POLITECNICO
MILANO 1863

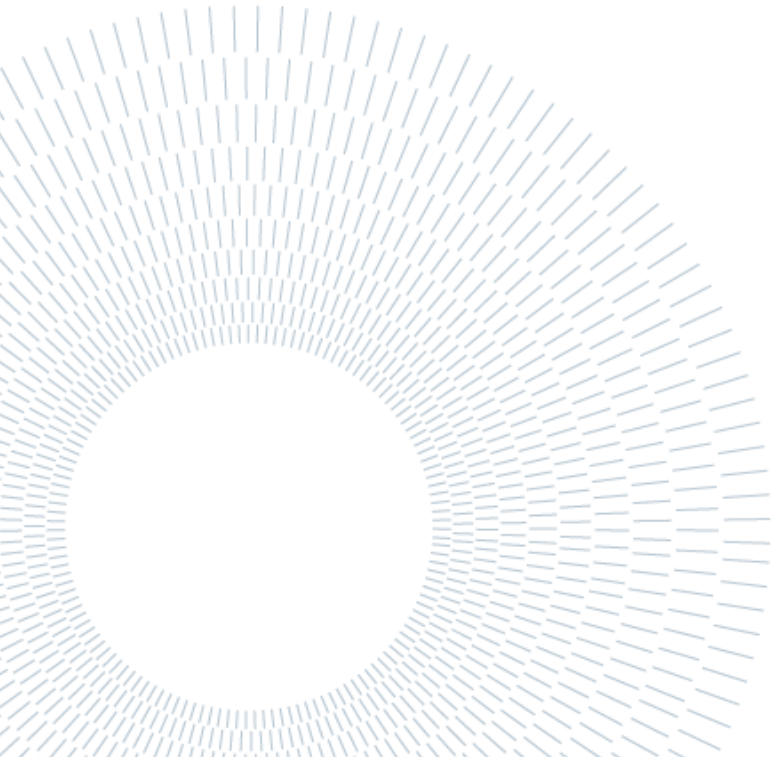
SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Comparative Performance Evaluation of PI and Deadbeat Current Control Architectures for PMSM Drives on FPGA

TESI DI LAUREA MAGISTRALE IN
ELECTRICAL ENGINEERING
INGEGNERIA ELETTRICA

Author: Batuhan Orhan

Student ID: 10943510
Advisor: Professor Luigi Piegari
Co-advisor: Sarper Öztürk
Academic Year: 2024-25



Abstract

This thesis presents the design and experimental validation of a fully FPGA-based control system for a Permanent Magnet Synchronous Motor (PMSM). The work focuses on the implementation and comparison of discrete-time current control strategies, namely a proportional–integral (PI) controller and a deadbeat controller, targeting high bandwidth, deterministic timing, and hardware-level parallelism.

The control architecture is implemented in VHDL on a Xilinx Zynq-7000 System-on-Chip platform. The system includes Clarke and Park transformations, mechanical angle acquisition from an incremental encoder, fixed-point arithmetic modules, sinusoidal PWM generation, and AXI-Stream–based communication between control stages. Attention is given to fixed-point format selection, pipeline latency management, and timing closure at high clock frequencies. The control loop operates with a sampling period in the order of microseconds, exploiting the inherent parallelism of FPGA fabric to achieve deterministic execution.

A detailed comparison between PI and deadbeat current control is carried out. The PI controller is tuned in fixed-point representation and validated experimentally. The deadbeat controller is derived from the discrete PMSM model and implemented using resource-optimized arithmetic structures. Experimental results demonstrate reduced current tracking error and improved dynamic response with the deadbeat strategy, at the expense of increased implementation complexity and sensitivity to parameter mismatch.

The thesis highlights the advantages of FPGA-based motor control in terms of latency, scalability, and architectural flexibility compared to conventional microcontroller implementations. The presented architecture provides a modular and extensible framework suitable for high-performance electric drive applications.

Key-words: FPGA; PMSM; PI control; Deadbeat control; Fixed-point implementation; Real-time motor control

Abstract in italiano

Questa tesi presenta la progettazione e la validazione sperimentale di un sistema di controllo completamente implementato su FPGA per un Motore Sincrono a Magneti Permanenti (PMSM). Il lavoro si concentra sull'implementazione e sul confronto di strategie di controllo di corrente in tempo discreto, in particolare un controllore proporzionale–integrale (PI) e un controllore deadbeat, con l'obiettivo di ottenere elevata banda passante, temporizzazione deterministica e parallelismo a livello hardware.

L'architettura di controllo è sviluppata in VHDL su piattaforma Xilinx Zynq-7000 System-on-Chip. Il sistema include le trasformazioni di Clarke e Park, l'acquisizione dell'angolo meccanico tramite encoder incrementale, moduli aritmetici in virgola fissa, la generazione di Sinusodial PWM e la comunicazione tra gli stadi di controllo tramite protocollo AXI-Stream. Particolare attenzione è dedicata alla scelta del formato in virgola fissa, alla gestione della latenza di pipeline e al rispetto dei vincoli temporali a frequenze di clock elevate. Il ciclo di controllo opera con un periodo di campionamento dell'ordine dei microsecondi, sfruttando il parallelismo intrinseco della logica FPGA per garantire un'esecuzione deterministica.

Viene effettuato un confronto dettagliato tra il controllo di corrente PI e deadbeat. Il controllore PI è tarato in rappresentazione a virgola fissa e validato sperimentalmente. Il controllore deadbeat è derivato dal modello discreto del PMSM e implementato mediante strutture aritmetiche ottimizzate in termini di risorse. I risultati sperimentali evidenziano una riduzione dell'errore di inseguimento della corrente e un miglioramento della risposta dinamica con la strategia deadbeat, a fronte di una maggiore complessità implementativa e di una più elevata sensibilità alla variazione dei parametri del modello.

La tesi mette in evidenza i vantaggi del controllo motore basato su FPGA in termini di latenza, scalabilità e flessibilità architetturale rispetto alle implementazioni convenzionali su microcontrollore. L'architettura proposta fornisce una struttura modulare ed estensibile, idonea ad applicazioni di azionamenti elettrici ad alte prestazioni.

Parole chiave: FPGA; PMSM; Controllo PI; Controllo deadbeat; Implementazione in virgola fissa; Controllo motore in tempo reale

Contents

1 Introduction

Permanent magnet synchronous machines (PMSMs) are widely adopted in high-performance electric drive applications due to their high efficiency, compact structure, and superior torque density. In modern drive systems, Field-Oriented Control (FOC) enables decoupled regulation of flux and torque components through transformation into a rotating dq reference frame. Within this structure, the inner current control loop determines the dynamic torque response and ultimately limits the achievable bandwidth of the entire drive system. As performance requirements increase in applications such as electric vehicles and industrial automation, the design of high-speed and high-accuracy current control strategies becomes a central challenge.

In industrial practice, proportional–integral (PI) current controllers remain the most widely used solution due to their robustness and well-established tuning methodologies. Discrete time formulations of current control in the rotating reference frame have been extensively studied, including improved decoupling strategies and digital-domain analysis [1]. These approaches provide predictable behavior and satisfactory steady-state performance. However, the achievable bandwidth of PI controllers is fundamentally constrained by sampling frequency, modulation delay, and computational latency. As switching frequencies increase and dynamic requirements become more stringent, these limitations motivate the exploration of predictive control techniques.

Deadbeat and model predictive current control strategies have been proposed as high-performance alternatives. By directly exploiting the discrete-time model of the PMSM, deadbeat current control computes the voltage vector required to force the stator current to its reference value within a single sampling interval. Multisampling and oversampling variants have been investigated to enhance dynamic response under low carrier frequencies [2], while Field-programmable gate array (FPGA) -based realizations of deadbeat and predictive current controllers demonstrate the feasibility of high-bandwidth implementations in hardware [3], [4]. These studies report improved transient behavior and reduced current tracking error compared to classical PI control. Nevertheless, predictive approaches exhibit increased sensitivity to parameter mismatch and to delay between measurement and actuation, which directly alters the one-step prediction model.

The impact of digital delay becomes particularly significant when the control algorithm is implemented in hardware. In practical drive systems, the control cycle

includes analog-to-digital conversion, coordinate transformations, controller computation, and pulse width modulation (PWM) update. Each stage introduces latency, which must be explicitly considered in discrete-time predictive formulations. Recent works on FPGA-based predictive cascaded speed and current control highlight the importance of parallel processing and pipeline optimization to reduce computational time [5]. Similarly, high-bandwidth vector control implementations on FPGA platforms exploit parallel architectures to achieve deterministic timing and reduced jitter [6]. These contributions confirm that hardware architecture and computational scheduling are integral parts of control system design.

FPGAs provide deterministic timing, massive parallelism, and flexibility in fixed-point arithmetic implementation. Compared to DSP-based solutions, FPGA platforms allow simultaneous execution of Clarke and Park transformations, current control laws, and PWM generation within deeply pipelined architectures. However, the resulting pipeline depth introduces a multi-cycle delay from current sampling to voltage applications. While classical PI controllers can tolerate such delay through conservative bandwidth design, predictive controllers such as deadbeat strategies require careful latency-aware modeling to preserve their theoretical performance.

Although predictive current control and FPGA implementation techniques have been individually addressed in the literature, a systematic experimental comparison between discrete-time PI and deadbeat current control implemented within the same fixed-point FPGA architecture remains limited. In particular, the interaction between pipeline latency, quantization effects, and predictive current control performance requires further investigation under realistic laboratory conditions.

The objective of this thesis is to design, implement, and experimentally evaluate two discrete-time current control strategies for PMSM drives: a classical PI controller and a deadbeat controller, both realized in fixed-point arithmetic on an FPGA platform. The work encompasses the derivation of discrete-time models, the development of fully pipelined VHDL architecture, the analysis of computational latency and its effect on closed-loop behavior, and experimental validation on a PMSM test bench. Special emphasis is placed on latency-aware predictive control implementation and on the comparative assessment of dynamic response and steady-state performance.

The remainder of this thesis is organized as follows. Chapter 2 presents the mathematical modeling of the PMSM and the principles of field-oriented control. Chapter 3 introduces the digital hardware fundamentals required for the implementation. Chapter 4 describes the FPGA-in-the-Loop (FIL) environment and the Simulink-based system design. Chapter 5 describes the custom VHDL implementation of the control algorithms. Chapter 6 reports the results obtained from the FPGA platform and provides a comparative performance analysis. Finally, Chapter 7 summarizes the main conclusions of the work and outlines possible directions for future development.

2 System Modeling and Control Fundamentals

2.1 Field Oriented Control

FOC, also referred to as vector control, is a commonly adopted control strategy for regulating the torque and speed of PMSM. The fundamental concept of FOC is the decoupling of the stator current components responsible for torque production and magnetic flux generation, enabling their independent regulation. This decoupling is accomplished by transforming the three-phase stator currents into a rotating reference frame that is synchronized with the rotor magnetic field, known as the dq reference frame. In this frame, the direct-axis current i_{sd} is aligned with the rotor flux, while the quadrature-axis current i_{sq} is oriented orthogonally to it. As a result, the electromagnetic torque is mainly governed by the q-axis current, whereas the d-axis current primarily influences the magnetic flux within the motor. A simplified control diagram illustrating the main functional blocks of the field-oriented control scheme is shown in Figure 2.1. [7]

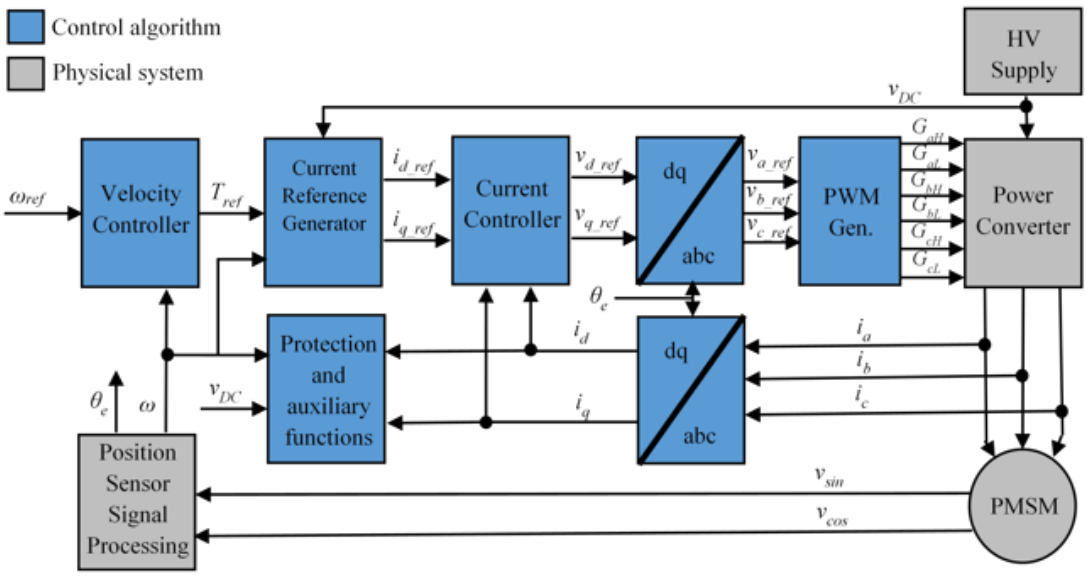


Figure 2.1 Block diagram of the field-oriented control scheme

2.2 PMSM Model

The electrical behavior of the PMSM can be initially described in the stationary three-phase (abc) reference frame. Under the assumption of balanced stator windings and neglecting zero-sequence components, the stator phase voltage equations can be written as

$$v_a = R_s i_a + \frac{d\Phi_a}{dt} \tag{2.1}$$

$$v_b = R_s i_b + \frac{d\Phi_b}{dt} \tag{2.2}$$

$$v_c = R_s i_c + \frac{d\Phi_c}{dt} \tag{2.3}$$

where v_a, v_b, and v_c are the stator phase voltages, i_a, i_b, and i_c are the stator phase currents, R_s is the stator resistance, andΦ_a, Φ_b, and $\Phi_c$denote the flux linkages of the corresponding stator phases.

For a more compact formulation of the three-phase quantities, the stator voltage, current, and flux linkage variables can be expressed using space-vector notation. The stator voltage and current space vectors, $\mathrm{v}_s$and i_s, are defined as

$$\mathrm{v}_s = \frac{2}{3}\left(v_a + v_b e^{j2\pi/3} + v_c e^{j4\pi/3}\right) \tag{2.4}$$

$$\mathrm{i}_s = \frac{2}{3}\left(i_a + i_b e^{j2\pi/3} + i_c e^{j4\pi/3}\right) \tag{2.5}$$

Similarly, the stator flux linkage space vector is defined as

$$\phi_s = \frac{2}{3}\left(\Phi_a + \Phi_b e^{j2\pi/3} + \Phi_c e^{j4\pi/3}\right) \tag{2.6}$$

Using the space-vector definitions in Equations (2.4), (2.5) and (2.6), and the three-phase stator voltages in Equations (2.1), (2.2) and (2.3) can be compactly rewritten as

$$\mathrm{v}_s = R_s \mathrm{i}_s + \frac{d\phi_s}{dt} \tag{2.7}$$

Equation (2.7) describes the PMSM electrical dynamics in the stationary reference frame using complex-valued quantities. Then the stationary space vectors are referred to a synchronous rotating reference frame aligned with the rotor magnetic field. This is achieved by multiplying the stationary space vectors by the complex exponential $e^{-j\theta_e}$, where θ_e denotes the electrical rotor position.

Accordingly, the stator voltage, current, and flux linkage space vectors in the rotating reference frame are defined as

$$\mathrm{v}_s^{\theta} = \mathrm{v}_s e^{-j\theta_e}, \qquad \mathrm{i}_s^{\theta} = \mathrm{i}_s e^{-j\theta_e}, \qquad \phi_s^{\theta} = \phi_s e^{-j\theta_e} \tag{2.8}$$

Substituting the rotating reference frame definitions in Equation (2.8) into the stationary reference frame voltage in Equation (2.7) allows the PMSM electrical dynamics to be referred to a synchronous reference frame rotating at the electrical angular speed of the rotor. By expressing the stator space vectors in this rotating frame and separating their real and imaginary components, the complex-valued formulation can be transformed into an equivalent real-valued representation. As a result, the PMSM electrical model can be expressed in the conventional dq reference frame, where the voltage equations explicitly include speed-dependent coupling terms that arise from the rotation of the reference frame.

$$v_{sd} = R_s i_{sd} + L_d \frac{d}{dt} i_{sd} - p\omega_r L_q i_{sq} \tag{2.9}$$

$$v_{sq} = R_s i_{sq} + L_q \frac{d}{dt} i_{sq} + p\omega_r L_d i_{sd} + p\omega_r \varphi_n \tag{2.10}$$

Here, p denotes the number of pole pairs and ω_r is the electrical rotor angular speed (rad/s). Since the machine under consideration is a surface-mounted PMSM, the stator inductances along the direct and quadrature axes can be assumed equal, i.e., $L_d = L_q = L_s$. Under this assumption, the voltage equations in the synchronous rotating reference frame simplify to

$$v_{sd} = R_s i_{sd} + L_s \frac{d}{dt} i_{sd} - p\omega_r L_s i_{sq} \tag{2.11}$$

$$v_{sq} = R_s i_{sq} + L_s \frac{d}{dt} i_{sq} + p\omega_r L_s i_{sd} + p\omega_r \varphi_n \tag{2.12}$$

This simplification is commonly adopted for surface-mounted PMSMs and allows the electrical dynamics of the d- and q-axis current control loops to be represented using identical stator inductance parameters. The resulting model forms the basis for the design of the stator current controllers presented in the following section.

The electromagnetic torque produced by a PMSM can be conveniently expressed in the synchronous rotating dq reference frame. In this frame, the torque is given by

$$T_e = \frac{3}{2} p\left(\varphi_n i_{sq} + \left(L_d - L_q\right) i_{sd} i_{sq}\right) \tag{2.13}$$

where T_e is the electromagnetic torque, p denotes the number of pole pairs, ψ_f is the permanent magnet flux linkage, and i_{sd} and i_{sq} are the direct- and quadrature-axis

stator currents, respectively. As stated, for surface mounted PMSMs, the direct- and quadrature-axis inductances are equal. Therefore, the reluctance torque component vanishes, and the torque expression simplifies to

$$T_e = \frac{3}{2} p \varphi_n i_{sq} \tag{2.14}$$

Based on the simplified torque expression, field-oriented control is typically implemented by imposing zero direct-axis current reference, $i_{sd}^* = 0$. This operating condition maximizes torque per ampere and simplifies the control structure, allowing the electromagnetic torque to be regulated exclusively through the control of the q-axis current.

As a result, the stator current control problem can be decoupled into two independent control loops, where the d-axis current loop regulates the flux-related component and the q-axis current loop directly governs torque production. This decoupled structure forms the basis for the design of the stator current controllers presented in the following section.

The mechanical dynamics of the PMSM can be described by the following differential equation.

$$T_e - T_L = B\omega_r + J\frac{d\omega_r}{dt} \tag{2.15}$$

where J denotes the rotor moment of inertia, ω_r is the mechanical rotor speed, T_e is the electromagnetic torque generated by the motor, T_L represents the load torque, and B is the viscous friction coefficient.

While the PMSM model can be conveniently expressed using space-vector notation and complex-valued quantities, such formulations are not well suited for direct implementation on digital hardware platforms. In FPGA-based motor control systems, real-time constraints, deterministic execution, and the extensive use of fixed-point arithmetic require control algorithms to be expressed using real-valued operations only. Although complex arithmetic can be emulated in hardware, it introduces additional computational overhead and increases resource utilization, making it less efficient for high-performance real-time control.

For this reason, the control algorithm is formulated using real-valued orthogonal reference frames through the application of the Clarke and Park transformations. These transformations map the three-phase stator quantities into stationary and rotating reference frames, respectively, allowing the PMSM equations to be expressed in a form that is more suitable for fixed-point digital implementation while preserving the decoupling properties fundamental to field-oriented control.

The Clarke transformation maps the three-phase stator quantities from the abc reference frame into a two-axis stationary orthogonal reference frame ($\alpha\beta$). This transformation reduces the three-phase system to two independent components under the assumption of balanced operation and zero-sequence components equal to zero.

For the stator currents, the Clarke transformation is defined as

$$\begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix} = \frac{2}{3}\begin{bmatrix} 1 & -1/2 & -1/2 \\ 0 & \sqrt{3}/2 & -\sqrt{3}/2 \end{bmatrix}\begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix} \tag{2.16}$$

The same transformation can be applied to stator voltages and flux linkages. The resulting $\alpha\beta$ quantities are expressed in a stationary reference frame fixed to the stator. Based on the space-vector formulation of Equations (2.4), (2.5) and (2.6), the Clarke transformation is here expressed in explicit matrix form. While mathematically equivalent to the previous derivation, restating the matrix allows the exact numerical coefficients used in the fixed-point VHDL implementation to be defined clearly.

The Park transformation rotates the stationary $\alpha\beta$ reference frame by the electrical rotor angle θ_e, yielding the synchronous rotating dq reference frame. This transformation allows the stator quantities to be expressed in a frame aligned with the rotor magnetic field, enabling decoupled control of torque and flux.

For the stator currents, the Park transformation is given by

$$\begin{bmatrix} i_{sd} \\ i_{sq} \end{bmatrix} = \begin{bmatrix} \cos(\theta_e) & \sin(\theta_e) \\ -\sin(\theta_e) & \cos(\theta_e) \end{bmatrix}\begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix} \tag{2.17}$$

Applying the Park transformation to the stator voltages and flux linkages yields the corresponding quantities in the synchronous rotating reference frame.

In FOC, the voltage references generated by the current controllers are expressed in the dq reference frame. Prior to pulse-width modulation, these voltage commands must be transformed back into the three-phase abc reference frame. This is achieved by applying the inverse Park and Clarke transformations.

The inverse Park transformation is defined as

$$\begin{bmatrix} v_\alpha \\ v_\beta \end{bmatrix} = \begin{bmatrix} \cos(\theta_e) & -\sin(\theta_e) \\ \sin(\theta_e) & \cos(\theta_e) \end{bmatrix}\begin{bmatrix} v_{sd} \\ v_{sq} \end{bmatrix} \tag{2.18}$$

while the inverse Clarke transformation maps the stationary $\alpha\beta$quantities back to the three-phase domain according to

$$\begin{bmatrix} v_a \\ v_b \\ v_c \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -1/2 & \sqrt{3}/2 \\ -1/2 & -\sqrt{3}/2 \end{bmatrix} \begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix} \quad (2.19)$$

These inverse transformations are an essential part of the control architecture, as they enable the synthesis of three-phase voltage commands suitable for inverter modulation.

2.3 Two Level Three Phase Inverter

In field-oriented control, the voltage commands generated by the digital controllers are applied to the motor through a power electronic converter. In this work, a two-level three-phase voltage source inverter is employed to interface the DC supply with the PMSM, as illustrated in Figure 2.2 [8].

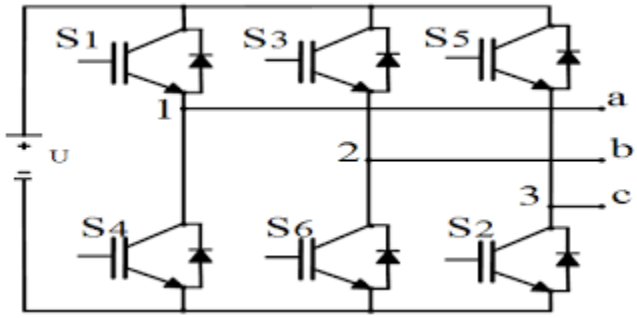


Figure 2.2 Two level three phase inverter circuit diagram

The inverter consists of three identical legs, each composed of two complementary switching devices connected to a common DC bus. By appropriately controlling the switching states of the inverter legs, the phase voltages applied to the motor terminals can be synthesized from the DC link voltage. Owing to its simplicity, robustness, and widespread adoption in motor drive applications, the two-level inverter represents a suitable and well-established choice for the control system under investigation.

For control and analysis purposes, the inverter is modeled by relating the phase voltages at the motor terminals to the switching states of the inverter. Let V_{AN}, V_{BN}, and V_{CN} denote the pole voltages of the inverter legs with respect to the DC bus midpoint, while V_{aN}, V_{bN}, and V_{cN} represent the phase voltages applied to the motor windings with respect to the motor neutral point.

Assuming a balanced three-phase system with an isolated neutral point, the motor phase voltages can be expressed as

$$V_{aN} = \frac{2V_{AN} - V_{BN} - V_{CN}}{3} \quad (2.20)$$

$$V_{bN} = \frac{2V_{BN} - V_{AN} - V_{CN}}{3} \quad (2.21)$$

$$V_{cN} = \frac{2V_{CN} - V_{AN} - V_{BN}}{3} \quad (2.22)$$

These expressions ensure that the sum of the three phase voltages is zero, satisfying the constraint imposed by the floating neutral point of the motor. This inverter model provides a convenient link between the voltage references generated by the control algorithm and the actual phase voltages applied to the PMSM.

In the proposed control structure, the inverter receives voltage reference commands generated in the synchronous rotating reference frame. These commands are subsequently converted into switching signals through a pulse width modulation (PWM) strategy, enabling the synthesis of the desired three-phase voltages at the motor terminals. The accuracy of this voltage synthesis directly affects the performance of the current control loops and, consequently, the overall drive performance.

2.4 Sinusoidal Pulse Width Modulation Generation

To synthesize the desired three-phase voltages at the inverter output, a PWM strategy is required. In this work, sinusoidal pulse-width modulation (SPWM) is employed due to its conceptual simplicity, widespread adoption in motor drive applications, and suitability for digital implementation.

In SPWM, each inverter leg is controlled by comparing a low-frequency reference voltage signal with a high-frequency carrier signal. The reference voltage represents the desired phase voltage command generated by the control algorithm, while the carrier signal determines the switching frequency of the inverter. By modulating the duty cycle of the inverter switches according to the reference waveform, the inverter output voltage can be shaped to approximate a sinusoidal waveform at the fundamental frequency.

The carrier signal is chosen as a triangular waveform with fixed amplitude and frequency. Its frequency directly defines the switching frequency of the inverter and is selected as a compromise between switching losses and current ripple. The reference voltage signals at steady state are sinusoidal waveforms whose amplitude and phase are determined by the control algorithm. For a three-phase system, three reference signals are generated, each phase-shifted by 120°with respect to the others.

The switching logic is based on a simple comparison process: when the instantaneous value of the reference voltage exceeds that of the carrier signal, the upper switch of the corresponding inverter leg is turned on; otherwise, the lower switch is activated. This comparison produces a sequence of switching pulses whose duty cycle varies in accordance with the reference signal. As a result, the fundamental component of the inverter output voltage follows the desired sinusoidal reference, while higher-frequency components are shifted around the switching frequency.

