Closed Loop Brushless Direct Current (BLDC) Motion Control User Guide

Reference Design

FPGA-RD-02308-1.3

October 2025

Disclaimers

Lattice makes no warranty, representation, or guarantee regarding the accuracy of information contained in this document or the suitability of its products for any particular purpose. All information herein is provided AS IS, with all faults, and all associated risk is the responsibility entirely of the Buyer. The information provided herein is for informational purposes only and may contain technical inaccuracies or omissions, and may be otherwise rendered inaccurate for many reasons, and Lattice assumes no obligation to update or otherwise correct or revise this information. Products sold by Lattice have been subject to limited testing and it is the Buyer's responsibility to independently determine the suitability of any products and to test and verify the same. LATTICE PRODUCTS AND SERVICES ARE NOT DESIGNED, MANUFACTURED, OR TESTED FOR USE IN LIFE OR SAFETY CRITICAL SYSTEMS, HAZARDOUS ENVIRONMENTS, OR ANY OTHER ENVIRONMENTS REQUIRING FAIL-SAFE PERFORMANCE, INCLUDING ANY APPLICATION IN WHICH THE FAILURE OF THE PRODUCT OR SERVICE COULD LEAD TO DEATH, PERSONAL INJURY, SEVERE PROPERTY DAMAGE OR ENVIRONMENTAL HARM (COLLECTIVELY, "HIGH-RISK USES"). FURTHER, BUYER MUST TAKE PRUDENT STEPS TO PROTECT AGAINST PRODUCT AND SERVICE FAILURES, INCLUDING PROVIDING APPROPRIATE REDUNDANCIES, FAIL-SAFE FEATURES, AND/OR SHUT-DOWN MECHANISMS. LATTICE EXPRESSLY DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY OF FITNESS OF THE PRODUCTS OR SERVICES FOR HIGH-RISK USES. The information provided in this document is proprietary to Lattice Semiconductor, and Lattice reserves the right to make any changes to the information in this document or to any products at any time without notice.

Inclusive Language

This document was created consistent with Lattice Semiconductor’s inclusive language policy. In some cases, the language in underlying tools and other items may not yet have been updated. Please refer to Lattice’s inclusive language FAQ 6878 for a cross reference of terms. Note in some cases such as register names and state names it has been necessary to continue to utilize older terminology for compatibility.

Contents

Figures

Tables

Abbreviations in This Document

A list of abbreviations used in this document.

Abbreviations	Definition
ADC	Analog to Digital Converter
AHB	Advanced High-performance Bus
APB	Advanced Peripheral Bus
BiSS-C	Bidirectional Synchronous Serial – Continuous
BLDC	Brushless Direct Current
CNC	Computer Numerical Control
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
DIP	Dual Inline Package
DSP	Digital Signal Processor
EBR	Embedded Block RAM
FPGA	Field-Programmable Gate Array
GPIO	General Purpose Input/Output
GUI	Graphical User Interface
IP	Intellectual Property
K_I	Integral Term
K_p	Proportional Term
LED	Light-Emitting Diode
LMMI	Lattice Memory Mapped Interface
MISO	Master In Slave Out
MOSI	Master Out Slave In
PC	Personal Computer
PI	Proportional-Integral
PDM	Predictive Data Maintenance
PLL	Phase-locked Loop
PWM	Pulse Width Modulation
RISC-V	Reduced Instruction Set Computer Five
SCD	Single-cycle Data
SPI	Serial Peripheral Interface
SVPWM	Space Vector Pulse Width Modulation
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus

1. Introduction

Motion control is a sub-field of automation that involves the precise movement of machine components. To construct a complete system, the following crucial components are required:

- Motion controller: Controls the movement trajectories.
- Actuators: Converts electrical energy into mechanical motion. For example, a brushless direct current (BLDC) motor.
- Drivers and amplifiers: Provides the necessary power to the actuators based on controller. For example, a motor driver board.
- Feedback sensors: Provides real-time data on position for accurate speed control. For example, an encoder.

There are two main types of control as follows:

- Open loop control: Operates without feedback, relying on predefined calculations.
- Closed loop control: Uses real-time feedback from feedback sensors to adjust the motion.

Closed loop control is preferred and crucial for applications that require high precision and reliability, such as precise movement control in robotics or accurate cutting and shaping of materials in computer numerical control (CNC) machines. Through continuous monitoring and adjusting the system based on feedback, closed loop control ensures precise positioning on the motor. There are other advantages of using closed loop control, such as enhancing safety to prevent unsafe motion affecting both equipment and human, adaptability on dynamic load condition changes, and autonomous error correction to reduce deviation from optimal performance and stability.

FPGA offers flexibility in design, allowing easy integration of feedback sensors such as encoders with various communication protocols and interfaces. Industrial encoders such as absolute encoder provide position data that can be processed by the FPGA to facilitate development of accurate motion control algorithms. In addition, FPGAs excel at handling real-time data processing where you can make immediate adjustments with low latency to the motor control signals with the continuous position feedback from absolute encoders. Lattice™ low power FPGAs offer flexibility on implementing custom control algorithms as necessary for different motor control requirements.

This reference design describes and demonstrates an implementation of the encoder on real-time position feedback for speed control in the closed loop BLDC motor control system on the Lattice Certus™-NX FPGA device. A demo graphical user interface (GUI) application is provided to configure and run the BLDC motor in this reference design.

1.1. Quick Facts

Download the reference design files from the Lattice reference design web page: Closed Loop BLDC Motion Control Reference Design.

Table 1.1. Summary of the Reference Design

General	Target devices	LFD2NX-40, LFCPNX-100
	Source code format	Not available
Simulation	Functional simulation	Performed[1]
	Timing simulation	Not performed
	Testbench	Available[1]
	Testbench format	System Verilog
Software Requirements	Software tool and version	Lattice Propel™ software 2024.2 Lattice Radiant™ software 2024.2 Siemens® Questa™ advanced simulator
	Intellectual Property (IP) version[4]	SPI Controller IP v2.3.0 UART IP v1.3.0 GPIO IP v1.6.2 RISC-V MC CPU IP v2.5.0 RISC-V RTOS IP v2.4.0 AHB-Lite Interconnect Module v1.3.0 APB Interconnect Module v1.2.1 AHB-Lite to APB Bridge Module v1.1.0

		AXI4 to AHB-Lite Bridge Module v1.3.0 AXI Register Slice v1.0.0 PLL IP v1.9.0 System Memory IP v2.2.0 EnDat2.2 Master IP v1.0.3[2, 3] MB100 BiSS Interface Master IP v1.0.1[2, 3] Motor Control IP v2.0.14[3] APB Encoder Position Requestor IP v1.0.0[3]
Hardware Requirements	Board	Certus-NX Versa evaluation board CertusPro-NX evaluation board Encoder RS485 transceiver board Trenz® TEP0002 motor driver board
	Component	24 V, 10 A DC power supply (BLDC motor) 12 V, 3 A DC power supply (EnDat encoder) Anaheim Automation® BLY171D brushless DC motor Heidenhain® ROQ 437 EnDat rotary encoder Heidenhain K17 diaphragm coupling SICK AG® HIPERFACE DSL® EKS36-2KF0B020A rotary encoder Hengstler® AC58 encoder
	Cable	8-pin M12 encoder cable (1133832-01) USB 2.0 cable (USB Type A plug to mini USB Type B plug) 24 V power supply and adapter

Notes:

1. The functional simulation demonstrates reset sequence, basic initialization of the system, and encoder initialization (BiSS only).
2. This IP is provided by a third-party vendor.
3. This IP component is available via patch installation provided for this reference design in the Lattice website.
4. All IP components are available in the Lattice Propel Builder software unless indicated otherwise.

1.2. Features

The key features of the closed loop BLDC motion control reference design are as follows:

- Consists of the following control features in the Motor Control IP:
  - Closed loop control using real-time feedback of the motor position from the EnDat, HIPERFACE DSL, or Bidirectional Synchronous Serial – Continuous (BiSS-C) encoder.
  - Open loop control without feedback.
  - Three-phase pulse width modulation (PWM) signals with frequency of 16 kHz.
  - Configurable parameters such as proportional term (K_p) and integral term (K_I) in the feedback controller for speed control.
- Performs serial peripheral interface (SPI) communication with the EnDat2.2 Master IP, HIPERFACE DSL Master IP, or MB100 BiSS Interface Master IP.
- Performs encoder (EnDat, Hiperface DSL, BiSS-C) initialization using the RISCV MC, SX CPU application.
- Uses the demo GUI application for configuration control and status monitoring of the BLDC motor.
- Supports three absolute rotary encoder types with very high positioning accuracy:
  - The Heidenhain EnDat ROQ437 rotary encoder provides up to a resolution of 25-bit for single-turn revolution and 12-bit for multiturn revolution.
  - The SICK AG HIPERFACE DSL EKS36-2KF0B020A rotary encoder provides up to a resolution of 20-bit for single-turn revolution.
    Note: The SICK AG HIPERFACE DSL encoder is built into a customized synchronous servo motor model that has the following characteristics and is tested as a whole:
    - 10 poles or 5 pole pairs
    - Maximum RPM up to 4,000
    - Maximum torque up to 1.4 Nm at 30 A
    - 24 V DC bus voltage

    - The Hengstler AC58 encoder provides resolution of up to 17-bit for single-turn revolution and 12-bit for multiturn revolution.
- Uses an Anaheim BLY17 Series 8 poles dual-shafts brushless DC motor with high power density that is capable to run up to 4,000 RPM.

1.3. Naming Conventions

1.3.1. Nomenclature

The nomenclature used in this document is based on Verilog HDL.

1.3.2. Signal Names

Signal names that end with:

- _n are active low signals
- _i are input signals
- _o are output signals
- _io are bi-directional input and output signals

2. Directory Structure and Files

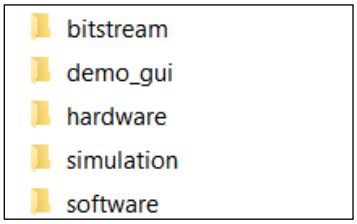

Figure 2.1. Directory Structure of the Closed Loop BLDC Motion Control Reference Design

Table 2.1. File List in the Reference Design Package

Attribute	Description
bitstream	Contains the Lattice Radiant software project bitstream that is compiled from the */hardware* folder.
demo_gui	Contains the demo GUI application for this reference design.
hardware	Contains the Lattice Radiant software project, the Lattice Propel Builder project, and the generated HDL files for the IP.
simulation	Contains the simulation scripts and testbench for this reference design.
software	Contains the software project and source code used in this reference design.

3. Functional Description

The reference design demonstrates a closed loop motor control system. This closed loop motor control system is created using the Lattice Propel Builder software.

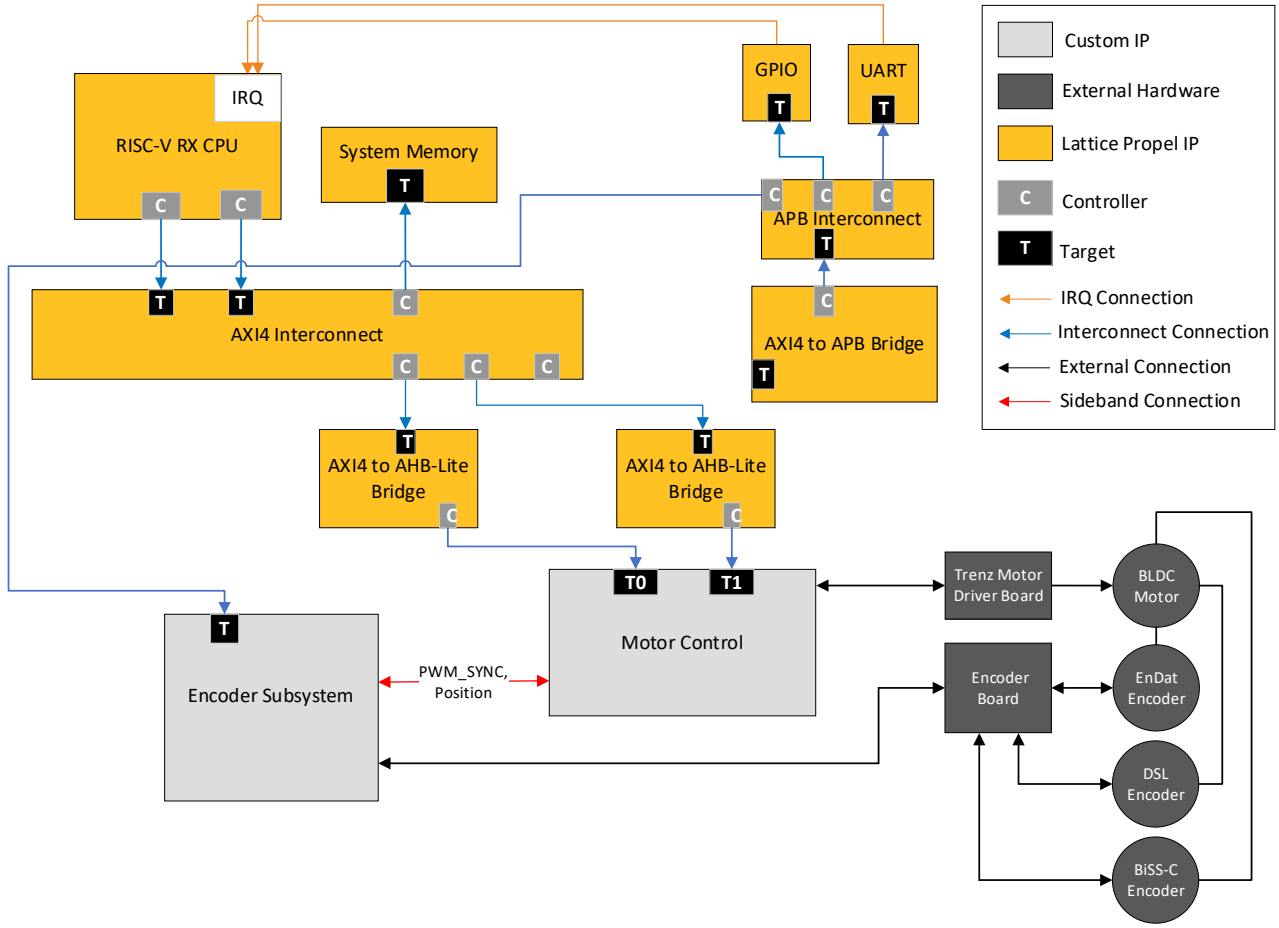

Figure 3.1. Closed Loop Motor Control System Architecture (CertusPro-NX)

For CertusPro-NX devices, the reference design is derived from the CertusPro-NX GHRD/GSRD reference design version 2.0 with reduced features. For more details, refer to the GHRD/GSRD Reference Design web page.

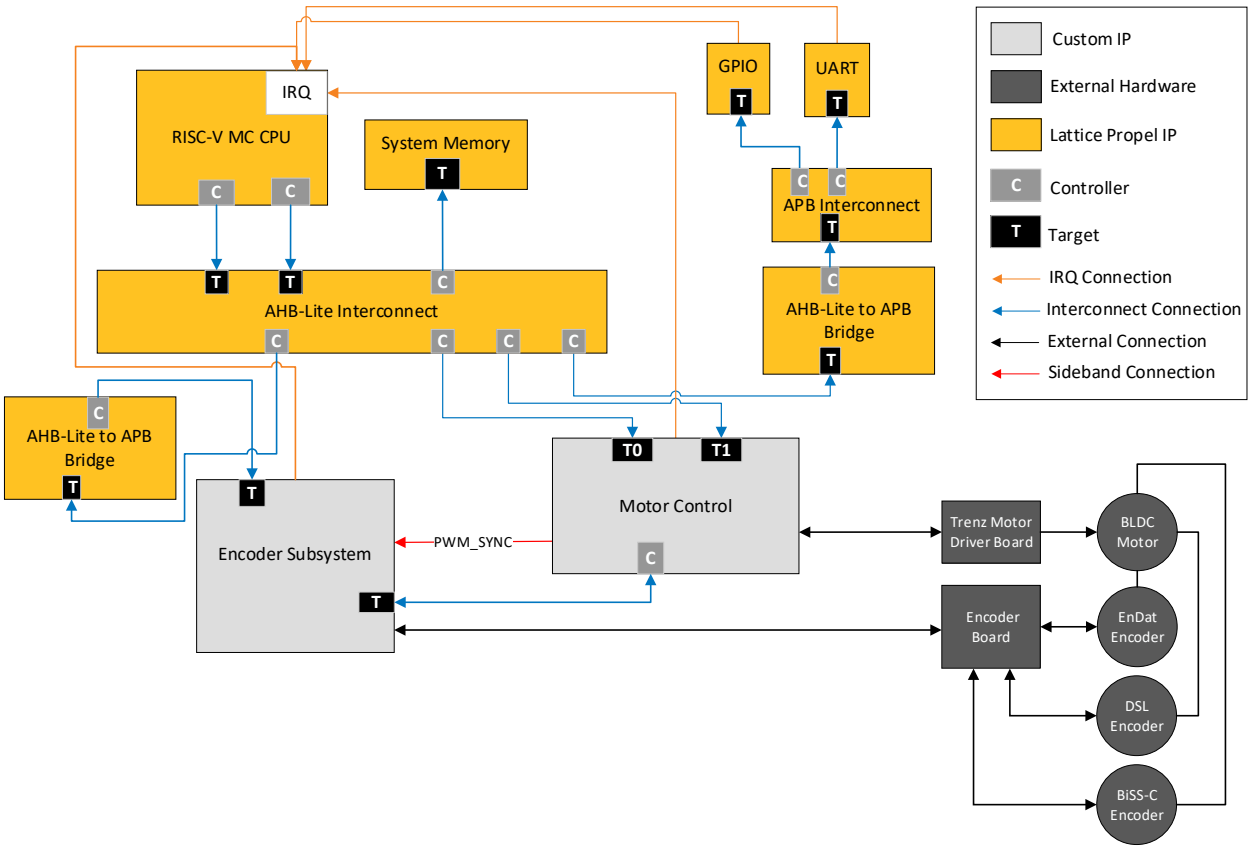

Figure 3.2. Closed Loop Motor Control System Architecture (Certus-NX)

The overall system consists of the following IP components:

- UART Controller
- GPIO
- RISC-V MC CPU
- Phase-locked loop (PLL)
- Motor Control
- AHB-Lite Interconnect
- APB Interconnect
- AHB-Lite to APB Bridge
- System Memory
- Encoder Subsystem

3.1. Design Components

3.1.1. RISC-V MC CPU

The RISC-V MC CPU IP has advanced high-performance bus (AHB)-Lite based instruction and data ports. The instruction port is connected to the System Memory that contains the system application for central processing unit (CPU) boot up when the FPGA device is programmed. The data port is connected to the System Memory and other peripherals for control and status read. The CPU also receives user input commands from the demo GUI application via the universal asynchronous receiver/transmitter (UART) serial communication.

For more information about the IP including the register map information, refer to RISC-V MC CPU IP User Guide (FPGA-IPUG-02252).

3.1.2. Encoder Subsystem

The encoder subsystem functions as a communication interface that allows any advanced peripheral bus (APB) controller to connect with third-party Master IPs, such as EnDat2.2, HIPERFACE DSL, and MB100 BiSS Interface, over SPI protocol. To enable autonomous position retrieval, the APB Encoder Position Requestor issues predefined instructions via the SPI controller, facilitating direct data access from the Master IP. This design simplifies integration by enabling consistent communication between the higher-level system and third-party IPs through standardized APB interconnect.

Note: The APB Encoder Position Requestor is instantiated within the encoder subsystem only in the CertusPro-NX reference design. For Certus-NX reference design, the position fetching is initiated from the motor control block.

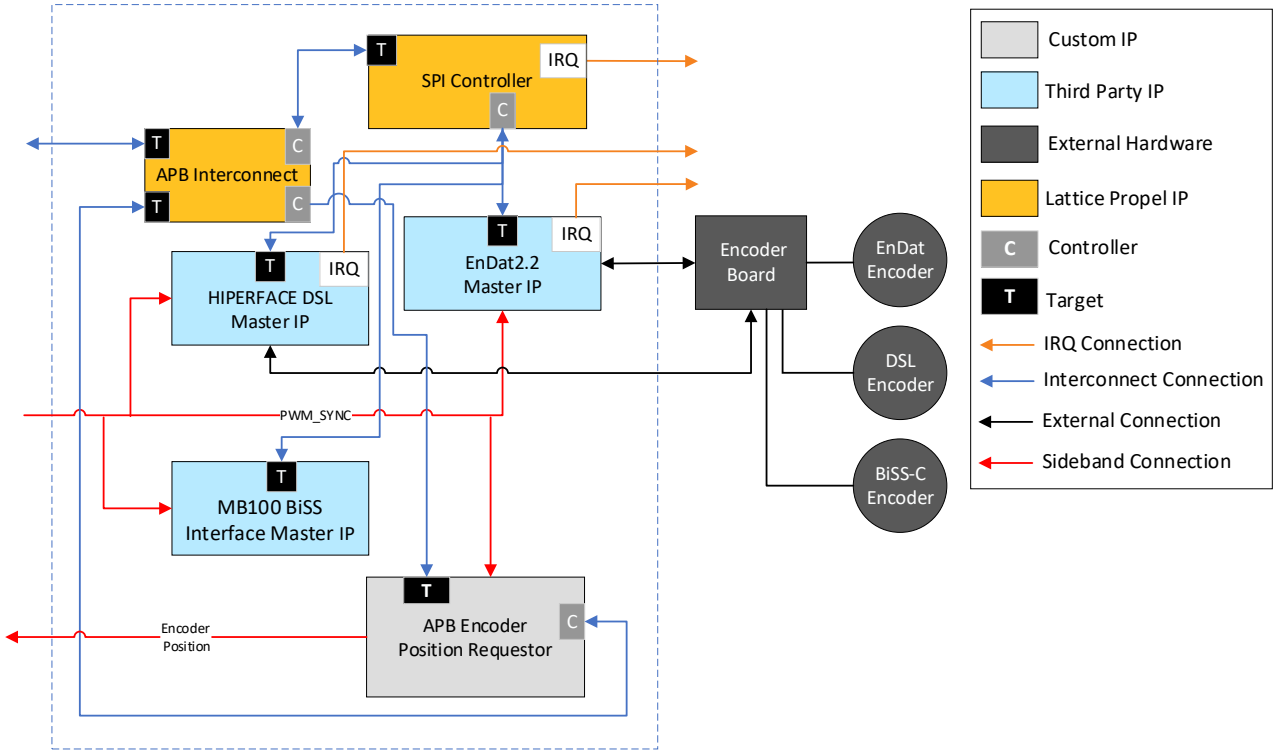

Figure 3.3. Encoder Subsystem Architecture

The encoder subsystem consists of the following IP components:

- EnDat2.2 Master
- HIPERFACE DSL Master
- MB100 BiSS Interface Master
- SPI Controller
- APB Interconnect
- APB Encoder Position Requestor

3.1.2.1. EnDat2.2 Master IP

The EnDat2.2 Master IP is a digital interface solution designed to manage communication between controllers and EnDat2.2-compatible encoders. This IP enables bidirectional, serial communication over four signal lines, allowing for the transmission of absolute position values, diagnostic data, and encoder parameters. This IP supports both incremental and absolute encoders, and is suited for safety-critical applications because of the robust error detection mechanisms, including CRC and redundant position data paths.

