */
    EPwm1Regs.HRPCTL.all = (EPwm1Regs.HRPCTL.all & ~0x72U) | 0x0U;
    EDIS;
  }

  /* InitializeConditions for Integrator: '<S23>/Integrator' */
  Final_Code_X.Integrator_CSTATE = 0.0;

  /* InitializeConditions for Integrator: '<S22>/Integrator' */
  Final_Code_X.Integrator_CSTATE_n = 0.0;

  /* InitializeConditions for Integrator: '<S1>/Integrator1' */
  Final_Code_X.Integrator1_CSTATE = 0.0;

  /* InitializeConditions for Integrator: '<S25>/Integrator' */
  Final_Code_X.Integrator_CSTATE_l = 0.0;

  /* InitializeConditions for Integrator: '<S24>/Integrator' */
  Final_Code_X.Integrator_CSTATE_g = 0.0;

  /* InitializeConditions for Integrator: '<S112>/Integrator' */
  Final_Code_X.Integrator_CSTATE_ly = 0.0;

  /* InitializeConditions for Integrator: '<S164>/Integrator' */
  Final_Code_X.Integrator_CSTATE_j = 0.0;

  /* InitializeConditions for Integrator: '<S60>/Integrator' */
  Final_Code_X.Integrator_CSTATE_c = 0.0;
}

/* Model terminate function */
void Final_Code_terminate(void)
{
  /* (no terminate code required) */
}
```