The operating principle of SPWM for a single inverter leg is illustrated in Figure 2.3 [9], where the sinusoidal reference signal, the triangular carrier signal, and the resulting PWM switching waveform are shown.

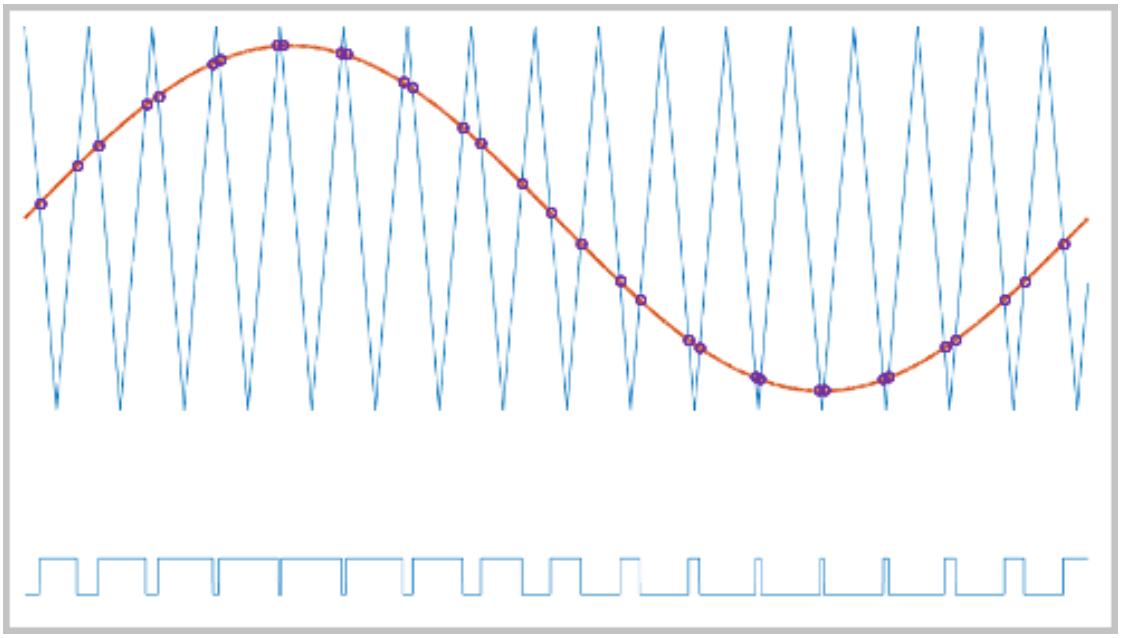


Figure 2.3 Principle of sinusoidal PWM

For clarity, the modulation principle is illustrated for a single phase only. The same modulation process is applied independently to the remaining inverter legs, using reference signals phase-shifted by 120°, thereby generating a balanced three-phase voltage system at the inverter output. In field-oriented control, the voltage references generated in the synchronous rotating reference frame are first transformed into the three-phase domain and subsequently processed by the SPWM modulator to produce the inverter switching signals.

In practical inverter implementations, a short dead-time interval is introduced between the switching commands of the upper and lower devices within each inverter leg to prevent shoot-through conditions. During this interval, both switches are temporarily turned off, ensuring safe operation of the power devices. Although dead time improves inverter reliability, it introduces a deviation between the commanded and applied phase voltages, which may affect current control performance.

2.5 Digital Control Methods

2.5.1 PI Current Control

Proportional–Integral (PI) controllers are widely used in FOC schemes to regulate the stator current components in the rotating reference frame. Independent PI controllers are typically employed for the direct-axis (d) and quadrature-axis (q) current loops to achieve decoupled control of magnetic flux and electromagnetic torque, respectively. The time-domain representation of a PI controller is given by

$$u(t) = K_p \cdot e(t) + K_i \int e(t)\, dt \tag{2.23}$$

where $K_p$and K_i are the proportional and integral gains, respectively. $e(t)$ denotes the current tracking error, defined as the difference between the reference current and the measured current in the corresponding axis, i.e., $e(t) = i^*(t) - i(t)$. The controller output $u(t)$ represents the commanded control voltage applied to the machine through the voltage source inverter, expressed in the same reference frame.

In the Laplace domain, the PI controller can be expressed by the transfer function

$$C(s) = \frac{U(s)}{E(s)} = K_p + K_i \cdot \frac{1}{s} \tag{2.24}$$

The proportional term provides an immediate response to the current error, improving transient performance, while the integral term eliminates steady-state error by accumulating the error over time.

The tuning of the PI controller gains is performed based on a frequency-domain design approach, in which the desired closed-loop bandwidth of the current controller is chosen according to switching frequency and noise filtering criteria. Typical choice for is $\omega_{cut} = 2\pi \frac{f_{sw}}{10}$.

By omitting the feedforward terms from the Equations (2.11) and (2.12) plant models simplify into

$$v_{sd} = r_s i_{sd} + L_s \frac{d}{dt} i_{sd} \tag{2.25}$$

$$v_{sq} = r_s i_{sq} + L_s \frac{d}{dt} i_{sq} \tag{2.26}$$

which can be simply treated as RL load whose transfer function can be written as below.

$$G(s) = \frac{I(s)}{V(s)} = \frac{1}{sL_s + R_s} \tag{2.27}$$

Using this plant transfer function closed loop transfer function can be written as below.

$$W(s) = \frac{C(s) \cdot G(s)}{1 + C(s) \cdot G(s)} = \frac{K_p s + K_i}{L_s s^2 + (R_s + K_p)s + K_i} \tag{2.28}$$

For the chosen cut-off frequency, frequency response of closed loop transfer function can be written as below.

$$W(j\omega_{cut}) = \frac{jK_p j\omega_{cut} + K_i}{-L_s j\omega_{cut}{}^2 + (R_s + K_p)j\omega_{cut} + K_i} \tag{2.29}$$

The cutoff angular frequency $\omega_c$is defined as the frequency at which the magnitude of the closed-loop transfer function decreases to −3 dB with respect to its low-frequency value.

$$|W(j\omega_{cut})| = \frac{\sqrt{(K_p\omega_{cut})^2 + K_i{}^2}}{\sqrt{(K_p + R_s)^2\omega_{cut}{}^2 + (K_i - L_s\omega_{cut}{}^2)^2}} = 10^{-3/20} \approx \frac{1}{\sqrt{2}} \tag{2.30}$$

This equation defines the relationship between the controller gains K_p and K_i for a given choice of cutoff frequency ω_c.

Squaring both sides to eliminate the square root yields

$$2\left[(K_p\omega_{cut})^2 + K_i{}^2\right] = (K_p + R_s)^2\omega_{cut}{}^2 + (K_i - L\omega_{cut}{}^2)^2 \tag{2.31}$$

Expanding the terms and rearranging the equation leads to a quadratic expression in K_i,

$$K_i{}^2 + 2\omega_{cut}{}^2 L_s K_i + \left[\omega_{cut}{}^2 K_p{}^2 - 2\omega_{cut}{}^2 R_s K_p - \omega_{cut}{}^2 R_s{}^2 - \omega_{cut}{}^4 L_s{}^2\right] = 0 \tag{2.32}$$

Solving the quadratic equation with respect to K_i yields

$$K_i = -\omega_{cut}{}^2 L \pm \omega_{cut}\sqrt{2\omega_{cut}{}^2 L^2 - K_p{}^2 + 2RK_p + R^2} \tag{2.33}$$

For the PI controller gains to be physically meaningful and practically implementable, two constraints must be satisfied.

1. Real-valued solution constraint

Since the integral gain K_i must be a real-valued quantity, the discriminant of the square root term must be non-negative. This requirement imposes the following condition on the proportional gain K_p:

$$\omega_{cut}{}^2 L_s{}^2 - K_p{}^2 + 2R_s K_p + R_s{}^2 \geq 0 \tag{2.34}$$

2. Positive integral gain constraint

Furthermore, to ensure closed-loop stability and proper integral action, the integral gain must satisfy non-negative K_i value. Applying this condition to the positive branch of the solution leads to

$$2\omega_{cut}{}^2 L_s{}^2 - K_p{}^2 + 2R_s K_p + R_s{}^2 \geq \omega_{cut}{}^2 L_s{}^2 \tag{2.35}$$

By squaring both sides and rearranging terms, the above inequality can be expressed as

$$K_p{}^2 - 2R_s K_p - R_s{}^2 \omega_{cut}{}^2 L_s{}^2 \leq 0 \tag{2.36}$$

Solving this inequality yields the admissible range for the proportional gain,

$$K_p = R_s \pm \sqrt{2R_s{}^2 + \omega_{cut}{}^2 L_s{}^2} \tag{2.37}$$

Since the same physical constraints also apply to the proportional gain and negative values of K_p are not meaningful in this context, the proportional gain is selected within the interval

$$0 \leq K_p \leq K_{p,max}, \qquad K_{p,max} = R_s + \sqrt{2R_s{}^2 + \omega_{\text{cut}}^2 L_s{}^2} \tag{2.38}$$

Among the infinitely many admissible (K_p, K_i) pairs, the pair that maximizes the phase margin of the open-loop system is selected. To this end, the open-loop phase evaluated at the cutoff frequency $\omega_c$is considered:

$$\angle G(j\omega_{cut}) = -\tan^{-1}\left(\frac{K_i}{\omega_{cut} K_p}\right) - \tan^{-1}\left(\frac{\omega_{cut} L_s}{R_s}\right) \tag{2.39}$$

From this expression, it can be observed that increasing the proportional gain K_p, while keeping the relationship $K_i(K_p)$ on the admissible branch, leads to an improvement in the phase margin.

Differentiating the integral gain $K_i$with respect to the proportional gain K_p yields:

$$\frac{dK_i}{dK_p} = \omega_{cut} \frac{R_s - K_p}{\sqrt{2\omega_{cut}{}^2 L_s^2 - K_p{}^2 + 2R_s K_p + R_s{}^2}} \tag{2.40}$$

According to the admissibility condition, the denominator of the above expression is always positive within the valid range. Therefore, the sign of the derivative is determined by the term $R_s - K_p$. To decrease the integral gain $K_i$while increasing the proportional gain K_p, thus improving the phase margin, the proportional gain must be selected within the range

$$R_s < K_p < K_{p,\max} \tag{2.41}$$

This tuning procedure provides a systematic method for selecting the PI controller gains for the d-axis and q-axis current control loops in field-oriented control, ensuring stable and responsive current regulation of the PMSM.

The PI controller design presented in the previously is formulated in continuous time. However, for real-time digital implementation, the control law must be expressed in discrete time. The continuous time PI control law is given in Equation (2.23). Let the sampling period be denoted by T_s, and let $e[k]$ and $u[k]$ represent the discrete-time error and controller output at the k-th sampling instant, respectively.

The integral term in Equation (2.23) can be approximated in discrete time by a cumulative sum of the error signal, yielding

$$\int_0^t e(\tau)\, d\tau \;\approx\; T_s \sum_{i=0}^{k} e[i] \tag{2.42}$$

Substituting this expression into Equation (2.23) the discrete-time PI control law can be written as

$$u[k] = K_p e[k] + K_i T_s \sum_{i=0}^{k} e[i] \tag{2.43}$$

To obtain an implementation-friendly recursive form, the controller output can be rewritten as a difference equation,

$$u[k] = u[k-1] + K_p(e[k] - e[k-1]) + K_i T_s e[k] \tag{2.44}$$

This formulation is particularly well suited for FPGA-based implementation, as it relies solely on additions, multiplications, and register delays, and can be efficiently realized using fixed-point arithmetic.

For discrete-time simulation purposes, the continuous-time PI controller defined in Equation (2.24) is implemented in discrete time. In this work, the discrete-time formulation adopted by the Simulink PI block is based on the bilinear (Tustin) transformation, which provides a stable and frequency-consistent mapping between the continuous-time and discrete-time domains.

Using the Tustin approximation

$$s \approx \frac{2}{T_s} \frac{1 - z^{-1}}{1 + z^{-1}} \tag{2.45}$$

the continuous-time PI transfer function in Equation (2.24) is mapped into the z-domain by substituting s with Equation (2.45)

$$\mathrm{C}(z) = K_p + K_i \frac{T_s}{2} \frac{z + 1}{z - 1} \tag{2.46}$$

This discrete-time transfer function corresponds to the internal realization used by the Simulink PI controller when the Tustin discretization method is selected.

2.5.2 Deadbeat Current Control

Deadbeat control is a discrete-time control strategy designed to drive the controlled variable to its reference value within a finite number of sampling periods. In motor drive applications, deadbeat current control has gained increasing attention in recent years due to the growing emphasis on high-bandwidth digital control, where fast current tracking and tight regulation directly translate into improved torque response and disturbance rejection. Unlike classical PI controllers, whose dynamics are typically shaped through bandwidth and phase-margin tuning, deadbeat control explicitly exploits a discrete-time model of the plant to compute the control action required to reach the desired value in the next sampling instant.

A key advantage of deadbeat control is that it is formulated directly in discrete time, which aligns naturally with the sampled data nature of digital control systems. This avoids the need to interpret continuous-time controller dynamics through a separate discretization step and allows the controller design to be expressed directly in terms of the sampling period. As a result, the controller's behavior is straightforward to analyze and implement at the chosen sampling rate, and the designer can explicitly reason about the effect of sampling and computational delay on the closed-loop response. Deadbeat control provides a direct mechanism to target very fast transient response, ideally achieving one-sample settling under nominal conditions.

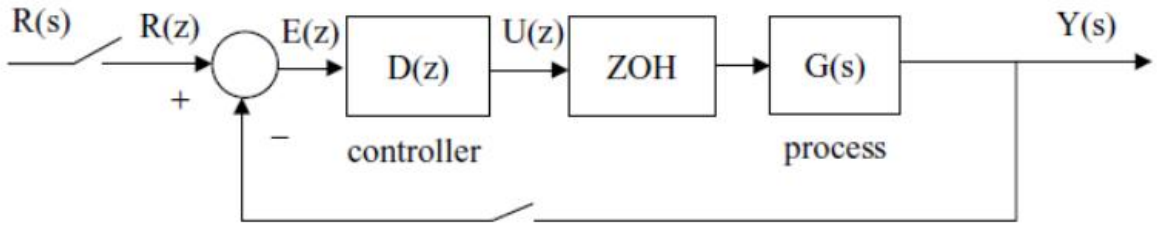


Figure 2.4 Discrete-time closed loop control diagram

The general structure of a digitally controlled system considered in this work is illustrated in Figure 2.4, where the controller is implemented in the discrete-time domain, while the plant is continuous-time. The reference signal is sampled and processed by a digital controller $D(z)$, whose output is converted into a continuous-time signal through a zero-order hold (ZOH) before being applied to the plant $G(s)$.

The presence of the ZOH is intrinsic to digital control systems, as the control signal is held constant between consecutive sampling instants. For a sampling period T_s, the transfer function of the zero-order hold is given by

$$G_{\text{ZOH}}(s) = \frac{1 - e^{-sT_s}}{s} \tag{2.47}$$

The combined effect of the continuous-time plant and the ZOH can be represented in the discrete-time domain by an equivalent discrete-time plant transfer function $G_{ZOH}(z)$. This discrete-time representation captures the sampled-data behavior of the system and forms the basis for deadbeat controller design.

The combined effect of the continuous-time plant and the ZOH can be represented in the discrete-time domain by an equivalent discrete-time plant transfer function $G_p(z)$. This discrete-time representation captures the sampled-data behavior of the system and forms the basis for deadbeat controller design.

$$G_{\text{cl}}(z) = \frac{Y(z)}{R(z)} = \frac{1}{z^k} \tag{2.48}$$

In discrete-time control, deadbeat behavior is achieved by explicitly specifying the desired closed-loop transfer function. For a deadbeat controller with a finite settling time of $k$sampling periods, the closed-loop transfer function is chosen as

$$G_{\text{cl}}(z) = \frac{1}{z} \tag{2.49}$$

This implies that the controlled variable reaches its reference in a single sampling period under ideal model conditions and without saturation.

Let $G_{ZOH}(z)$ denote the discrete-time transfer function of the plant obtained from the ZOH-equivalent representation. The closed-loop transfer function of the digital control system can be written as

$$W(z) = \frac{D(z)\,G_{ZOH}(z)}{1 + D(z)\,G_{ZOH}(z)} \tag{2.50}$$

Imposing the deadbeat condition in Equation (2.49), the controller transfer function can be directly obtained as

$$D(z) = \frac{W(z)}{G_{ZOH}(z)\big(1 - W(z)\big)} \tag{2.51}$$

This expression highlights the model-based nature of deadbeat control, as the controller explicitly depends on the discrete-time plant dynamics. While the transfer-function formulation provides a clear theoretical interpretation of deadbeat behavior, practical implementation is more conveniently expressed in the time domain.

For real-time digital implementation, particularly on FPGA-based platforms, the deadbeat controller is therefore implemented as a difference equation derived from the discrete-time plant model and the desired closed-loop response. This time-domain formulation enables deterministic execution using fixed-point arithmetic and is presented in the following section.

2.5.2.1 Deadbeat Control Design Based on Discrete-Time RL Model

The stator current dynamics of the PMSM can be approximated by a first order RL model. Expressing this model directly in discrete time using a backward-difference approximation for the derivative, the voltage–current relationship can be written as

$$v[n+1] = R_s\, i[n+1] + L_s\, \frac{i[n+1] - i[n]}{T_s} \tag{2.52}$$

where $v[n]$ is the applied stator voltage, $i[n]$ is the measured stator current at the nth sampling instant, $R$and L are the stator resistance and inductance, respectively, and T_s is the sampling period. The discrete time current model is obtained using a backward Euler approximation of the current derivative and is written in implicit form at the n+1 sampling instant. This choice enables direct enforcement of the deadbeat control objective and yields a numerically stable control law suitable for digital implementation.

Rearranging the above expression to obtain the current update equation yields

$$i[n+1] = \frac{T_s}{L_s + R_s T_s}\, v[n+1] + \frac{L_s}{L_s + R_s T_s}\, i[n] \tag{2.53}$$

This equation represents the discrete-time plant model used for deadbeat current controller design.

For one-step deadbeat control, the objective is to drive the current to its reference value within a single sampling period. This requirement can be expressed as

$$i[n+1] = i^*[n] \tag{2.54}$$

Substituting the discrete-time plant model into the above condition gives

$$i^*[n] = \frac{T_s}{L_s + R_s T_s}\, v[n+1] + \frac{L_s}{L_s + R_s T_s}\, i[n] \tag{2.55}$$

Solving for the control input $v[n+1]$ yields the deadbeat control law

$$v[n+1] = \frac{L_s + R_s T_s}{T_s}\, i^*[n] - \frac{L_s}{T_s}\, i[n] \tag{2.56}$$

The resulting deadbeat controller is implemented as a difference equation evaluated at each sampling instant. The control input depends on the current reference and the previously measured current and does not require integral action or dynamic tuning parameters.

Under nominal conditions and assuming accurate knowledge of the plant parameters, this control law ensures one-sample settling of the current response, provided that the resulting voltage command remains within the available inverter voltage limits. In practice, the control output is therefore subject to saturation to account for DC-link and modulation constraints.

2.5.3 Comparison Between PI and Deadbeat

The PI and deadbeat current controllers considered in this work differ fundamentally in their design philosophy and dynamic behavior. The PI controller shapes the closed-loop response through bandwidth and phase-margin tuning, resulting in a gradual convergence of the current to its reference. This approach offers good robustness against parameter variations and modeling uncertainties, making it well suited for practical applications where plant parameters may vary with temperature or operating conditions.

In contrast, deadbeat control is explicitly model-based and aims to drive the current to its reference value within a finite number of sampling periods, ideally in a single step. This results in a significantly faster transient response under nominal conditions. However, the reliance on an accurate discrete-time model makes deadbeat control more sensitive to parameter mismatches, delays, and voltage saturation effects. As a result, while deadbeat control can achieve superior dynamic performance, its robustness is inherently lower than that of PI control.

From an implementation perspective, both controllers are well suited for FPGA-based realization. The PI controller requires accumulation and gain scaling, while the deadbeat controller directly computes the control action from the plant model and reference. The choice between the two strategies therefore involves a trade-off between dynamic performance and robustness, which is further investigated in the experimental and simulation results presented in the subsequent chapters.

Table 2.1 Summary of PI and deadbeat current controller characteristics

Feature	**PI Current Control**	**Deadbeat Current Control**
Control formulation	Continuous-time design with discrete implementation	Purely discrete-time design
Transient response	Gradual, bandwidth-limited	Finite-step (ideally one-sample)
Settling time	Determined by cutoff frequency	Determined by sampling period
Model dependence	Low	High
Robustness to parameter mismatch	High	Moderate to low
Sensitivity to delay	Low	High
Sensitivity to saturation	Moderate	High
FPGA implementation	Simple and robust	Simple but model-dependent
Tuning complexity	Low	Low (requires accurate model)

2.5.4 PI Speed Control

From Equation (2.15), the mechanical dynamics of the PMSM were defined and shown to be governed by a first-order differential equation. When expressed in the Laplace domain, the corresponding transfer function relating the electromagnetic torque to the mechanical speed can be written as

$$G_m(s) = \frac{\Omega_m(s)}{T_e(s)} = \frac{1}{Js + B} \tag{2.57}$$

Due to the structural similarity between the mechanical dynamics in Equation (2.15) and the electrical RL dynamics used for current control, the same PI control algorithm can be employed for speed regulation, with the mechanical subsystem acting as the controlled plant instead of the stator electrical circuit.

In the cascaded control architecture, the speed controller operates as an outer loop and generates a reference torque command, which is subsequently converted into a q-axis current reference for the inner current control loop, as expressed below. This reference i_{sq}^* is then supplied to the inner current control loop, which regulates the stator currents to produce the desired electromagnetic torque.

$$i_{sq}^* = T_e \frac{2}{3p\,\varphi_n} \tag{2.58}$$

To ensure proper time-scale separation and stable operation, the bandwidth of the speed control loop is selected significantly lower than that of the current control loops. In particular, the speed-loop cutoff frequency is selected to be approximately one decade lower than that of the inner current loops.

In this chapter, the PMSM drive model and the FOC framework were established, and the current-control problem was expressed in discrete time to match the sampled-data nature of the final implementation. Based on this formulation, both the discrete PI approach and the deadbeat current controller were derived as candidate solutions for regulating the d- and q-axis currents, which directly shape torque production and flux behavior. However, moving from control equations to a realizable hardware implementation requires additional constraints. The next chapter focuses on the FPGA-oriented design considerations like fixed-point representation, bit-width growth and rescaling, pipelining and scheduling, and data interfacing which form the basis for implementing the controllers in hardware while preserving the behavior predicted by the discrete-time model.

3 Hardware-Aware Design Methodology

3.1 Field-Programmable Gate Arrays in Real-Time Control Applications

3.1.1 Overview of FPGA Technology

A Field-Programmable Gate Array (FPGA) is a reconfigurable digital hardware device composed of an array of configurable logic blocks interconnected through a programmable routing network. Unlike general-purpose processors, which execute instructions sequentially according to predefined architecture, FPGAs allow the hardware structure itself to be defined by the designer, enabling application-specific data paths and highly parallel execution.

Modern FPGAs integrate several types of resources optimized for different computational tasks. These include configurable logic elements for implementing control logic, dedicated digital signal processing (DSP) blocks for efficient arithmetic operations, and on-chip memory resources for buffering and data storage. All logic on the FPGA operates synchronously with respect to one or more clock signals, and the behavior of the system is fully determined by the hardware description programmed into the device.

From a control perspective, the defining characteristic of FPGA platforms is that computations are performed through spatial parallelism rather than sequential instruction execution. Multiple arithmetic and logical operations can be evaluated simultaneously within a single clock cycle, provided sufficient hardware resources are available. This execution model fundamentally differs from that of microcontroller or DSP-based systems and has direct implications for the implementation of real-time control algorithms.

3.1.2 FPGA Execution Model Compared to Processor-Based Control

In conventional digital motor control systems based on microcontrollers or digital signal processors, control algorithms are executed as sequences of instructions within a software program. The control loop is typically triggered by periodic interrupts, and its execution time depends on instruction scheduling, memory access, and peripheral latency. As a result, the effective control delay may vary slightly from one iteration to the next, especially in systems that handle multiple tasks or interrupts.

In contrast, FPGA-based control systems implement the control algorithm as dedicated hardware logic. Each operation bis realized as a hardware block that operates concurrently with other blocks. The execution latency of the control loop is therefore fixed and deterministic, defined by the number of clock cycles required for the signal to propagate through the hardware pipeline. Once synthesized and placed, this latency does not change during operation.

This deterministic execution behavior is particularly advantageous for high-performance current control and predictive control strategies, where precise knowledge of sampling instants and control delay is essential. In such cases, the control delay becomes a design parameter that can be explicitly accounted for in the controller formulation rather than an uncertain artifact of software execution.

3.1.3 Relevance of FPGA Platforms for Digital Motor Control

The characteristics of FPGA execution make these platforms well suited for real-time motor control applications requiring high sampling rates, low and predictable latency, and tight synchronization between measurement, control computation, and modulation. The ability to implement parallel signal processing pipelines allows multiple control functions to be executed concurrently without increasing the control period.

Moreover, the deterministic nature of FPGA execution facilitates the implementation of control strategies that are formulated directly in discrete time, such as deadbeat current control. Since the timing behavior of the hardware is fixed and repeatable, the assumptions made during controller design regarding sampling and delay remain valid in the implemented system. This alignment between control theory and hardware execution is a central motivation for the FPGA-based approach adopted in this work.

3.2 Fixed-Point Arithmetic in FPGA-Based Control

Digital control algorithms are commonly developed and validated using floating-point arithmetic, which provides a wide dynamic range and high numerical precision. In most software-based environments, floating-point operations conform to the IEEE 754 standard, which defines the representation and behavior of real numbers using sign, exponent, and mantissa fields. This format enables accurate representation of very small and very large values and simplifies algorithm development by largely eliminating concerns related to numerical overflow and scaling.

While IEEE 754 floating-point arithmetic is well suited for high-level modeling and simulation, its direct implementation on FPGA platforms is associated with significant hardware overhead. Floating-point operators require complex arithmetic units, including exponent alignment, normalization, and rounding logic, which result in

increased resource utilization and longer computational latency compared to integer-based operations. Moreover, floating-point pipelines typically introduce multi-cycle latency, which may complicate timing analysis in real-time control applications where precise and deterministic execution is required.

For these reasons, although floating-point arithmetic can be implemented on modern FPGA devices, it is generally avoided in high-performance real-time motor control systems. Instead, fixed-point arithmetic is preferred, as it allows numerical operations to be expressed using integer arithmetic with well-defined scaling. Fixed-point implementations offer reduced hardware complexity, lower latency, and fully deterministic execution, making them particularly attractive for control applications operating at high sampling rates.