The EnDat2.2 Master IP facilitates high speed, synchronous data exchange with a clock rate up to 16 MHz, and supports advanced features like electronic ID labels, datum shifting, and real-time diagnostics. The compact design and minimal wiring requirements help reduce system complexity and cost, making the IP ideal for use in CNC machines, robotics, and industrial automation systems.

The EnDat2.2 Master IP consists of the following interfaces:

- EnDat interface communicates to the external EnDat encoder during initialization stage and normal operation stage for control and monitoring.
- SPI interface for the RISC-V MC CPU IP to perform initialization sequence as required by the EnDat encoder. During normal operation, the SPI interface is accessed by the motor control block to retrieve encoder position values through receive registers as defined in the EnDat2.2 Master IP.

For more details about the EnDat2.2 Master IP, refer to the Heidenhain web page.

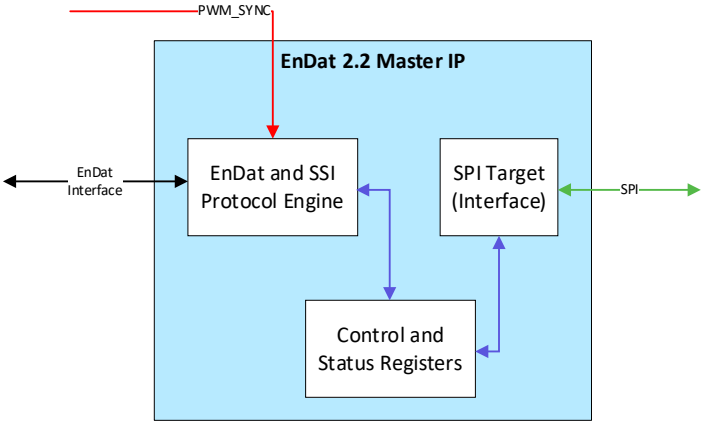

Figure 3.4. EnDat2.2 Master IP

3.1.2.2. HIPERFACE DSL Master IP

The HIPERFACE DSL Master IP is a specialized IP core designed to implement the HIPERFACE DSL motor feedback protocol within servo drives. This protocol enables high speed, digital communication between servo drives and SICK motor feedback encoders using a single-cable solution, which simplifies wiring and reduces system costs. The IP core is integrated into an FPGA and is available in two variants:

- Standard version for basic applications.
- Safe version that supports advanced safety functions and diagnostics in compliance with safety standards.

The HIPERFACE DSL Master IP consists of the following interfaces:

- DSL interface that communicates to the external DSL encoder during initialization stage and normal operation stage for control and monitoring.
- SPI interface for the RISC-V MC CPU IP to perform initialization sequence as required by the DSL encoder. During normal operation, the SPI interface is accessed by the motor control block to retrieve encoder position values through position registers as defined in the HIPERFACE DSL Master IP.

For further enquiry or the IP source of the HIPERFACE DSL Master IP, refer to the SICK AG web page.

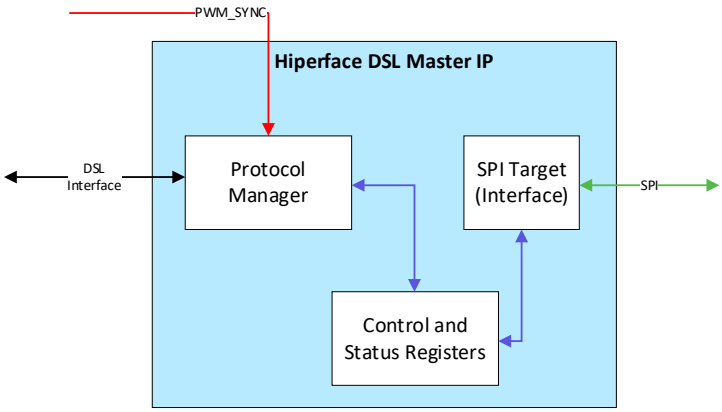

Figure 3.5. HIPERFACE DSL Master IP

3.1.2.3. MB100 BiSS Interface Master IP

The MB100 BiSS Interface Master IP is an IP designed to implement the BiSS-C mode protocol, which is an open-source digital interface for real-time communication with sensors and actuators. This IP enables high speed, bidirectional communication using two unidirectional lines (clock and data), making it compatible with the Synchronous Serial Interface (SSI) standard while offering enhanced features like cyclic data transmission, register access, and CRC-protected communication for increased reliability.

The MB100 BiSS Interface Master IP is often used in applications requiring precise position control, such as motor drives and industrial automation systems. This IP supports continuous data acquisition and control communication, allowing the master to both read sensor data and send commands. The IP is typically implemented in FPGAs or ASICs for the robustness in noisy industrial environments, because of features like line delay compensation and high speed transmission (up to 10 MHz or more depending on the physical layer used).

The MB100 BiSS Interface Master IP consists of the following interfaces:

- BISS-C interface that communicates to the external BiSS encoder during the initialization stage and normal operation stage for control and monitoring.
- SPI interface for the RISC-V MC CPU IP to perform initialization sequence as required by the BiSS encoder. During normal operation, the SPI interface is accessed by the motor control block to retrieve encoder position values through position registers as defined in the MB100 BiSS Interface Master IP.

For more details about the MB100 BiSS Interface Master IP, refer to the iC-Haus web page.

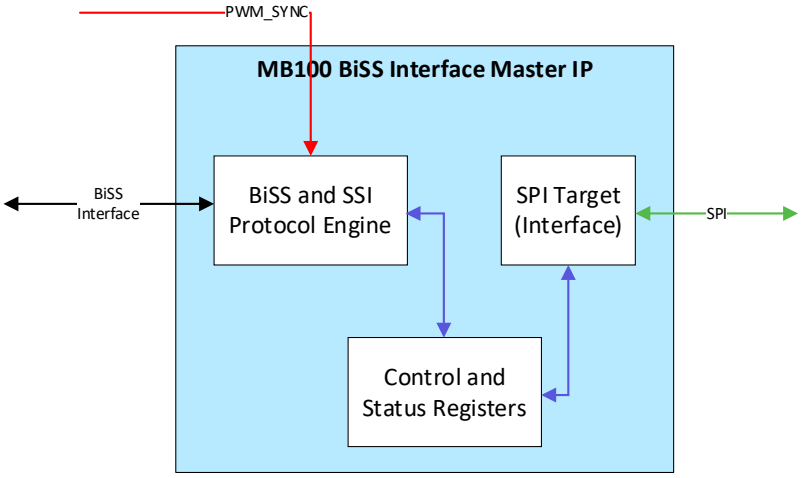

Figure 3.6. MB100 BiSS Interface Master IP

3.1.2.4. APB Encoder Position Requestor

The APB Encoder Position Requestor is a customized IP block designed to continuously issue instructions to the SPI controller for retrieving position data via the SPI interface. The requestor interacts with third-party Master IPs within the encoder subsystem. When a PWM SYNC pulse is received from the motor control block, the requestor triggers a set of predefined instructions to the SPI controller to fetch position data from the Master IP, based on the currently selected encoder type.

To use other encoder types not included in the reference design, refer to the Integrating with a Different Encoder section.

Note: Because of some issues in the IP Packager, some errors are reported in the Lattice Propel Builder when you connect the APB_S00 interface of the APB Encoder Position Requestor IP to an APB controller interface. This error will be fixed in future releases.

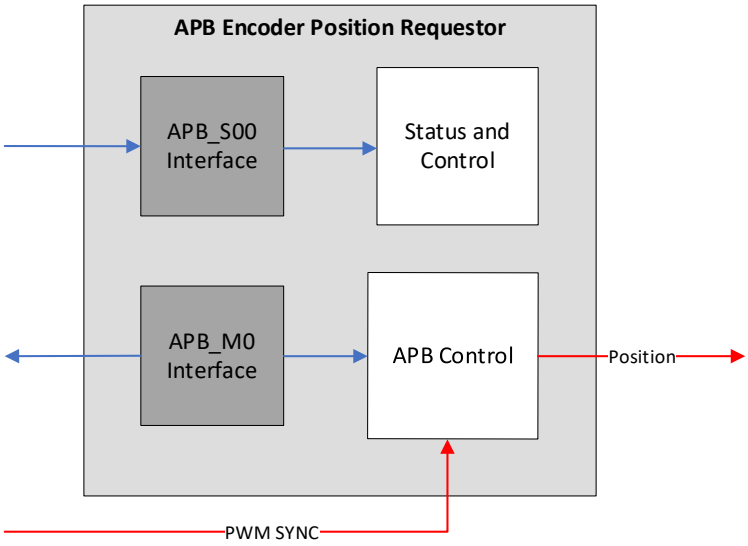

Figure 3.7. APB Encoder Position Requestor IP

3.1.2.5. APB Encoder Position Requestor IP Parameter

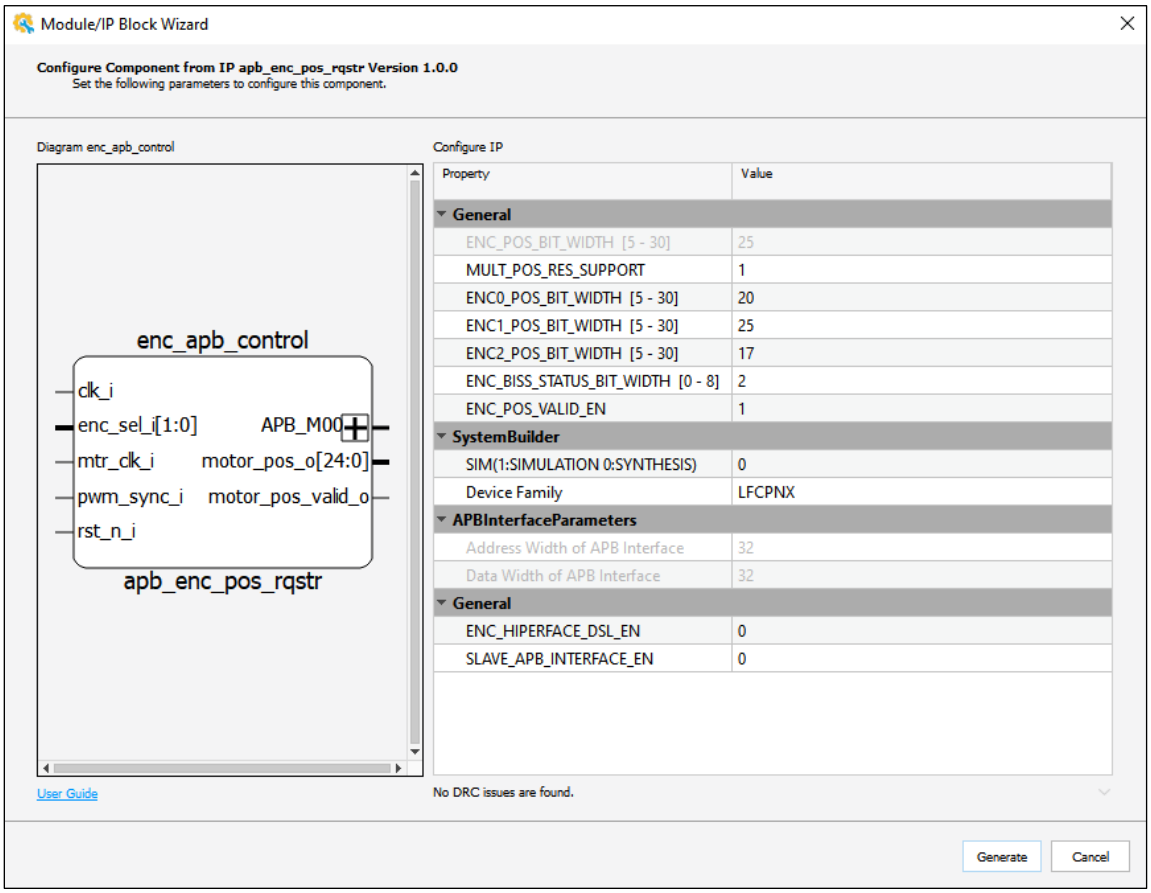

Figure 3.8. APB Encoder Position Requestor IP Block Wizard

Table 3.1. APB Encoder Position Requestor IP Attributes

Attribute	Selectable Values	Default	Dependency on Other Attributes
ENC_POS_BIT_WIDTH	5–30	25	Editable when MULT_POS_RES_SUPPORT is set to 0.
ENC0_POS_BIT_WIDTH	5–30	20	Editable when MULT_POS_RES_SUPPORT is set to 1.
ENC1_POS_BIT_WIDTH	5–30	25	Editable when MULT_POS_RES_SUPPORT is set to 1.
ENC2_POS_BIT_WIDTH	5–30	17	Editable when MULT_POS_RES_SUPPORT is set to 1.
ENC_BISS_STATUS_BIT_WIDTH	0–8	2	—
MULT_POS_RES_SUPPORT	0, 1	1	—
SLAVE_APB_INTERFACE_EN	0,1	0	—
ENC_HIPERFACE_DSL_EN	0,1	0	—
ENC_POS_VALID_EN	0,1	1	—
SIM	0, 1	0	—
DEVICE Family	LFD2NX, LFCPNX	LFD2NX	—

Table 3.2. APB Encoder Position Requestor IP Attributes Description

Attribute	Description
ENC_POS_BIT_WIDTH	Indicates the single-turn resolution width of the encoder position. This is the position value received from encoder based on one full revolution. For example, with a resolution of 25 bits, the position value of motor shaft located at 360 degrees is equal to 33554431 in decimal value.
ENC0_POS_BIT_WIDTH	Indicates the single-turn resolution width of the position for HIPERFACE DSL type encoder.
ENC1_POS_BIT_WIDTH	Indicates the single-turn resolution width of the position for EnDat2.2 type encoder.
ENC2_POS_BIT_WIDTH	Indicates the single-turn resolution width of the position for BiSS-C type encoder.
ENC_BISS_STATUS_BIT_WIDTH	Indicates the encoder status bit width for BiSS-C type encoder within the single-cycle data frame. The Motor Control IP uses this information to correctly extract the status bits value from the data frame when reading from data registers via MB100 BiSS Interface Master IP. For example, the Hengstler AC58 encoder used in this reference design has the following 31-bit data frame content: • Multi-turn resolution 12-bit • Single-turn resolution 17-bit • 2-bit of encoder status bits Hence, by specifying value of 2 for this field, the Motor Control IP correctly identifies the LSB 2 bits for status. The exact width of the encoder status bits may vary depending on the encoder manufacturer and the specific application. This value must be configured according to the encoder data sheet to ensure the correct bit width is used.
MULT_POS_RES_SUPPORT	Indicates the resolution support mode. 0 – Only a single resolution is supported based on POS_BIT. 1 – Supports up to three fixed encoder types, selectable via the enc_pos_bit_sel input.
SLAVE_APB_INTERFACE_EN	Indicates if the control APB interface is exposed. 0 – None 1 – Exposed
ENC_HIPERFACE_DSL_EN	Indicates if the HIPERFACE DSL type encoder is used. 0 – None 1 – The enc_pos_rdy_i input port is exposed to be connected to the fast position ready output port from the HIPERFACE DSL Master IP.
ENC_POS_VALID_EN	Indicates if the motor_pos_valid_o output port is exposed out. The motor_pos_valid_o indicates the valid status of encoder position output from the APB Encoder Position Requestor. 0 – None

Encoder	enc_sel_i Input	Resolution
HIPERFACEDSL	2'b00	ENC0_POS_BIT_WIDTH
EnDat2.2	2'b01	ENC1_POS_BIT_WIDTH
BiSS-C	2'b10	ENC2_POS_BIT_WIDTH

Attribute	Description
	1 – Exposed
SIM	Determines the type of IP generation. 0 – Simulation 1 – Synthesis
DEVICE Family	Specifies the supported device families: LFD2NX or LFCPNX.

3.1.2.6. Control and Status Registers

All registers are accessed through the APB_S00 interface.

Table 3.3. Access Types

Access Type	Behavior on Read Access	Behavior on Write Access
RO	Returns register value	Ignores write access
WO	Returns 0	Updates register value
RW	Returns register value	Updates register value
RC	Returns register value	Reads to clear or reset the register bit to the default value
RSVD	Returns 0	Ignores write access

Table 3.4. APB Encoder Position Requestor IP Registers

Register Offset	Register Name	Description	Access Point
0x00	APB_CTRL_REG	Control Register	APB_S00
0x04	APB_STATUS_REG	Status Register	APB_S00
0x08 onwards	RSVD	Reserved	—

Table 3.5. APB_CTRL_REG

Field	Name	Access	Default	Description
31:1	Reserved	RSVD	0	Reserved.
0	Enable	RW	1	Set this bit to enable the APB Encoder Position Requestor. Clear this bit to disable operation, such as when software needs to directly access register data from the Encoder Master IP. Before initiating any SPI Controller transactions, software must first verify that the Idle bit in APB_STATUS_REG is asserted, indicating the requestor is inactive. When the required access is complete, software must re-enable this bit to allow the APB Encoder Position Requestor to resume position-fetching operations.

Table 3.6. APB_STATUS_REG

Field	Name	Access	Default	Description
31:1	Reserved	RSVD	0	Reserved.
0	Idle	RO	1	0 – The APB encoder position requestor is active. 1 – The APB encoder position requestor is inactive.

3.1.3. Motor Control

The motor control block is a customized IP to drive and control the speed of the BLDC motor via three-phase space vector pulse width modulation (SVPWM) signals.

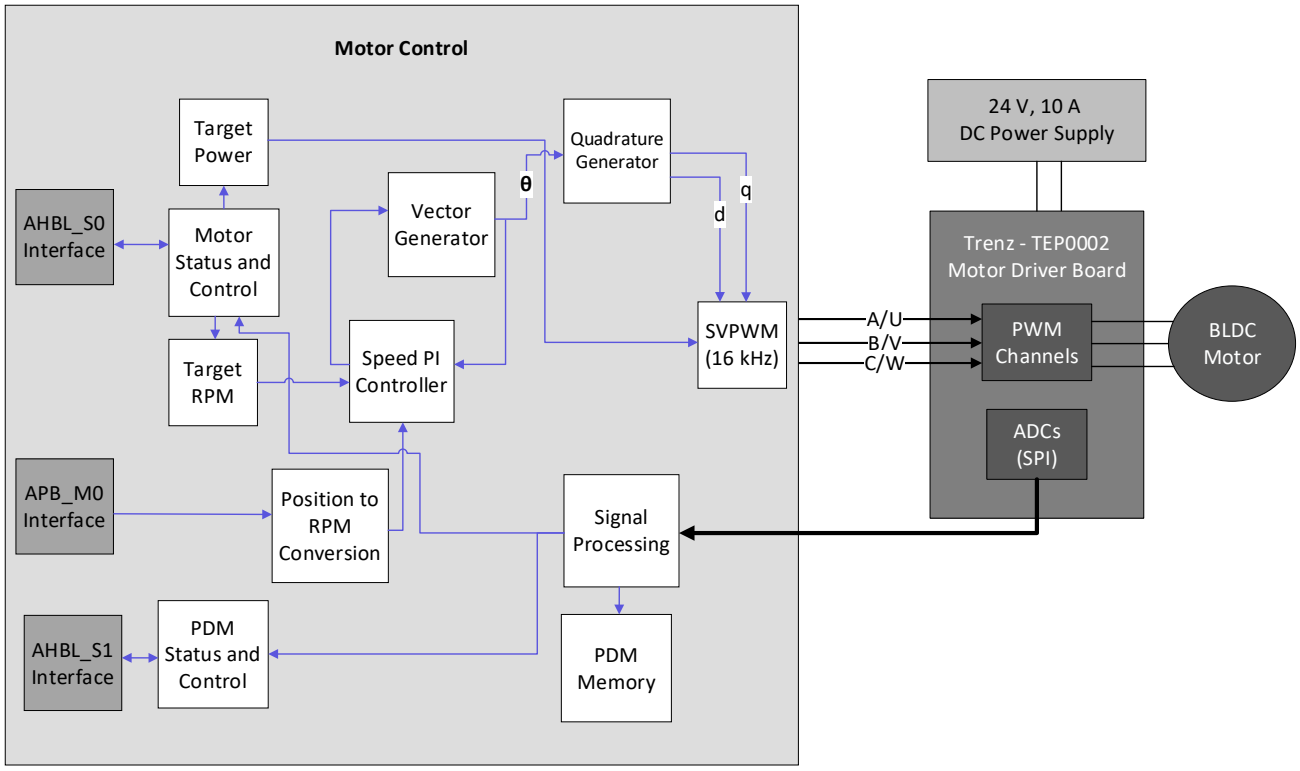

Figure 3.9. Motor Control IP

The motor control block provides the following capabilities:

- Open loop control (non-feedback)
- Closed loop control (real-time position feedback)
- Three-phase PWM signals with PWM SYNC pulse frequency of 16 kHz
- Autonomous encoder position fetching (closed loop control)

The motor control block has two AHB-Lite target interfaces and one APB controller interface:

- AHBL_S0 interface: Access to motor configuration and status registers.
- AHBL_S1 interface: Access to predictive maintenance control and status registers.
  Note: The predictive maintenance control feature is not demonstrated in this reference design.
- APB_M0 interface: Initiate position fetching and update operation to the encoder subsystem.
  Note: The APB_M0 interface is enabled only in the Motor Control IP for Certus-NX reference design. For CertusPro-NX reference design, the position comes from the APB Encoder Position Requestor block within the encoder subsystem.

3.1.3.1. Motor Control Operation

Table 3.7. Subblocks Operation of the Motor Control IP

Subblock	Operation
Motor status and control	Defines a list of configurable control and status registers that are accessible via AHBL_S0 and AHBL_S1 interfaces. This subblock ensures the feedback current from the analog-to-digital converter (ADC) channels located on the motor driver board does not exceed the maximum current value configured in the MTRCR1 register.
Target RPM	Converts the target RPM value to theta period (angular position or phase angle of the rotor in relation to the stator).
Target power	Takes in the values configured in the MTRCR0 and MTRCR1 registers and derives the high pulse width duration of the PWM output.
Position to RPM conversion	Takes in real-time encoder position and converts the value to theta period.
Speed PI controller	Takes in reference theta period generated from the following locations and drives the motor to the desired speed under both open loop control and closed loop control systems: • Target RPM subblock • Position to RPM conversion subblock • K_p and K_i values configured in the MTRCR2 and MTRCR3 registers
Vector generator	Converts the theta period into V_d and V_q values (two-phase rotating reference frame).
Quadrature generator	Converts V_d and V_q values into V_{Alpha} and V_{Beta} (two-phase orthogonal stationary reference frame) using the Inverse Park transform.
SVPWM	Converts V_{Alpha} and V_{Beta} into V_a, V_b, and V_c (three-phase stationary reference frame) using the Inverse Clarke transform.

The following steps summarize the operation of the motor control block:

1. The motor status and control subblock receives write request on motor configuration from RISC-V MC CPU via the AHBL_S0 interface.
2. The target RPM subblock derives the expected theta period based on the configured target RPM.
3. The target power subblock derives the high pulse width duration for the PWM output.
4. The position to RPM conversion subblock derives the expected theta period based on the feedback encoder position.
5. Depending on the open loop control or closed loop control operation, the speed PI controller subblock takes in the reference theta period from either the target RPM subblock or the position to RPM conversion subblock to derive the current theta period by taking into consideration the K_p and K_i values in calculation.
6. The vector generator subblock derives V_d and V_q based on the current theta period.
7. The quadrature generator subblock derives V_{Alpha} and V_{Beta}.
8. The SVPWM subblock derives V_a, V_b, and V_c, and drives out to the motor driver board.

3.1.3.2. Motor Control Parameter

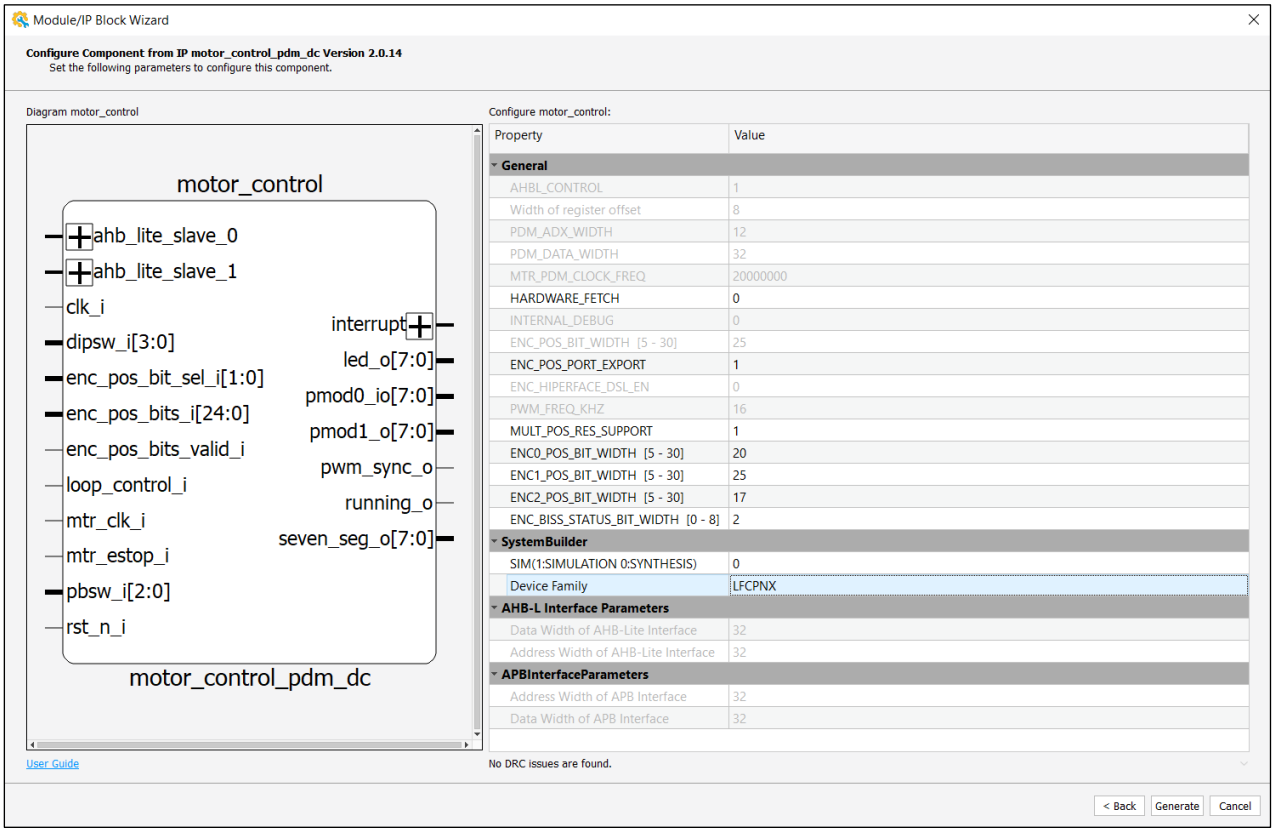

Figure 3.10. Motor Control IP Block Wizard

Table 3.8. Motor Control IP Attributes

Attribute	Selectable Values	Default	Dependency on Other Attributes
HARDWARE_FETCH	0, 1	1 (Certus-NX), 0 (CertusPro-NX)	—
ENC_POS_BIT_WIDTH	5–30	25	Editable when MULT_POS_RES_SUPPORT is set to 0.
ENC0_POS_BIT_WIDTH	5–30	20	Editable when MULT_POS_RES_SUPPORT is set to 1.
ENC1_POS_BIT_WIDTH	5–30	25	Editable when MULT_POS_RES_SUPPORT is set to 1.
ENC2_POS_BIT_WIDTH	5–30	17	Editable when MULT_POS_RES_SUPPORT is set to 1.
ENC_BISS_STATUS_BIT_WIDTH	0–8	2	—
MULT_POS_RES_SUPPORT	0, 1	1	—
ENC_POS_PORT_EXPORT	0, 1	0 (Certus-NX) 1 (CertusPro-NX)	Editable when HARDWARE_FETCH is set to 0.
SIM	0, 1	0	—
DEVICE Family	LFD2NX, LFCPNX	LFD2NX	—

Table 3.9. Motor Control IP Attributes Description

Attribute	Description
HARDWARE_FETCH	Indicates whether the Motor Control IP performs the encoder position fetching after a system reset. 0 – Does not perform the encoder position fetching after a system reset. 1 – Performs the encoder position fetching automatically via the APB_M0 interface after a system reset when the ENABLE bit is set in the MTRCR6 register. You must set this parameter to 0 if you are using encoder types other than EnDat2.2, HIPERFACE DSL, or BiSS-C.
ENC_POS_BIT_WIDTH	Indicates the single-turn resolution width of the encoder position. This is the position value received from encoder based on one full revolution. For example, with a resolution of 25 bits, the position value of motor shaft located at 360 degrees is equal to 33554431 in decimal value.
ENC0_POS_BIT_WIDTH	Indicates the single-turn resolution width of the position for HIPERFACE DSL type encoder.
ENC1_POS_BIT_WIDTH	Indicates the single-turn resolution width of the position for EnDat2.2 type encoder.
ENC2_POS_BIT_WIDTH	Indicates the single-turn resolution width of the position for BiSS-C type encoder.
ENC_BISS_STATUS_BIT_WIDTH	Indicates the encoder status bit width for BiSS-C type encoder in a single-cycle data frame. The Motor Control IP uses this information to isolate the position values from the data frame when reading from data registers via MB100 BiSS Interface Master IP. For example, the Hengstler AC58 encoder used in this reference design has the following 31-bit data frame content: • Multi-turn resolution 12-bit • Single-turn resolution 17-bit • 2-bit of encoder status bits The exact width of the encoder status bits may vary depending on the encoder manufacturer and the specific application. This value must be configured according to the encoder datasheet to ensure the correct bit width is used.
MULT_POS_RES_SUPPORT	Indicates the resolution support mode: 0 – Only a single resolution is supported based on POS_BIT. 1 – Supports up to three fixed encoder types, selectable via the enc_pos_bit_sel input.
ENC_POS_PORT_EXPORT	Indicates if Motor Control IP received an encoder position via self-encoder position fetching or from exported position port. 0 – The Motor Control IP receives an encoder position via self-encoder position fetching if HARDWARE_FETCH is set to 1. 1 – The Motor Control IP receives an encoder position through exported enc_pos_bit_sel port. When ENC_POS_PORT_EXPORT is set to 0, the Motor Control IP ignores the value from the enc_pos_bit_sel port.
SIM	Determines the type of IP generation. 0 – Simulation 1 – Synthesis
DEVICE Family	Specifies the supported device families: LFD2NX or LFCPNX.

Encoder	enc_pos_bit_sel Input	Resolution
HIPERFACEDSL	2'b00	ENC0_POS_BIT_WIDTH
EnDat2.2	2'b01	ENC1_POS_BIT_WIDTH
BiSS-C	2'b10	ENC2_POS_BIT_WIDTH
—	2’b11	Reserved

3.1.3.3. Control and Status Registers

All registers are accessed through the AHBL_S0 interface or the AHBL_S1 interface.

Table 3.10. Access Types

Access Type	Behavior on Read Access	Behavior on Write Access
RO	Returns register value	Ignores write access
WO	Returns 0	Updates register value
RW	Returns register value	Updates register value

Access Type	Behavior on Read Access	Behavior on Write Access
RC	Returns register value	Reads to clear or reset the register bit to the default value
RSVD	Returns 0	Ignores write access

Table 3.11. Motor Control IP Registers

Register Offset	Register Name	Description	Access Point
0x00	MTRCR0	Minimum RPM	AHBL_S0
0x04	MTRCR1	Maximum RPM	AHBL_S0
0x08	MTRCR2	RPM PI Ki	AHBL_S0
0x0C	MTRCR3	RPM PI KP	AHBL_S0
0x10–0x14	RSVD	Reserved	—
0x18	MTRCR6	Sync delay and control	AHBL_S0
0x1C	MTRCR7	Target RPM	AHBL_S0
0x20–0x24	RSVD	Reserved	AHBL_S0
0x28	MTRSR0	Status RPM	AHBL_S0
0x2C	MTRSR1	System status	AHBL_S0
0x30	PDMCR0	Predictive maintenance control 0	AHBL_S0
0x34	RSVD	Reserved	—
0x38	PDMSR	Predictive maintenance status	AHBL_S0
0x3C	PDMDDR	Predictive maintenance ADC data	AHBL_S0
0x40	PDMQDR	Predictive maintenance ADC data	AHBL_S0
0x50	RSVD	Reserved	—
0x54	BRDLEDS	Versa board light-emitting diodes (LEDs)	AHBL_S0
0x58–0x5C	RSVD	Reserved	—
0x60	ENC_POS	Encoder position	AHBL_S0
0x64	RSVD	Reserved	—
0x68	PWM_SYNC_IRQ	PWM_SYNC IRQ status	AHBL_S0
0x6C–0x74	RSVD	Reserved	—

Table 3.12. Minimum RPM

Field	Name	Access	Default	Description
31:24	RPM_PI_DELAY	RW	0	RPM PI update rate in the speed PI controller block. Valid values are 1 to 255.
23:16	MTRPOLES	RW	0	Number of motor stator pole pairs. The value must be configured according to the data sheet for the specific motor.
15:8	Reserved	RSVD	0	Reserved.
7:0	MINPWR	RW	0	Minimum power for the motor. The value configured impacts the high pulse width duration of the PWM output.

Table 3.13. Maximum RPM

Field	Name	Access	Default	Description
31:24	MAXAMPS	RW	0	Breaker amps for the motor. The value configured limits the maximum current drawn from the motor driver board.
23:16	PWRGAIN	RW	0	Power gain for the motor. The value configured impacts the high pulse width duration of the PWM output.
15:0	MAXRPM	RW	0	Maximum RPM is the upper limit RPM. Valid values are up to the maximum RPM supported by the motor. In this reference design, the maximum RPM configured must not exceed 2,000.

Table 3.14. RPM PI Ki

Field	Name	Access	Default	Description
31:16	RPMINT_MIN	RW	0	Integrator anti-windup threshold. Valid values are 1 to (2^{16} – 1).
15:0	RPMINTK	RW	0	Gain of the integrator part of the RPM PI control loop. Valid values are 1 to (2^{16} – 1).

Table 3.15. RPM PI Kp

Field	Name	Access	Default	Description
31:16	RPMINT_LIM	RW	0	Integrator anti-windup clamp. Valid values are 1 to (2^{16} – 1).
15:0	RPMPRPK	RW	0	Gain of the proportional part of the RPM PI control loop. Valid values are 1 to (2^{16} – 1).

Table 3.16. Synchronization Delay and Control

Field	Name	Access	Default	Description
31	ENGAGE	RW	0	Sync signal to latch all control registers from AHB-L clock domain (50 – 100 MHz) to motor clock domain (20 MHz). Writes to all other control registers first (including this register when the bit is set to 0). Writes to this register (read-modify-write) to set this bit. This register can also be used to synchronize multiple nodes. 0 – No updates to motor or predictive data maintenance (PDM) control registers. 1 – Transfers all control register from AHB-L holding registers to motor PDM active registers.
30	DIRECTION	RW	0	Indicates the direction of motor, depending on the MTR_TYPE value. **MTR_TYPE** / **Direction** 0 / 0 – Clockwise rotation, 1 – Counterclockwise rotation 1 / 1 – Clockwise rotation, 0 – Counterclockwise rotation
29	Reserved	RO	0	Reserved.
28	ENABLE	RW	0	Enables motor drivers. 0 – Disables motor drivers. 1 – Enables motor drivers.
27	ESTOP	RW	0	Emergency stop. 0 – Normal operation. 1 – Engages E-brakes without sync delay or MTR_ENGAGE.
26	APB_DISABLE	RW	0	Disables the APB controller. Software can set this bit to disable the internal APB controller from initiate position fetching operation if software needs to access to retrieve register information from the Encoder Master IP. Software must clear this bit to enable the APB controller to resume the position fetching operation. 0 – Enables APB controller to initiate position fetching operation. 1 – Disables APB controller from initiate position fetching operation.
25	STOP	RW	0	Holds the motor in position. 0 – Normal operation. 1 – Stops the motor rotation.
24	RESET_PI	RW	0	Resets the RPM PI control. 0 – Normal operation. 1 – Forces the output to match the input (zero input values force the output to default of 120 RPM).
23:0	Reserved	RO	0	Reserved.

Table 3.17. Target RPM

Field	Name	Access	Default	Description
31:17	Reserved	RO	0	Reserved.
16	MTR_TYPE	RW	0	Determines the behavior of the value in the DIRECTION to be interpreted by the Motor Control IP. For Anaheim motor, this bit is set to 0.
15:0	TRGRPM	RW	0	Target RPM. Valid values are 0 to (2^{16} – 1). The value must not exceed the value configured in MAXRPM of MTRCR1 register.

Table 3.18. Status RPM

Field	Name	Access	Default	Description
31:16	Reserved	RO	0	Reserved.
15:0	MTRSTRPM	RO	0	Current motor RPM. Valid values are 0 to (2^{16} – 1).

Table 3.19. System Status

Field	Name	Access	Default	Description
31	ENC_LINK_STAT	RO	0	0 – Encoder link is not established. 1 – Encoder link is established.
30	RSVD	RO	0	Reserved.
29	APB_IDLE	RO	1	0 – The APB controller is active. 1 – The APB controller is inactive.
28:11	RSVD	RO	0	Reserved.
10:9	ENC_POS_BIT_SEL	RO	0	2'b00 – HIPERFACE DSL encoder. 2'b01 – EnDat encoder. 2'b10 – BiSS-C encoder. 2'b11 – Reserved.
8	ECB_TRIPPED	RO	0	0 – ECB tripped does not occur. 1 – ECB tripped occurs because feedback current received from the motor driver board exceeded the value.
7	DRIVE_FAULT	RO	0	0 – Drive fault does not occur. 1 – Drive fault occurs. This bit is from motor driver board that drives to the actual motor when overcurrent fault is detected from the protection circuit.
6	I_LOOP_CONTROL	RO	—	0 – Open loop. 1 – Closed loop.
5	VLD_RPM	RO	0	0 – RPM to theta period calculation is in process or invalid RPM request. 1 – RPM to theta period calculation is complete.
4	STOP	RO	0	0 – Motor is at non-zero RPM. 1 – Motor is at zero RPM.
3	RPM_LOCK	RO	0	Indicates whether motor is at target RPM. 0 – Motor is not at target RPM. 1 – Motor is at target RPM.
2	DECL	RO	0	Indicates whether motor is deaccelerating. 0 – Motor is not deaccelerating. 1 – Motor is deaccelerating.
1	ACCEL	RO	0	Indicates whether motor is accelerating. 0 – Motor is not accelerating. 1 – Motor is accelerating.
0	MOV	RO	0	Indicates whether motor is moving. 0 – Motor stops or is coasting. 1 – Motor is moving under control.

Table 3.20. Predictive Maintenance Control0

Field	Name	Access	Default	Description
31:16	DCREVS	RW	0	Data collection revolutions used to capture PDM data. Valid values are 1 to 65,536.
15:8	PREREVS	RW	0	Pre-data collection revolutions. Number of theta (field vector) revolutions to ignore before data collection.
7	ADCH	RW	0	ADC channel select for PDMDDR and PDMQDR registers. 0 – ADC channel = Amps 1 – ADC channel = Volts
6	CALIB	RW	0	ADC offset calibration. 0 – Normal operation 1 – Calibrates ADC offsets (motor is not running)
5	RSVD	RO	0	Reserved.
4	CONTINUOUS	RW	0	Collects data as long as START = 1. 0 – Fixed – Collects PDM data for a set number of rotations 1 – Continuous – Collects PDM data continuously
3	2FOLDEN	RW	0	Enables double folding of PDM data. 0 – Double folding disabled 1 – Double folding enabled
2	FOLDEN	RW	0	Enables single folding of PDM data. 0 –Single Fold disabled 1 – Single Fold enabled
1	PKDTEN	RW	0	PDM normalization peak detect enable. 0 – PDM peak detect disabled 1 – PDM peak detect enabled
0	START	RW	0	Starts PDM data collection. 0 – Collection not started 1 – Collection started

Table 3.21. Predictive Maintenance Status

Field	Name	Access	Default	Description
31:16	ROT	RO	0	Current count of theta rotations for PDM data collection.
15:4	RSVD	RO	0	Reserved
3	READY	RO	0	PDM data collector status 0 – Not ready to collect data 1 – Ready to collect data
2	CAL_DONE	RO	0	ADC offset calibration status 0 – Offset calibration is not complete 1 – Offset calibration is complete
1	BUSY	RO	0	PDM activity status 0 – PDM is not active 1 – PDM is collecting data
0	DONE	RO	0	PDM activity status 0 – Data collection is not complete 1 – Data collection is complete

Table 3.22. Predictive Maintenance Current/Voltage Data (for Register Offset 0x3C)

Field	Name	Access	Default	Description
31:16	ADC1	RO	0	Voltage or current reading phase A
15:0	ADC0	RO	0	Voltage or current reading phase B

Table 3.23. Predictive Maintenance Current/Voltage Data (for Register Offset 0x40)

Field	Name	Access	Default	Description
31:16	ADC3	RO	0	Voltage or current reading of DC supply
15:0	ADC2	RO	0	Voltage or current reading phase C

Table 3.24. Versa Board LED

Field	Name	Access	Default	Description
31:16	RSVD	RO	0	Reserved
15:8	7SEG	RW	0	7SEG [0]: D36 segment a – 0 = On, 1 = Off 7SEG [1]: D36 segment b – 0 = On, 1 = Off 7SEG [2]: D36 segment c – 0 = On, 1 = Off 7SEG [3]: D36 segment d – 0 = On, 1 = Off 7SEG [4]: D36 segment e – 0 = On, 1 = Off 7SEG [5]: D36 segment f – 0 = On, 1 = Off 7SEG [6]: D36 segment g – 0 = On, 1 = Off 7SEG [7]: D36 segment dp – 0 = On, 1 = Off
7:0	RSVD	RO	0	Reserved

Table 3.25. Encoder Position

Field	Name	Access	Default	Description
31:0	ENC_POS	RO	0	Motor position received from encoder. The single-turn resolution depends on the model of the encoder. All unused bits are tied to zero. For EnDat encoder, the resolution is up to 25-bit (bit [24:0]). For HIPERFACE DSL encoder, the resolution is up to 20-bit (bit [19:0]). For BiSS-C encoder, the resolution is up to 17-bit (bit [16:0]).