In this work, fixed-point arithmetic is adopted for the implementation of the control algorithms on FPGA hardware. This choice enables efficient utilization of the available computational resources while maintaining predictable timing behavior, which is essential for the reliable execution of discrete-time control strategies such as deadbeat current control.

3.2.1 Fixed-Point Representation and Q-Format Notation

In fixed-point arithmetic, real-valued quantities are represented by scaling an integer variable by a constant power-of-two. A fixed-point number x can be expressed as

$$x = \frac{X_{\text{int}}}{2^n} \tag{3.1}$$

where X_{int} is the stored integer value and n denotes the number of fractional bits. This representation allows fixed-point arithmetic to be implemented using integer adders, subtractors, and multipliers, followed by appropriate bit shifting operations.

Depending on the nature of the represented signal, both signed and unsigned fixed-point formats are employed in FPGA-based control systems.

Signed Fixed-Point Representation

Signed fixed-point formats are used for quantities that may assume both positive and negative values, such as stator currents, voltage commands, and internal controller signals. These values are typically represented using two's complement arithmetic. For a signed fixed-point format with m integer bits (excluding the sign bit) and n fractional bits, the representable range is given by

$$X_{\text{int}} \in [-2^{m+n}, 2^{m+n} - 1] \tag{3.2}$$

which corresponds to the real-valued range

$$x \in [-2^m, 2^m - 2^{-n}] \tag{3.3}$$

The resolution of the representation, defined as the smallest representable increment, is

$$\Delta x = 2^{-n} \tag{3.4}$$

This symmetric range around zero makes signed fixed-point formats well suited for control signals expressed in rotating reference frames, where both positive and negative values naturally occur.

Unsigned Fixed-Point Representation

Unsigned fixed-point formats are employed for quantities that are inherently non-negative, such as duty cycles, modulation indices, and timing-related signals. In this case, the stored integer value satisfies

$$X_{\text{int}} \in [0, 2^{m+n} - 1] \tag{3.5}$$

leading to the real-valued range

$$x \in [0,\ 2^m - 2^{-n}] \tag{3.6}$$

By eliminating the sign bit, unsigned formats maximize the usable dynamic range for positive quantities and are therefore preferred whenever bidirectional representation is not required.

The choice between signed and unsigned fixed-point representations is dictated by the physical meaning of the corresponding signal and directly affects the achievable dynamic range and numerical resolution. In FPGA-based motor control systems, signed formats are typically adopted for measurement and control signals, while unsigned formats are used for modulation and timing-related quantities. The specific word lengths and scaling factors employed in this work are introduced later in the context of fixed-point simulation and hardware implementation.

3.2.2 Scaling and Normalization of Control Signals

In FPGA-based control systems, physical quantities such as currents, voltages, and control commands are represented in the digital domain using fixed-point arithmetic. Since fixed-point numbers have a finite dynamic range, a direct representation of physical quantities in their original units is generally not possible. Instead, signals are first normalized with respect to suitable base values before being processed by the control algorithm.

Let x_{phys} denote a physical quantity expressed in engineering units (e.g., amperes or volts) and let x_{base} represent the corresponding base value chosen according to the expected operating range. The normalized digital representation is defined as

$$x_{\text{norm}} = \frac{x_{\text{phys}}}{x_{\text{base}}} \tag{3.7}$$

Conversely, the physical quantity can be recovered from its normalized representation as

$$x_{\text{phys}} = x_{\text{norm}} \cdot x_{\text{base}} \tag{3.8}$$

Through this normalization process, all signals handled by the digital control algorithm are mapped into a dimensionless range that is compatible with the fixed-point representation. The base values are selected such that the normalized signals remain within the representable interval

$$x_{\text{norm}} \in [-2^{m}, 2^{m} - 2^{-n}] \tag{3.9}$$

for signed fixed-point formats, thereby preventing numerical overflow during normal operation.

Normalization also provides a consistent scaling framework across different processing stages, allowing arithmetic operations such as addition and multiplication to be performed without repeated rescaling. This is particularly important in FPGA-based implementations, where inconsistent scaling may lead to unnecessary truncation, loss of precision, or overflow.

In practice, different base values may be selected for different signal types. For example, stator currents may be normalized with respect to the rated or maximum allowable current, while voltage commands are normalized with respect to the available DC-link voltage. The specific base values and scaling factors adopted in this work are introduced later in the context of fixed-point simulation and hardware implementation.

3.2.3 Saturation and Finite Numerical Range

Due to the finite dynamic range of fixed-point representations, arithmetic operations performed within the control algorithm may produce values that exceed the representable interval. Unlike floating-point arithmetic, where overflow is rarely encountered in typical control applications, fixed-point arithmetic requires explicit handling of such conditions. In FPGA-based implementations, this is commonly achieved through saturation logic, which limits the signal to the maximum or minimum representable value.

For a signed fixed-point representation with range represented in Equation (3.3) the saturation operation can be expressed as

$$x_{\text{sat}} = \begin{cases} 2^m - 2^{-n}, & x > 2^m - 2^{-n} \\ x, & -2^m \le x \le 2^m - 2^{-n} \\ -2^m, & x < -2^m \end{cases} \tag{3.10}$$

Saturation ensures that numerical overflow does not result in unintended wrap-around behavior, which could otherwise lead to severe degradation of control performance or instability. While saturation introduces a nonlinearity into the control loop, its presence is an inherent consequence of fixed-point arithmetic and must be considered when transitioning control algorithms from idealized simulations to hardware execution.

3.2.4 CORDIC-Based Trigonometric Computation for FPGA Implementation

Coordinate Rotation Digital Computer (CORDIC) is a well-established algorithm for evaluating trigonometric functions using iterative shift-and-add operations, making it particularly suitable for fixed-point digital hardware and FPGA-based systems [10].

In FPGA platforms, the use of CORDIC enables efficient and deterministic computation of sine and cosine values without requiring hardware multipliers or division units. This property is especially advantageous in real-time motor control applications, where predictable latency and efficient resource utilization are critical. For this reason, CORDIC-based architectures are widely adopted in FPGA designs and are supported through vendor-optimized IP cores.

In this work, CORDIC functionality is realized using the standard Xilinx CORDIC IP, which supports multiple operational modes and architectural configurations. Since the CORDIC algorithm is provided as a pre-designed and verified hardware block, its internal iterative operation is not addressed in detail. Instead, the block is treated as a deterministic functional unit within the control pipeline, supplying sine and cosine values required for the Park and inverse Park transformations.

A high-level block diagram of the Xilinx CORDIC core, illustrating the input stage, iterative computation engine, and output stage, is shown in Figure 3.1.

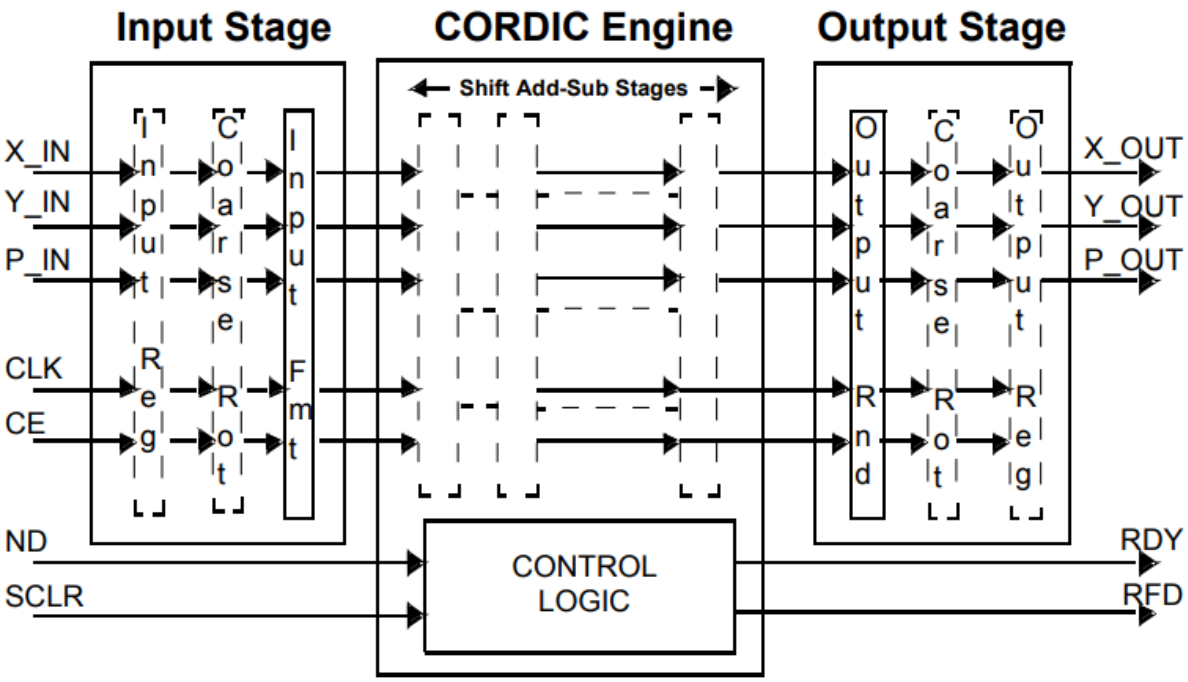


Figure 3.1 Block diagram of CORDIC

3.3 Timing, Determinism, and Pipelining

In FPGA-based systems, all computations are performed within a synchronous, clock-driven execution model. Signals are updated at discrete instants defined by the rising edge of a global clock, and intermediate values are stored in registers that hold their contents between clock cycles. As a result, time in an FPGA is naturally discretized in terms of clock cycles, and the temporal behavior of a control algorithm is fully determined by the structure of the underlying hardware.

A fundamental property of this execution model is determinism. Once an FPGA design has been synthesized and implemented, the latency between an input signal and the corresponding output response is fixed and repeatable. Unlike processor-based control systems, where execution time may vary due to instruction scheduling, memory access, or interrupt handling, FPGA-based implementations exhibit a constant timing behavior that is independent of data values and operating conditions. From a control perspective, this deterministic behavior allows execution delays to be treated as known parameters rather than uncertain disturbances.

FPGA hardware enables spatial parallelism, as multiple arithmetic and logical operations can be evaluated concurrently within the same clock cycle. In control applications, this allows different parts of the computation to be executed simultaneously rather than sequentially. Spatial parallelism increases computational throughput without reducing the control period and forms the basis for high-performance real-time execution.

In practice, complex control computations are often divided into multiple stages separated by registers, giving rise to a pipelined data path as illustrated in Figure 3.2. Each pipeline stage performs a portion of the overall computation within one clock cycle, and intermediate results are stored in registers before being passed to the next stage. Once the pipeline is filled, new input samples can be accepted while previous samples are still being processed, resulting in the concurrent evaluation of multiple data samples across different stages.

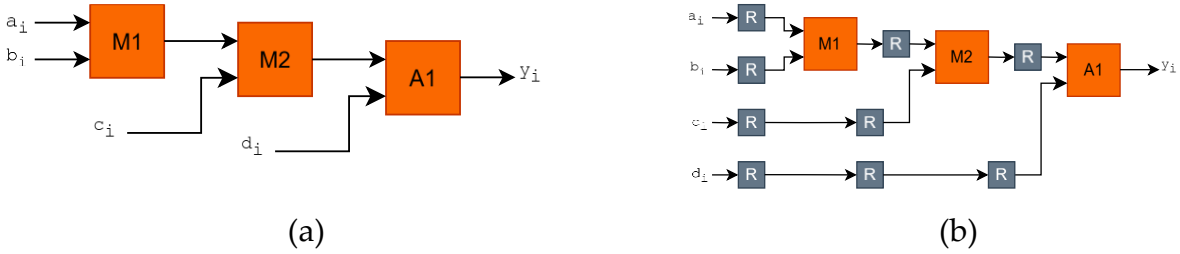


Figure 3.2 Non-pipelined (a) and pipelined (b) datapath

In addition to pipelined data paths, FPGA-based control systems frequently employ finite state machines (FSMs) to coordinate sequential operations and manage control flow. FSMs define a set of discrete states and transitions that are evaluated synchronously with the clock, allowing complex behaviors to be decomposed into a sequence of well-defined steps. From a timing perspective, each state typically corresponds to one or more clock cycles, and state transitions therefore contribute deterministically to the overall execution latency.

While pipelining and FSM-based sequencing are both clocked mechanisms, they serve distinct roles within an FPGA design. Pipelining introduces temporal parallelism by overlapping the processing of multiple data samples across different stages, whereas FSMs control the order and timing of operations by sequencing shared hardware resources. In practical FPGA-based control implementations, these two mechanisms are commonly combined: pipelined data paths are used for arithmetic-intensive computations, while FSMs supervise data movement, synchronization, and control flow. Together, they define a fixed and predictable execution schedule.

The trade-off between latency and throughput introduced by pipelining represents a fundamental design consideration in FPGA-based control systems. Increasing pipeline depth allows higher clock frequencies and improved throughput, at the cost of increased latency. Conversely, reducing pipeline depth lowers latency but may limit achievable clock frequency or resource efficiency. In control applications, the objective is therefore not to eliminate latency, but to ensure that it remains bounded, deterministic, and explicitly known, enabling its inclusion in the overall control design.

3.4 Streaming Dataflow and AXI-Stream Background

The implementation of high-performance control algorithms on FPGA platforms naturally leads to a streaming dataflow model, where data samples are processed sequentially as they propagate through a chain of processing blocks. In such architectures, each block operates on incoming data as soon as it becomes available, enabling efficient exploitation of pipelined data paths and deterministic timing behavior. To support this execution model in a standardized and modular manner, the AXI4-Stream protocol is widely adopted in FPGA-based systems.

AXI4-Stream is a point-to-point, unidirectional streaming interface designed to transfer data between a master and a slave without the need for address information. Communication is synchronous and governed by a global clock and reset, as shown conceptually in Figure 3.3. Unlike memory-mapped interfaces, AXI4-Stream is optimized for continuous data transfer, making it particularly well suited for signal-processing and control pipelines.

Data transfer over an AXI4-Stream interface is controlled by a handshake mechanism based on the signals TVALID and TREADY. The master asserts TVALID to indicate that valid data is available on the data bus, while the slave asserts TREADY to signal its ability to accept data. A transfer occurs only when both signals are asserted simultaneously during a clock cycle, as illustrated in Figure 3.3. This mechanism allows either side of the interface to stall the data flow without loss of synchronization, thereby supporting back-pressure in streaming pipelines.

An important characteristic of the AXI4-Stream handshake is that the assertion of TVALID must not depend on the value of TREADY. Once asserted, TVALID must remain high until a transfer takes place, and the associated data must remain stable during this period. This behavior ensures reliable data delivery and preserves determinism even in the presence of variable downstream processing delays. The timing relationships between TVALID, TREADY, and the data bus are exemplified in Figure 3.3.

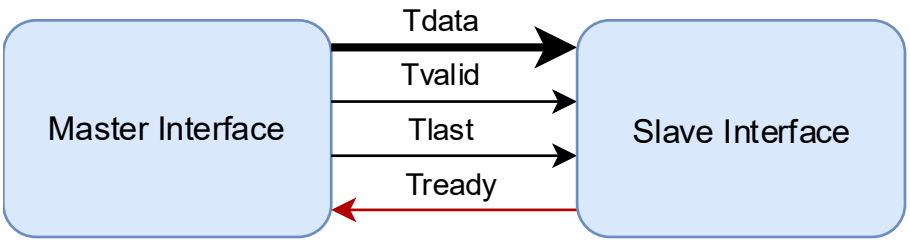


Figure 3.3 AXI4 Stream interface dataflow

The primary payload of the AXI4-Stream interface is conveyed through the TDATA signal, whose width is defined as an integer multiple of bytes. Additional sideband signals may be used to convey auxiliary information alongside the data stream. Among these, TLAST is commonly employed to indicate packet boundaries, enabling

the grouping of consecutive data samples into logical frames, as shown in Figure 3.3. While packetization is not always required in control applications, the availability of TLAST provides flexibility for extending the streaming model to more complex data structures.

From a system-level perspective, AXI4-Stream interfaces facilitate the construction of modular processing chains, where independent blocks can be connected without explicit knowledge of each other's internal timing. The handshake mechanism ensures correct data transfer even when processing latencies differ across blocks, allowing pipelined data paths and FSM-based control logic to coexist within a unified framework.

This chapter translated the controller requirements into an FPGA-compatible viewpoint by defining how signals are represented, transported, and computed under real-time constraints. In particular, the discussion of fixed-point formats and scaling establishes a consistent numerical interpretation for all measured and computed variables, while the treatment of bit-width growth, saturation, and rescaling outlines how mathematical operations can be implemented without overflow or loss of critical resolution. With these constraints in place, the next step is to validate that the complete control chain behaves as expected when transitioning from floating-point simulations to hardware-oriented models. Chapter 4 therefore presents the simulation workflow and FPGA-in-the-Loop framework used to progressively move from a continuous-time floating-point model to a fixed-point and FIL-validated design, providing a controlled environment to identify numerical and timing-related deviations before the final VHDL implementation.

4 Simulation and FPGA-in-the-Loop Framework

This chapter introduces the simulation and validation framework adopted in this work, with particular emphasis on the use of FPGA-in-the-Loop (FIL) techniques for the development and verification of digital motor control algorithms. The objective of this chapter is to describe how the proposed control strategies are progressively transitioned from idealized simulation environments toward hardware-realistic execution, while preserving a consistent control structure throughout all stages. In the FPGA-in-the-loop configuration adopted in this work, the control algorithm executes on the FPGA hardware, while the inverter and PMSM plant remain simulated, allowing hardware execution effects to be isolated without introducing power-stage nonidealities

In the context of digital motor control, conventional simulation approaches based solely on continuous-time models are insufficient to capture the effects introduced by sampled-data operation, computational delay, and hardware execution constraints. While continuous-time simulations remain useful for initial controller tuning and conceptual validation, the final control performance is ultimately determined by discrete-time execution on a digital platform. This is particularly relevant for control strategies such as deadbeat current control, whose behavior is explicitly defined in terms of sampling instants and discrete-time plant models.

FIL simulation represents an intermediate validation step between pure software simulation and full experimental testing. In an FIL setup, the control algorithm is synthesized and executed on the target FPGA hardware, while the plant model and supervisory logic remain in the simulation environment. This approach enables the controller to operate under realistic timing conditions, including clock-driven execution, pipeline latency, and interface handshaking, without requiring a fully instrumented power stage or physical motor.

The primary motivation for employing FIL in this work is to verify that the discrete-time control algorithms behave as expected when subjected to the deterministic execution model of FPGA hardware. Unlike processor-based platforms, FPGA implementations exhibit fixed and repeatable latency, which directly influences closed-loop dynamics. FIL simulation allows these effects to be observed and analyzed early in the design process, reducing the risk of discrepancies between simulated and real-time behavior.

Within the framework presented in this chapter, continuous-time simulations are first used only as a reference to validate the overall control structure. The focus then shifts to discrete-time simulations, which form the foundation of the controller design. The same discrete-time structure is subsequently employed for FPGA-in-the-Loop validation, ensuring methodological consistency across all stages. In this way, differences observed in later chapters can be attributed directly to numerical representation and hardware execution effects, rather than to changes in the underlying control formulation.

The results obtained using the simulation and FIL framework described in this chapter are presented and discussed in Chapter 6, where the performance of the proposed PI and deadbeat control strategies are evaluated under increasing levels of implementation realism.

4.1 Continuous Time Simulation as a Reference

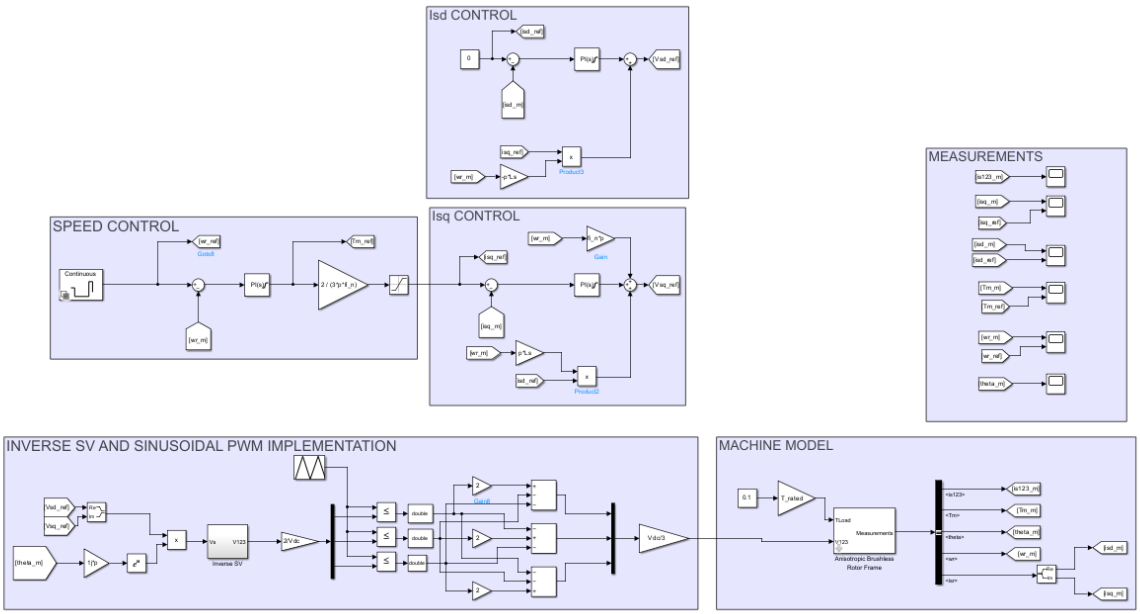


Figure 4.1 Continuous time control diagram

Figure 4.1 provides a high-level overview of the complete continuous-time control system implemented in Simulink and used as a reference in this work. The model includes the cascaded speed and current control loops, inverse coordinate transformations, pulse-width modulation, inverter, and PMSM plant. The figure is intended as an overall orientation map; each functional block is described individually in the following subsections to improve clarity.

The continuous-time model is formulated under idealized assumptions and is used solely to establish the control architecture and signal flow prior to the introduction of discrete-time execution and hardware-related constraints.

4.1.1 PMSM Model

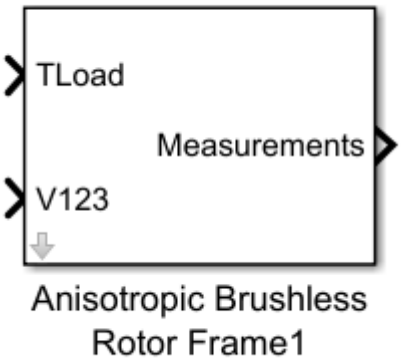


Figure 4.2 PMSM model block diagram used in simulations

The PMSM plant used in the continuous time Simulink model is based on the electrical and mechanical dynamics introduced in Section 2.2. The stator electrical equations are expressed in the synchronous rotating reference frame, while the mechanical dynamics relate the electromagnetic torque to rotor speed through the rotor inertia and viscous friction. Figure 4.2 shows the PMSM subsystem used in the simulation environment.

The motor parameters adopted in this work are summarized in Table 4.1. These parameters are used consistently across all simulation stages (continuous-time reference, discrete-time simulation, fixed-point simulation, and FPGA-in-the-loop), ensuring that any difference observed in later chapters can be attributed to execution and numerical effects rather than changes in the plant model.

Table 4.1 PMSM parameters

Parameter	Symbol	Value
Stator resistance	R_s	650 $m\Omega$
Stator inductance	L_s	1.2 mH
Permanent magnet flux linkage	φ_n	4.55 mWb
Number of pole pairs	p	4
Rotor inertia	J	$6.7014 \times 10^{-6} kgm^2$
Viscous friction coefficient	B	$1.8026 \times 10^{-5} Nms$

4.1.2 Inverter and PWM Modeling

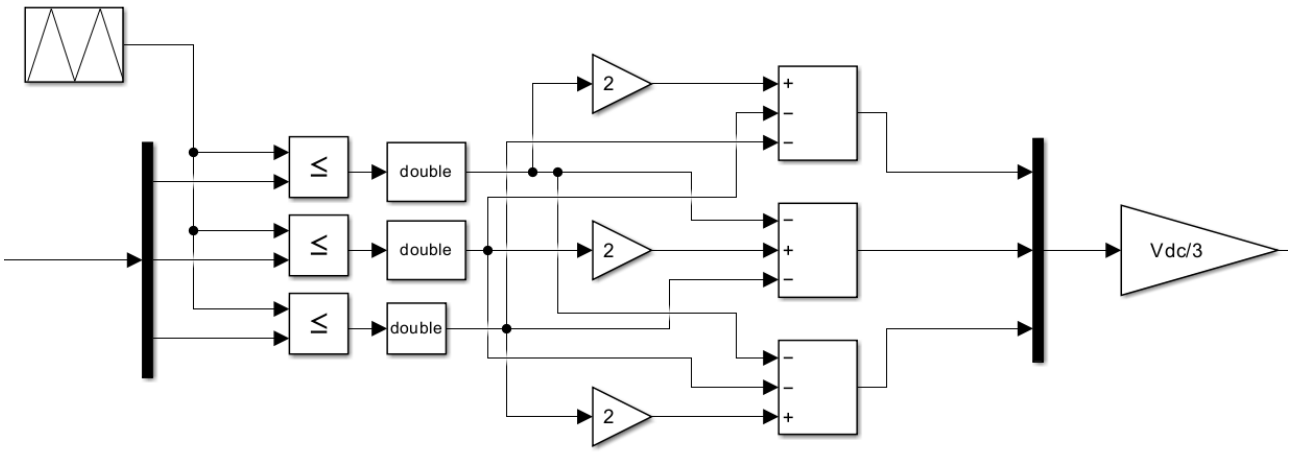


Figure 4.3 Inverter diagram

The PMSM is driven by a two-level three-phase voltage source inverter operated using SPWM. In the continuous time Simulink model, the inverter and modulation stage are represented under idealized conditions to focus on the control design rather than on power electronic nonidealities. Figure 4.3 shows the inverter and PWM subsystem used in the simulation.

The inverter is supplied by a constant DC-link voltage of $V_{dc} = 24\,\mathrm{V}$ which corresponds to the nominal DC bus voltage of the experimental setup and the rated supply voltage of the motor. This value is used consistently across all simulation stages

and experimental validation to ensure coherence between the control design and the physical system.

The sinusoidal PWM strategy generates switching signals by comparing three-phase voltage reference signals with a high-frequency triangular carrier. This modulation technique is selected due to its simplicity, widespread adoption in industrial motor drive applications, and straightforward mapping between reference voltages and the fundamental component of the applied phase voltages. These properties make SPWM particularly suitable for control-oriented design and FPGA-based implementation.