Table 3.26. PWM_SYNC IRQ Status

Field	Name	Access	Default	Description
31:1	RSVD	RO	0	Reserved.
0	PWM_SYNC_IRQ	RW	0	IRQ status when PWM_SYNC is issued from the Motor Control IP.

3.1.4. SPI Controller

The SPI is a high-speed synchronous, serial, and full-duplex interface that allows a serial bitstream of configured length, 8, 16, 24, or 32 bits to be shifted into and out of the device at a programmed bit-transfer rate. The Lattice SPI Controller IP core is normally used to communicate with external SPI target devices such as display drivers, SPI EPROMS, and ADCs. For the SPI Controller IP within the encoder subsystem, the IP is used to communicate with the third-party Master IP (EnDat2.2 Master IP, HIPERFACE DSL Master IP, and MB100 BiSS Interface Master IP) for data communication on the encoder initialization and status monitor purposes.

For more information about the IP including register map information, refer to the SPI Controller IP User Guide (FPGA-IPUG-02069).

Note: For interoperability with the HIPERFACE DSL Master IP, a customized version of the SPI controller is provided in the reference design.

3.1.5. UART

The UART Transceiver IP core performs serial-to-parallel conversion of data characters received from a peripheral UART device and parallel-to-serial conversion of data characters received from the host locater inside the FPGA through an APB interface. In this system, UART is used to interface with the demo GUI.

For more information about the IP including register map information, refer to the UART IP User Guide (FPGA-IPUG-02105).

3.1.6. GPIO

The general purpose input/output (GPIO) Peripheral Soft IP is a simple IP designed to control GPIOs via Lattice memory mapped interface (LMMI) or APB interface. When configured as an input, the IP can detect the state of a GPIO by reading the state of the associated register. When configured as an output, the IP takes the value written into the associated register and controls the state of the controlled GPIO. In this system, GPIO has inputs connected to the dual inline package (DIP) switches on the board. The RISC-V MC CPU can read the input values via the APB interface.

For more information about the IP including register map information, refer to the GPIO IP User Guide (FPGA-IPUG-02076).

3.2. Data Flow

When the FPGA device is programmed, the RISC-V MC CPU streams the application from the system memory and initializes all peripherals in the design system. In addition, the application is also responsible to initialize the external encoder (EnDat, HIPERFACE DSL, or BiSS-C) using the SPI Controller IP and the corresponding Master IP (EnDat2.2 Master IP, HIPERFACE DSL Master IP, or MB100 BiSS Interface Master IP).

After the initialization stage is complete, the RISC-V MC CPU application monitors for any incoming request you send using the demo GUI application.

The figures below show the general application flow and the flow based on specific user input command via the demo GUI application in this reference design system.

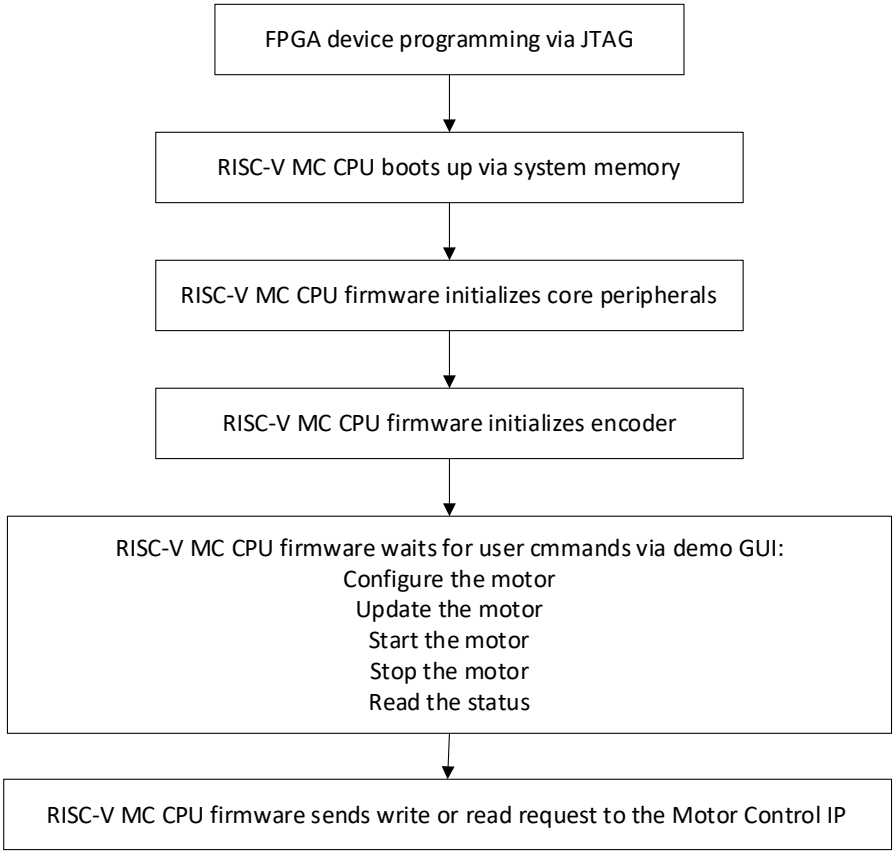

Figure 3.11. General Application Flow

3.3. Clocking Scheme

The clocking scheme implemented in this reference design system is described in the table below.

Table 3.27. Clocking

Attribute	Clock Frequency	Description
clk_100_i	100 MHz	PLL reference clock input from an on-board oscillator. This clock also drives to the clock input of the EnDat2.2 Master IP.
clkop_o	75 MHz	Clock output generated from PLL as the system clock. This clock also drives to the clock input of the HIPERFACEDSL Master IP.
clkos_o	20 MHz	Clock output generated from PLL to the Motor Control IP.
clkos2_o	20 MHz	Clock output generated from PLL to the MB100 BiSS Interface Master IP.
clkos3_o	20 MHz	Clock output generated from PLL (phase shift of 180 degrees) to the MB100 BiSS Interface Master IP.

3.3.1. Clocking Overview

The figures below show the clocking domain of the overall system and the encoder subsystem.

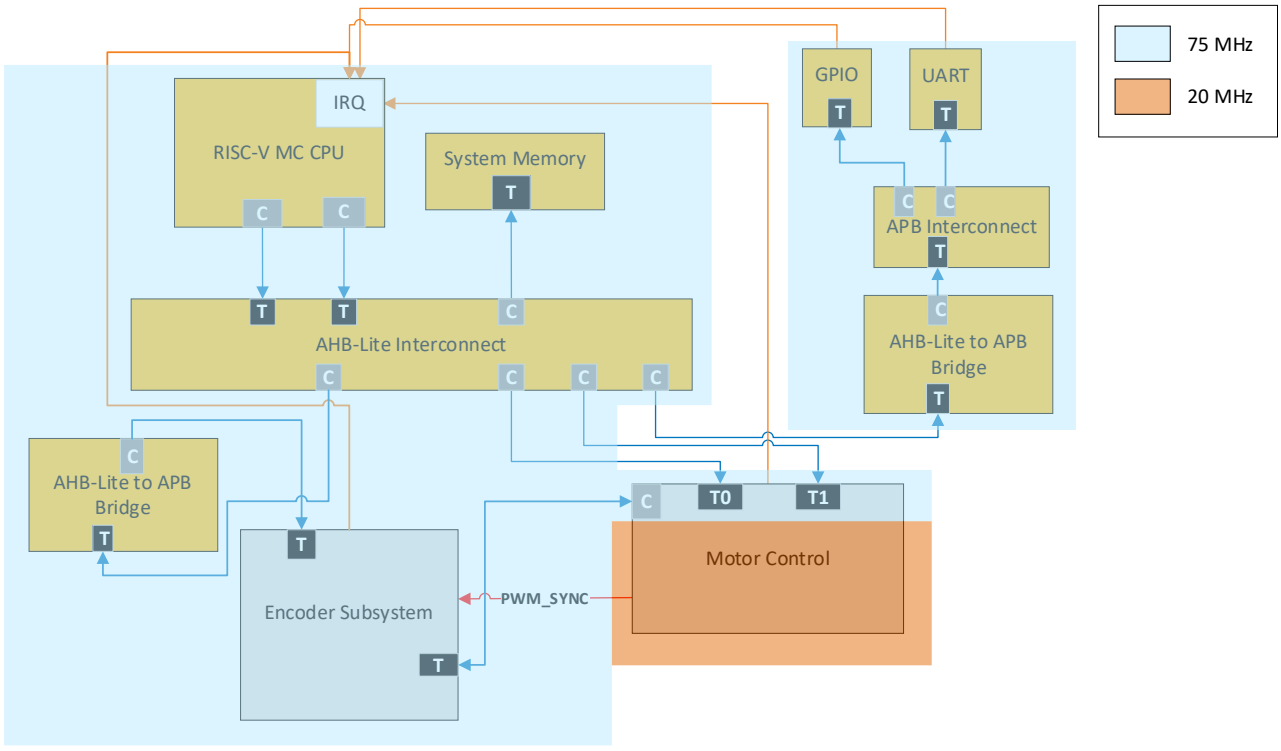

Figure 3.12. Overall System Clocking Domain

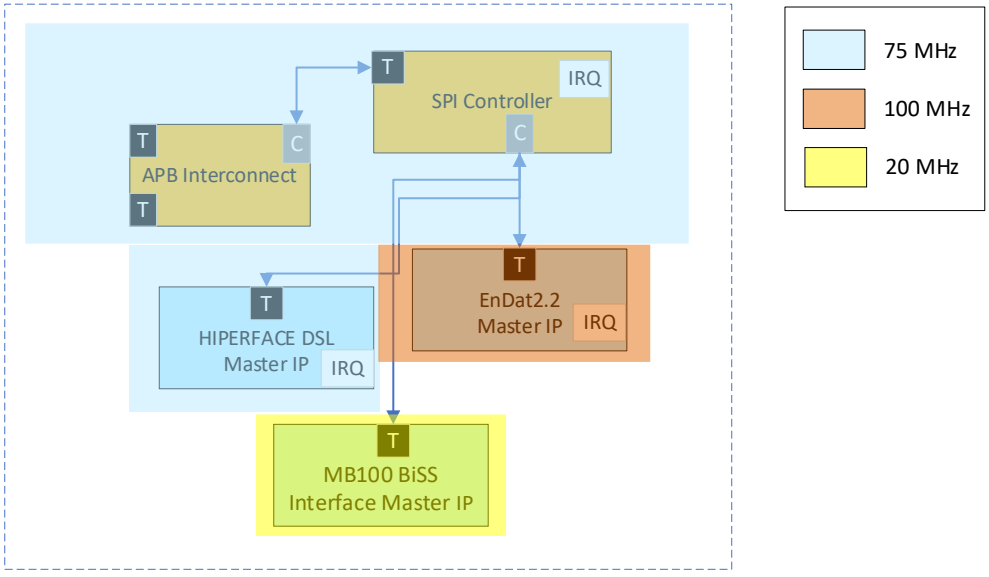

Figure 3.13. Encoder Subsystem Clocking Domain

3.4. Reset Scheme

The reset scheme implemented in this reference design system is described in the table below.

Table 3.28. Reset

Attribute	Description
rstn_i	Active low asynchronous reset input from on-board pushbutton. This input resets the entire system including the PLL.

Attribute	Description
lock_o	Locks output from PLL. Hold the RISC-V MC CPU in reset until this output is asserted.
system_resetn_o	Active low system reset output from RISC-V MC CPU. This output holds all peripherals in reset except the PLL.

3.4.1. Reset Overview

The figure below shows the reset sequence of this reference design system when you power up the FPGA device.

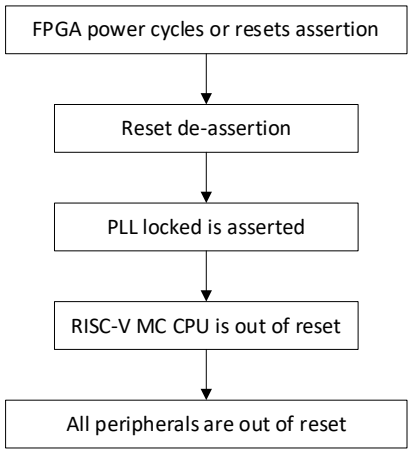

Figure 3.14. Reset Sequence Flow

3.5. Software

The following software is provided in this reference design:

- RISC-V bare metal program
- Drivers for peripherals, EnDat2.2 Master IP, HIPERFACE DSL Master IP, and MB100 BiSS Interface Master IP
- Demo GUI application

3.5.1. RISC-V Bare Metal Program

The bare metal program is built on-top of the RISC-V MC/RTOS CPU Hello World template that is generated using the Lattice Propel software.

3.5.1.1. Main APIs

Table 3.29. Main APIs

Main API Function	Description
int main (void)	When you power cycle the FPGA device or assert the system reset, the main function initializes and configures all peripherals in the system. After that, this function waits for the user input commands received from the demo GUI via UART communication.
void system_init(void)	Called within the main function which is responsible for initializing and configuring the interrupts and timer of the RISC-V MC CPU. In addition, this function also initializes the encoder (EnDat, HIPERFACE DSL, or BiSS-C) and peripherals such as GPIO, UART, and SPI.
void enc_init (void)	Called within the system_init function to initialize the encoder (EnDat, HIPERFACE DSL, or BiSS-C) using the respective encoder driver provided in this reference design.

3.5.2. Encoder Driver for the EnDat2.2 Master IP

An encoder driver is provided in this reference design based on the EnDat2.2 Master IP that is tested with the EnDat encoder. The encoder driver is located within the following directory:

<local directory>/Close-Loop-BLDC-Motion-Control-RD-Source-XXXXX/software/baremetal_test/src/encoder

For more details, refer to the Heidenhain web page for the EnDat2.2 Master IP and data sheet.

3.5.2.1. Encoder Driver APIs for the EnDat2.2 Master IP

Table 3.30. Encoder Driver APIs for the EnDat2.2 Master IP

API Function	Description
uint8_t init_endat22_master (uint8_t sim_en)	Called within the enc_init function to initialize the EnDat encoder via the EnDat2.2 Master IP and SPI Controller IP according to the initialization sequence specified in the IP data sheet.
void read_endat_id (uint32_t *data)	Reads the identification register available in the EnDat2.2 Master IP.
void cmd_mode0_transmit_position (void)	Initiates EnDat command mode 0 to the EnDat encoder via the EnDat2.2 Master IP.
void cmd_mode7_transmit_position_w_add_info (void)	Initiates EnDat command mode 7 to the EnDat encoder via the EnDat2.2 Master IP.
void cmd_mode1_selection_memory_area (uint8_t mrs_code_c)	Initiates EnDat command mode 1 targeting a specific address in the EnDat encoder via the EnDat2.2 Master IP.
void cmd_mode3_receive_parameters (uint8_t param_h, uint8_t param_l, uint8_t mrs_code_addr)	Initiates EnDat command mode 0 targeting a specific address and parameter value to be written to the EnDat encoder via the EnDat2.2 Master IP.
void cmd_mode4_transmit_parameters (uint8_t mrs_code_addr)	Initiates EnDat command mode 4 targeting a specific address in the EnDat encoder via the EnDat2.2 Master IP.
void cmd5_reset_encoder (uint8_t mrs_code_addr)	Initiates EnDat command mode 5 to reset the EnDat encoder via the EnDat2.2 Master IP.
void datum_shift_zero_alignment (uint8_t word0_byte_h, uint8_t word0_byte_l, uint8_t word1_byte_h, uint8_t word1_byte_l, uint8_t word2_byte_h, uint8_t word2_byte_l)	Performs electronic zeroing alignment in the EnDat encoder via the EnDat2.2 Master IP.
void set_recovery_tm (void)	Enables shorter recovery time in the EnDat encoder via the EnDat2.2 Master IP. The shorter recovery time is applicable only for type 2.2 mode commands.

3.5.3. Encoder Driver for the HIPERFACE DSL Master IP

An encoder driver is provided in this reference design based on the HIPERFACE DSL Master IP that is tested with the DSL encoder. The encoder driver is located within the following directory:

<local directory>/Close-Loop-BLDC-Motion-Control-RD-Source-XXXXX/software/baremetal_test/src/encoder

For more details, refer to the SICK AG web page for the HIPERFACE DSL Master IP and data sheet.

3.5.3.1. Encoder Driver APIs for the HIPERFACE DSL Master IP

Table 3.31. Encoder Driver APIs for the HIPERFACE DSL Master IP

API Function	Description
uint8_t init_hiperface_dsl (void)	Called within the enc_init function to initialize the DSL encoder via the HIPERFACE DSL Master IP and SPI Controller IP according to the initialization sequence specified in the IP data sheet.
uint8_t read_dsl_register (uint8_t *data, uint8_t addr_offset, uint8_t status_en)	Initiates register read operation to the HIPERFACE DSL Master IP.
uint8_t write_dsl_register (uint8_t data, uint8_t addr_offset, uint8_t status_en)	Initiates register write operation to the HIPERFACE DSL Master IP.

3.5.4. Encoder Driver for the MB100 BiSS Interface Master IP

An encoder driver is provided in this reference design based on the MB100 BiSS Interface Master IP that is tested with the BiSS-C encoder. The encoder driver is located within the following directory:

<local directory>/Close-Loop-BLDC-Motion-Control-RD-Source-XXXXX/software/baremetal_test/src/encoder

For more details, refer to the iC-Haus web page for the MB100 BiSS Interface Master IP and data sheet.

3.5.4.1. Encoder Driver APIs for the MB100 BiSS Interface Master IP

Table 3.32. Encoder Driver APIs for the MB100 BiSS Interface Master IP

API Function	Description
uint8_t init_biss_master (uint8_t sim_en)	Called within the enc_init function to initialize the BiSS-C encoder via the MB100 BiSS Interface Master IP and SPI Controller IP according to the initialization sequence specified in the IP data sheet.
uint8_t write_biss_register (uint8_t data, uint8_t addr_offset)	Initiates register write operation to the MB100 BiSS Interface Master IP.
uint8_t read_biss_register (uint8_t *data, uint8_t addr_offset)	Initiates register read operation to the MB100 BiSS Interface Master IP.
uint8_t read_biss_status_register (void)	Initiates register read operation to the status register of the MB100 BiSS Interface Master IP.

3.5.5. Demo GUI Application

The reference design provides a demo GUI application that supports the following features:

- Sends and receives characters via UART communication between the demo GUI application and the RISC-V program
- Configures the motor
- Starts the motor
- Stops the motor
- Monitors runtime motor status
- RPM versus time chart

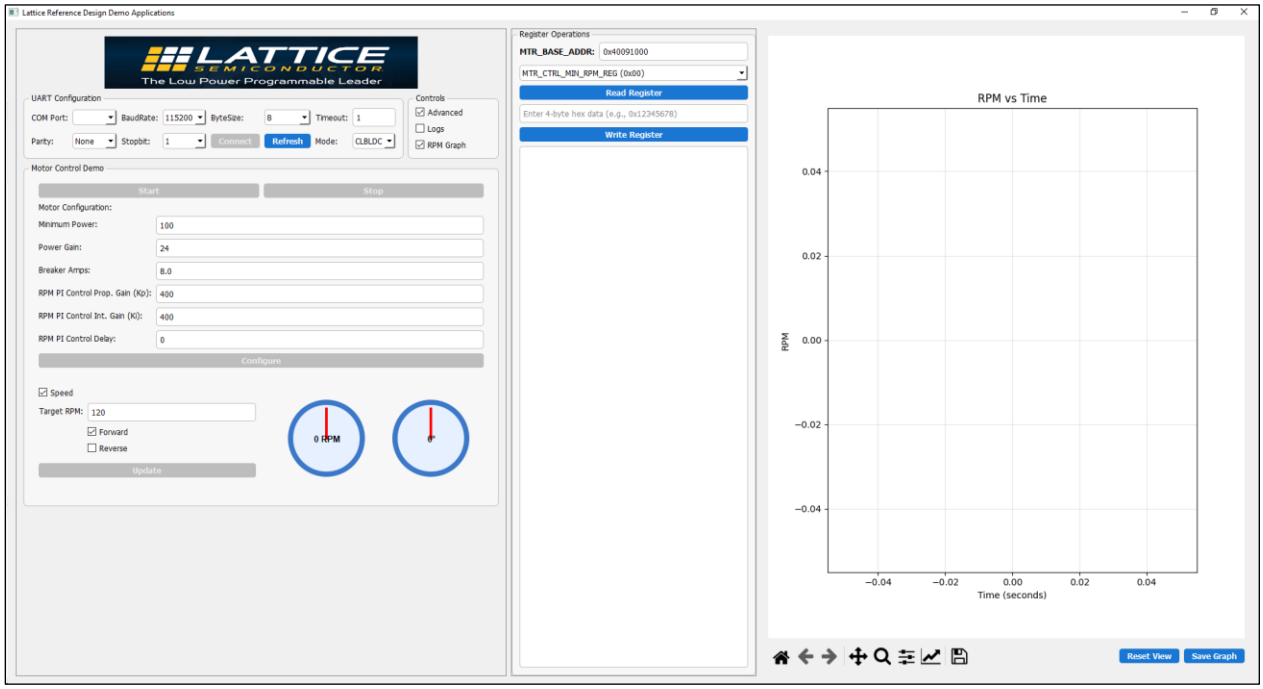

Figure 3.15. Demo GUI Application

4. Signal Description

Table 4.1. Primary I/O Interface Signals for the multi_encoder_demo_top Module

Port Name	I/O	Width	Description
clk_100	In	1	Input reference clock to PLL for system clock generation. The clock frequency is 100 MHz.
rstn_i	In	1	System reset. Active low asynchronous reset.
rxd_i	In	1	UART serial data input from the communication link.
txd_o	Out	1	UART serial data output to the communication link.
motor_control_closeloop_en	In	1	Controls input from DIP switch (SW4) on board to the motor control block. 0 – Open loop control (non-feedback) 1 – Closed loop control (real-time position feedback) **Note:** You must power cycle the FPGA device after changing the value.
encoder_interface	In	1	Controls input from DIP switch (SW3) on board to select the encoder interface. 0 – 2-wire RS485 serial interface (for HIPERFACE DSL encoder) 1 – 4-wire RS485 serial interface (for EnDat or BiSS-C encoder)
encoder_id_in	In	2	Controls input from DIP switches (SW1, SW2) on board to select the encoder type. 2'b00 – HIPERFACE DSL encoder 2'b01 – EnDat encoder 2'b10 – BiSS-C encoder 2'b11 – Reserved SW1 – bit [1] SW2 - bit [0]
encoder_datain_2	In	1	Serial data input for the HIPERFACE DSL encoder via the PMOD2 connector.
encoder_de_2	Out	1	Serial data enable output for the HIPERFACE DSL encoder via the PMOD2 connector.
encoder_datain_4	In	1	Serial data input for the EnDat encoder or BiSS-C encoder via the PMOD2 connector.
encoder_de_4	Out	1	Serial data enable output for the EnDat encoder via the PMOD2 connector.
encoder_dataout	Out	1	Serial data output for the EnDat encoder or BiSS-C encoder via the PMOD2 connector.
encoder_clk	Out	1	Serial clock output for the EnDat encoder or BiSS-C encoder via the PMOD2 connector.
encoder_clk_in	In	1	Serial clock input for the EnDat encoder via the PMOD2 connector. **Note:** In this release, this signal is not used.
encoder_clk_dir	Out	1	Serial clock direction output for the EnDat encoder via the PMOD2 connector. 0 – Input 1 – Output **Note:** Only the value of 1 is supported.
motor_control_pmod0_io	In/Out	8	ADC controls the input and output from/to the motor driver board to/from the motor control block via the PMOD0 connector. [0]: ADCs analog input channel select [1]: ADCs SPI clock [2]: ADCs SPI chip select [3]: Sensor fault from the motor driver board [4]: ADC channel 0 SPI data output [5]: ADC channel 1 SPI data output [6]: ADC channel 2 SPI data output [7]: ADC channel 3 SPI data output
motor_control_pmod1_o	Out	8	PWM controls the output from the motor control block to the motor driver board via the PMOD1 connector.

Port Name	I/O	Width	Description
			Three-phase PWM channel signals [0]: PWM channel A/U [1]: PWM channel B/V [2]: PWM channel C/W [3]: Reserved Three-phase PWM channel enable signals [4]: PWM channel enable A/U [5]: PWM channel enable B/V [6]: PWM channel enable C/W [7]: Reserved
motor_control_seven_seg_o	Out	8	LED output from the motor control block to the 7-Segment LED of the Certus-NX Versa evaluation board. [0]: Segment A [1]: Segment B [2]: Segment C [3]: Segment D [4]: Segment E [5]: Segment F [6]: Segment G [7]: Segment DP

5. Running the Reference Design

This section describes how to run the closed loop BLDC motion control reference design using the following Lattice software:

- Lattice Radiant software 2024.2
- Lattice Propel Builder software 2024.2
- Lattice Propel SDK software 2024.2
- Questa advanced simulator

5.1. Opening the Reference Design in the Lattice Propel Builder Software

To open the reference design using the Lattice Propel Builder software, follow these steps:

1. Open the Lattice Propel Builder software, as shown in the figure below.

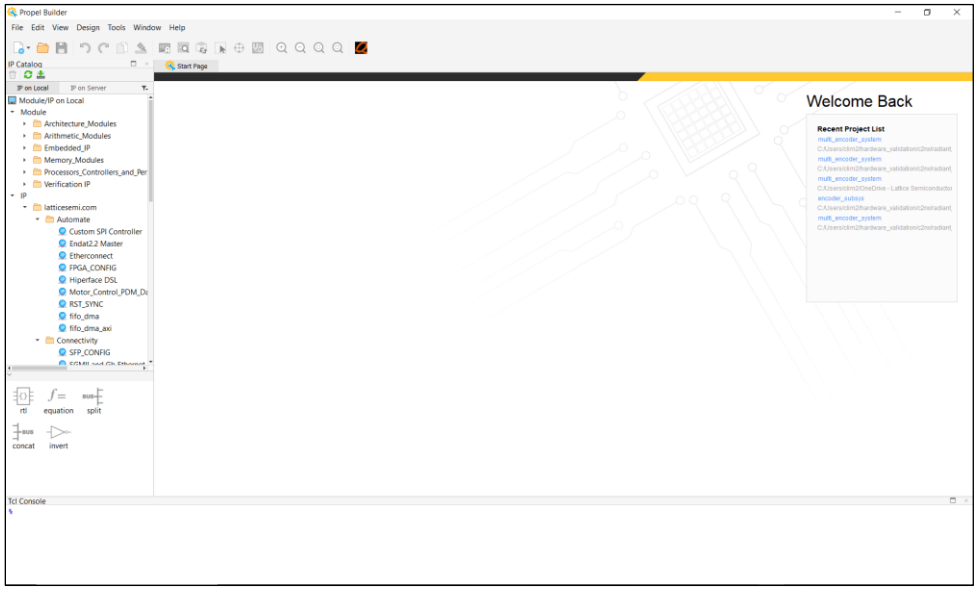

Figure 5.1. Lattice Propel Builder Software

2. Click **File > Open Design** and from the project database, open the Propel Builder Sbx file (*.sbx*) from the *<local directory>/Close-Loop-BLDC-Motion-Control-RD-Source-XXXXX/hardware/multi_encoder_system/multi_encoder_system* folder, as shown in the figure below. You may navigate to **Schematic** or **Address** tab to look at the connections of the overall design system and the base address for the corresponding target and controller.

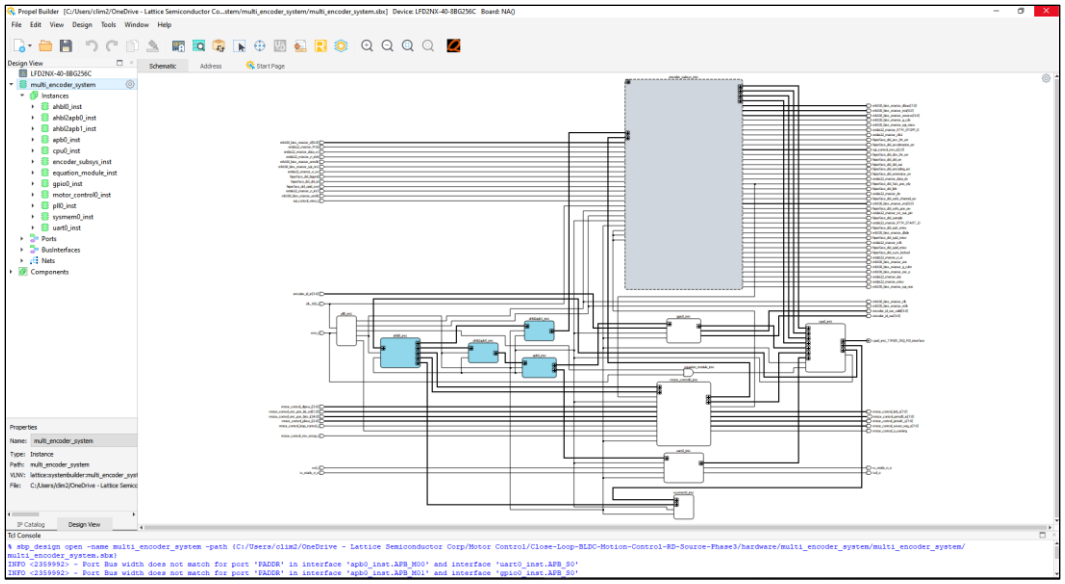

Figure 5.2. Opening the Design in the Lattice Propel Builder Software

5.2. Compiling the Reference Design in the Lattice Radiant Software

To compile the reference design using the Lattice Radiant software, follow these steps:

1. Open the Lattice Radiant software, as shown in the figure below.

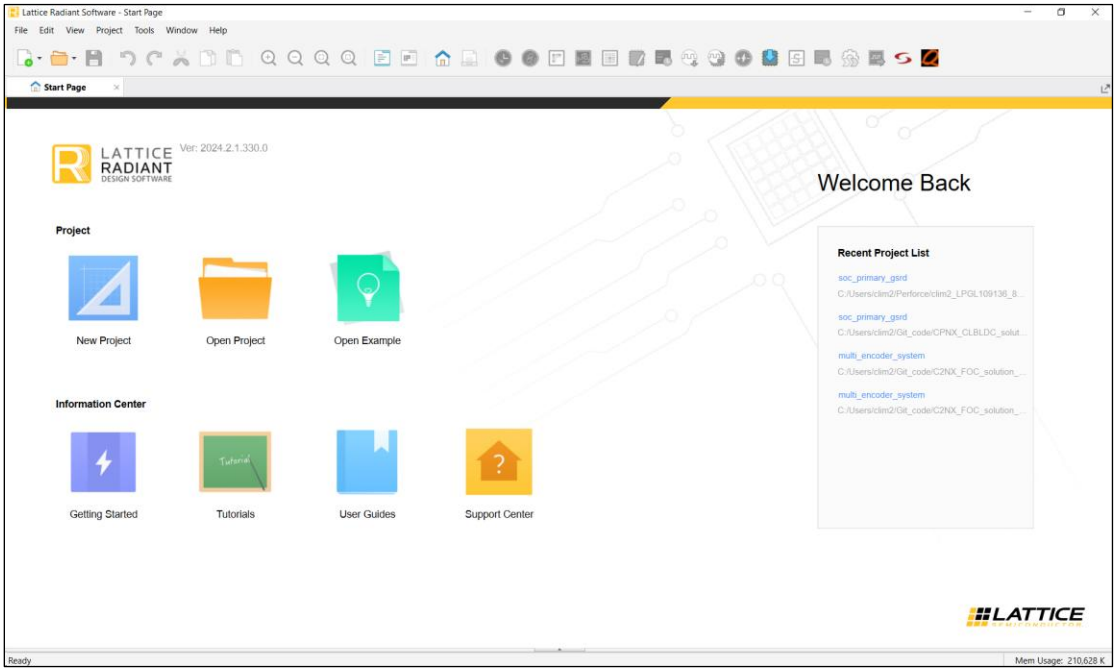

Figure 5.3. Lattice Radiant Software

2. Click **File > Open Project** and from the project database, open the Radiant project file (*.rdf*) from the *<local directory>/Close-Loop-BLDC-Motion-Control-RD-Source-XXXXX/hardware/multi_encoder_system* folder, as shown in the figure below.

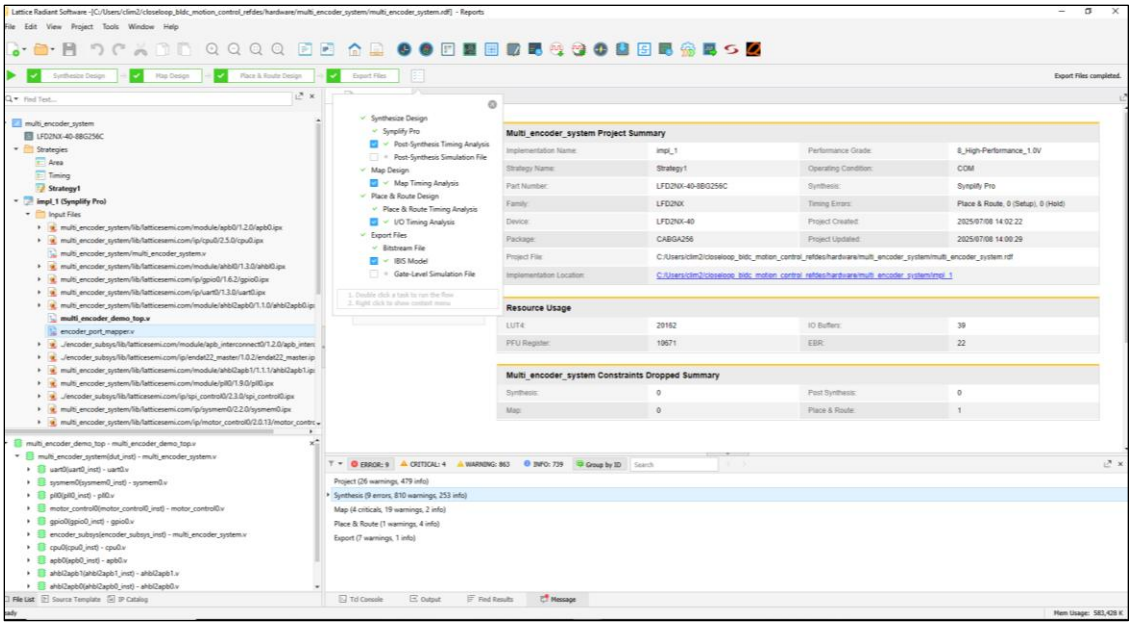

Figure 5.4. Opening the Project File in the Lattice Radiant Software

3. Click **Export Files** to initiate full compilation to generate the bitstream file with the following options selected:
   - Post-Synthesis Timing Analysis
   - Map Timing Analysis
   - I/O Timing Analysis
   - IBIS Model

   The bitstream file is generated in this path:

   <localdirectory>/Close-Loop-BLDC-Motion-Control-RD-Source-XXXXX/hardware/multi_encoder_system/impl_1/multi_encoder_system_impl_1.bit

Note: The bitstream file is pre-generated in this release, thus, compilation steps are optional. The error messages reported from the full compilation are expected because of some known issues in the Lattice Radiant software and will be fixed in future releases. Ignore the error messages as the compilation is not impacted.

5.3. Programming the Bitstream into the FPGA Device

To program the generated bitstream into the FPGA device using the Lattice Radiant software, follow these steps:

1. Click **Tools > Programmer**, the Lattice Radiant Programmer window opens as shown in the figure below.

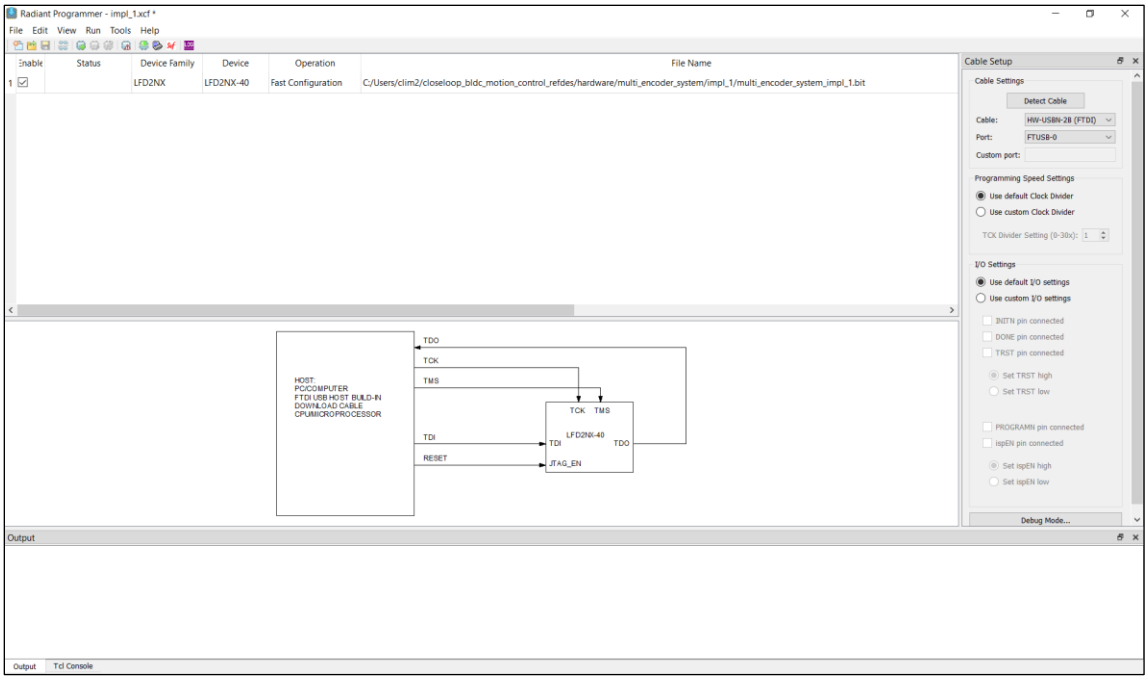

Figure 5.5. Lattice Radiant Programmer

2. Click **Detect Cable** to ensure the FPGA device is connected and detected properly via JTAG.
3. Click **Program Device**. The output console displays the operation successful message when the bitstream is programmed successfully as shown in the figure below.

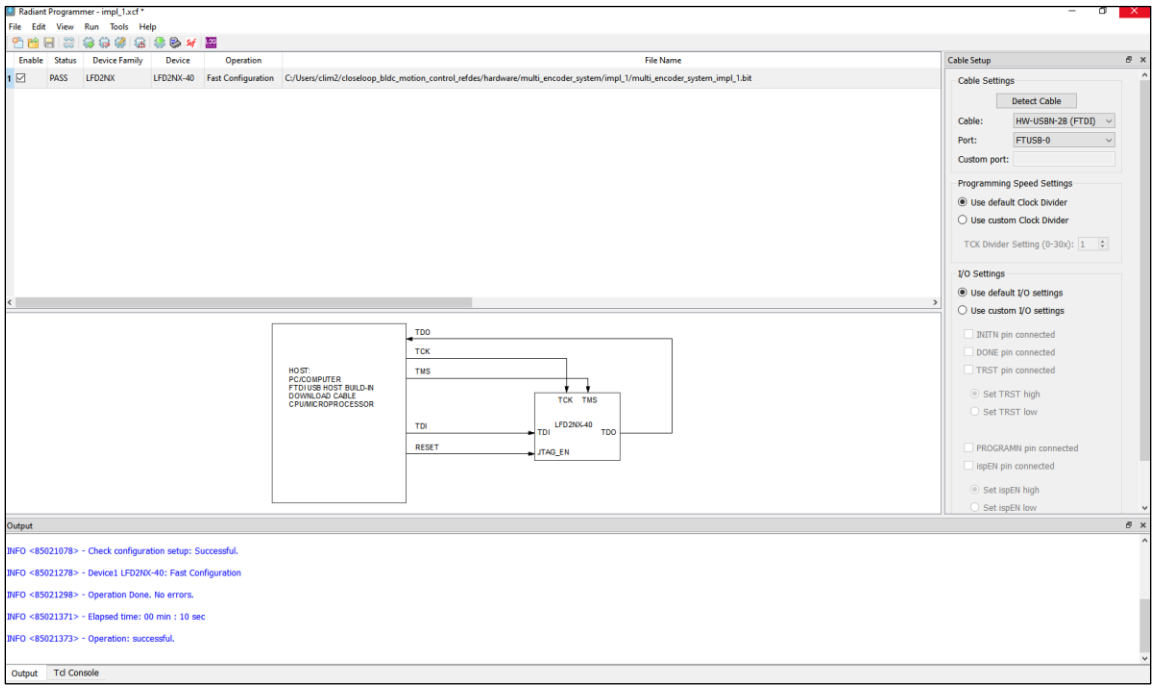

Figure 5.6. Successful Bitstream Programming

Note: If you encounter error during bitstream programming, follow these steps:

1. Disconnect the HW-USBN-2B cable from PC.
2. Power cycle the board (switch off and on).

5.4. Simulating the Reference Design

To simulate the reference design using the QuestaSim™ simulator, follow these steps:

1. In the Lattice Propel Builder software, click the QuestaSim simulation tool icon to open the simulator software, as shown in the figure below.

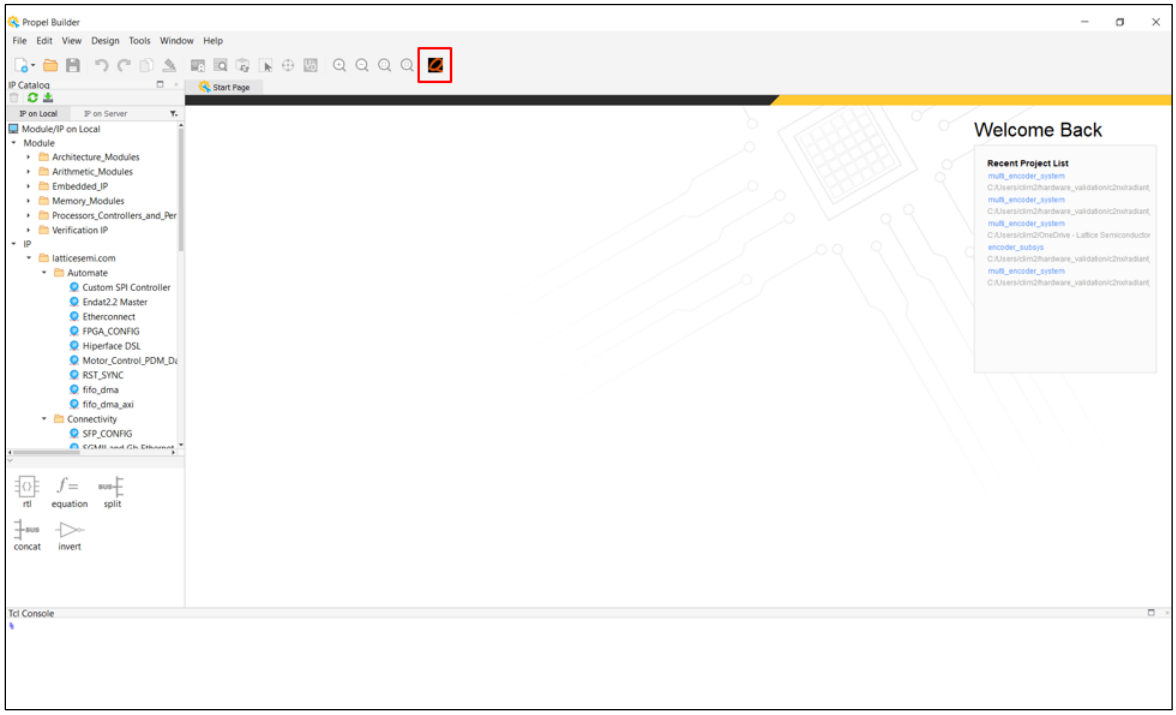

Figure 5.7. Opening the QuestaSim Software in the Lattice Propel Builder Software

2. Click **File > Change Directory** to the *<localdirectory>/Close-Loop-BLDC-Motion-Control-RD-Source-XXXXX/simulation/*.
   Note: Prior to performing the reference design simulation, you must manually map the simulation library to your local directory where the Lattice Propel software version 2024.1 is installed. Navigate to *<localdirectory>/Close-Loop-BLDC-Motion-Control-RD-Source-XXXXX/simulation/* and edit line 10 from the *msim.do* file to the correct directory path: *vmap pmi_work_r {<localdirectory>/lscc/propel/2024.1/questasim/libs/pmi_work_r}*.
3. To start the simulation, click **File > Load > Macro File**, select the *msim.do* file, and click **OK**.
   Note: The HIPERFACE DSL Master IP is not available in simulation. For CertusPro-NX devices, only basic initialization sequence is simulated to shorten simulation time.