The switching frequency is selected as $f_{\mathrm{sw}} = 10\,\mathrm{kHz}$. This value represents a compromise between current ripple, control bandwidth, and implementation constraints. In particular, the switching frequency is chosen to be sufficiently higher than the bandwidth of the current control loops, allowing the inverter dynamics to be neglected during controller design and enabling the use of an average voltage model. At the same time, a switching frequency of 10 kHz limits switching losses and computational burden, making it suitable for real-time execution on FPGA hardware.

In the continuous-time reference model, the inverter is assumed to apply the commanded phase voltages instantaneously and without distortion. Dead time, semiconductor voltage drops, switching delays, and PWM-induced ripple are neglected. This abstraction allows the controller structure and parameters designed in Chapter 2 to be evaluated independently of power electronic nonidealities, which are progressively introduced in the discrete-time and FPGA-in-the-loop stages.

4.1.3 Inverse Space Vector Transformation

The voltage references generated by the current control loops are expressed in the synchronous rotating reference frame as direct- and quadrature-axis components. In this work, the transformation from the rotating reference frame to three-phase voltage references is implemented using a space-vector formulation. The corresponding Simulink implementation is illustrated in Figure 4.4

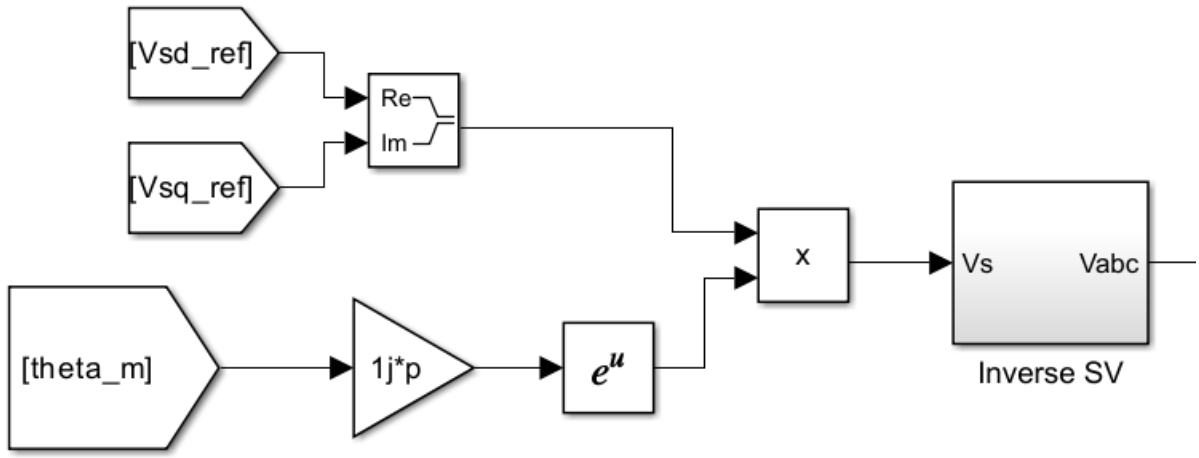


Figure 4.4 Inverse space vector transformation

As shown in Figure 4.4, the direct- and quadrature-axis voltage components are first combined into a complex voltage vector,

$$\underline{v}_{dq} = v_d + jv_q \tag{4.1}$$

which is then rotated into the stationary reference frame through multiplication by the complex exponential

$$\underline{v}_{\alpha\beta} = \underline{v}_{dq}\, e^{jp\theta_m} \tag{4.2}$$

where θ_m denotes the measured mechanical rotor position and p is the number of pole pairs. This operation is mathematically equivalent to the inverse Park transformation and yields the stationary-frame voltage space vector.

The stationary voltage vector is subsequently processed by the inverse space-vector mapping shown in Figure 4.5. This block converts the complex stationary-frame voltage into three-phase voltage references (v_a, v_b, v_c), which are then supplied to the PWM modulator.

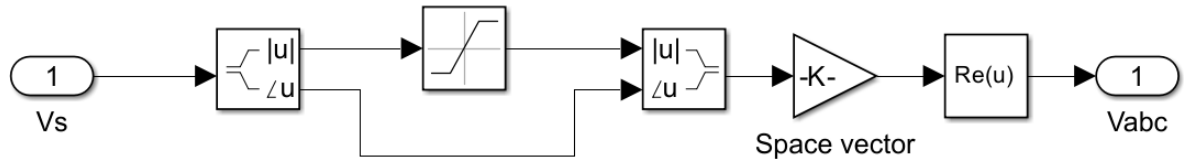


Figure 4.5 Internal structure of the inverse space vector transformation

The inverse space-vector mapping employed in this work is based on a complex-vector formulation. The stationary-frame voltage vector is converted into three-phase voltage

references by projection onto a set of phase-shifted basis vectors. This operation can be expressed as

$$\begin{bmatrix} v_a \\ v_b \\ v_c \end{bmatrix} = \Re \left\{ \underline{v}_{\alpha\beta} \begin{bmatrix} 1 \\ e^{-j\frac{2\pi}{3}} \\ e^{-j\frac{4\pi}{3}} \end{bmatrix} \right\} \tag{4.3}$$

where $\Re\{\cdot\}$ denotes the real-part operator. The complex coefficients

$$\mathrm{K} = \left[1 \; e^{-j\frac{2\pi}{3}} \; e^{-j\frac{4\pi}{3}}\right] \tag{4.4}$$

define the phase-axis basis vectors separated by 120° in the complex plane. This formulation is mathematically equivalent to the inverse Clarke transformation commonly used in field-oriented control, and it produces balanced three-phase voltage references from the stationary-frame space vector. This mapping is equivalent to the inverse Clarke transformation commonly used in field-oriented control schemes.

By adopting a space-vector-based formulation, the inverse coordinate transformations are expressed in a compact and mathematically concise manner. This representation preserves the same functional behavior as the conventional inverse Park and Clarke transformations while providing a structure that is well suited for simulation.

4.1.4 Current Control Design

4.1.4.1 Q-Axis Current Controller

The quadrature-axis current control loop regulates the torque-producing component of the stator current in the synchronous rotating reference frame. The control structure implemented in the continuous-time Simulink model is shown in Figure 4.6 and follows the design methodology introduced in Section 2.5.1.

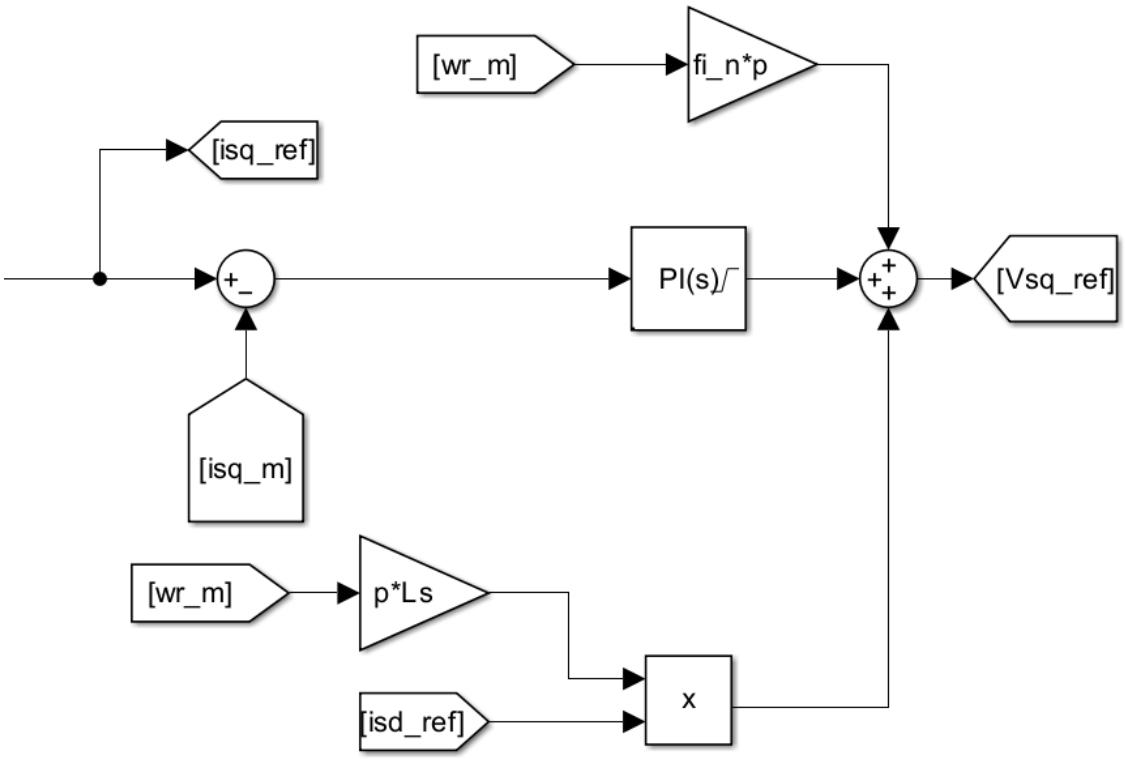


Figure 4.6 Continuous time Q-Axis current controller block diagram

The measured quadrature-axis current i_{sq} is compared with its reference value i_{sq}^{*}, generated by the outer speed control loop. The resulting current error is processed by a continuous-time PI controller. As derived in Chapter 2, the general transfer function of the PI controller is given in Equation (2.24). After applying the design procedure described therein, the controller implemented in the continuous-time Simulink model can be expressed as

$$C_{i_q}(s) = 3.3978 + 2797.5\frac{1}{s} \tag{4.5}$$

The values of these proportional and integral gains are selected according to the tuning procedure described in Section 2.5.1.

The bandwidth of the quadrature-axis current controller is chosen as $f_{c,i} = 500$ Hz corresponding to an angular bandwidth of $\omega_{c,i} = 2\pi \cdot 500$ rad/s. This bandwidth is selected to ensure fast current regulation while maintaining sufficient separation from the inverter switching frequency, which is set to 10 kHz, as discussed in Section 4.1.2. Under this condition, the inverter dynamics can be neglected, and the applied voltages can be assumed to follow their reference values.

In the controller design presented in Section 2.5.1, the electrical dynamics of the PMSM in the synchronous rotating reference frame were approximated as an RL load with

additional speed-dependent cross-coupling terms. Under this assumption, the voltage equations along the d- and q-axes include coupling components proportional to the electrical angular speed and the orthogonal current component. If left uncompensated, these terms introduce dynamic interaction between the current control loops.

To address this effect, feedforward decoupling terms derived from the RL-based plant model are incorporated into the i_q current controller, as shown in Figure 4.6. These terms compensate for the speed-dependent cross-coupling introduced by the synchronous reference frame formulation, allowing the PI regulator to operate on an approximately decoupled first-order system. As a result, the i_q control loop can be designed using standard linear control techniques, with the closed-loop bandwidth selected independently of the rotor speed.

In the continuous-time reference model, the feedforward compensation is assumed to be exact, and the resulting current dynamics closely match the ideal RL load behavior assumed during controller design in Section 2.5.1.

4.1.4.2 D-Axis Current Controller

The direct-axis current control loop regulates the flux-producing component of the stator current in the synchronous rotating reference frame. The control structure implemented in the continuous time Simulink model is shown in Figure 4.7.

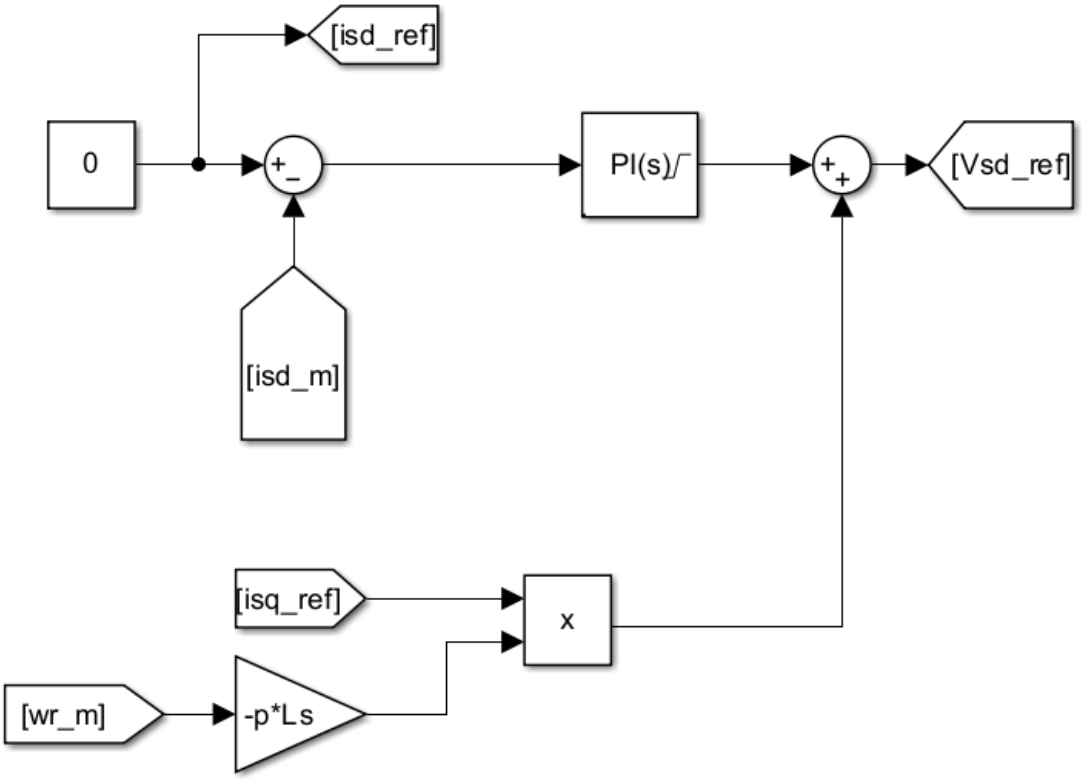


Figure 4.7 Continuous time D-Axis current controller block diagram

In this work, the direct-axis current reference is set to $i_{sd}^{*} = 0$, corresponding to standard field-oriented control operation without flux weakening. This choice is appropriate for surface-mounted PMSMs, for which the direct- and quadrature-axis inductances are approximately equal. Under this condition, torque production is maximized by allocating the stator current entirely along the q-axis, while maintaining a zero direct-axis current.

As discussed in the previous subsection, the electrical dynamics of the PMSM in the synchronous rotating reference frame can be approximated as an RL load with additional speed-dependent cross-coupling terms. As mentioned in Section 2.2, for the surface-mounted PMSM considered in this work, the stator parameters are identical along the direct and quadrature axes. Under this condition, the d- and q-axis current dynamics are equivalent. Consequently, the same PI controller structure used for the $i_q$current loop can be adopted for the i_{sd} loop. The controller is therefore defined according to the general PI formulation given in Equation (4.5), with gains selected based on the same bandwidth considerations.

4.1.5 Speed Control Design

The outer speed control loop regulates the mechanical speed of the PMSM and provides the torque-producing current reference for the inner current control loops. The control structure implemented in the continuous-time Simulink model is shown in Figure 4.8.

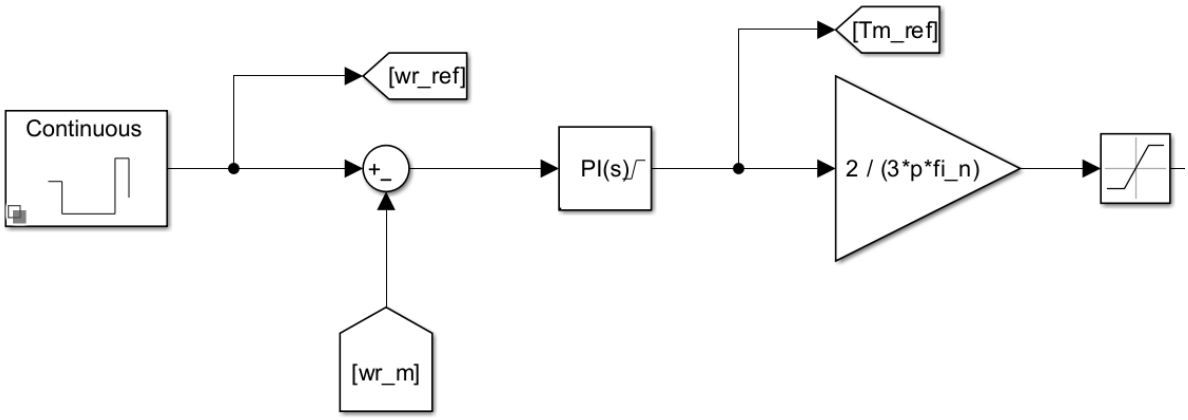


Figure 4.8 Continuous time speed controller block diagram

The measured rotor speed ω_{r_m} is compared with the reference speed ω_r^{*}, and the resulting speed error is processed by a continuous-time PI controller. As derived in Section 2.5.4, the general PI controller formulation is given in Equation (2.24). After applying the design procedure described therein, the speed controller implemented in the continuous-time model can be expressed as

$$C_\omega(s) = 0.0019 + 0.0606\frac{1}{s} \tag{4.6}$$

The output of the speed controller corresponds to the electromagnetic torque reference T_m^*. This torque reference is converted into a quadrature-axis current reference i_q^* using the PMSM torque relationship given in Equation (2.48). For the motor considered in this work, the resulting conversion gain is equal to 36.63, as shown in Figure 4.8.

The bandwidth of the speed control loop is selected as $f_{c,\omega} = 50\,\text{Hz}$, which is one decade lower than the bandwidth of the inner current control loops. This choice ensures a clear time-scale separation between the electrical and mechanical dynamics, effectively decoupling the speed control loop from the inner current regulation. Under this assumption, the current loops can be considered instantaneous from the perspective of the speed controller, and the electromagnetic torque is assumed to track its reference without significant delay. Such bandwidth separation is a standard and necessary condition for the stability, robustness, and modular design of cascaded speed–current control structures.

In the continuous time reference model, the speed controller is assumed to operate without sampling, computational delay, or numerical constraints. This idealized formulation allows the dynamic interaction between the speed and current loops to be evaluated prior to discretization and FPGA-based implementation.

4.2 Discrete Time Control Design and Simulation

This section presents the discrete time formulation of the control system introduced in Section 4.1. At this stage, the continuous-time controllers are reformulated for discrete-time execution and implemented using floating-point arithmetic in Simulink. The inverter and the PMSM plant remain modeled in continuous time, and their dynamics are therefore unchanged with respect to the continuous-time reference model.

The discrete time simulation serves as an intermediate design step between the ideal continuous time formulation and the FIL implementation which will be discussed in Section 4.3. By discretizing only, the control algorithms while retaining a continuous-time plant, the effects of sampling and discrete-time execution can be evaluated independently of hardware-related constraints such as fixed-point quantization, pipeline latency, and interface delays.

Both proportional–integral and deadbeat current control strategies are considered in discrete time. The same control architecture and parameter set established in the continuous-time design are retained wherever possible, ensuring consistency across all simulation stages and enabling a systematic assessment of discretization effects.

4.2.1 Sampling Rate Selection

The selection of the sampling frequency plays a critical role in the discrete time implementation of the control system, as it directly affects closed-loop performance, stability margins, and the accuracy with which the continuous-time dynamics are captured. In this work, the sampling rate is selected in relation to both the inverter switching frequency and the bandwidth of the control loops defined in Section 4.1.

The inverter switching frequency is set to 10 kHz, while the bandwidth of the inner current control loops is selected as 500 Hz. To ensure sufficient time resolution and to minimize discretization-induced performance degradation, sampling frequency is chosen as $f_s = 100$ kHz, corresponding to a sampling period of $T_s = 10\ \mu s$. This choice provides a clear separation between the controller update rate and the current-loop bandwidth, allowing the discrete-time controllers to closely approximate their continuous time behavior while preserving adequate phase margin.

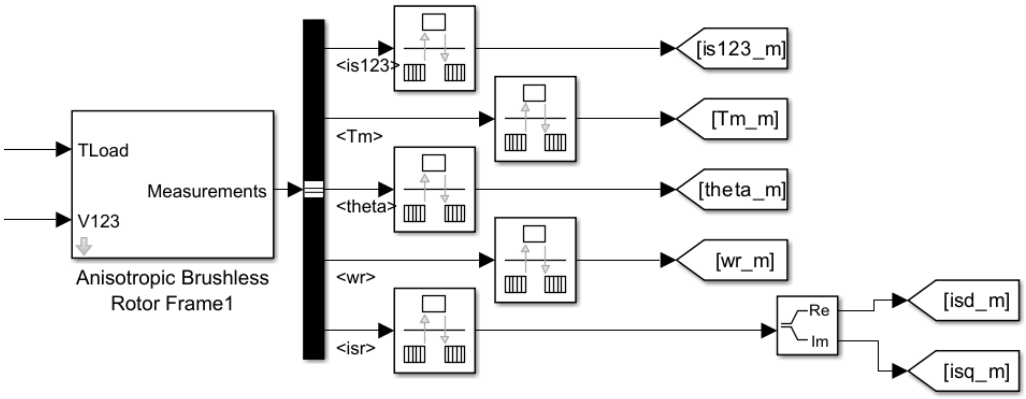


Figure 4.9 Sampling of continuous time measurements

As shown in Figure 4.9, the sampling of continuous-time measurements is implemented using Simulink Rate Transition blocks. These blocks enforce a deterministic transfer of signals from the continuous-time PMSM and inverter models to the discrete-time control algorithms. The Rate Transition mechanism ensures that measurement signals are sampled at the controller update rate and held constant between sampling instants, thereby defining the timing reference for all discrete-time control computations.

From an implementation perspective, the selected sampling frequency is compatible with the data acquisition capabilities of the target FPGA platform. In particular, the XADC peripheral available on the device can be configured with sampling rates of up to 1 MSPS, which provides ample margin for acquiring current and voltage measurements at the selected controller update rate. As a result, the 100 kHz sampling

frequency adopted in the discrete-time simulation can be retained in the FPGA-in-the-loop implementation without being limited by the analog-to-digital conversion stage.

In the discrete-time floating-point simulation presented in this section, the sampling frequency is assumed to be fixed and perfectly periodic.

4.2.2 Current Controllers in Discrete Time

4.2.2.1 Discrete Time PI Current Controllers

The discrete-time PI current controller is obtained by discretizing the continuous-time PI structure introduced in Section 4.1.4, while preserving the same control architecture, parameter values, and decoupling strategy. The discrete time implementation of the quadrature-axis current controller is shown in Figure 4.10.

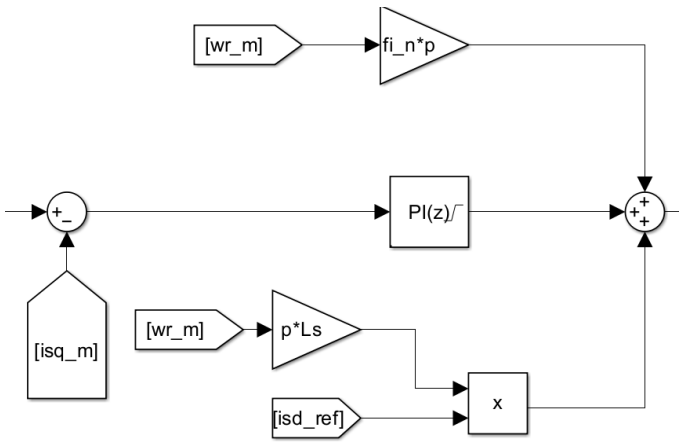


Figure 4.10 Discrete time Q-Axis PI current controller

In discrete time, the i_{sq} current controller is implemented in parallel PI form and operates at the sampling frequency selected in Section 4.2.1. The controller gains are identical to those used in the continuous-time design. The discrete-time PI controller can be expressed as

$$C(z) = K_p + K_i \frac{T_s}{2} \frac{z+1}{z-1} \tag{4.7}$$

As in the continuous-time formulation, feedforward decoupling terms derived from the RL-based plant model discussed in Section 2.5.1 are retained in the discrete-time implementation. These terms compensate for the speed-dependent cross-coupling between the current components and allow the PI regulator to operate on an approximately decoupled first-order system.

To prevent integrator windup under actuator saturation, an anti-windup mechanism is enabled in the discrete-time PI controller. The same anti-windup strategy is also employed in the continuous-time simulations presented in Section 4.1, ensuring consistent controller behavior across simulation stages.

The discrete time direct-axis PI current controller is given in Figure 4.11. The controller is implemented using the same PI structure, discretization method, and gain values as the quadrature-axis controller. As discussed in Section 4.1.4, this choice is justified by the symmetric electrical characteristics of the surface-mounted PMSM considered in this work. Consequently, no additional modifications are required for the i_{sd}controller apart from the reference signal and the corresponding feedforward decoupling terms.

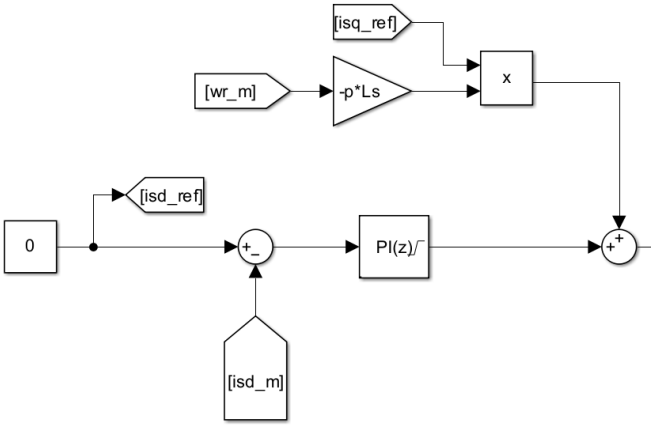


Figure 4.11 Discrete time D-Axis PI current controller

4.2.2.2 Discrete Time Deadbeat Current Controllers

In addition to the discrete time PI current controller, a deadbeat current control strategy is implemented and evaluated in discrete time. Unlike the PI controller, deadbeat control is inherently a discrete-time control method and does not admit a meaningful continuous-time formulation. For this reason, deadbeat current control was not included in the continuous-time simulations presented in Section 4.1 and is introduced for the first time at the discrete-time stage.

The control structure of the deadbeat current controller is illustrated in Figure 4.12 for the quadrature-axis current loop. In this configuration, the PI controller used in the discrete-time implementation is replaced by a deadbeat controller, while all remaining elements of the control architecture, including feedforward decoupling terms, saturation blocks, and reference generation, are preserved. This allows a direct comparison between PI and deadbeat current control under identical operating conditions.

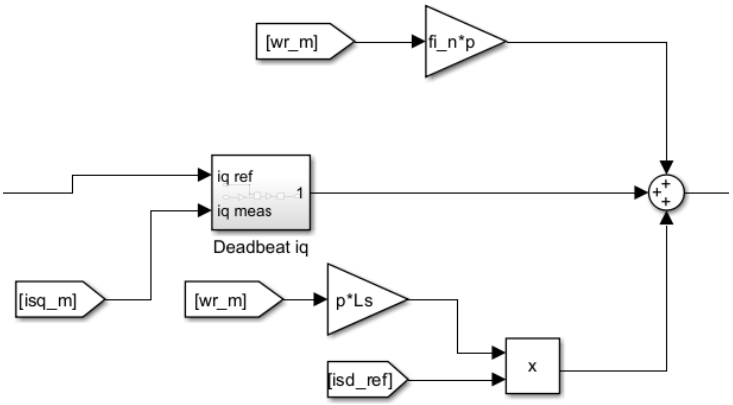


Figure 4.12 Discrete time Q-Axis deadbeat current control diagram

The internal structure of the deadbeat controller is shown in Figure 4.13. The control law is derived directly from the discrete-time model of the PMSM current dynamics obtained by discretizing the RL load approximation discussed in Section 2.5.2.

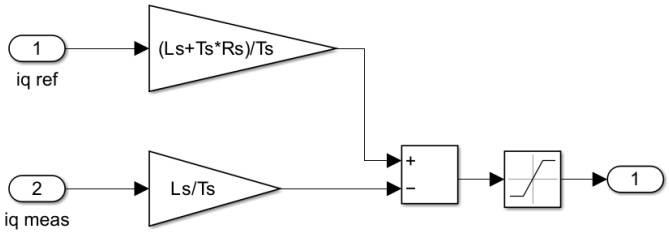


Figure 4.13 Internal Structure of deadbeat Current Control

The resulting deadbeat control law computes the required voltage command such that the current error is driven to zero within a single sampling interval under ideal modeling assumptions. Using the Equation (2.56) the controller can be expressed as

$$v[n+1] = 120.65\, i^{*}[n] - 120\, i[n] \tag{4.8}$$

where $i^{*}[n]$ and $i[n]$ denote the reference and measured current at the k-th sampling instant, respectively. For the system considered in this work, the discrete-time model coefficients are determined by the stator resistance, stator inductance, and the selected sampling period

As with the PI controller, the deadbeat current controller is implemented independently for both the i_{sq} and i_{sd} current loops. The direct-axis implementation is illustrated in Figure 4.14. The same internal structure in Figure 4.13 is used for the direct-axis implementation. Consistent with the previous designs, the direct-axis current reference is set to zero, and the same decoupling terms are applied to compensate for speed-dependent cross-coupling effects.

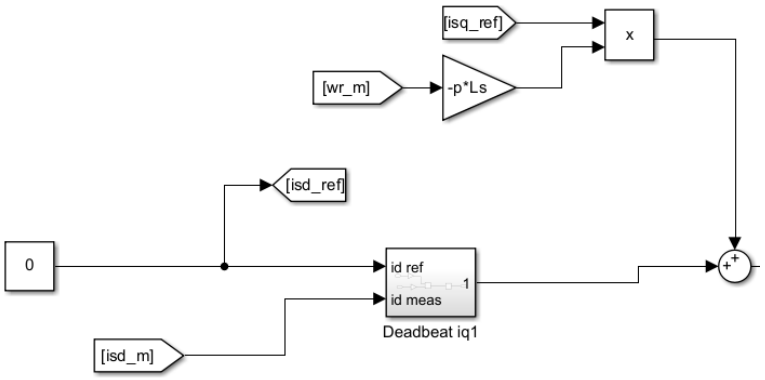


Figure 4.14 Discrete time D-Axis deadbeat current control diagram

In the discrete-time floating-point simulations presented in this section, the deadbeat controller is assumed to operate under ideal conditions, with exact knowledge of the plant parameters and without quantization or computational delay.

4.2.3 Discrete Time Speed Controller

The speed control loop is implemented in discrete time using the same execution framework adopted for the discrete-time current controllers discussed in Section 4.2.2. The discrete time implementation of the speed controller is shown in Figure 4.15. As in the continuous-time formulation, the speed controller operates as the outer loop of the cascaded control structure and generates the torque-producing current reference for the inner current control loops.

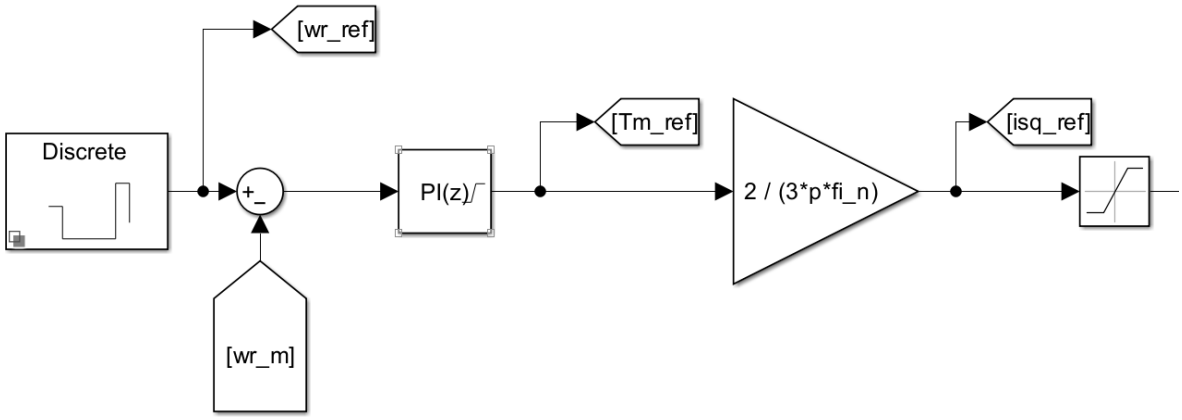


Figure 4.15 Discrete time speed controller block diagram

The discrete-time speed controller is implemented as a PI controller operating at the sampling frequency selected in Section 4.2.1. The controller gains are identical to those used in the continuous-time design, and no re-tuning is performed at this stage. The discrete-time PI speed controller can be expressed as

$$C(z) = 0.0019 + 0.0606\,\frac{10^{-5}}{2}\,\frac{z+1}{z-1} \tag{4.9}$$

The PI controller is discretized using the trapezoidal (Tustin) method, ensuring close correspondence between the continuous and discrete time frequency responses within the operating bandwidth of the speed loop.

Due to the significant separation between the speed-loop bandwidth and the bandwidth of the inner current control loops, the electrical dynamics can be considered instantaneous from the perspective of the speed controller. This time-scale separation effectively decouples the speed control loop from the inner current dynamics, allowing the discrete-time speed controller to be designed and analyzed independently of the current control implementation.

An anti-windup mechanism is retained in the discrete-time speed controller to prevent integrator windup under actuator saturation, consistent with the continuous-time formulation. As in the discrete-time current control simulations, floating-point arithmetic is used and no additional computational delay or quantization effects are modeled at this stage.

4.3 FIL Implementation

Following the discrete-time formulation and floating-point validation of the control system presented in the previous section, the focus now shifts toward a hardware-consistent realization suitable for FPGA execution. At this stage, the control algorithms themselves are no longer modified; instead, attention is directed to how measured quantities, internal control signals, and voltage references are represented, processed, and propagated within a fixed-point, clock-driven architecture. This transition introduces practical constraints related to numerical representation, execution latency, and timing closure, which are inherent to FPGA-based implementations. The objective of this section is therefore to describe how the discrete-time control structure is mapped onto a hardware-aware framework while preserving the same control logic, sampling rate, and signal flow established in the simulation environment.

4.3.1 Fixed-Point Selection for Measured Quantities

The selection of fixed-point formats for measurement signals represents a critical design step in the transition from discrete-time simulation to FPGA-based execution. In a hardware-implemented control system, measured quantities constitute the entry point of the control pipeline, and any limitation in numerical range or resolution introduced at this stage propagates through all subsequent computations. Consequently, fixed-point formats cannot be chosen arbitrarily; they must simultaneously accommodate the physical operating limits of the motor, the resolution

and dynamic range of the measurement hardware, and the numerical requirements of the control algorithms. In this work, the fixed-point representations of current, speed, and position measurements are selected through a systematic process that balances physical constraints, hardware capabilities, and control performance considerations.

The stator current measurement range is defined based on the absolute maximum allowable current of the motor, which is taken as 4.81 A in this work. In addition to this physical constraint, current measurements are acquired through the on-chip XADC peripheral, which provides a 16-bit digital output while internally operating with a 12-bit successive approximation register architecture. For this reason, the fractional resolution of the fixed-point format is chosen such that its least significant bit corresponds to a resolution comparable to the effective quantization step of the XADC, avoiding unnecessary precision that would not translate into improved measurement accuracy.

In the FPGA-in-the-loop simulation environment, the conversion between physical units and fixed-point representation is handled implicitly by the simulation framework; therefore, no explicit normalization or rescaling logic is introduced at this stage of the design. The selected formats are instead verified through simulation to ensure that the full current range remains representable and that quantization effects do not degrade control performance, while preserving consistency between simulated numerical behavior and the intended hardware execution.

Similarly, the speed measurement range is defined based on the motor's nominal operating speed. The rated mechanical speed of the machine is 4000 rpm, corresponding to approximately 420 rad/s, and this value is used as the reference upper bound for speed normalization to cover the full operating envelope while maintaining sufficient numerical resolution at low and medium speeds.

In the FPGA-in-the-loop implementation, a total of five measured signals are processed by the control pipeline. Among these, the three phase current measurements share identical numerical characteristics and are therefore represented using the same fixed-point format. The stator currents are represented using a signed 16-bit fixed-point format with 12 fractional bits, corresponding to a Q3.12 representation. This format provides sufficient dynamic range to accommodate the maximum expected current while offering high fractional resolution, ensuring that quantization at the measurement stage does not adversely affect the inner current control dynamics.

The mechanical speed measurement is represented using a signed 16-bit fixed-point format with 4 fractional bits, corresponding to a Q11.4 representation. Due to the slower dynamics of the mechanical subsystem and the significantly lower bandwidth of the speed control loop compared to the current loops, the speed signal is less sensitive to quantization. Consequently, allocating fewer fractional bits is sufficient to preserve accurate speed regulation while avoiding unnecessary numerical precision.

The electrical rotor position is represented using an unsigned 16-bit fixed-point format with 12 fractional bits, corresponding to a Q4.12 representation. Although the electrical angle is inherently periodic and could be represented with fewer bits, a 16-bit word length is intentionally maintained to preserve numerical consistency across all measurement signals. The increased fractional resolution reduces phase quantization effects in coordinate transformations, which is particularly important for the Park and inverse Park transformations used in the current control loops. This choice improves numerical robustness while maintaining uniform data width throughout the control pipeline. For clarity, the fixed-point representations adopted for each measured quantity in the FPGA-in-the-loop implementation are summarized in Table 4.2.

Table 4.2 Fixed-point representation of measurement signals used in FPGA-in-the-loop simulation

Signal	Physical Quantity	Signed / Unsigned	Q-Format	Total Bits	Fractional Bits
i_a	Phase current (A)	Signed	Q3.12	16	12
i_b	Phase current (A)	Signed	Q3.12	16	12
i_c	Phase current (A)	Signed	Q3.12	16	12
ω_r	Mechanical speed (rad/s)	Signed	Q11.4	16	4
θ_m	Rotor position (rad)	Unsigned	Q4.12	16	12

During fixed-point implementation, arithmetic operations such as multiplication and accumulation inherently lead to an increase in bit width. In particular, the multiplication of two fixed-point numbers results in a product whose word length is equal to the sum of the operand word lengths, while the number of fractional bits is equal to the sum of the fractional bits of the operands. If left unaccounted for, this growth may lead to numerical overflow or unnecessary resource usage in hardware.

To ensure numerical correctness, all internal arithmetic operations within the control pipeline are therefore performed using extended precision. Intermediate results are allowed to grow in word length during computation, and explicit rescaling is applied only at well-defined pipeline boundaries. This approach avoids premature truncation and preserves numerical accuracy while maintaining deterministic execution.

As a representative example, consider the proportional term of the current PI controller, where the current error $e[k]$ represented in Q3.12 format is multiplied by a proportional gain K_p represented in Qx.y format. The resulting product temporarily occupies a wider fixed-point format with increased integer and fractional resolution. After the multiplication, the result is rescaled by discarding the appropriate number of fractional bits and, if necessary, saturated before being stored back into a 16-bit fixed-point representation compatible with the rest of the control pipeline.

By systematically managing bit-width growth in this manner, numerical precision is preserved without compromising hardware efficiency. The same strategy is applied consistently across all arithmetic operations in the controller, including integral accumulation and coordinate transformations, ensuring predictable behavior and timing closure in the FPGA-based implementation.

4.3.2 Pipeline-Aware and Timing-Constrained Design

To ensure correct and deterministic execution on FPGA hardware, the proposed control system is structured according to a modular design philosophy. The overall control algorithm is decomposed into a set of functional modules, each corresponding to a well-defined stage of the control loop, such as angle processing, current transformation, current control, voltage reconstruction, and PWM generation. This modular organization improves design clarity and allows each block to be analyzed independently in terms of numerical precision, latency, and hardware complexity, while preserving a clear signal flow across the complete control pipeline. The resulting modular control architecture, including the explicit insertion of delay elements for signal alignment, is shown in Figure 4.16.

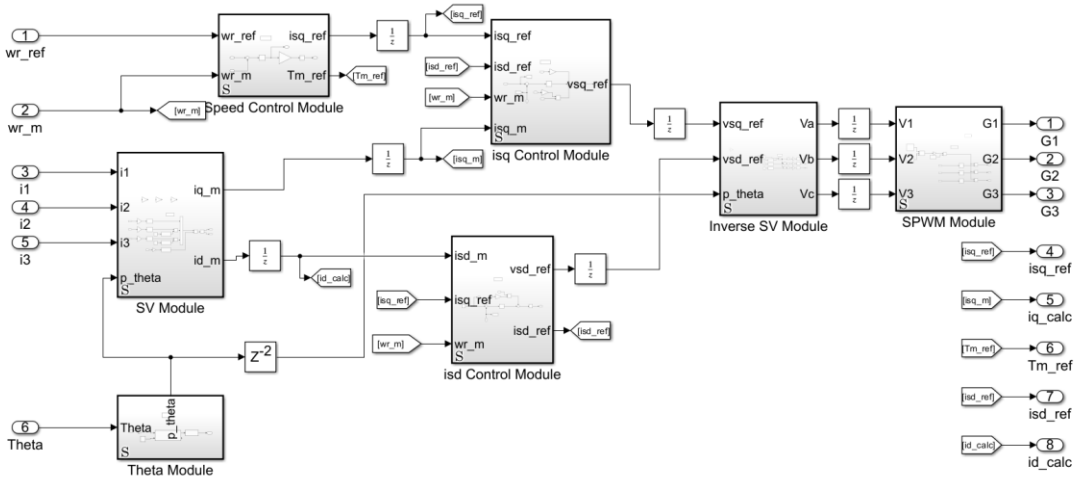


Figure 4.16 Timing aware modular control architecture

In FPGA-based implementations, arithmetic operations are inherently distributed over multiple clock cycles due to fixed-point multiplications, accumulations, and the use of iterative computational blocks. As a result, each module introduces a

deterministic processing latency. Rather than treating these latencies as implicit synthesis artifacts, they are explicitly accounted for at the system level through the insertion of sample delays between modules. These delays act as pipeline stages that align signals belonging to the same control iteration, ensuring that measured quantities, reference signals, and controller outputs remain temporally consistent as they propagate through the control architecture. By explicitly modeling these delays, pipelining becomes an intentional design mechanism rather than an unintended consequence of hardware execution.

The timing-aware nature of the design is further reflected in the selection of the system clock frequency. During HDL code generation, timing performance was evaluated using HDL Coder by targeting different clock frequencies and analyzing the resulting timing reports. Based on the Worst Negative Slack (WNS) values obtained from this analysis, a clock frequency of 25 MHz was identified as the highest reliably achievable frequency for the complete control architecture. At this frequency, all timing constraints are satisfied without violations, ensuring stable operation and timing closure across all modules.

4.3.3 FPGA Optimized Electrical Angle Calculation

In FOC, accurate synchronization between the stator reference frame and the rotor magnetic field is essential for achieving decoupled control of torque and flux. This synchronization is established through the rotor electrical angle, which defines the orientation of the rotating reference frame in which the current control loops operate. All current regulations in the proposed control architecture are therefore performed in the synchronous rotating reference frame, making the computation of the electrical angle and the associated coordinate transformations a fundamental part of the control pipeline. In the FPGA-based implementation considered in this work, these operations are executed deterministically using fixed-point arithmetic and form the first processing stage following the acquisition of current and position measurements.

The rotor position measurement available to the control system corresponds to the mechanical angle θ_m. To synchronize the rotating reference frame with the rotor magnetic field, this quantity must be converted into the electrical angle θ_e, which is obtained by scaling the mechanical angle by the number of pole pairs. The resulting electrical angle is periodic with a period of 2π and must therefore be wrapped to remain within a bounded interval. In the proposed implementation, this operation is realized by applying a modulo-2π to the scaled angle, as illustrated in Figure 4.17.

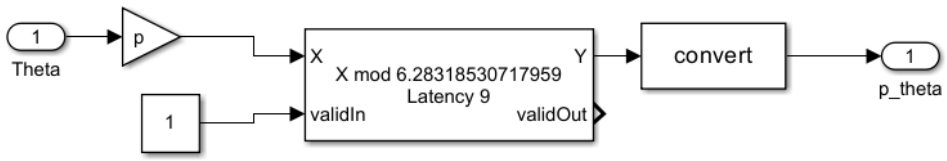


Figure 4.17 Mechanical to electrical angle conversion with Modulo 2π wrapping

From an implementation perspective, a direct modulo operation using a generic arithmetic block is not suitable for FPGA synthesis, particularly when the modulus is a non-power-of-two constant such as 2π. Basic modulo implementation would require division-based arithmetic or iterative comparison logic, which leads to inefficient hardware utilization and may not be supported by standard HDL synthesis flows. For this reason, a conventional modulo block was not employed in the proposed design.

Instead, the electrical angle wrapping is implemented using the `Modulo by Constant (HDL Optimized)` block available in the Simulink HDL library. This block is specifically designed to generate synthesizable and resource-efficient hardware for modulo operations with constant divisors. Internally, it relies on deterministic arithmetic structures tailored for fixed-point execution, avoiding division operators and conditional branching while ensuring predictable latency. As a result, the electrical angle is reliably confined to the $[0,2\pi)$ interval in a manner that is fully compatible with FPGA implementation constraints. The wrapped electrical angle is subsequently reformatted to the internal fixed-point representation and used as the synchronization variable for the coordinate transformations in the current control loops.

4.3.4 Three Phase Current into the dq Reference Frame Transformation

The stator phase currents measured at the inverter output are naturally expressed in the three-phase stationary reference frame. However, in FOC current regulation is performed in a synchronous rotating reference frame aligned with the rotor magnetic field. Consequently, the measured phase currents must be transformed into direct- and quadrature-axis components prior to entering the current control loops.

In the proposed implementation, this transformation is realized using a space-vector formulation rather than explicit Clarke and Park transformation stages. This approach allows stationery and rotating frame transformations to be merged into a single, hardware-efficient structure that is well suited for FPGA-based execution. The resulting processing chain is illustrated in Figure 4.18.

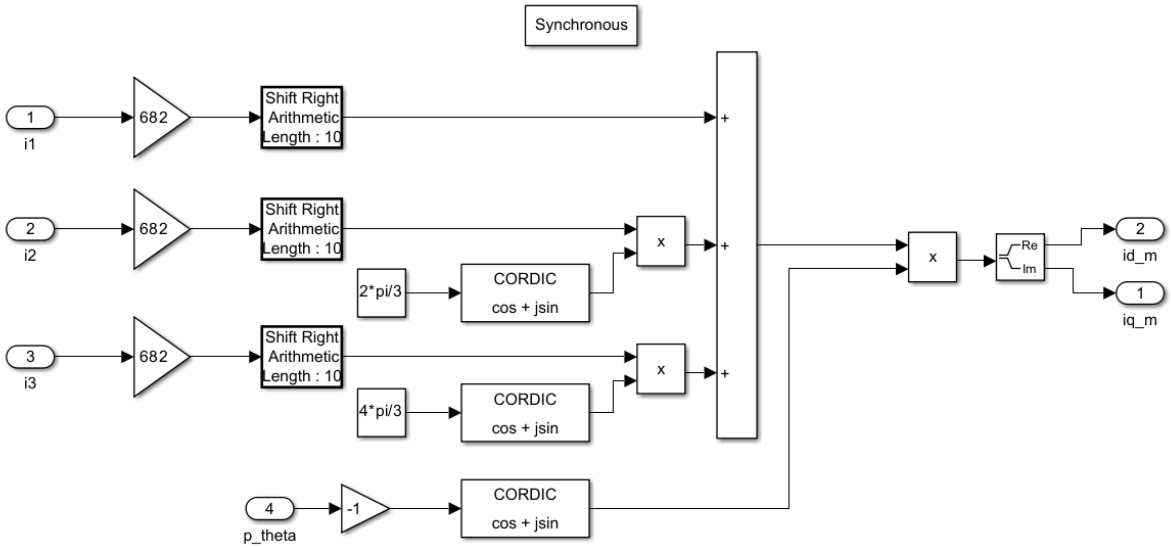


Figure 4.18 Three phase to dq frame current transformation

Each measured phase current is first scaled by a factor of $2/3$ which is implemented as multiplication by 682 followed by 10 bits shift right operation and projected onto a corresponding unit vector in the complex plane. These unit vectors are phase-shifted by 0, $2\pi/3$, and $4\pi/3$, respectively, reflecting the spatial displacement of the three stator phases. The required sine and cosine values associated with these fixed phase offsets are generated using dedicated CORDIC trigonometric blocks. The resulting complex-valued products are then summed to form a single stationary current space vector, which represents the combined contribution of the three phase currents in the stationary reference frame. Although the phase shifts could be implemented using fixed constant coefficients, the CORDIC-based implementation was intentionally preserved. This choice ensures consistent numerical scaling and avoids fixed-point type inconsistencies encountered during HDL Coder generation when combining constant multipliers with CORDIC-generated outputs.

This choice ensures consistent numerical scaling, identical pipeline latency among branches, and avoids fixed-point type inconsistencies encountered during HDL Coder generation when combining constant multipliers with CORDIC-generated outputs. The resulting architecture remains homogeneous and facilitates verification of the fixed-point datapath.

To obtain the current components in the synchronous rotating reference frame, the stationary current space vector is subsequently rotated by the negative of the electrical rotor angle. This rotation is implemented through a complex multiplication with the exponential term $e^{-j\theta_e}$, where the electrical angle θ_e is computed as described in the previous subsection. The trigonometric terms required for this rotation are generated using the same CORDIC approach, ensuring consistent numerical behavior and deterministic execution latency.

Following the rotation, the real and imaginary parts of the resulting complex signal correspond directly to the direct-axis and quadrature-axis current components, i_{sd} and i_{sq}, respectively. These signals are then forwarded to the current control loops operating in the rotating reference frame.

4.3.5 Sinusoidal PWM Generation

In the proposed control architecture, pulse-width modulation is employed to translate the continuous-valued phase voltage references into binary switching signals suitable for driving the three-phase inverter. A sinusoidal PWM (SPWM) scheme is adopted, in which each phase voltage reference is compared against a common triangular carrier waveform. The resulting comparator outputs directly determine the switching states of the inverter legs.

The amplitude of the triangular carrier is defined based on the relationship between the system clock frequency and the selected switching frequency. In the proposed implementation, the PWM carrier is generated digitally using integer arithmetic, with one complete carrier period corresponding to one switching period. Given a system clock frequency of 25 MHz and a switching frequency of 10 kHz, one switching period contains 2500 clock cycles. Accordingly, a symmetric triangular carrier is defined with a peak value of 1250 counts, resulting in a carrier range of $(-1250, 1250]$. This choice allows the carrier to be centered around zero, which simplifies the modulation logic by enabling symmetric treatment of positive and negative voltage references without requiring additional offsets in the comparison stage.

In the FPGA-based implementation, the triangular carrier waveform is generated digitally using integer arithmetic. Due to the limited availability of a bidirectional counter in the Simulink-HDL Coder design environment, the carrier is synthesized by combining an up-counting and a down-counting sequence. A switching logic selects between these two sequences to emulate the behavior of a triangular waveform over one switching period. This approach enables deterministic carrier generation using simple counting logic without the need for dedicated analog or mixed-signal components.

The initially generated carrier spans a range between 0 and 1250 counts, corresponding to half of the switching period. To obtain a symmetric triangular waveform centered around zero, the carrier is first scaled by a factor of two and subsequently offset by subtracting 1250. As a result, the final carrier waveform spans the interval $(-1250, 1250]$. This representation is particularly convenient for PWM comparison, as it allows positive and negative voltage references to be treated symmetrically and avoids the need for additional offset handling in the comparator stage. The complete carrier generation and comparison logic is illustrated in Figure 4.19.

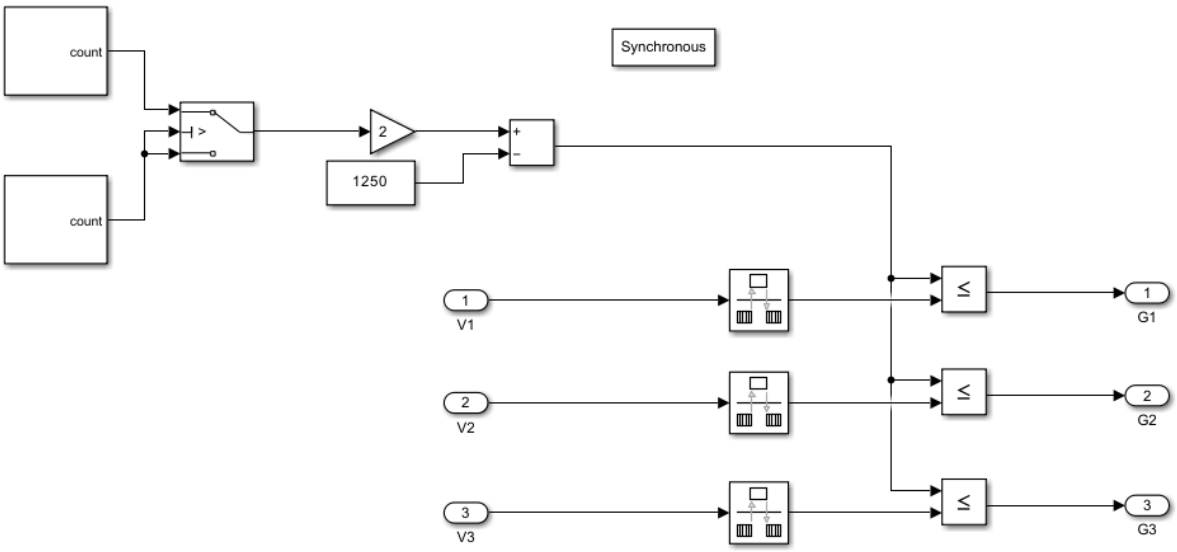


Figure 4.19 PWM generation logic optimized for FPGA

Each reconstructed phase voltage reference is then compared against the common triangular carrier using a simple relational operator. The gate signals are generated through a relational comparison between each phase voltage reference and the triangular carrier. The corresponding gate output is set to logic high when the voltage reference is greater than or equal to the instantaneous carrier value and set to logic low otherwise. This process is applied identically to all three phases, ensuring consistent modulation behavior across the inverter legs.