The simulation demonstrates the following initialization sequence of this reference design system:

1. The lock_o signal is asserted from the IOPLL to indicate lock to the reference clock input.
2. The rst_n_i input is asserted high to RISC-V MC CPU.
3. The system_resetn_o is asserted high to indicate the RISC-V MC CPU is up and running.
4. The RISC-V MC CPU instruction controller fetches the software application from the system memory.
5. The RISC-V MC CPU data controller initializes all peripherals in the reference design system.

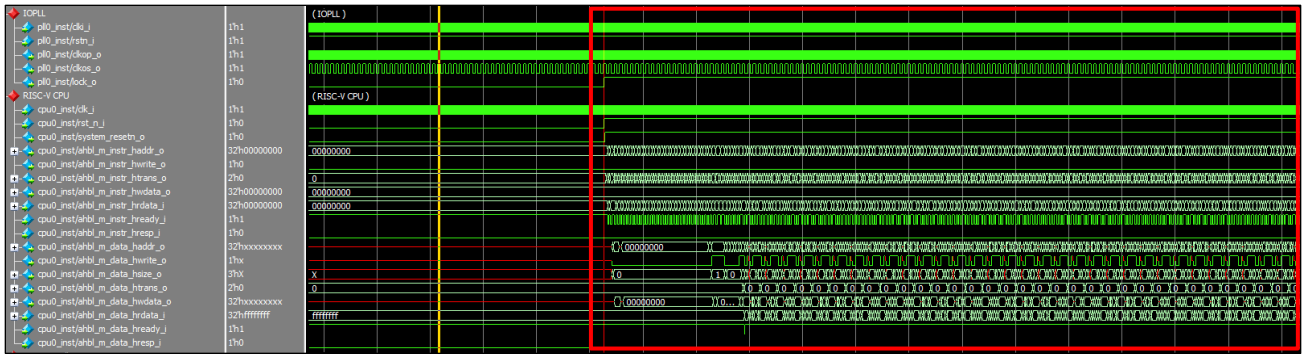

Figure 5.8. RISC-V MC CPU Out of Reset

6. The RISC-V MC CPU data controller performs the following transactions via the SPI controller block to the encoder IP selected based on the encoder_id_in input:

   For EnDat2.2 Master IP:

   Initiates read transaction to the EnDat2.2 Master IP to retrieve ID register information in the following sequence:

   a. Transmitted header and address bytes via Master Out Slave In (MOSI) line while disabling data reception on Master In Slave Out (MISO) line using the SPI controller (highlighted in the red grid in the figure below).

   b. Transmitted dummy bytes via MOSI line and enables data reception on MISO line to receive the read response using the SPI controller (highlighted in the yellow grid in the figure below).

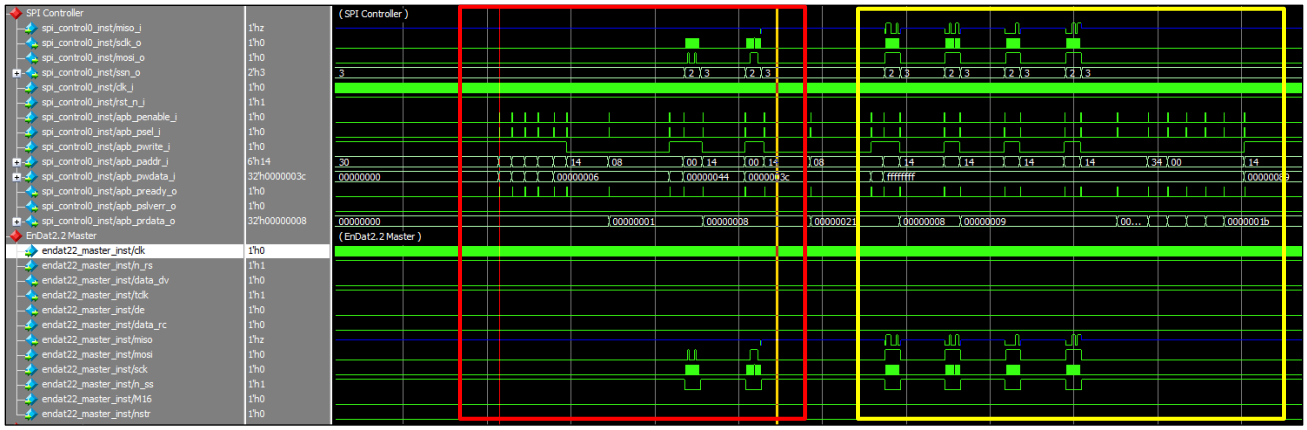

Figure 5.9. EnDat2.2 Master SPI Transaction

For MB100 BiSS Interface Master IP:

a. Initiates read transaction to the MB100 BiSS Interface Master IP to retrieve ID and version register information.

b. Initiates write transaction to the MB100 BiSS Interface Master IP to execute the BREAK sequence to the BiSS simulated receiver model.

c. Initiates multiple write transactions to the MB100 BiSS Interface Master IP to configure settings such as MA clock frequency, REGVERS, slave ID, SCD length, and CRC length.

d. Initiates write transaction to the MB100 BiSS Interface Master IP to execute the INIT sequence to the BiSS simulated receiver model.

e. Initiates write transaction to the MB100 BiSS Interface Master IP to perform register communication with the BiSS simulated receiver model to retrieve device register information.

f. Initiates write transaction to the MB100 BiSS Interface Master IP to retrieve SCD information from BiSS simulated receiver model.

g. Initiates write transaction to the MB100 BiSS Interface Master IP to execute the BREAK sequence to the BISS simulated receiver model.

h. Initiates write transaction to the MB100 BiSS Interface Master IP to switch to hardware strobe mode (PWM SYNC).

i. Initiates write transaction to the MB100 BiSS Interface Master IP to execute the INIT sequence to the BISS model.

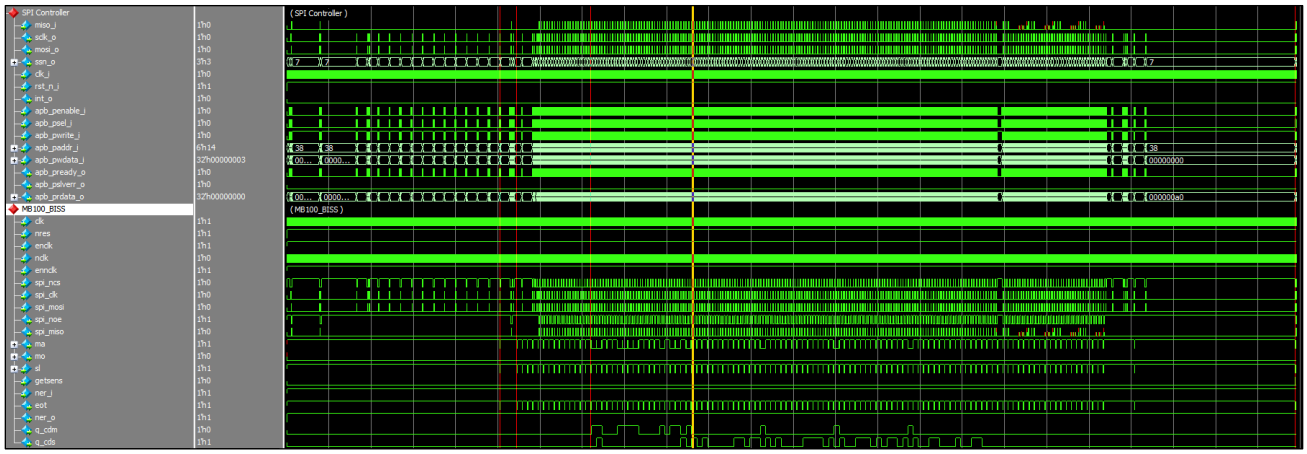

Figure 5.10. MB100 BiSS Transaction

The following figure shows example of read transaction from the SPI Controller IP to the MB100 BiSS Interface Master IP in the following sequence:

a. Transmitted header and address bytes via MOSI line while disabling data reception on MISO line using the SPI Controller IP (highlighted in the orange grid). The cpha bit is set to 0x0 in the SPI Controller IP to shift out the data on the MOSI line at the falling edge of SCLK onward.

b. Transmitted dummy bytes via MOSI line and enables data reception on MISO line to receive the read response using the SPI controller (highlighted in the blue grid). The cpha bit is set to 0x1 in the SPI Controller IP to sample the data on the MISO line at the rising edge from the second clock transition of SCLK onward.

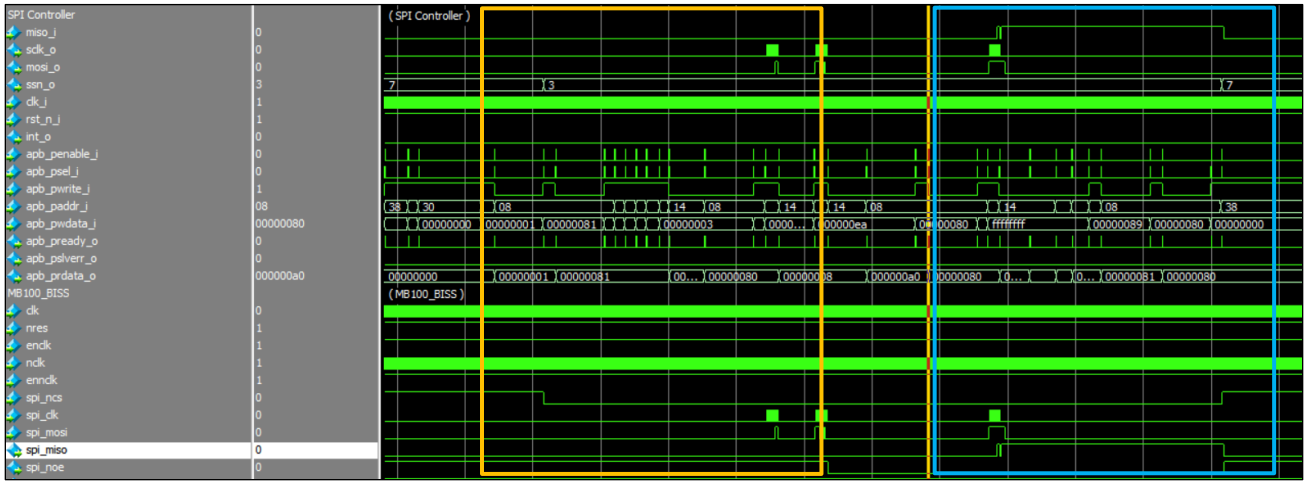

Figure 5.11. MB100 BiSS Transaction (Zoom In)

7. The RISC-V MC CPU data controller initiates write transactions to the motor control block to update configuration register setting.

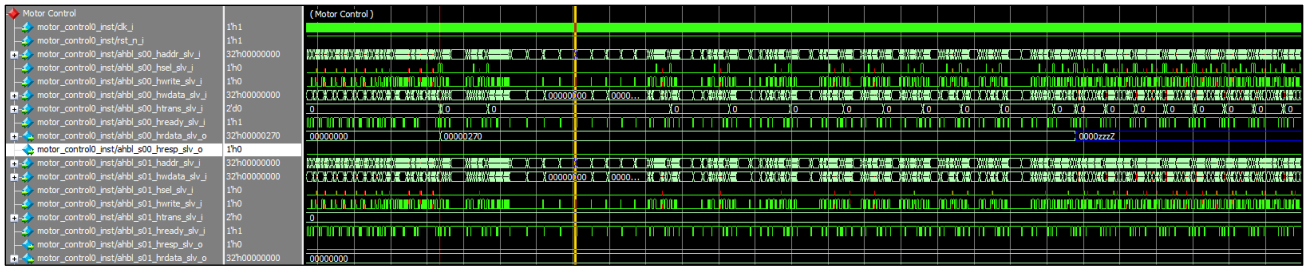

Figure 5.12. Motor Control Configuration Register Updates

The following figure shows example of write transaction from RISC-V CPU targeting the address at 32'h00011818 to the Motor Control IP.

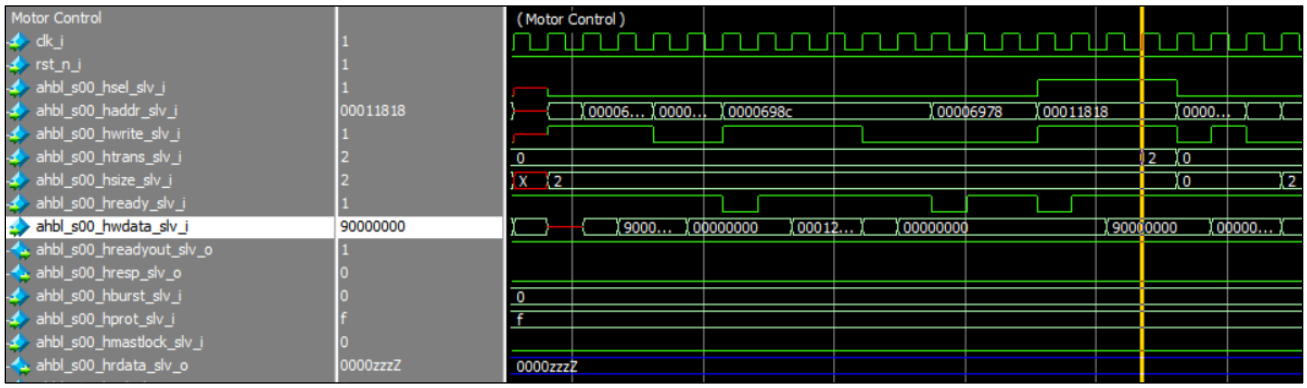

Figure 5.13. Motor Control Configuration Register Updates (Zoom In)

8. The RISC-V MC CPU data controller initiates write transactions to the GPIO block to indicate the completion of the initialization sequence.

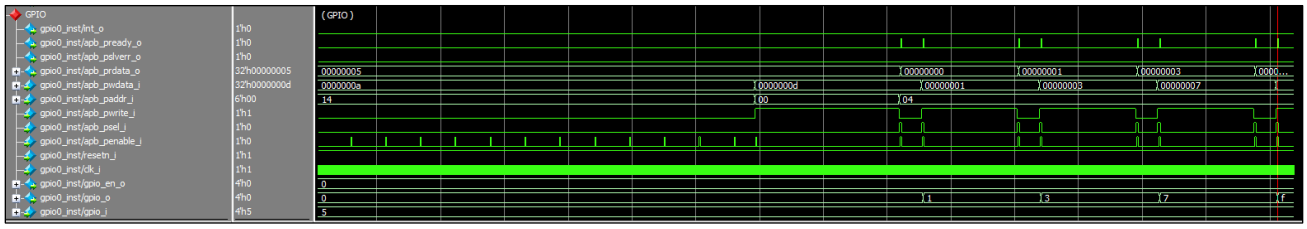

Figure 5.14. GPIO Output Updates

6. Implementing the Reference Design on Board

This section describes the steps to run the closed loop BLDC motion control reference design using the prebuilt bitstream file and the demo GUI application in the design package.

The following Lattice development boards are used to run the design:

- Certus-NX Versa evaluation board
- CertusPro-NX evaluation board

6.1. Extracting the Reference Design Files

Before running the reference design, download the *Close-Loop-BLDC-Motion-Control-RD-Source-XXXXX.zip* file from the reference design web page in the Quick Facts section.

Unzip the *.zip* file to your local directory. For example, unzip to *C:/workspace*.

6.2. Setting Up the Evaluation Board

This section provides the steps to set up the Certus-NX Versa evaluation board or CertusPro-NX evaluation board, shown in the figures below.

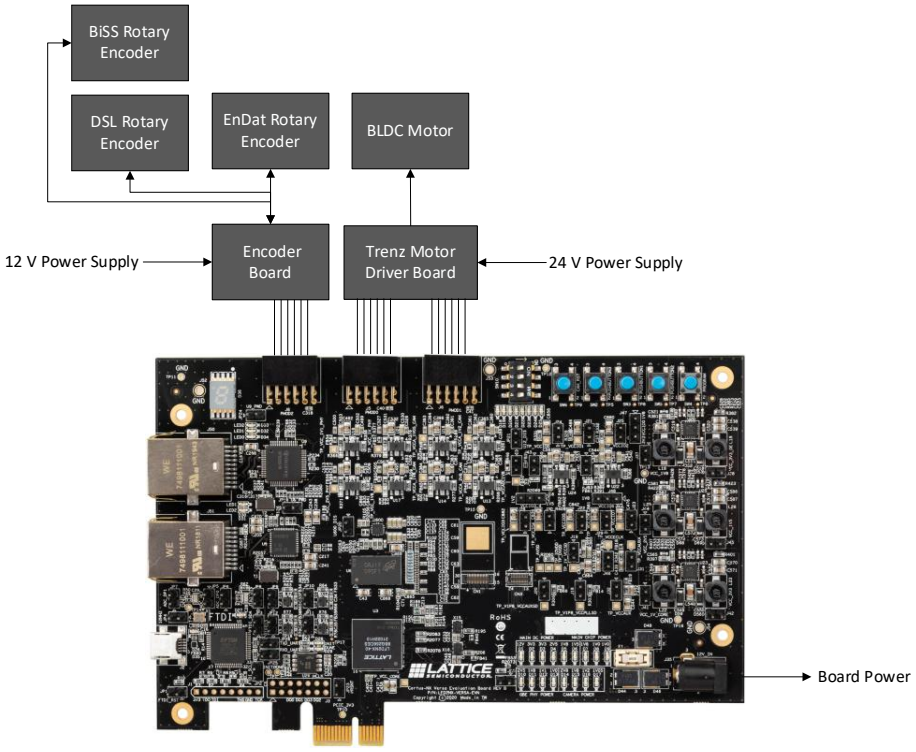

Figure 6.1. Certus-NX Versa Evaluation Board Setup

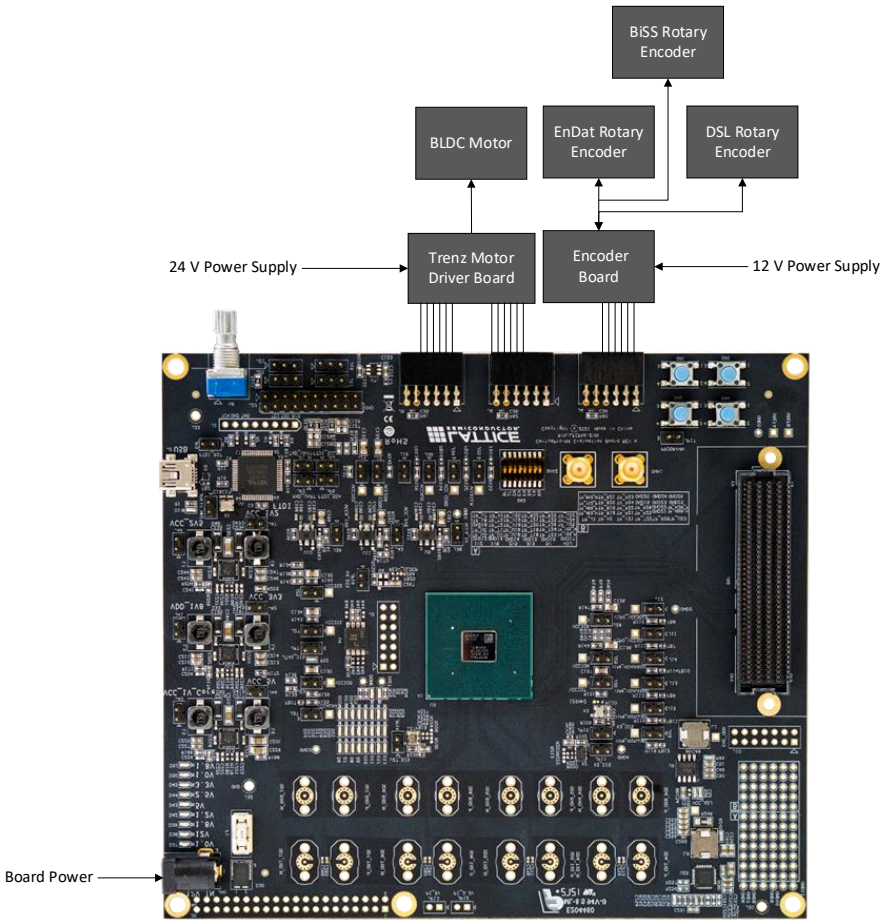

Figure 6.2. CertusPro-NX Evaluation Board Setup

To set up the evaluation board, follow these steps:

1. Connect the 12 V power adaptor to J35.
2. Connect the 24 V with 10 A power supply to the Trenz motor driver board.
3. Connect the Trenz motor driver board to the PMOD connectors based on the corresponding evaluation board:
   - Certus-NX Versa: PMOD0 and PMOD1 connectors
   - CertusPro-NX: PMOD0 and PMOD2 connectors
4. Connect the encoder board to the PMOD connector on the corresponding evaluation board:
   - Certus-NX Versa: PMOD2 connector
   - CertusPro-NX: PMOD1 connector
5. Connect the wires from BLDC motor to the Trenz motor driver board according to the Anaheim and Trenz data sheets.
6. Connect the mini USB Type A cable from the PC to J2.
7. Connect (using jumpers) pin 1 and pin 2 on both J25 and J26 to enable UART.

Note: The noise caused by mechanical vibration when the motor is moving may impact the accuracy of encoder position detection. In this reference design, both BLDC motor and EnDat encoder are installed on a solid fixture to stabilize both BLDC motor and encoder to reduce the vibration noise when the motor is moving.

6.3. Setting Up the Encoder Board

The encoder board consists of RS485 transceivers that are used for data transmission between the Certus-NX Versa evaluation board or the CertusPro-NX evaluation board with the respective encoder (EnDat, HIPERFACE DS, or BiSS-C).

For encoder with 2-wire interface implementation, the terminal J6 is used to connect with the encoder.

For encoder with 4-wire or 6-wire interface implementation, the terminals J2–J4 are used to connect with the encoder.

To set up the encoder board with the respective encoder (EnDat, HIPERFACE DS, or BiSS-C), follow these steps:

1. Connect the 12 V with 3 A power supply to terminal J5.
2. For the EnDat encoder, you need to connect the wires to each of the following terminal based on the encoder data sheet from the Heidenhain web page:
   - J2—Differential clock signals
   - J3—Differential data signals
   - J4—Power signals
3. For the HIPERFACE DSL encoder, you need to connect the wires to the following terminal based on the encoder data sheet from the SICK AG web page:
   - J6—Differential data signals
4. For BiSS-C encoder, you need to connect the wires to each of the following terminal based on the encoder data sheet from the Hengstler web page:
   - J2—Differential clock signals
   - J3—Differential data signals
   - J4—Power signals

For more details, refer to the RS485 Encoder Transceiver Board web page.

6.4. Setting Up the UART Terminal

The software code in this reference design displays messages on the terminal through the UART interface.

There are two ways to set up the UART terminal as follows:

- Terminal emulator software such as Tera Term and PuTTY. This software is useful for debugging purposes only.
- Demo GUI application provided in this reference design.

6.4.1. Setting Up the Tera Term Software

To set up the Tera Term software, follow these steps:

1. Launch the Tera Term software.
2. In the New connection dialog box, select **Serial**.
3. Select the COM port from the **Port** drop-down menu. The correct COM port is the larger number from the drop-down menu. For example, if you see COM17 and COM18, select **COM18** and click **OK**.

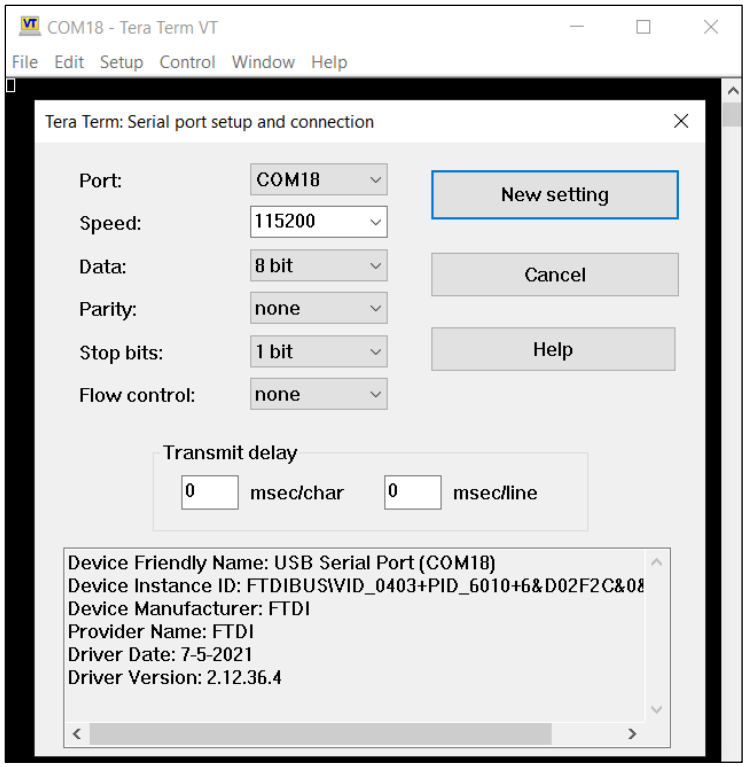

Figure 6.3. Tera Term Software Setup

4. Click **Setup > Serial Port**.
5. In the Serial port setup and connection interface, select **Speed: 115200**.
6. Select the rest of the options based on the settings in Figure 6.3.
7. Click **New setting** to complete the process.

6.4.2. Setting Up the Demo GUI Application

Program the bitstream design into the Certus-NX Versa evaluation board or CertusPro-NX evaluation board. For bitstream programming to the FPGA device, refer to the Programming the Bitstream into the FPGA Device section.

To set up the demo GUI application, follow these steps:

1. Launch the *closeloop_bldc_motion_control_gui.exe* described in the Directory Structure and Files section.
2. Select **Mode: CLBLDC**.
3. Select **BaudRate: 115200**.
4. Select **ByteSize: 8**.
5. Click **Connect** to establish the UART connection.
6. Optionally, select **Advanced**, **Logs**, and **RPM Graph** to enable the corresponding tabs in the GUI application.

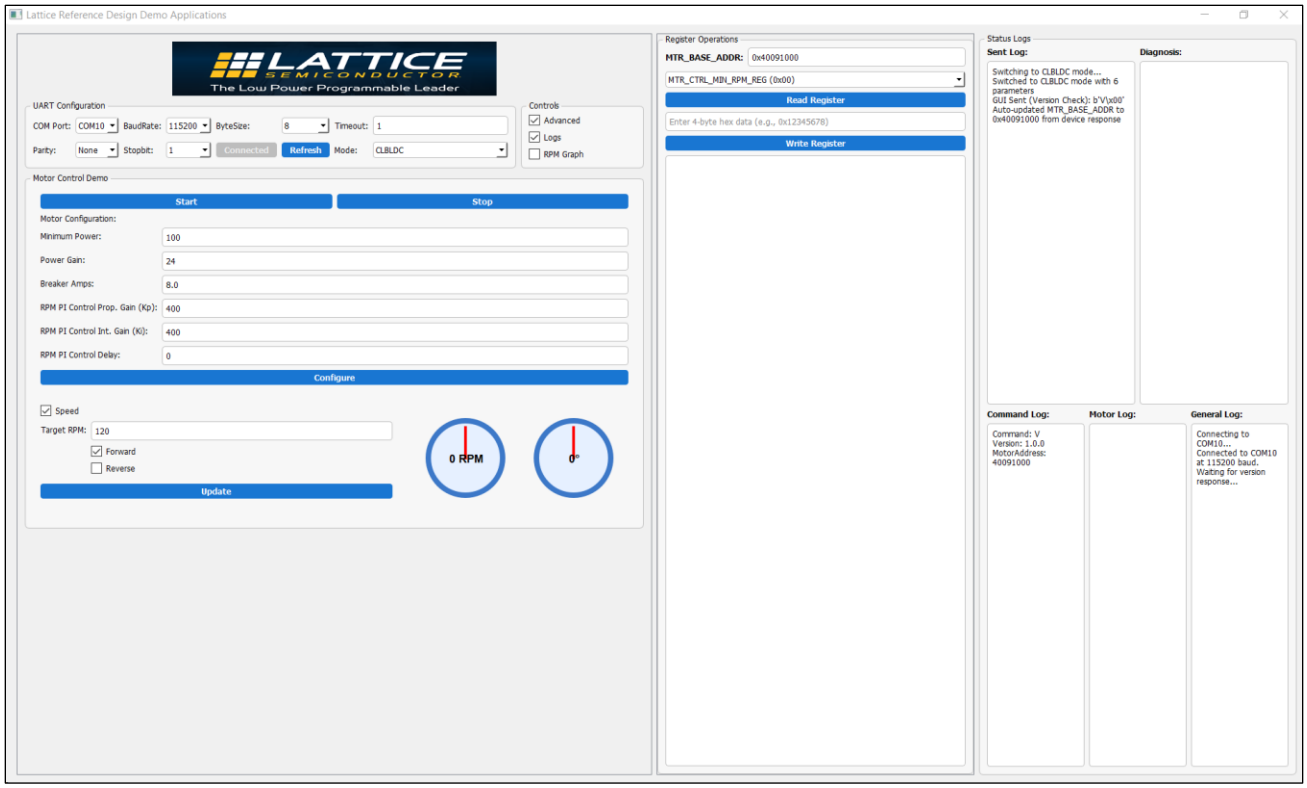

Figure 6.4. Demo GUI Application UART Test Tab Setup

6.5. Running the Motor from Demo GUI Application

To configure and run the motor, follow these steps:

1. Click **Configure** to initiate the control setting.
2. Click **Start** to enable and run the motor. The motor runs up to the speed as specified in **Target RPM**.
   Note: The Motor Control IP is default to startup at 120 RPM. You may specify any value in **Target RPM**, not exceeding 2,000 RPM for closed loop control option. For open loop control, you must not specify the value in **Target RPM** exceeding 1,600 RPM. You must increase or decrease the **Target RPM** gradually. For example: 400 RPM, 800 RPM, 1200 RPM, 1600 RPM.
3. Optionally, you may perform the following options via the demo GUI application:
   - Click **Stop** to stop the motor.
   - Specify the desired RPM value in **Target RPM** and click **Update** to accelerate or decelerate the motor.
   - Select **Forward** and click **Update** to run the motor in clockwise direction.
   - Select **Reverse** and click **Update** to run the motor in counterclockwise direction.

   Note: Selecting **Forward** or **Reverse** does not have any effect if the motor is already running in the specified direction.
4. The RPM versus time chart is plotted continuously as long as the motor is running.

The configuration settings in this reference design are optimized only for the specific BLDC motor model listed in the Quick Facts section. Settings may differ for other BLDC motor models.

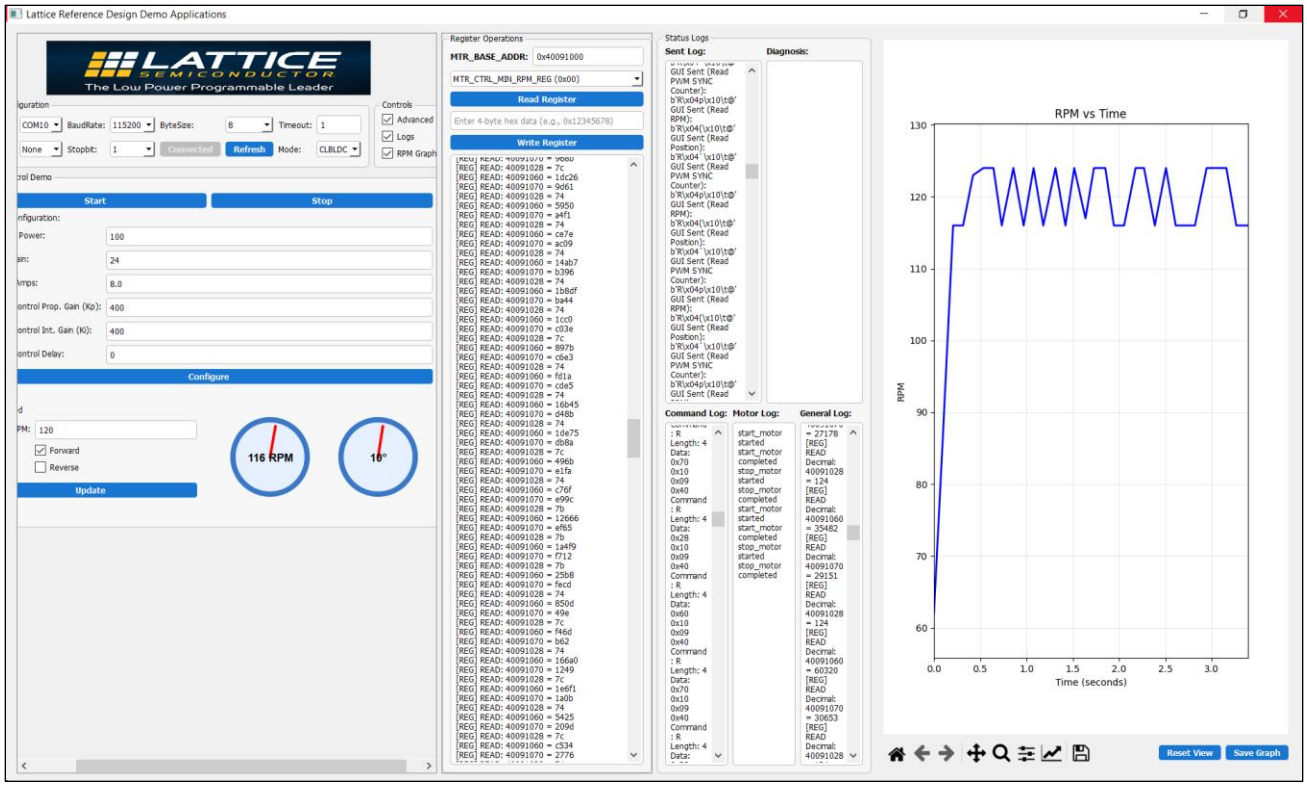

Figure 6.5. Demo GUI Application Motor Control Demo Tab

7. Resource Utilization

The resource utilization for Certus-NX devices is shown in the table below.
Note: The values are only for reference and actual usage may vary.

Table 7.1. Resource Utilization for Certus-NX Devices

Configuration	LUTs (Utilization/Total)	Registers (Utilization/Total)	EBRs (Utilization/Total)	I/O (Utilization/Total)	DSP (Utilization/Total)
Closed loop BLDC motion control reference design targeting Certus-NX devices	20,162/32,256	10,680/32,811	22/84	46/185	35/56
Closed loop BLDC motion control reference design targeting CertusPro-NX devices	24,872/79,872	14,270/80,769	100/208	35/299	35/156

8. Debugging

This section lists possible issues and suggested troubleshooting steps that you can follow.

8.1. Motor Unable to Run After Configured in Either Closed Loop or Open Loop Control

The motor is not able to run after configured in either closed loop or open loop control system.

To resolve this issue, follow these guidelines:

- Make sure to use the power supply with the correct voltage and current as listed in the Quick Facts section.
- Make sure the driver fault is not asserted by motor driver board if an overcurrent is detected.

8.2. Motor Unable to Accelerate or Decelerate Correctly for Closed Loop Control

The motor is unable to achieve the specified target RPM when running in the closed loop control system.

To resolve this issue, follow these guidelines:

- Make sure the coupling is installed properly and tightly screwed between the motor and the encoder. You may run the design with the open loop control system to verify the installation.
- Make sure the encoder is initialized correctly, and the encoder position is updated continuously when the motor is running.

8.3. Proportional and Integration Constant Tuning

In this reference design, the proportional (K_p) and integration (K_i) constant values used in the PI controller within the Motor Control IP are applicable to the specified Anaheim motor model only. For Anaheim motor, customized motor for HIPERFACE DSL, and other motor models, you may change and test the different combinations of K_p and K_i values to achieve the desired performance in the speed PI controller.

When tuning with K_p and K_i constant values, it is recommended to vary only one of the values at a time to verify the impacts on the motor control system.

In this reference design, the K_p and K_i constant values used are 300 each. You may use the values as a reference point to increase or decrease the K_p or K_i values accordingly based on the different motor model. For example, you may increase or decrease K_p or K_i values by 50 for each attempt on only one parameter at a time.

The rpm_ki_gain (RPMINTK) and rpm_kp_gain (RPMPRPK) variables are defined under the ANAHEIM define section at the top section in the *motor.c* file. Refer to the Software Project Compilation section to recompile the software after modifying the values.

The table below describes the key metrics to be evaluated in the performance of PI controller in a motor control system.

Table 8.1. K_p and K_i Gain Effects in Closed Loop System

Metrics	Definition	Gain Effects	
		Kp	Ki
Rise Time	The time taken for the motor speed or position to go from 0% to 90% of the desired setpoint.	Higher value generally decreases the rise time, making the system respond faster.	Higher value can reduce the rising time as an integral action to accumulate error faster and apply corrective action more aggressively.
Overshoot	The extent to which the motor speed or position exceeds the desired setpoint before settling.	Higher value can increase overshoot because the system reacts more aggressively to errors.	Reduces overshoot by correcting accumulated errors, but if the value is too high, the value can also contribute to overshoot.

Metrics	Definition	Gain Effects	
		Kp	Ki
Settling Time	The time taken for the motor speed or position to remain within a certain percentage (usually 2% or 5%) of the desired setpoint.	Higher value can reduce settling time by making the system more responsive, but excessive Kp can cause oscillations, increasing settling time.	Proper tuned value can reduce settling time by eliminating steady-state errors, but if the value is too high, the value can cause slow convergence and instability.
Stability	The ability of the system to return to the desired setpoint without oscillating or diverging.	Higher value can make the system less stable if the value causes excessive oscillations.	Proper tuned value can improve stability by correcting steady-state errors, but if the value is too high, the value can destabilize the system.
Steady State Error	The difference between the desired setpoint and the actual motor speed or position after the system has settled.	Higher value reduces steady-state error but cannot eliminate the error entirely.	Crucial for eliminating steady-state error by integrating the error over time and making continuous adjustments.

8.4. GUI Hang, Not Responding, or Exit Abruptly

The demo GUI application may hang or non-responding after any command is selected.

To resolve this issue, follow these guidelines:

- Restart the demo GUI based on steps in the Setting Up the Demo GUI Application section and check if the issue persists.
- Make sure the UART connection is established correctly.

8.5. UART Serial Terminal Prints Incorrect Characters

Incorrect (junk) characters may be displayed on the UART serial terminal.

To resolve this issue, follow these guidelines:

- Make sure the **BaudRate** or **Speed** parameter in the terminal software is set to **115200**.
- Make sure the clock input from the on-board oscillator matches the **System Clock Frequency** value configured in the UART IP GUI from the Lattice Propel Builder software.

9. Integrating with a Different Motor

This section provides the considerations when integrating the Motor Control IP with a motor that is not listed in the Quick Facts section.

9.1. Overcurrent Fault

The Trenz motor driver board used in this reference design has built-in current sensors that flag an overcurrent fault to the Motor Control IP if the detected current exceeds the threshold value defined in the data sheet. When an overcurrent fault occurs, the Motor Control IP disables all PWM signals driving to the Trenz motor driver board to prevent board damage. As different motor models have different electrical characteristics, ensure the power source connected to the Trenz motor driver board follows the data sheet specifications.

To achieve the optimal current source for driving the motor, try with different combinations of power gains and minimum power values as described in Table 3.12 and Table 3.13. Higher power gain and minimum power values result in more current being drawn to the motor. Additionally, increasing the Target RPM value, as specified in Table 3.17, also raises the current draw. Both higher Target RPM and power parameter values lead to an increase pulse width of the PWM signals. Consequently, the half-bridge gate driver amplifies these PWM signals, driving the gates of the high-side and low-side switches more rapidly, allowing more current to flow into the motor.

If overcurrent fault is reported from the Trenz motor driver board, you may use the following guidelines to identify the root cause when working with a different motor model.

Note: When overcurrent fault occurs, you need to power cycle the entire system before attempting to perform the troubleshooting steps described in the subsequent sections.

9.1.1. Power Parameter Tuning

To perform power parameter tuning, consider the following guidelines:

- Reduce the PWRGAIN value in the register as specified in Table 3.13. For troubleshooting, start by setting the value to 0.
- Decrease the MINPWR value in the register as specified in in Table 3.12. Begin with a setting of 30 and gradually increase by 10 for each attempt until just before the overcurrent fault occurs.
- The min_power (MINPWR) and power_gain (PWRGAIN) variables are defined under the ANAHEIM define section at the top section in the *motor.c* file. Refer to the Software Project Compilation section to recompile the software after modifying the values. Alternately, these values can be configured via the **Minimum Power** and **Power Gain** settings in the GUI.

Note: You must not change the power parameter during runtime after initialization.

9.1.2. Reducing Target RPM Value

To reduce the target RPM value, consider the following guidelines:

- Reduce the Target RPM value in the register as specified in Table 3.17. Begin with RPM value of 20 and gradually increase by 20 for each attempt until just before the overcurrent fault occurs.

9.1.3. Debugging Mode from the Motor Control IP

The Motor Control IP supports an internal debugging mode to troubleshoot overcurrent faults caused by improper startup RPM settings for a different motor model.

The following formula shows the default startup theta period (angular velocity) calculation implemented in the Motor Control IP:

Theta period = **BLDC_SYS_CLOCK_FREQ** / (**RPM** × 6 × **Number of Motor Pole Pairs**)

- BLDC_SYS_CLOCK_FREQ = 20 MHz (Motor Control IP clock frequency, fixed)
- Number of Motor Pole Pairs = 4

By default, at power-up, the IP operates at theta period of 120 RPM based on the Anaheim motor with 4 pole pairs.

Theta period = 20,000,000 / (120 × 6 × 4) = 6,944

This value may need to be updated in the Motor Control IP for a different motor model with different number of motor pole pairs.

If the overcurrent fault occurs at the startup even with low Target RPM value, you may change theta period value to a lower value via software application as described in the subsequent sections.

The following subsections describe the methods to enable the internal debugging mode in the Motor Control IP.

9.1.3.1. Disable Initialization

The design system was compiled with the initialization file (*.mem*) configured in the system memory for the system to be boot up automatically when bitstream is programmed into the FPGA device.

For troubleshooting, you may disable the initialization option from the System Memory by following these steps:

1. Open the Lattice Propel Builder software.
2. Open the Sbx file (*.sbx*). For details, refer to the Opening the Reference Design in the Lattice Propel Builder Software section.
3. Double-click the **sysmem0_inst** instance to open the Module/IP Block Wizard window.
4. From Module/IP Block Wizard window, untick **Initialize Memory** and set **Initialization File** to **none**.