The use of integer-based carrier generation and comparator-based PWM ensures fully deterministic timing behavior and efficient utilization of FPGA resources. Moreover, defining the carrier amplitude explicitly in terms of integer counts establishes a clear numerical domain for the voltage reference scaling process, which is addressed in the following subsection.

4.3.6 Voltage Reference Reconstruction and Three-Phase Reference Synthesis

The voltage commands generated by the current control loops are expressed in the synchronous rotating reference frame as direct- and quadrature-axis components v_{sd} and v_{sq}. To apply these commands to the three-phase voltage source inverter, they must be transformed back into phase voltage references compatible with the pulse-width modulation stage. This subsection describes the reconstruction of three-phase voltage references from the rotating reference frame in a form suitable for fixed-point FPGA-based implementation.

In the proposed control architecture, the voltage reconstruction is realized using an inverse space-vector formulation. The direct- and quadrature-axis voltage components are first combined into a complex voltage vector in the rotating reference frame. This

vector is then rotated into the stationary reference frame through a complex multiplication with the exponential term $e^{j\theta_e}$, where θ_e denotes the electrical rotor angle. The trigonometric values required for this rotation are generated using a CORDIC-based implementation, ensuring numerical consistency and deterministic latency across the control pipeline.

The stationary voltage space vector obtained after the inverse rotation is subsequently projected onto three phase-shifted unit vectors separated by 120°, yielding the three-phase voltage references. These projections are implemented through complex multiplications with fixed phase offsets of 0, $2\pi/3$, and $4\pi/3$, followed by extraction of the real components. This formulation is mathematically equivalent to the conventional inverse Park and inverse Clarke transformations but is expressed in a compact form that is well suited for Simulink interface and HDL coder. The resulting structure is illustrated in Figure 4.20.

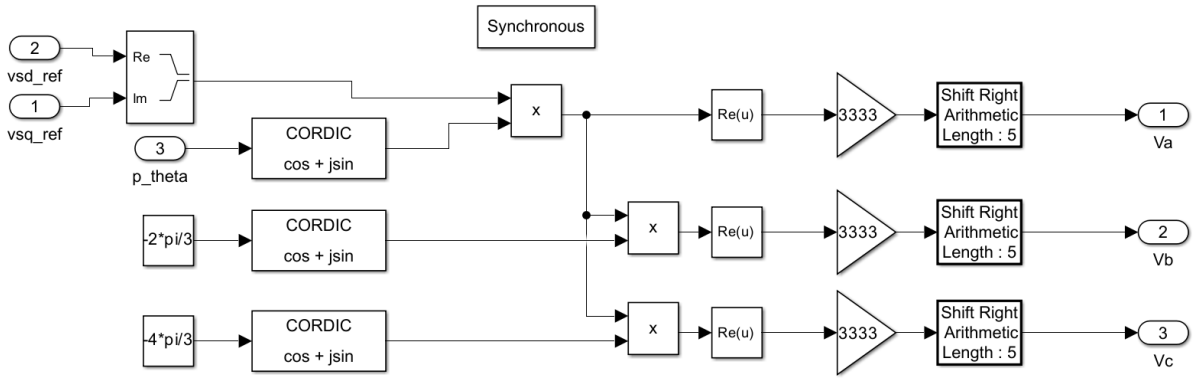


Figure 4.20 dq frame to three phase voltage transformation

Consistent with the design choice in Figure 4.18 CORDIC blocks are also used in for the evaluation of the required trigonometric terms. While constant coefficients could be employed for the phase-shifted components, the CORDIC-based realization was retained to ensure uniform fixed-point scaling and to prevent HDL Coder type-consistency issues.

To interface the reconstructed phase voltage references with the pulse-width modulation stage, the voltage signals must be appropriately normalized and scaled. In the discrete-time control design, sinusoidal PWM normalization is employed by scaling the voltage references by a factor of $2/V_{dc}$, where V_{dc} denotes the DC-link voltage. This normalization ensures that a modulation index of unity corresponds to the maximum achievable fundamental voltage and yields normalized reference signals in the interval $[-1,1]$.

However, in the FPGA-based implementation, the PWM carrier is not represented as a normalized signal but as a digitally generated triangular waveform with a finite integer amplitude. Therefore, the voltage references produced by the control algorithm, which are normalized to the interval $[-1,1]$ according to the sinusoidal

PWM formulation, must be rescaled to match the numerical domain of the carrier prior to PWM comparison. These rescaling maps the normalized voltage references into the carrier range and, in principle, requires multiplication by a factor proportional to $2 \cdot 1250/V_{dc}$. Direct implementation of this scaling factor would require division by the DC-link voltage, which in this case is 24 V. Division by an arbitrary constant is inefficient in FPGA hardware and is therefore avoided due to its high resource cost. To address this limitation, the scaling operation is implemented using a rational approximation that can be efficiently realized using fixed-point multiplication followed by a power-of-two division.

Specifically, the factor $2 \cdot 1250/24$ is approximated by the ratio $3333/32$. The multiplication by 3333 provides the required gain with sufficient numerical accuracy, while the subsequent arithmetic right shift by five bits implements an exact division by 32. This approach eliminates the need for explicit division logic, preserves deterministic execution, and introduces only a negligible scaling error relative to the resolution of the PWM carrier. As a result, the phase voltage references are efficiently mapped into the carrier domain in a manner that is consistent with the discrete-time SPWM normalization used in the control design and fully compatible with FPGA implementation constraints.

5 VHDL Implementation

5.1 XADC Interface Module

The acquisition of stator current measurements in the proposed control system is performed using the on-chip XADC peripheral available on the target FPGA platform. The XADC provides a digitized representation of analog input signals through an internal successive-approximation register (SAR) architecture with an effective resolution of 12 bits, while exposing a 16-bit digital output word through the Dynamic Reconfiguration Port (DRP) interface. The XADC is configured using the XADC Wizard IP, which handles sampling, channel sequencing, and internal calibration, and allows deterministic access to conversion results through synchronous DRP transactions.

Since the XADC analog input range is limited to a maximum input voltage of 1 V, the phase current measurement signals are externally scaled before being applied to the ADC inputs. In the implemented setup, the voltage generated by the inverter current sensing stage is passed through a resistive voltage divider composed of a 23 kΩ series resistor and a 10 kΩ shunt resistor, as shown in Figure 5.1. This divider maps the maximum expected sensor output voltage safely within the allowable XADC input range while preserving sufficient resolution for current regulation at low and medium operating levels.

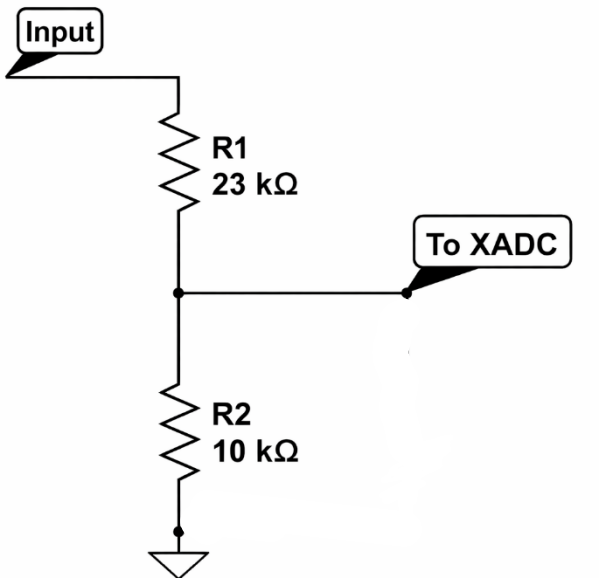


Figure 5.1 Voltage divider used for scaling the current sensing signal to the XADC input range

The resulting relationship between the sensor output voltage $V_{\text{sens}}$and the voltage applied to the XADC input V_{XADC} is given by

$$V_{\text{XADC}} = V_{\text{sens}} \cdot \frac{10\,\text{k}\Omega}{23\,\text{k}\Omega + 10\,\text{k}\Omega} \tag{5.1}$$

which corresponds to a fixed attenuation factor of $1/3.3$. From a system-level perspective, this analog scaling stage constitutes a static, linear transformation whose effect is absorbed into the subsequent digital normalization and fixed-point representation. As a result, the presence of the voltage divider does not alter the structure of the control algorithm but directly influences the mapping between physical current values and their digital representation.

A custom VHDL interface module manages the DRP read operations and extracts the conversion results corresponding to the phase current channels. The acquired 16-bit DRP words contain the 12-bit conversion result aligned according to the XADC output format. At this stage, the module forwards the data as unsigned 16-bit values through a streaming interface compatible with the rest of the FPGA-based architecture.

No numerical interpretation, scaling, offset removal, or signed conversion is performed within this interface block. All signal conditioning and fixed-point processing are delegated to the subsequent processing modules in the control pipeline. This separation ensures that the DRP interface remains strictly responsible for data acquisition and formatting, while arithmetic interpretation and normalization are handled in dedicated computational stages.

5.2 Rotary Encoder Interface Module

5.2.1 Quadrature Decoder and Position Accumulator

The rotor position is measured using an incremental quadrature encoder (LPD3806-600BM-G5-24C) with a nominal resolution of 600 pulses per mechanical revolution (PPR). By exploiting both rising and falling edges of the A and B channels, the effective resolution is increased by a factor of four, resulting in 2400 position increments per mechanical revolution. This value corresponds to the generic parameter `POS_PER_REV` defined in the VHDL implementation. The resulting angular resolution directly determines the quantization of the mechanical angle and, consequently, the electrical angle used in the field-oriented control algorithm.

The quadrature decoding logic is implemented in the `Encoder Reader` module. The encoder input signals are first synchronized to the FPGA clock domain through two cascaded flip-flops per channel. This double-registering stage reduces metastability risk and ensures that the decoding logic operates on stable, clock-aligned signals. The synchronized encoder state is represented by the two-bit vector formed from the registered signals A_2 and B_2. Four valid states are therefore possible: 00, 01, 11, and 10.

Position updates are determined by comparing the current state with the previously stored state. Under ideal quadrature operation, the encoder transitions only between adjacent states. A forward rotation corresponds to the cyclic sequence

$$00 \rightarrow 01 \rightarrow 11 \rightarrow 10 \rightarrow 00,$$

while reverse rotation follows the opposite order

$$00 \rightarrow 10 \rightarrow 11 \rightarrow 01 \rightarrow 00.$$

Each valid adjacent transition produces a signed increment of $+1$ or -1. Any non-adjacent transition, which may arise due to noise or timing imperfections, is treated as invalid and ignored. The state transition structure implemented in hardware is illustrated in Figure 5.2.

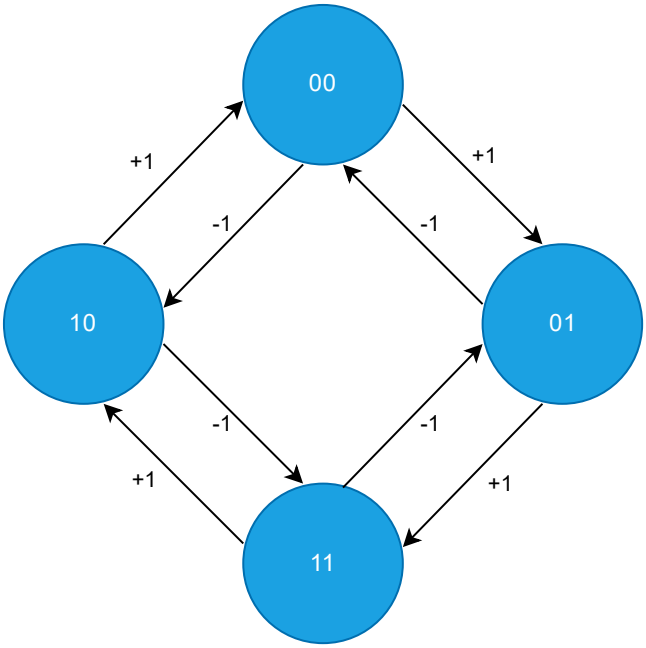


Figure 5.2 Quadrature encoder state transition diagram

Two position representations are maintained in parallel within the decoder architecture. The first is a position counter confined to a single mechanical revolution, denoted as `pos_in_rev`. This counter is constrained to the range $0 \ldots \text{POS_PER_REV} - 1$and wraps around at the boundaries. It therefore represents the instantaneous angular position within one mechanical revolution and provides a modular position signal suitable for electrical angle computation.

In parallel with this local counter, a 32-bit signed accumulator, denoted as `full_pos`, is maintained without wrap-around. This register increments or decrements by one count for each valid quadrature transition and thus represents the total multi-turn position of the rotor. The signed accumulator enables continuous position tracking over an arbitrary number of revolutions and forms the basis for discrete-time speed estimation described in the following subsection.

A direction flag is updated whenever a valid transition occurs and reflects the most recent motion direction, with logic high indicating forward rotation and logic low indicating reverse rotation. The decoding logic operates at every system clock cycle and updates the position registers only when a valid state change is detected. As a result, the module runs continuously and independently of the control sampling event, ensuring that encoder pulses are captured as long as the encoder transition frequency remains below the limit imposed by the FPGA clock frequency.

The parallel realization of synchronization, state decoding, and dual position accumulation exploits the inherent spatial parallelism of the FPGA fabric. All operations are performed within a single clocked process without sequential software-style execution, yielding deterministic and low-latency position tracking suitable for high-bandwidth motor control applications.

5.2.2 Mechanical-to-Electrical Angle Conversion

The `Mech_Angle_Calculator` module converts the measured mechanical position into an electrical angle expressed in the fixed-point format required by the CORDIC IP core. The input `pos_in_rev_in` represents the mechanical position within one revolution in encoder counts, and the conversion is triggered at each control sampling instant by the `eos_input` signal.

The encoder provides 600 pulses per revolution (PPR), and with x4 quadrature decoding the effective resolution is 2400 counts per mechanical revolution. The mechanical-to-electrical relation is given in Equation (5.2).

$$\theta_e = p\,\theta_m \tag{5.2}$$

where p is the number of pole pairs. In the VHDL implementation, this multiplication is performed directly in the count domain by left-shifting the position value by 2 bits. For $p = 4$, the mechanical count is multiplied by four, producing an intermediate electrical count `pos_e_raw`.

Because this multiplication may exceed one electrical revolution, the value must be reduced modulo one electrical period. This is implemented by subtracting fixed thresholds (2400, 4800, 7200) so that the electrical position is mapped back into the range

$$0 \leq pos\_e < POS\_PER\_REV \tag{5.3}$$

This operation performs the modulo reduction in the integer domain before conversion into angular units.

The downstream CORDIC IP expects its phase input to be represented as a signed fixed-point quantity in the interval $[-\pi, \pi)$. In the present design, the electrical angle is expressed in Q3.13 format, meaning that one radian is scaled by a factor of 2^{13}. Consequently,

$$\pi \approx 25736$$

which corresponds to the constant `PI_2P13`, computed as $\pi \cdot 2^{13}$. Similarly,

$$2\pi \approx 51472$$

which appears explicitly in the wrap-around logic.

To convert encoder counts to radians in this fixed-point domain, the scaling constant

$$K = \frac{2\pi \cdot 2^{13}}{\text{POS\_PER\_REV}} \tag{5.4}$$

is precomputed and stored in Q10 format as `K_Q10`. The additional Q10 scaling enables the multiplication to be performed using integer arithmetic while preserving precision. The intermediate product `pos_e * K_Q10` therefore carries an extra factor of 2^{10}, which is removed by a right shift of `K_SHIFT = 10`. After this shift, the resulting value `theta_u` represents the electrical angle in Q3.13 format over the range $[0,2\pi)$.

Although the computed angle naturally lies in the interval $[0,2\pi)$, the CORDIC IP is configured to operate on signed phase inputs within the principal interval $[-\pi,\pi)$. To comply with this requirement, an explicit wrap-around step is implemented. Pseudo VHDL algorithm is given in Algorithm 1.

Algorithm 1 : Conversion of Unsigned Phase to Signed Phase

```
If θu > π then
    θs <= θu - 2π;
else
    θs <= θu;
end if;
```

In fixed-point form, this corresponds to subtracting the constant 51472 (equal to $2\pi \cdot 2^{13}$) when the angle exceeds `PI_2P13`. The result `theta_s` is therefore a signed Q3.13 electrical angle strictly confined to $[-\pi,\pi)$, matching the phase convention required by the CORDIC-based sine and cosine computation.

The entire conversion process is implemented as a finite-state machine triggered by `eos_input`. Mechanical position capture, electrical scaling, fixed-point multiplication, and phase wrapping are performed in successive clock cycles. The computed signed angle is transmitted once `m_axis_tready` is asserted, ensuring deterministic latency and consistent phase alignment within the control pipeline.

5.2.3 Discrete-Time Speed Estimation

The rotor speed is estimated from the multi-turn position accumulator `full_pos`, produced by the quadrature decoder described in Section 5.2.1. Rather than computing an instantaneous derivative over a single control period, the implemented approach performs window-based accumulation over a fixed number of samples, defined by the generic parameter `N_WINDOW`.

At each control sampling instant, the current position `full_pos_in` is captured and compared with the previously stored value `full_pos_prev`. The difference

$$\Delta\text{pos}[k] = \text{full\_pos}[k] - \text{full\_pos}[k-1] \tag{5.5}$$

represents the incremental mechanical displacement during one sampling period. This discrete differentiation is performed entirely in signed arithmetic to preserve direction information.

To improve robustness against quantization effects and encoder jitter, the incremental displacement is accumulated over a window of N_{WINDOW} samples. The internal accumulator `acc_delta` sums the successive position increments. Formulation is given in Equation (5.6).

$$\mathrm{acc\_delta} = \sum_{i=0}^{N_{\mathrm{WINDOW}}-1} \Delta\mathrm{pos}[k-i] \tag{5.6}$$

When the window counter reaches `N_WINDOW - 1`, the accumulated value is transferred to `speed_reg`, the accumulator is reset, and the flag `speed_ready` is asserted. In the current configuration, $N_{\mathrm{WINDOW}} = 1000$. With a control sampling frequency of 100 kHz, this corresponds to a 10 ms estimation window, providing a trade-off between noise reduction and response speed.

The resulting speed value is therefore proportional to the average displacement over the window duration. Since encoder counts are used directly, the computed quantity is expressed in counts per window interval. Conversion to physical units (e.g., rad/s) can be performed externally by applying the appropriate scaling factor derived from the encoder resolution and sampling frequency.

The architecture separates numerical computation from data transmission. A first process performs position differencing, accumulation, and window management. A second finite-state machine handles the AXI-Stream output toward the speed controller. When `speed_ready` is asserted, the value stored in `speed_reg` is presented on `m_axis_tdata` with `m_axis_tvalid` and `m_axis_tlast` asserted. Upon reception of `m_axis_tready`, the transaction is completed and an acknowledgment signal clears the ready flag.

The complete implementation operates synchronously with the control sampling pulse and introduces a deterministic estimation latency equal to the window length. Because all arithmetic operations are performed using fixed-width signed registers, the design remains fully synthesizable and hardware-efficient while providing noise-attenuated speed feedback suitable for closed-loop operation

5.3 Park and Clarke Transformation in Digital Design

5.3.1 Clarke Transformation

The Clarke transformation converts the three measured phase currents into the stationary two-axis ($\alpha\beta$) reference frame required for the subsequent Park transformation. Formulation for Clarke transformation was given in Equation (2.16). In the implemented architecture, the transformation is simplified by assuming balanced three-phase operation as given in Equation (5.7).

$$i_a + i_b + i_c = 0 \tag{5.7}$$

Under this condition, the Clarke equations reduce to

$$i_\alpha = i_a \tag{5.8}$$

$$i_\beta = \frac{1}{\sqrt{3}}(i_b - i_c) \tag{5.9}$$

This formulation eliminates the need for averaging terms and reduces arithmetic complexity, which improves timing closure and minimizes DSP usage in the FPGA. To avoid division, the constant $\frac{1}{\sqrt{3}} \approx 0.577$ is approximated using fixed-point scaling. The implemented multiplier uses the integer constant 74 followed by a right shift of seven bits. Equation (5.9) can be approximated as

$$i_\beta = \frac{74}{2^7}(i_b - i_c) \tag{5.10}$$

Since

$$\frac{74}{128} = 0.578125$$

the approximation error is negligible relative to ADC quantization and switching ripple. This approach allows the multiplication to be realized using integer arithmetic followed by a power-of-two division, which is efficient in FPGA fabric.

The intermediate product is computed in 32-bit precision to accommodate bit growth resulting from the subtraction and multiplication. After the arithmetic right shift, the result is resized back to 16 bits. This preserves numerical stability while maintaining a uniform data width throughout the control pipeline.

The module receives phase-current samples through an AXI4-Stream interface. Although the XADC delivers 16-bit words through the DRP interface, only the upper 12 bits correspond to meaningful conversion data. Consequently, each received sample is right-shifted by four bits prior to numerical processing. This operation aligns

the ADC output with the effective 12-bit resolution of the converter and avoids propagating unused least-significant bits through the control pipeline.

The three phase currents are transmitted sequentially over the same AXI-Stream channel using a three-beat packet structure rather than allocating three parallel data buses. The phase identity of each sample is encoded in the two-bit `tuser` field. The encoding convention is:

- "00" → phase A
- "01" → phase B
- "10" → phase C

This approach enables all three phase measurements to be conveyed using a single streaming datapath while preserving explicit phase association. The use of `tuser` avoids additional decoding logic or multi-channel routing and therefore reduces interconnect complexity and LUT utilization. Compared to a parallel three-channel architecture, the three-beat streaming mechanism minimizes routing congestion and register duplication, which is particularly beneficial in FPGA fabrics where wide parallel buses increase placement pressure and routing delay.

The packet is terminated using `tlast` on the third beat (phase C). This explicitly marks the completion of one three-phase measurement set and allows the finite state machine to transition deterministically to the processing stage. By structuring the acquisition as a fixed three-beat sequence, the design guarantees temporal coherence between phase samples while keeping the interface resource-efficient and timing-friendly.

After acquisition, a fixed DC offset is removed from each phase current to compensate for analog front-end bias, current-sensor midpoint deviation, and XADC offset error. The offsets are implemented as compile-time generics with the values

- OFFSET_A = 2065
- OFFSET_B = 2020
- OFFSET_C = 2036

These values were experimentally identified from the steady-state ADC readings at zero current and reflect channel-dependent bias introduced by the analog sensing stage and ADC conversion chain.

Offset compensation is performed immediately after the 4-bit right shift and before any coordinate transformation or multiplication stage. The subtraction is executed in signed 16-bit arithmetic.

$$\begin{aligned} i_a &= \text{phase\_raw} - 2065 \\ i_b &= \text{phase\_raw} - 2020 \\ i_c &= \text{phase\_raw} - 2036 \end{aligned} \tag{5.11}$$

Performing offset removal at this early stage prevents DC bias from propagating through the Clarke and Park transformations. If uncompensated, such bias would introduce artificial components in both the $\alpha\beta$ and dq frames, distort the balanced-current assumption, and reduce available dynamic range in the fixed-point datapath.

By aligning the digital representation such that zero physical current corresponds to zero in the signed fixed-point domain, symmetry of the transformation equations is preserved and the stationary-frame components remain centered around zero. This is particularly important due to the simplification made for Clarke transformation.

The transformation entity itself is fully synchronous and introduces one clock-cycle latency due to registered outputs. In the enclosing `Clarke_Transformation_Module`, a finite state machine coordinates AXI reception, offset compensation, and output transmission. Once all three phases are received, the module waits for the control sampling event (`eos_input`) before asserting the output valid signal. The computed $i_\alpha$and i_β values are left-shifted by three bits before transmission to align with the internal scaling used in the subsequent Park transformation stage.

The result is a deterministic, fixed-point implementation of the Clarke transformation with minimal arithmetic complexity, single-cycle computational latency in the core transformation block, and full compatibility with the AXI-stream-based control pipeline.

5.3.2 Park Transformation

The Park transformation rotates the stationary-frame current vector (i_α, i_β) into the synchronous reference frame (i_{sd}, i_{sq}). In this implementation, the rotation is performed using a CORDIC IP core configured in rotation mode. Mathematical formulation for Park transformation was given in Equation (2.17). The objective is to multiply the stationary current vector by $e^{-jp\theta_m}$ where p is the number of pole pairs and θ_m is the mechanical rotor angle. The negative sign in the exponent indicates that the vector is rotated by $-p\theta_m$, aligning the rotating reference frame with the rotor flux.

The CORDIC core expects the current inputs in Q2.14 and the angle input in Q3.13 format. To comply with this interface, the outputs of the Clarke transformation are left-

shifted by three bits prior to entering the Park stage. This operation increases fractional precision and converts the internal current representation to Q2.14 format without introducing additional multipliers. Since left-shifting by three corresponds to multiplication by 2^3, numerical proportionality is preserved while expanding fractional resolution.

The electrical angle $p\theta_m$ is represented in Q3.13 format, allowing a full $\pm\pi$ range while maintaining sufficient angular resolution for accurate trigonometric rotation.

The configured CORDIC introduces a 25 clock cycle latency. This latency is fixed and deterministic, which simplifies pipeline alignment across the control loop. The delay is explicitly accounted for in the top-level integration, ensuring that the transformed currents correspond to the correct sampling instant.

5.3.3 Inverse Park Transform

The inverse Park transformation converts the voltage references from the synchronous reference frame (v_{sd}, v_{sq}) back to the stationary frame (v_α, v_β) before the inverse Clarke transformation and PWM generation stage. Mathematically, this operation corresponds to multiplying the synchronous voltage vector by $e^{+jp\theta_m}$.

The same CORDIC IP core configuration used for the forward Park transformation is employed here. The core remains in rotation mode, and the only difference is the sign of the supplied electrical angle. The input formats are identical v_{sd}, v_{sq} in Q2.14 and electrical angle $p\theta_m$ in Q3.13 format

No additional scaling is required, since the controller outputs are already aligned to the CORDIC input format. This preserves numerical consistency across forward and inverse transformations and avoids redundant shift operations.

To preserve temporal consistency across the control pipeline, the same electrical angle $p\theta_m$ used in the forward Park transformation is also used in the inverse Park transformation, but with appropriate synchronization. Since the forward Park block introduces a fixed 25 clock cycle latency and the controller stage adds additional deterministic delay, the voltage references v_{sd} and v_{sq} correspond to a current sample taken at an earlier instant. Feeding the inverse Park transformation with the most recent angle value would therefore introduce a phase mismatch between the computed voltage vector and the original measurement instant.

To prevent this misalignment, the electrical angle is delayed by one control sample and pipelined consistently with the transformation latency before being supplied to the inverse Park CORDIC. This guarantees that both rotations, $e^{-jp\theta_m}$ in the forward Park and $e^{+jp\theta_m}$ in the inverse Park, operate on angle values corresponding to the same physical sampling instant. As a result, the synchronous-to-stationary transformation remains temporally coherent, avoiding artificial phase shifts or unintended cross-coupling effects in the reconstructed voltage vector.

5.3.4 Inverse Clarke Transformation

The inverse Clarke transformation converts the stationary-frame voltage components (v_α, v_β) into three-phase voltage references (v_a, v_b, v_c) for the PWM generation stage. The implemented equations follow the inverse Clarke relation introduced in Equation (2.19), and are realized in fixed-point arithmetic with constant-coefficient approximations to reduce resource usage and simplify timing closure.

$$v_a = v_\alpha \tag{5.12}$$

$$v_b = -\frac{1}{2}v_\alpha + \frac{\sqrt{3}}{2}v_\beta \tag{5.13}$$

$$v_c = -\frac{1}{2}v_\alpha - \frac{\sqrt{3}}{2}v_\beta \tag{5.14}$$

The core arithmetic is implemented in `InvClarkeTransform` as a synchronous, one-clock-cycle registered datapath. The $-\frac{1}{2}v_\alpha$ term is implemented using an arithmetic right shift by one bit, avoiding a multiplier. The $\frac{\sqrt{3}}{2}v_\beta$ term is approximated using an integer multiply and a power-of-two division.

$$\frac{110}{128} = 0.859375 \approx \frac{\sqrt{3}}{2} = 0.866025$$

The inverse Clarke Equations can be written with the approximation as

$$v_a = v_\alpha \tag{5.15}$$

$$v_b = -\frac{1}{2}v_\alpha + \frac{110}{2^7}v_\beta \tag{5.16}$$

$$v_c = -\frac{1}{2}v_\alpha - \frac{110}{2^7}v_\beta \tag{5.17}$$

The error introduced by this approximation is small relative to the quantization and switching effects present in the system, while providing a low-cost implementation using fixed-point operations. To safely accommodate bit growth, the multiplication is performed with a 32-bit intermediate bit size increase and truncated back to 16 bits after the right shift.

The inverse Clarke wrapper (`Inverse_Clarke`) receives (v_α, v_β) as a packed 32-bit AXI-Stream word and forwards the reconstructed phase voltages through a resource-efficient three-beat master stream. Instead of transmitting (v_a, v_b, v_c) simultaneously on parallel channels, the design reuses a single 16-bit AXI stream and sends the three phase references sequentially as a fixed-length packet. The phase identity is encoded in `m_axis_tuser` using the same convention as the measurement path:

- "00" → phase A
- "01" → phase B
- "10" → phase C

The final beat (phase C) asserts `m_axis_tlast` to mark packet completion. This three-beat transmission mechanism reduces routing complexity and avoids duplicating AXI infrastructure, resulting in a timing-friendly and resource-efficient interface while retaining explicit phase association.

The complete acquisition, computation, and three-beat transmission process is coordinated by a finite state machine to guarantee deterministic sequencing and synchronization with the sampling event.

Finally, the reconstructed phase voltages are scaled for PWM compatibility. Each phase reference is multiplied by the triangular-carrier limit (2500) and right-shifted by 14 bits, mapping the fixed-point voltage representation to the integer domain used by the PWM comparator. This scaling stage is applied immediately before streaming the three phase outputs, ensuring consistent amplitude mapping across all phases in the PWM generation stage.

5.4 Current Controller Implementation

5.4.1 Controller Wrapper

The `Controller_Wrapper` module coordinates the execution of the current control stage and interfaces the PI and deadbeat controllers with the AXI-stream-based control pipeline. Its primary role is to ensure deterministic sequencing between current measurement acquisition, controller computation, optional feed-forward addition, and transmission of the voltage references.

The wrapper receives the measured synchronous-frame current pair (i_{sq}& i_{sd}) as a packed 32-bit AXI-stream word. Upon a valid handshake, the measurements and the external i_{sq}^{*} reference are latched internally. A single-cycle `start_PI` pulse is then generated to trigger the PI controller cores. This pulse-based activation guarantees that each control update corresponds to a unique sampling instant.

To preserve timing determinism, the wrapper includes a controlled delay stage (`DUMMY`) governed by the generic `CONTROLLER_CLOCK_CYCLE`. This stage ensures that the controller FSM has completed its internal four-state update before the computed voltage references are sampled. As a result, the control pipeline operates with a known and fixed latency.

A dedicated `FEED_FORWARD` state is reserved for the addition of a voltage feed-forward term. This term, derived from the decoupling and back-EMF compensation equations presented in Equations (2.11) and (2.12), follows the same structure used in the FPGA-in-the-loop implementation and is expressed in the same fixed-point format as the PI output. Although only the feedback PI contribution is active in the current snapshot, the architectural separation of this stage allows feed-forward integration without altering pipeline timing or state sequencing.

The computed voltage references (v_{sq}^{*}&v_{sd}^{*}) are transmitted via AXI-stream only when the global sampling event is asserted. This guarantees synchronization between measurement acquisition and actuation. The wrapper completes the transfer through an explicit `OUTPUT_HANDSHAKE` state, waiting for `m_axis_cartesian_tready` before returning to `IDLE`.

5.4.2 PI Controller Implementation

The PI current controller is implemented in VHDL as two independent regulators acting on the synchronous-frame current components i_{sq} and i_{sd}. The discrete-time PI formulation is introduced in Equation (2.44). Both controllers share the same architecture and the same controller parameters, only difference being is the i_{sd} controller tracks a zero reference ($i_{sd}^{*} = 0$), while the i_{sq} controller tracks the externally provided i_{sq}^{*} reference.

To enable efficient fixed-point arithmetic, the continuous-domain gains are converted into integer constants prior to synthesis. The proportional gain is stored in fix20_14 format (KP_F20_14), while the integral gain is stored in fix16_14 format as the discrete product $K_i T_s$(KI_FP). This allows gain multiplications to be followed by a right shift of 14 bits to restore scaling, avoiding division hardware:

$$K_{p,\text{F20\_14}} = \lfloor K_p \cdot 2^{14} \rceil$$

$$K_{iT_s,\text{F16\_14}} = \lfloor K_i T_s \cdot 2^{14} \rceil$$

The controller core (`isq_PI_module`, and equivalently `isd_PI_module`) is implemented as a four-state FSM to enforce deterministic sequencing and maintain timing-friendly arithmetic:

- ERROR_CALC: compute $e[k] = i^{*}[k] - i[k]$when start_PI is asserted high
- P_STATE: compute the proportional term using the incremental error $(e[k] - e[k-1])$
- I_STATE: compute the integral increment $K_i T_s\, e[k]$

- SUM_AND_SAT_STATE: accumulate contributions, apply output saturation, and update stored states

The computed control command is saturated to a symmetric bound (U_LIM_F16_14) to prevent over-modulation and to keep the voltage reference within the admissible range of the inverter. After saturation, the controller updates both the previous error and the previous output state for the next sampling instant. The FSM architectures of the PI controller is illustrated in Figure 5.3.

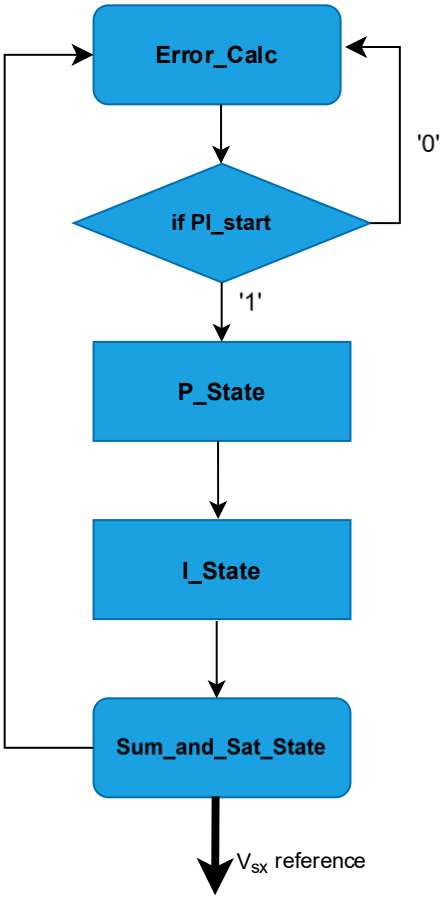


Figure 5.3 Finite state machine of the PI current controller

5.4.3 Deadbeat Controller Implementation

The deadbeat control law used for the inner current loop was derived in Section 2.5.2, where the discrete-time model of the PMSM and the corresponding deadbeat formulation were introduced. The analytical expression of the control law is therefore not repeated here. This section focuses exclusively on its RTL realization on FPGA.

The same hardware structure is used for both i_{sq} and i_{sd} channels. Only difference is d-axis controller has hard coded zero reference while q-axis controller receives the reference value. This guarantees architectural symmetry between d- and q-axis control paths and simplifies verification.

The deadbeat voltage command is generated using a small finite-state machine, as shown in Figure 5.4. In the `Idle` state, the module waits for `deadbeat_start`. When asserted, the controller transitions to the `Deadbeat` state, where the deadbeat equation is evaluated using the most recent current reference and measurement values. The resulting unsaturated voltage command is then forwarded to the `Sat_State`, where output limiting is applied to enforce the defined voltage bounds and ensure safe operation under transients. After saturation, the final voltage reference is registered and held constant until the next activation, providing a stable interface to the subsequent stages of the control pipeline. This sequencing introduces a small, fixed multi-cycle latency, but preserves fully deterministic timing and repeatable update behavior, which is critical for the inner current-loop implementation on FPGA.

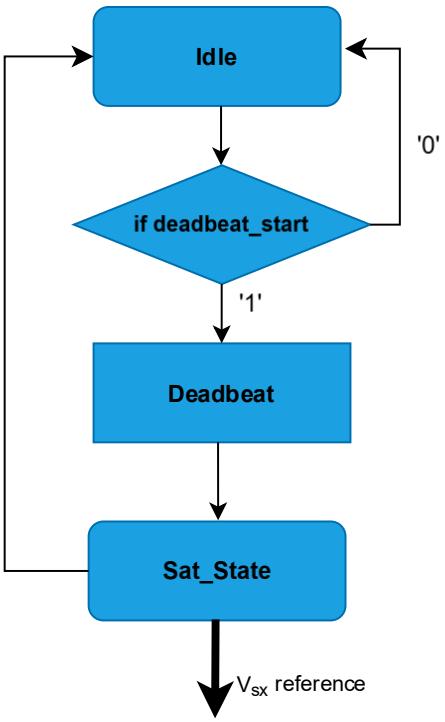


Figure 5.4 Finite state machine of the Deadbeat current controller

In Equation (4.8), the deadbeat difference equation was introduced, where the dominant coefficient evaluates to approximately 120.65. While the term corresponding to 120 can be directly implemented as an integer constant without loss of accuracy, the coefficient 120.65 cannot be represented exactly in the chosen fixed-point format. Therefore, a rational approximation is adopted as multiplication by 965 followed by right shift by 3 bits. With the approximation deadbeat current controller equation can be written as

$$v[n+1] = \frac{965}{2^3}\, i^*[n] - 120\, i[n] \tag{5.18}$$

This approximation enables an efficient hardware realization. The multiplication by 965 is mapped to a DSP block, while the division by 8 is implemented as an arithmetic

right shift by 3 bits. Consequently, the scaling operation is realized using a single multiplier followed by a shift, avoiding explicit division logic and maintaining a compact datapath.

It should be emphasized that deadbeat control is highly sensitive to parameter accuracy, since the controller directly relies on the discrete-time plant model. Any coefficient approximation may introduce a slight deviation from the ideal deadbeat response, potentially resulting in minor performance degradation. However, the selected rational approximation represents a practical trade-off between numerical precision and hardware efficiency.

A more precise implementation would require additional fractional bits and a wider internal datapath, increasing DSP usage and overall resource utilization. Although this could improve numerical fidelity and theoretical performance, it would also complicate timing closure and increase hardware cost. The chosen approach therefore balances implementation efficiency and control performance within the constraints of the FPGA platform.

5.5 PWM Generator

The `sin_pwm` module generates six complementary gate signals for a three-phase inverter by comparing phase voltage references against an internally generated triangular carrier. The modulation principle itself is introduced in Section 2.4. Therefore, this section focuses on the digital implementation, AXI-stream interfacing, and the parallel PWM architecture.

The module receives the phase voltage references through a single AXI4-Stream channel using a three-beat packet from Inverse Clarke Module. Each beat carries one 16-bit phase reference (s_axis_tdata), and the phase identity is encoded in `s_axis_tuser`:

- "00" → phase A reference
- "01" → phase B reference
- "10" → phase C reference

The packet boundary is indicated using `s_axis_tlast`. This structure mirrors the phase-tagging approach used in the measurement chain and reduces routing pressure and interface duplication, while maintaining explicit phase association. Incoming beats are first stored into buffer registers (phase_*_ref_buf). Once the packet is complete, the buffered values are synchronously transferred to the active reference registers (phase_*_ref) in a dedicated output state. This two-stage buffering mechanism guarantees that all three phase references are updated coherently as a set,

preventing partial updates that could otherwise occur if each phase were applied immediately upon reception.

The PWM logic is realized using multiple clocked processes operating in parallel. A dedicated process generates the triangular carrier using an up/down counter. The counter limits are derived from the switching-frequency requirement and the system clock, resulting in a symmetric carrier swing between ±QUARTER, where

$$\text{QUARTER} = \text{PERIOD\_CYCLE\_COUNT}/4$$

and

$$\text{PERIOD\_CYCLE\_COUNT} = \text{CLK\_FREQUENCY}/\text{SWITCHING\_FREQUENCY}.$$

This carrier generator runs continuously and is shared by all three phases.

In parallel with the carrier generator, the module implements three independent per-phase PWM state machines, one for each inverter leg (A, B, and C). Each per-phase FSM executes the same sequence:

1. wait for an update event (`enable_pulse`)
2. compare the phase reference against the current carrier value
3. determine whether the upper or lower device must be commanded
4. enforce dead-time during commutation
5. update the complementary gate outputs

By separating phases into independent FSMs, the design avoids large shared combinational blocks and enables clean timing closure through localized logic. The comparison decision is based on the sign of the inequality

$$v_x^* > v_{\text{tri}}$$

which produces a raw PWM selection for the corresponding phase leg. The gate outputs are complementary by construction (`pwm_x_upper, pwm_x_lower`), and switching transitions are handled explicitly to prevent shoot-through. When a commutation is required, both devices are forced off and held in a transition state for DEADTIME_CYCLES clock cycles before enabling the next device. This dead-time insertion is fully deterministic and implemented through counters local to each phase ensuring consistent behavior independent of modulation waveform dynamics.

Finally, the separation of the design into (i) one carrier-generation process, (ii) one AXI-stream acquisition/buffering process, and (iii) three per-phase PWM FSM processes results in a modular and highly parallel architecture. This organization improves readability, simplifies timing analysis, and ensures deterministic switching behavior while maintaining a resource-efficient single-channel AXI-stream interface for phase command updates.

5.6 Top-Level Integration

The clocking strategy was defined progressively during the design process. In the initial phase of development, the primary objective was functional correctness and stable integration of all subsystems, including current acquisition, coordinate transformations, control execution, and PWM generation. For this reason, the target system clock was conservatively set to 50 MHz. This frequency provided a comfortable timing budget for arithmetic operations implemented in fixed point format and allowed incremental verification of each module without aggressive timing pressure. At this stage, emphasis was placed on ensuring deterministic behavior, correct numerical scaling, and proper synchronization between processing stages. After complete integration of the pipeline and successful synthesis, the timing analysis revealed that the worst negative slack was not only eliminated but exceeded the requirement by a substantial margin. WNS value at 50MHz clock frequency was greater than 11ns which indicated using a faster clock was possible. Therefore, after the design is concluded clock frequency was updated to 100MHz. The overall pipelined control architecture and the identified longest datapath are illustrated in Figure 5.5.

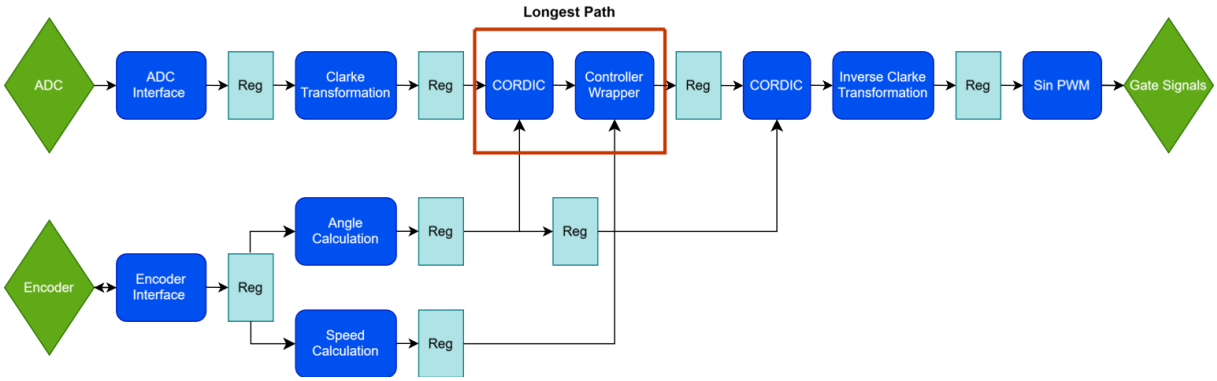


Figure 5.5 Top level pipelined control architecture and longest datapath identification

The longest combinational region, located between the Park transformation CORDIC and the controller wrapper, remained well within the allowable delay at 50 MHz. The significant positive slack indicated that the design was effectively over constrained at this frequency. Because the architecture had already been structured with explicit register boundaries between all major processing stages, increasing the clock frequency did not require structural modification. The clock constraint was therefore updated to 100 MHz to reduce computational latency and improve timing robustness.

With the PI controller configuration, the longest registered datapath spans 34 clock cycles from sampled current input to voltage reference propagation toward the PWM stage. When the deadbeat controller is selected, the depth is reduced to 33 clock cycles due to the absence of one accumulation stage. At 50 MHz, where the clock period is 20 ns, these depths correspond to total computational latencies of approximately 680 ns for the PI implementation and 660 ns for the deadbeat implementation. After

increasing the clock frequency to 100 MHz, where the clock period is 10 ns, the latencies are reduced to approximately 340 ns and 330 ns respectively.

It is essential to distinguish between computational latency and control update time. The effective control time is defined by the sampling frequency, which is set to 100 kHz. This corresponds to a sampling period of 10 μs. The controller output is updated once per sampling interval, synchronized with the inverter switching event. Even at the initial 50 MHz clock frequency, the internal computational delay occupies less than 7 percent of the 10 μs sampling period. At 100 MHz, this proportion decreases to approximately 3.3 percent.

From a discrete time control standpoint, the FPGA implementation introduces a fixed delay equivalent to 33 or 34 clock cycles. Since this delay remains significantly smaller than the sampling period, it does not dominate the closed loop dynamics. The primary timing constraint of the control system remains the 100 kHz sampling frequency, which determines the discrete time model and achievable bandwidth. Increasing the FPGA clock from 50 MHz to 100 MHz therefore improves computational margin and reduces absolute latency, while preserving the fundamental control behavior defined by the sampling rate.

5.7 UART Interface for Experimental Validation

To support experimental validation of the FPGA-based motor-control architecture, a minimal software interface was developed in Vitis on the Zynq Processing System (PS). While the inner control loops and the signal-processing chain execute fully in programmable logic, the PS software is used as a practical bridge for observability during testing. In particular, the software accesses selected internal variables made available through an AXI-Lite memory-mapped register interface and streams them to a host PC for visualization and sanity checks.

For simplicity and direct hardware access, the PS application was implemented using a bare-metal setup avoiding an operating system and related overhead. This choice reduced complexity during bring-up and made register-level communication straightforward, which was beneficial for rapid debugging and iteration.

A UART connection on the PS side was selected as the telemetry channel due to its simplicity and reliability. The link was configured at 115200 baud and used to transmit key signals such as i_{sd}, i_{sq}, the torque-producing current reference i_{sq}^{*}, and speed measurement quantities. This setup enabled quick verification of critical behaviors without introducing a more complex communication infrastructure. This low-complexity UART approach provides less detailed measurement capability than higher-bandwidth interfaces, but since it is used only for visual monitoring, the limitation is acceptable.

6 Simulation, FIL and Experimental Results

This chapter presents the performance evaluation of the proposed PMSM drive control system under progressively increasing levels of implementation realism. The results are organized to clearly separate the effects of controller formulation from those introduced by digital execution and hardware constraints. For this reason, the chapter begins with floating-point simulations, where numerical precision is ideal, and then transitions to fixed-point and FPGA-in-the-loop (FIL) validation, before concluding with real-time experimental results. All controllers are evaluated under two standardized disturbance conditions: a constant-load case set to 10% of the motor rated torque, and a ramp-load case where the load torque is increased linearly with a slope equal to 1/10 of the rated torque per second, providing a structured disturbance-rejection benchmark.

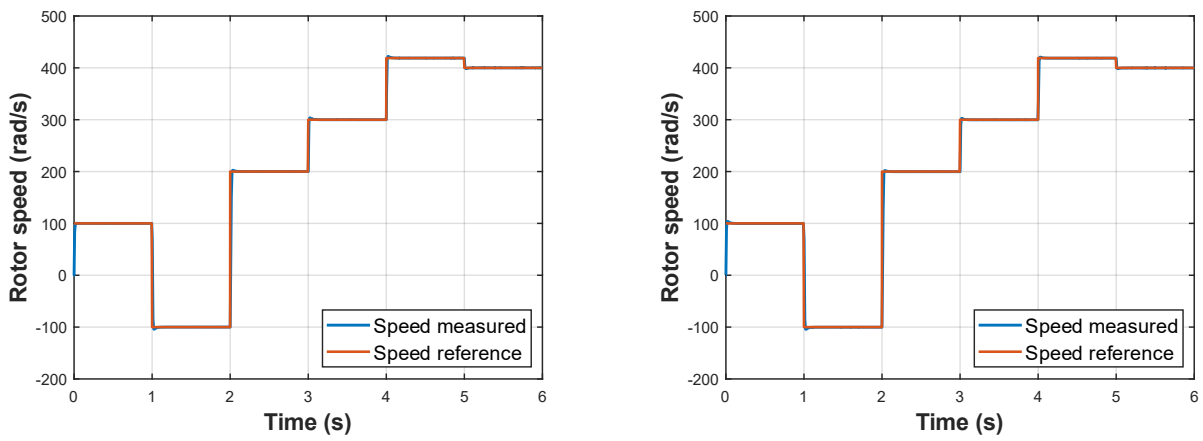
In the floating-point part, a continuous-time PI-controlled model is first used as an ideal reference to verify the overall control architecture and to provide a baseline response without sampling or computational delay. The same PI strategy is then evaluated in discrete time using floating-point arithmetic to isolate the impact of sampling on the closed-loop dynamics while keeping numerical effects negligible. Finally, deadbeat current control is introduced and evaluated in discrete time, since deadbeat control is inherently a discrete-time strategy and does not admit an equivalent continuous-time formulation. This organization enables a direct and fair comparison between discrete-time PI and deadbeat controllers under identical operating conditions.

To maintain consistency across all result sections and enable direct visual comparison, figures are reported using fixed signal ordering. For each test case, the rotor speed response is presented first, followed by the quadrature-axis current, the direct-axis current, and the electromagnetic torque. Each set of plots is complemented by zoomed-in views of the rotor speed and over a representative transient interval, which is used consistently throughout the chapter to highlight differences in dynamic behavior.

6.1 Floating Point Simulation Results

6.1.1 Continuous Time PI Current Control

Figure 6.1 compares the rotor speed response obtained with the continuous-time PI control structure under the constant-load and ramp-load scenarios. In both cases, the measured speed follows the commanded reference steps accurately, exhibiting smooth transients after each reference change and converging to the reference value. Under constant load (Figure 6.1 (a)) the operating point is stationary between speed steps, providing a baseline benchmark for subsequent discrete-time and hardware-oriented results. Under ramp load (Figure 6.1(b)), the speed remains well-regulated despite the continuously increasing disturbance torque; however, the operating point evolves over time due to the rising load, making this scenario more representative for assessing disturbance rejection.

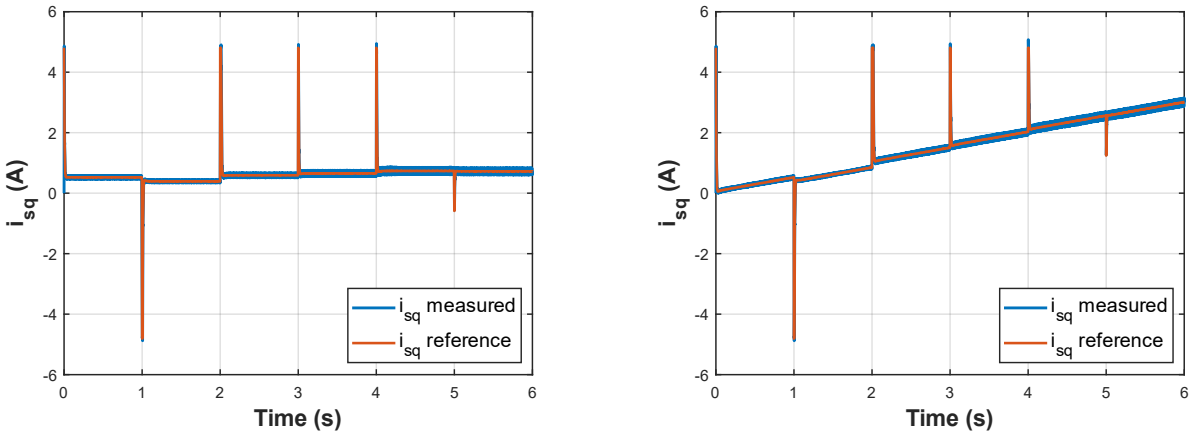


(a) (b)

Figure 6.1 Rotor speed response with continuous-time PI current control under (a) constant load equal to 10% of rated torque and (b) ramp load with slope equal to one-tenth of rated torque per second

Figure 6.2 reports the torque-producing current component i_{sq} for the constant-load and ramp-load scenarios. During each speed transition, i_{sq} increases to generate the electromagnetic torque required for acceleration and then settles toward a steady value once the reference speed is reached. Under constant load (Figure 6.2 (a)), the steady-state level remains approximately constant between speed steps, whereas under ramp load (Figure 6.2 (b)) it increases approximately linearly over time, reflecting the progressively increasing torque demand imposed by the ramping disturbance. In both cases, a high-frequency ripple is visible around the average i_{sq} trajectory. This ripple primarily originates from PWM switching and the finite stator inductance, which produce nonzero current ripple even when the average current is

well regulated. Moreover, the ripple becomes more noticeable at higher operating speeds: as the electrical fundamental frequency f_e increases with speed, the modulation ratio $m_f = f_{sw}/f_e$ decreases, reducing spectral separation and making switching-related harmonic content less effectively attenuated, which results in a wider visible ripple band. The combined plot therefore provides a baseline for subsequent discrete-time and fixed-point results, where the same trends may appear with additional quantization- and execution-related effects.