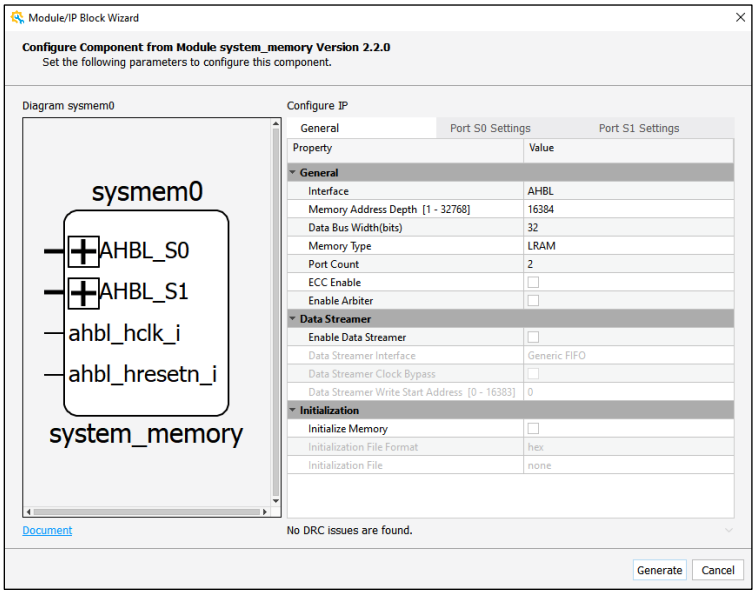

Figure 9.1. System Memory IP Block Wizard

5. Click **Generate** and **Finish**.
6. From the tab menu, click **Validate Design** to validate the design setting.
7. Click **Generate** to update the design system.
8. Click **Save** to save the updated design system.

9.1.3.2. Hardware Project Compilation

To compile the hardware project, follow these steps:

1. Open the Lattice Radiant software.
2. Open the Lattice Radiant design project. For details, refer to the Compiling the Reference Design in the Lattice Radiant Software section.
3. Navigate to the following folder:

   <localdirectory>/Close-Loop-BLDC-Motion-Control-RD-Source-XXXXX/hardware/multi_encoder_system/multi_encoder_system/lib/latticesemi.com/ip/motor_control0/2.x.xx/rtl
4. Open the *motor_control0.v* file.

5. Under the motor_control0_ipgen_motor_control_pdm_dc module, set the **INTERNAL_DEBUG** parameter to 1.
6. Save the changes in the *motor_control0.v* file.
7. Trigger a full compilation on the Lattice Radiant design project.
8. Locate the newly generated bitstream in the following folder:
   <localdirectory>/Close-Loop-BLDC-Motion-Control-RD-Source-XXXXX/hardware/multi_encoder_system/impl_1

9.1.3.3. Software Project Compilation

To compile the software project, follow these steps:

1. Open the Lattice Propel SDK software. For details, refer to the Lattice Propel 2024.2 SDK User Guide (FPGA-UG-02218).
2. From the Project Explorer window, right-click any area of the blank space and select **Import**.
3. From Import window, select **Existing Projects into Workspace** and click **Next**.
4. Click **Browse** and navigate to the following folder:
   <localdirectory>/Close-Loop-BLDC-Motion-Control-RD-Source-XXXXX/software/baremetal_test
5. To complete importing projects, click **Select Folder** and click **Finish**.
6. Navigate to the following folder:
   <localdirectory>/Close-Loop-BLDC-Motion-Control-RD-Source-XXXXX/software/baremetal_test/src/motor
7. Open the *motor.c* file.
8. At the top section in the *motor.c* file, locate the **pi_control_reset** variable declaration.
9. Modify the **pi_control_reset** variable based on the theta period formula. For details, refer to the Debugging Mode from the Motor Control IP section. For example, you may change the theta period value with a lower RPM value by decremental of 20 for each attempt, such as 100, 80, 60, 40, 20 RPM.

   For RPM **100** for motor model with **Number of Motor Pole Pairs** of **5**, the theta period is calculated as follows:

   Theta period = **BLDC_SYS_CLOCK_FREQ** / (**RPM** × 6 × **Number of Motor Pole Pairs**)

   = 20,000,000 / (100 × 6 × 5)

   = 33,333
10. Within the **motor_configure** function, locate the **motor_pi_reset_debug** function calling line.
11. Uncomment the **motor_pi_reset_debug** function calling line.
12. Save the changes in the *motor.c* file.
13. In the Project Explorer window, right-click the **baremetal_test** project.
14. Select the build project to recompile the software.
15. Locate the newly generated initialization (*.mem*) and application (*.elf*) files in the following folder:
    <localdirectory>/Close-Loop-BLDC-Motion-Control-RD-Source-XXXXX/ software/baremetal_test/Debug

9.1.3.4. Bitstream and Application Programming

To program the bitstream and application, follow these steps:

1. Program the new bitstream into the FPGA device. For details, refer to the Programming the Bitstream into the FPGA Device section.
2. Load and test the application generated from the Software Project Compilation section. For details, refer to the *Programming and On-Chip-Debugging Flow* chapter in the Lattice Propel 2024.2 SDK User Guide (FPGA-UG-02218).

When you have completed all steps in the following sections, restart the motor with the new application to verify the changes.

- Disable Initialization
- Hardware Project Compilation
- Software Project Compilation
- Bitstream and Application Programming

To test using lower **pi_control_reset** values until the overcurrent fault does not occur, you may need to repeat step 9–15 from the Software Project Compilation section and step 1–2 from the Bitstream and Application Programming section. You need to power cycle the entire system before rerunning these steps if an overcurrent fault occurs.

10. Integrating with a Different Encoder

This section provides the considerations when integrating with a different encoder model that is not listed in the Quick Facts section.

10.1. Encoder Resolutions

The absolute encoders used in this reference design have the following resolution bits:

- EnDat encoder—Width of 25 single-turn resolution bits
- HIPERFACE DSL encoder—Width of 20 single-turn resolution bits
- BiSS-C encoder—Width of 17 single-turn resolution bits

If you use any identical encoder type with different resolution bits than those listed above, you must configure the following settings in the IP Block Wizard for the corresponding IP.

Table 10.1. Motor Control IP Settings for Other Encoder Models

Attributes	Value
ENC_POS_BIT_WIDTH	Single-turn resolution bit width, set according to the encoder data sheet
MULT_POS_RES_SUPPORT	0
ENC_POS_PORT_EXPORT	0
HARDWARE_FETCH	1 (If APB Encoder Position Requestor IP is not used)
ENC_STATUS_BIT_WIDTH	**Encoder Type** / **Value** EnDat or HIPERFACE DSL / 0 BiSS-C / Set according to the encoder data sheet

Table 10.2. APB Encoder Position Requestor IP Settings for Other Encoder Models

Attributes	Value
ENC_POS_BIT_WIDTH	Single-turn resolution bit width, set according to the encoder data sheet
MULT_POS_RES_SUPPORT	0
ENC_STATUS_BIT_WIDTH	**Encoder Type** / **Value** EnDat or HIPERFACE DSL / 0 BiSS-C / Set according to the encoder data sheet

Additionally, you must update the data length value in the corresponding encoder driver to match the resolution bits of the encoder model used.

For EnDat2.2 encoder driver:

Locate the **data_word_length** variable in the init_endat22_master function and modify according to the resolution bits of the encoder model.

For MB100 BiSS encoder driver:

Locate the **scdlen** variable in the config_biss_master_point_to_point_connection function and modify according to the data bits of the encoder model.

Refer to the Software Project Compilation section to recompile the project after modifying the encoder driver.

10.2. Custom Encoder Master IPs

If your custom Master IP is supported on the SPI protocol communication, use the following steps to integrate your custom IPs into the encoder subsystem for optimized integration with your higher-level design system:

1. Navigate to the following folder:

   <localdirectory>/Close-Loop-BLDC-Motion-Control-RD-Source-XXXXX/hardware/encoder_subsys

2. Copy the entire *encoder_subsys* folder to your desired directory.
3. Open the Lattice Propel Builder software.
4. Navigate to the following folder:
   <localdirectory>/encoder_subsys/encoder_subsys
5. Open the Sbx file (*.sbx*). For details, refer to the Opening the Reference Design in the Lattice Propel Builder Software section.
6. Instantiate your custom Master IP.
7. Connect the SPI interface with the SPI controller block.
8. Update the setting of SPI Controller IP in the Module/IP Block Wizard window if required.
9. Click **Validate Design** from the tab menu to validate the design setting.
10. Click **Generate** to update the *encoder_subsys* Sbx.
11. Click **Save** to save the updated *encoder_subsys* Sbx.
12. Close the Lattice Propel Builder software.
13. Navigate to the following folder:
    <localdirectory>/Close-Loop-BLDC-Motion-Control-RD-Source-XXXXX/hardware/encoder_subsys/encoder_subsys/lib/latticesemi.com/ip/enc_apb_control/1.0.0/rtl
14. Open the *enc_apb_control.sv* file.
15. Locate the **enc_apb_control_ipgen_lsc_custom_enc_package** package module.
16. Modify parameters defined inside the package accordingly.
17. Save and close the *enc_apb_control.sv* file.
18. Add the *encoder_subsys* Sbx into your higher-level design system through the Lattice Propel Builder software.

References

- Certus-NX web page
- CertusPro-NX web page
- Certus-NX Versa Evaluation Board web page
- CertusPro-NX Evaluation Board web page
- RS485 Encoder Transceiver Board web page
- Closed Loop BLDC Motion Control Reference Design web page
- GHRD/GSRD Reference Design web page
- SPI Controller IP Core web page
- UART IP Core web page
- GPIO IP Core web page
- RISC-V MC CPU IP Core web page
- AHB-Lite Interconnect Module web page
- APB Interconnect Module web page
- AHB-Lite to APB Bridge Module web page
- Trenz Electronic web page
- Anaheim Automation web page
- Heidenhain web page
- SICK AG web page
- Hengstler web page
- iC-Haus web page
- BiSS Interface web page
- Siemens Questa Advanced Simulator web page
- Lattice Radiant Software web page
- Lattice Propel Design Environment web page
- Lattice Insights for Lattice Semiconductor training courses and learning plans

Technical Support Assistance

Submit a technical support case through www.latticesemi.com/techsupport or contact a local sales support representative.

For frequently asked questions, refer to the Lattice Answer Database at www.latticesemi.com/Support/AnswerDatabase.

Revision History

Revision 1.3, October 2025

Section	Change Summary
All	Performed minor formatting and editorial edits.
Abbreviations in This Document	Updated list of abbreviations.

Introduction	• Updated Table 1.1. Summary of the Reference Design as follows:
	  • Added support for CertusPro-NX devices.
	  • Updated software versions.
	  • Updated IP support and IP versions.
	  • Added support for CertusPro-NX evaluation board.
	• Updated encoder (EnDat, Hiperface DSL, BiSS-C) initialization using the RISCV MC, SX CPU application in the Features section.

Directory Structure and Files	• Updated Figure 2.1. Directory Structure of the Closed Loop BLDC Motion Control Reference Design. • Added the bitstream attribute in Table 2.1. File List in the Reference Design Package.

Functional Description	• Added description for the CertusPro-NX devices in the Functional Description section and added Figure 3.1. Closed Loop Motor Control System Architecture (CertusPro-NX).
	• Renamed figure from *Closed Loop Motor Control System Architecture* to Figure 3.2. Closed Loop Motor Control System Architecture (Certus-NX), and updated figure.
	• Updated the Encoder Subsystem section and Figure 3.3. Encoder Subsystem Architecture, and added the following subsections:
	  • APB Encoder Position Requestor
	  • APB Encoder Position Requestor IP Parameter
	  • Control and Status Registers
	• Added note for the APB_M0 interface in the Motor Control section.
	• Updated Figure 3.10. Motor Control IP Block Wizard.
	• Updated the default values for HARDWARE_FETCH and ENC_POS_PORT_EXPORT in Table 3.8. Motor Control IP Attributes.
	• Changed field 23:0 to *Reserved* in Table 3.16. Synchronization Delay and Control.
	• Removed the following figures:
	  • *Application Flow (Configure Button)*
	  • *Application Flow (Update Button)*
	  • *Application Flow (Start Button)*
	  • *Application Flow (Stop Button)*
	  • *Demo GUI Application (Motor Control Demo Tab)*
	• Updated the template for the bare metal program in the RISC-V Bare Metal Program section.
	• Updated the encoder driver directory path in the following sections:
	  • Encoder Driver for the EnDat2.2 Master IP
	  • Encoder Driver for the HIPERFACE DSL Master IP
	  • Encoder Driver for the MB100 BiSS Interface Master IP
	• Removed the encoder position meter feature in the Demo GUI Application section and updated Figure 3.15. Demo GUI Application.

Running the Reference Design	• Updated the software versions in the Running the Reference Design section. • Updated the folder path in the Opening the Reference Design in the Lattice Propel Builder Software section. • Updated the file and folder paths, and updated Figure 5.3. Lattice Radiant Software in the Compiling the Reference Design in the Lattice Radiant Software section. • Updated the directory path and added a note on basic initialization sequence for CertusPro-NX devices in the Simulating the Reference Design section.
Implementing the Reference Design on Board	• Added the CertusPro-NX evaluation board in the Implementing the Reference Design on Board section.

Section	Change Summary

	• Updated the reference design file name in the Extracting the Reference Design Files section.
	• Renamed section from *Setting Up the Certus-NX Versa Evaluation Board* to Setting Up the Evaluation Board, and updated the content as follows:
	  • Added support for the CertusPro-NX evaluation board.
	  • Updated Figure 6.1. Certus-NX Versa Evaluation Board Setup.
	  • Added Figure 6.2. CertusPro-NX Evaluation Board Setup.
	• Added support for the CertusPro-NX evaluation board in the Setting Up the Encoder Board section.
	• Updated the Setting Up the Demo GUI Application section and Figure 6.4. Demo GUI Application UART Test Tab Setup.
	• Updated the Running the Motor from Demo GUI Application section, removed figure: *Demo GUI Application Status Log*, and updated Figure 6.5. Demo GUI Application Motor Control Demo Tab.

Resource Utilization	Added resource utilization for closed loop BLDC motion control reference design targeting CertusPro-NX devices in Table 7.1. Resource Utilization for Certus-NX Devices.
Debugging	• Updated K_p and K_i constant values in the Proportional and Integration Constant Tuning section. • Updated the guidelines in the GUI Hang, Not Responding, or Exit Abruptly section.
Integrating with a Different Motor	Updated the folder path in the following sections: • Hardware Project Compilation. • Software Project Compilation.
Integrating with a Different Encoder	• Updated the Integrating with a Different Encoder section. • Updated the Encoder Resolutions section to add support for the APB Encoder Position Requestor IP. • Added the Custom Encoder Master IPs section.
References	Updated references.

Revision 1.2, August 2025

Section	Change Summary
All	Performed minor formatting and editorial edits.
Abbreviations in This Document	Added definition for BiSS-C, IP, LED, MISO, MOSI, PC, PLL, and SCD.

Introduction	• Updated the description about the motion controller in the Introduction section.
	• Updated Table 1.1. Summary of the Reference Design as follows:
	  • Updated the version of the PLL IP, EnDat2.2 Master IP, and Motor Control IP.
	  • Added the MB100 BiSS Interface Master IP.
	  • Updated the note about the functional simulation.
	• Added support for the MB100 BiSS Interface Master IP and BiSS-C encoder in the Features section.

Functional Description	• Updated the following figures:
	  • Figure 3.1. Closed Loop Motor Control System Architecture
	  • Figure 3.2. Encoder Subsystem Architecture
	  • Figure 3.7. Motor Control IP Block Wizard
	  • Figure 3.8. General Application Flow
	  • Figure 3.14. Encoder Subsystem Clocking Domain
	  • Figure 3.17. Demo GUI Application (Motor Control Demo Tab)
	• Added Figure 3.9. Application Flow (Configure Button).
	• Added support for the MB100 BiSS Interface Master IP in the following sections:
	  • Encoder Subsystem
	  • SPI Controller
	  • Data Flow

Section	Change Summary

	• Software
	• Updated the EnDat2.2 Master IP and HIPERFACE DSL Master IP sections.
	• Added the MB100 BiSS Interface Master IP section.
	• Removed table: *Motor Control IP Parameters*.
	• Added the following tables:
	  • Table 3.2. Motor Control IP Attributes
	  • Table 3.3. Motor Control IP Attributes Description
	• Updated *MAXRPM* description in Table 3.7. Maximum RPM.
	• Updated *ENC_POS_BIT_SEL* description in Table 3.13. System Status.
	• Updated *ENC_POS* description in Table 3.19. Encoder Position.
	• Updated *clkos_o*, and added *clkos2_o* and *clkos3_o* in Table 3.21. Clocking.
	• Updated description for void system_init(void) and void enc_init (void) in Table 3.23. Main APIs.
	• Updated section title from *APIs* to Encoder Driver APIs for the EnDat2.2 Master IP, and table title from *APIs* to Table 3.24. Encoder Driver APIs for the EnDat2.2 Master IP. Updated the table as follows:
	  • Replaced uint8_t init_endat22_master (void) with uint8_t init_endat22_master (uint8_t sim_en).
	  • Added void set_recovery_tm (void).
	• Updated section title from *APIs* to Encoder Driver APIs for the HIPERFACE DSL Master IP, and table title from *APIs* to Table 3.25. Encoder Driver APIs for the HIPERFACE DSL Master IP.
	• Added the Encoder Driver for the MB100 BiSS Interface Master IP section.

Signal Description	Updated the description for motor_control_closeloop_en, encoder_interface, encoder_id_in, encoder_datain_4, encoder_dataout, and encoder_clk in Table 4.1. Primary I/O Interface Signals for the multi_encoder_demo_top Module.

Running the Reference Design	• Updated the following figures:
	  • Figure 5.2. Opening the Design in the Lattice Propel Builder Software
	  • Figure 5.4. Opening the Project File in the Lattice Radiant Software
	• Updated the Simulating the Reference Design section.

Implementing the Reference Design on Board	• Added support for the BiSS-C encoder in the Setting Up the Encoder Board section.
	• Updated the following figures:
	  • Figure 6.4. Demo GUI Application Status Log
	  • Figure 6.5. Demo GUI Application Motor Control Demo Tab

Resource Utilization	Updated Table 7.1. Resource Utilization for Certus-NX Devices.
Integrating with a Different Motor	Updated the guidelines on min_power (MINPWR) and power_gain (PWRGAIN) variables settings in GUI in the Power Parameter Tuning section.
Integrating with a Different Encoder	• Updated the Encoder Resolutions section. • Removed the *Encoder Types* section.
References	Updated references.

Revision 1.1, May 2025

Section	Change Summary
All	Performed minor formatting and editorial edits.

Introduction	• Updated Table 1.1. Summary of the Reference Design.
	  • Updated IP version for the SPI Controller IP, EnDat2.2 Master IP, and Motor Control IP.
	  • Added component: SICK AG HIPERFACE DSL EKS36-2KF0B020A rotary encoder.
	  • Updated the note on functional simulation.
	• Added support for the HIPERFACE DSL Master IP in the Features section.

Functional Description	• Added support for the HIPERFACE DSL Master IP in the following sections:

Section	Change Summary
	• Encoder Subsystem • SPI Controller • Data Flow • Software • Added the following sections: • HIPERFACE DSL Master IP • Encoder Driver for the HIPERFACE DSL Master IP • Added features in the Demo GUI Application section. • Updated the following figures: • Figure 3.1. Closed Loop Motor Control System Architecture • Figure 3.2. Encoder Subsystem Architecture • Figure 3.6. Parameter Settings of the Motor Control IP • Figure 3.7. General Application Flow • Figure 3.12. Encoder Subsystem Clocking Domain • Figure 3.14. Demo GUI Application (UART Test Tab) • Figure 3.15. Demo GUI Application (Motor Control Demo Tab) • Updated Table 3.2. Motor Control IP Parameters. • Added the *APB_DISABLE* register and removed *Reserved* register in Table 3.9. Synchronization Delay and Control. • Updated Field 30, 29, 10:9, 6, and 4 in Table 3.12. System Status. • Renamed table title from *Predictive Maintenance Current/Voltage Data* to Table 3.15. Predictive Maintenance Current/Voltage Data (for Register Offset 0x3C). • Renamed table title from *Predictive Maintenance Current/Voltage Data* to Table 3.16. Predictive Maintenance Current/Voltage Data (for Register Offset 0x40). • Updated Table 3.18. Encoder Position. • Updated the *clkop_o* attribute in Table 3.20. Clocking. • Updated the *void system_init(void)* and *void enc_init (void)* APIs in Table 3.22. Main APIs.
Signal Description	Updated Table 4.1. Primary I/O Interface Signals for the multi_encoder_demo_top Module. • Updated the *encoder_interface*, *encoder_id_in*, and *motor_control_pmod1_o* signals. • Added the *encoder_datain_2* and *encoder_de_2* signals.
Running the Reference Design	Added a note about the HIPERFACE DSL Master IP is not available in simulation in the Simulating the Reference Design section.
Implementing the Reference Design on Board	• Updated the steps in the Setting Up the Certus-NX Versa Evaluation Board section. • Updated the following figures: • Figure 6.1. Certus-NX Versa Evaluation Board Setup • Figure 6.3. Demo GUI Application UART Test Tab Setup • Figure 6.4. Demo GUI Application Status Log • Figure 6.5. Demo GUI Application Motor Control Demo Tab • Added the Setting Up the Encoder Board section. • Updated the steps in the Running the Motor from Demo GUI Application section.
Resource Utilization	Updated Table 7.1. Resource Utilization for Certus-NX Devices.
Debugging	• Updated the Motor Unable to Accelerate or Decelerate Correctly for Closed Loop Control section. • Reworked section *9.2 Proportional and Integration Constant Tuning* and moved to section 8.3 Proportional and Integration Constant Tuning.
Integrating with a Different Motor	• Removed the following sections: • *Encoder Resolutions* • *Proportional and Integration Constant Tuning* • Updated the Overcurrent Fault section and added the following subsections: • Power Parameter Tuning • Reducing Target RPM Value • Debugging Mode from the Motor Control IP

Section	Change Summary
Integrating with a Different Encoder	Added this section.
References	Updated references.

Revision 1.0, February 2025

Section	Change Summary
All	Initial release.

www.latticesemi.com