(a) (b)

Figure 6.2 Quadrature-axis current tracking with continuous-time PI control (a) constant load equal to 10% of rated torque and (b) ramp load with slope equal to one-tenth of rated torque per second

Figure 6.3 shows the flux-producing current component i_{sd} for the constant-load and ramp-load scenarios. As expected for surface-mounted PMSM operation without flux weakening, the controller regulates i_{sd} around its reference ($i_{sd}^* = 0$)throughout the full simulation interval. Notably, the i_{sd} response is essentially the same in both load cases (Figure 6.3 (a)–(b)). Apart from brief deviations at the speed step instants, no sustained d-axis current component is introduced even when the load torque increases over time. This confirms correct field-oriented operation and effective decoupling between the d- and q-axis current dynamics in the continuous-time baseline, with the additional torque demand being handled primarily through the i_{sq}channel. Similar to i_{sq}, a small high-frequency ripple is present in i_{sd} due to PWM switching of the inverter. While the average i_{sd} remains at zero, the ripple becomes slightly more visible at higher operating speeds, consistent with the reduced frequency modulation ratio as the electrical fundamental frequency increases.

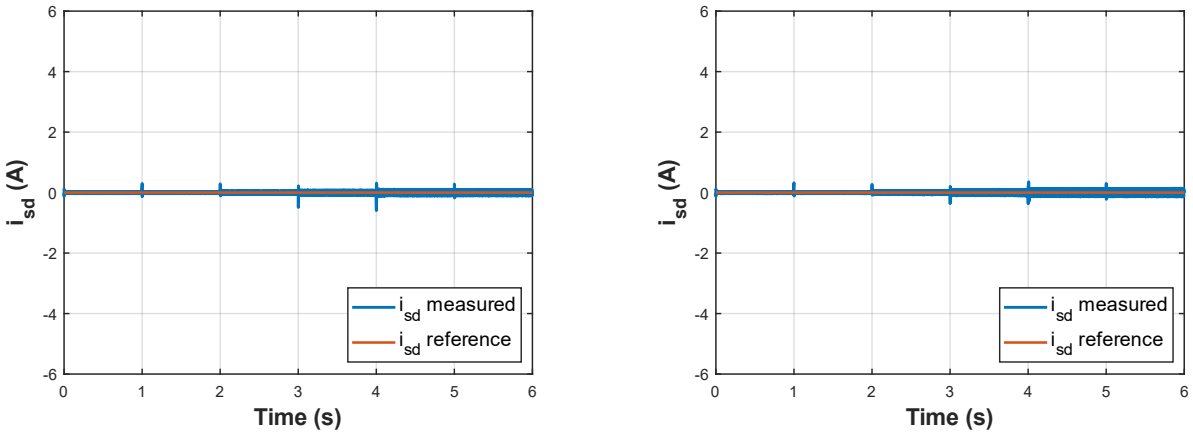


Figure 6.3 Direct-axis current tracking under continuous-time PI control (a) constant load equal to 10% of rated torque and (b) ramp load with slope equal to one-tenth of rated torque per second

The corresponding electromagnetic torque response is shown in Figure 6.4 for the constant-load and ramp-load scenarios. The torque waveform follows the same overall shape as the torque-producing current component i_{sq}, since electromagnetic torque is primarily proportional to i_{sq} under field-oriented control with $i_{sd} = 0$. Accordingly, torque peaks occur during the speed steps, reflecting the transient current demand required for acceleration and deceleration. Under constant load (Figure 6.4 (a)), the torque then settles to the level required to maintain the commanded speed, whereas under ramp load (Figure 6.4 (b)) the steady-state torque increases over time in line with the growing disturbance torque. In both cases, the response remains smooth and well behaved in the continuous-time baseline, without oscillatory components, providing a useful reference for later discrete-time and hardware-oriented results where sampling, quantization, and implementation effects may introduce additional torque ripple.

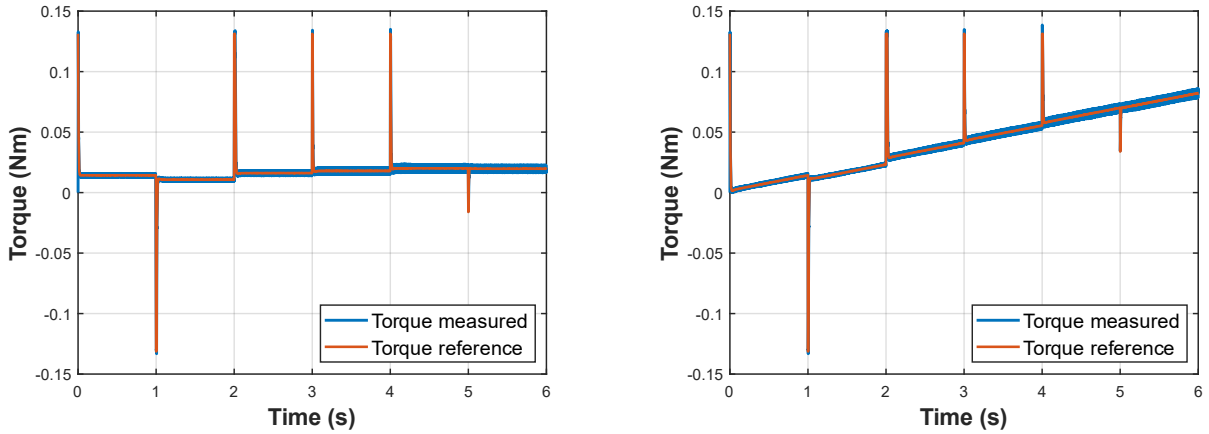


Figure 6.4 Electromagnetic torque response under continuous-time PI control (a) constant load equal to 10% of rated torque and (b) ramp load with slope equal to one-tenth of rated torque per second

To highlight the transient behavior, Figure 6.5 presents a zoomed view of the speed response around the representative speed step used throughout this chapter for both the constant-load and ramp-load scenarios. In both cases, the response remains well damped and converges smoothly to the new operating point, providing a clear reference for comparison with discrete-time results. Under ramp-load conditions, the transient settling remains fast; however, the underlying operating point continues to evolve due to the increasing load torque, which makes this case more representative for assessing disturbance rejection during nonstationary operation.

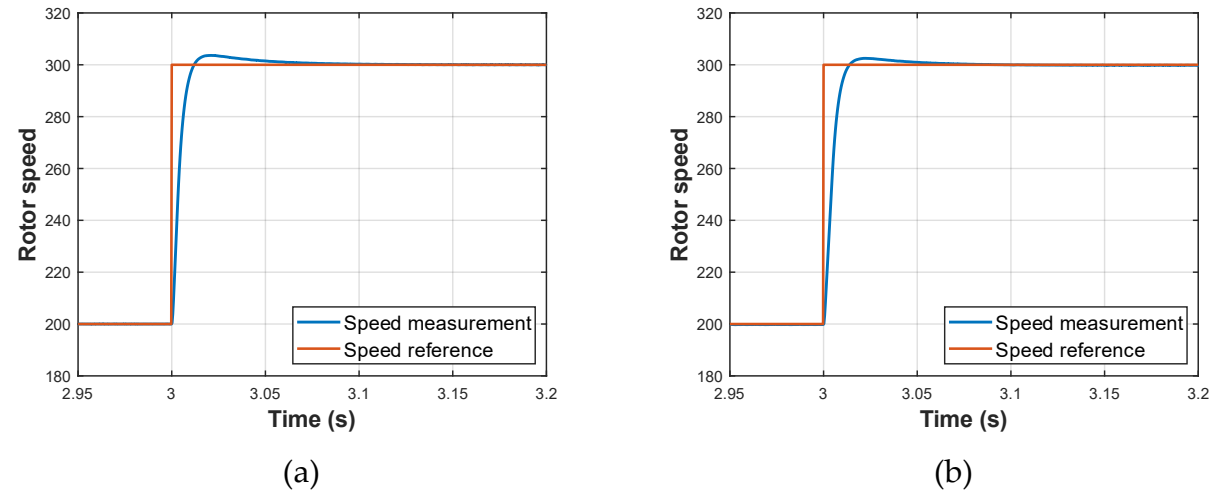
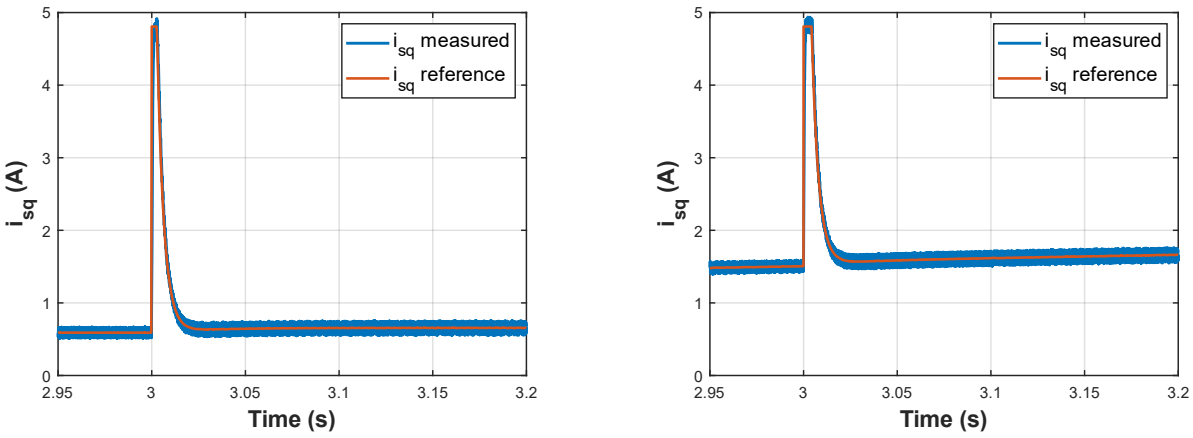


Figure 6.5 Zoomed speed response around the representative speed step (a) constant load equal to 10% of rated torque and (b) ramp load with slope equal to one-tenth of rated torque per second

Figure 6.6 shows the corresponding zoomed response of i_{sq} around the representative speed step for (a) the constant-load case and (b) the ramp-load case. In both scenarios, the torque-producing current reacts promptly to the speed reference change, generating a sharp transient peak to provide the required acceleration torque, and then settles rapidly once the transient ends. Under constant load (Figure 6.6 (a)), i_{sq} returns to a constant steady-state level, whereas under ramp load (Figure 6.6 (b))it settles back onto a slowly increasing trajectory consistent with the growing torque demand. This zoomed view provides a clear baseline for later discrete-time, fixed-point, and FIL comparisons: any staircase behavior, increased ripple band, or transient distortion can be attributed to numerical and implementation constraints rather than to the continuous-time control structure itself.



(a) (b)

Figure 6.6 Zoomed i_{sq} transient around the same representative speed step (a) constant load equal to 10% of rated torque and (b) ramp load with slope equal to one-tenth of rated torque per second

6.1.2 Discrete Time PI Current Control

The discrete-time PI controller is obtained by implementing the same control structure used in the continuous-time simulations but executing the current and speed control loops at a fixed sampling period T_s as selected in Section 4.2.1. Since this subsection still uses floating-point arithmetic, the purpose of these results is to isolate the effect of discretization from later fixed-point and hardware-related effects. Overall, the time-domain waveforms closely match the continuous-time results, indicating that the selected sampling period is sufficiently fast for the operating range considered.

Figure 6.7 shows the rotor speed response under discrete-time PI control for the constant-load and ramp-load scenarios. In both cases, the measured speed accurately tracks the reference across all commanded speed steps, with short transients occurring at each transition. Compared to the continuous-time baseline, the discrete-time response remains very similar, with only minor variations during the transient

intervals that are attributable to the sampled-data execution of the control loop. Under ramp-load conditions (Figure 6.7 (b)), the speed remains well regulated without steady-state error despite the continuously increasing disturbance torque, confirming that the discrete-time PI formulation preserves the disturbance rejection capability observed in the continuous-time case.

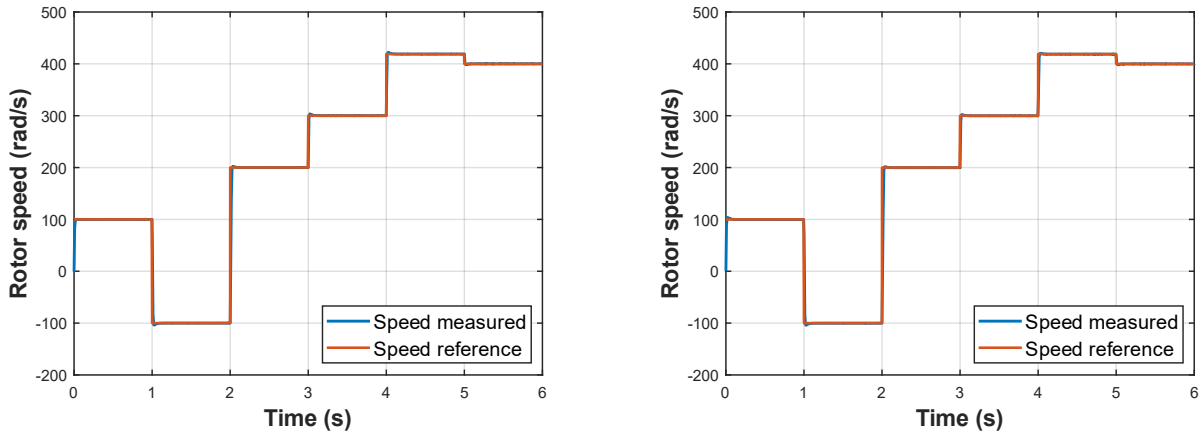
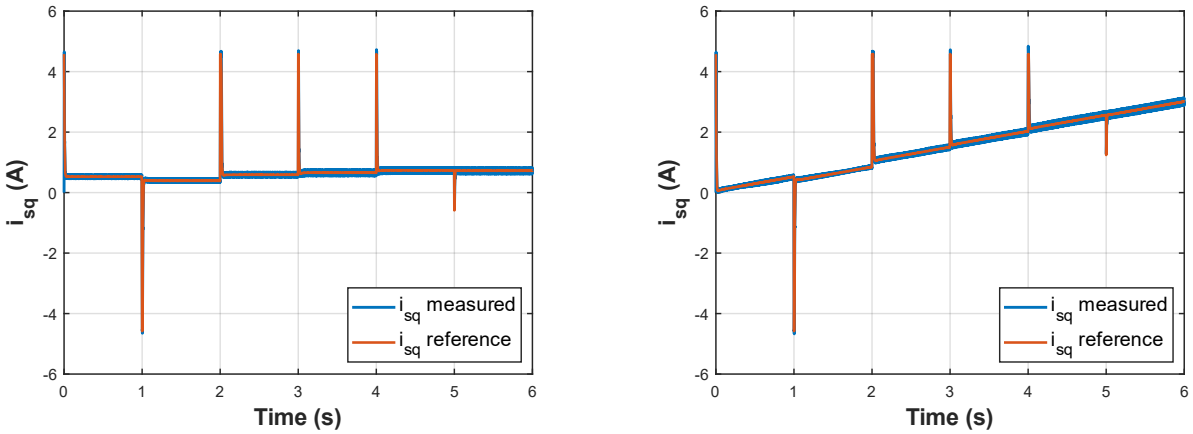


Figure 6.7 Speed response with discrete-time PI control (a) constant load equal to 10% of rated torque and (b) ramp load with slope equal to one-tenth of rated torque per second

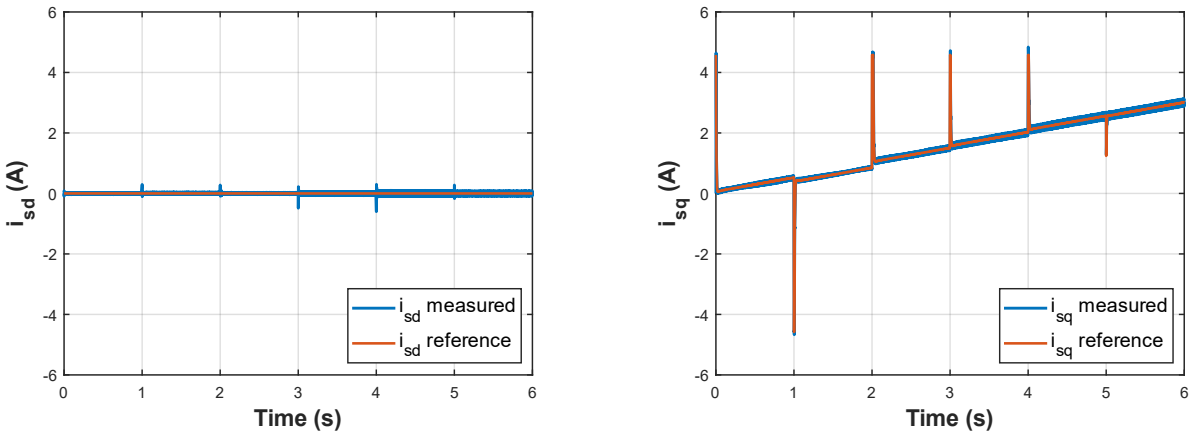
Figure 6.8 reports the quadrature-axis current i_{sq} under discrete-time PI control for the constant-load and ramp-load scenarios. As in the continuous-time case, i_{sq} exhibits sharp peaks at the instants of speed changes to provide the required acceleration/deceleration torque, and it then settles once the transient ends. Under constant load (Figure 6.8 (a)), i_{sq} returns to a constant steady-state level, whereas under ramp load (Figure 6.8 (b)) the average i_{sq} trajectory increases over time to supply the progressively increasing torque demand. The switching-related current ripple discussed previously is again visible in both cases, indicating that the dominant ripple component is introduced by PWM inverter switching rather than by discretization of the controller. In addition, the ripple becomes more noticeable at higher operating speeds, consistent with the continuous time case.



(a) (b)

Figure 6.8 Quadrature-axis current tracking with discrete-time PI control (a) constant load equal to 10% of rated torque and (b) ramp load with slope equal to one-tenth of rated torque per second

Figure 6.9 presents the direct-axis current i_{sd} under discrete-time PI control for the constant-load and ramp-load scenarios. Consistent with the continuous-time baseline, the current remains regulated around its zero reference ($i_{sd}^{*} = 0$)throughout the full interval, confirming that field orientation and d–q decoupling are preserved in the sampled implementation. Notably, the i_{sd} response is essentially unchanged by the load profile: apart from brief deviations during speed transitions no sustained d-axis component is introduced, indicating that the increasing torque demand is handled primarily through the i_{sq} channel. A small switching-related ripple is visible and becomes slightly more pronounced at higher operating speeds, while the mean i_{sd} remains around zero.



(a) (b)

Figure 6.9 Direct-axis current tracking with discrete-time PI control (a) constant load equal to 10% of rated torque and (b) ramp load with slope equal to one-tenth of rated torque per second

The corresponding electromagnetic torque response is shown in Figure 6.10 for the constant-load and ramp-load scenarios. As expected, the torque waveform closely follows the same overall shape as the torque-producing current i_{sq} same as continuous time case. Accordingly, torque transients appear at the speed step instants, reflecting the acceleration/deceleration torque requirements, while the post-transient torque level follows the operating point demand. Under constant load the torque settles to an approximately constant value between speed steps, whereas under ramp load it increases over time in line with the imposed disturbance profile, with a small, superimposed ripple component. In both cases, the torque behavior remains consistent with the continuous-time baseline, indicating that discretization at the selected sampling period $T_s$does not introduce a meaningful degradation in torque tracking.

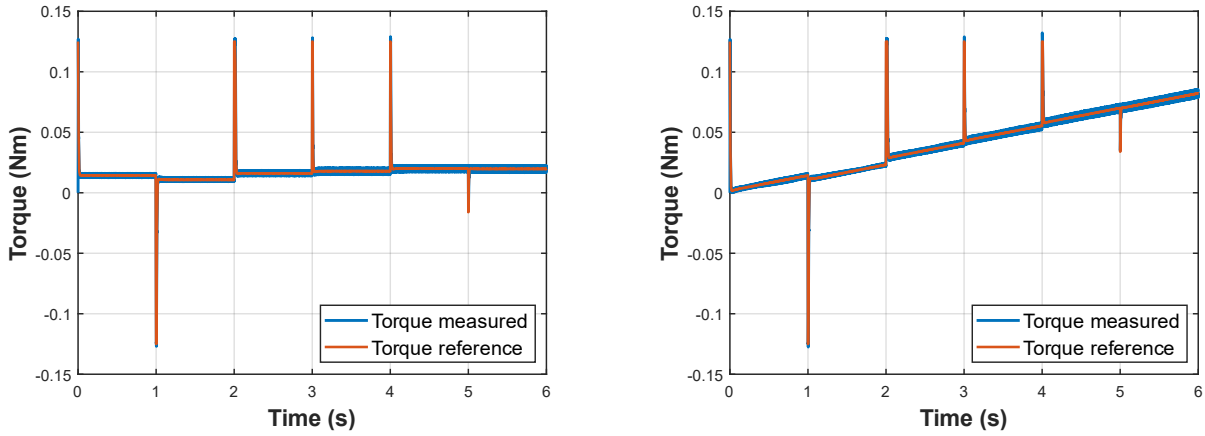


Figure 6.10 Electromagnetic torque response with discrete-time PI control (a) constant load equal to 10% of rated torque and (b) ramp load with slope equal to one-tenth of rated torque per second

To highlight the transient behavior, Figure 6.11 shows a zoomed view of the rotor speed response around the representative speed step for the constant-load and ramp-load scenarios. In both cases, the discrete-time PI controller preserves a well-damped response with fast settling, closely matching the continuous-time baseline. Any minor deviations are attributable to the sampled data nature of the loop, i.e., the control update at T_s, and remain negligible in practice.

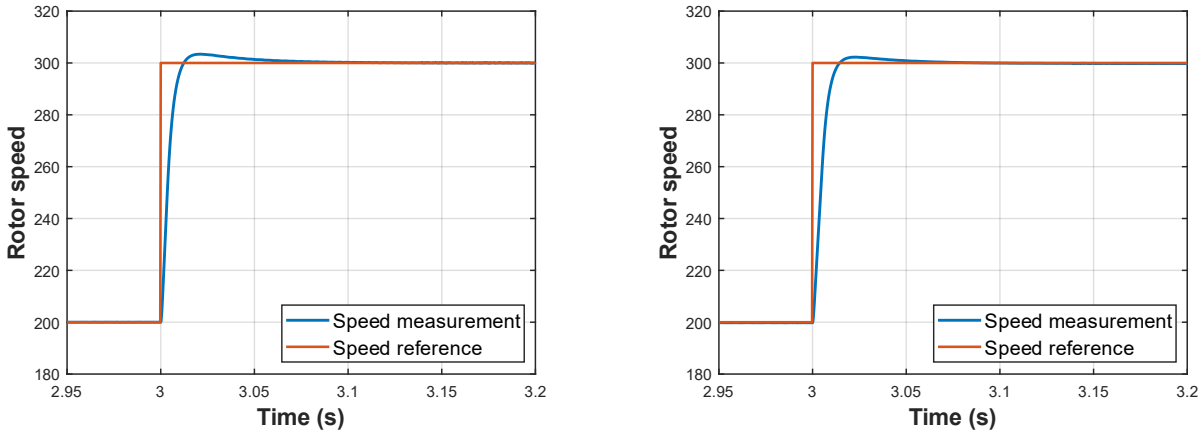


Figure 6.11 Zoomed speed response around the representative speed step (a) constant load equal to 10% of rated torque and (b) ramp load with slope equal to one-tenth of rated torque per second

Figure 6.12 shows the corresponding zoomed response of i_{sq} around the representative speed step for both load scenarios. In both cases, the current peak associated with the speed transition is followed by a rapid decay once the transient ends. Under constant load (Figure 6.12 (a)), i_{sq} settles to a constant steady-state level, whereas in the ramp-load case (Figure 6.12 (b)) the mean current continues to vary slowly after the transient to track the increasing load torque demand. The ripple around the current reference is again visible in both scenarios and remains dominated by switching effects

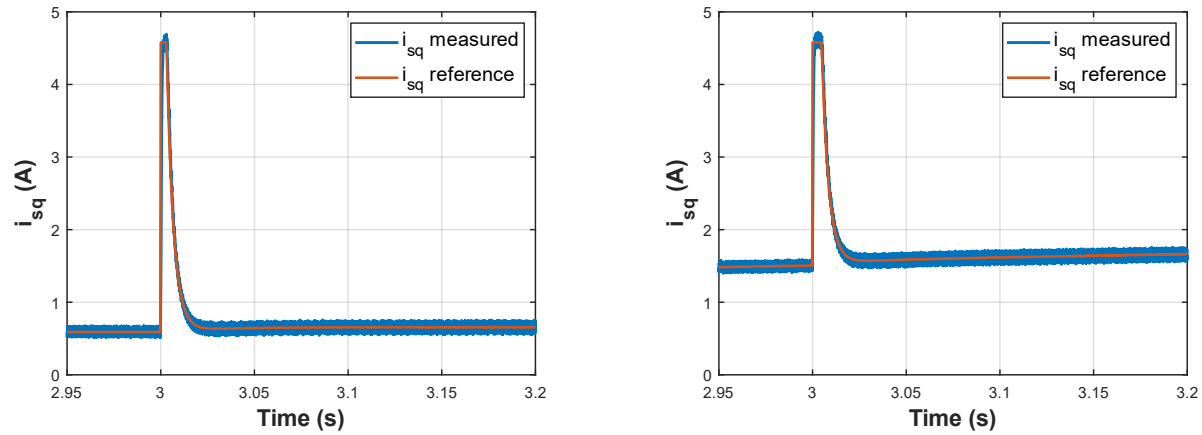
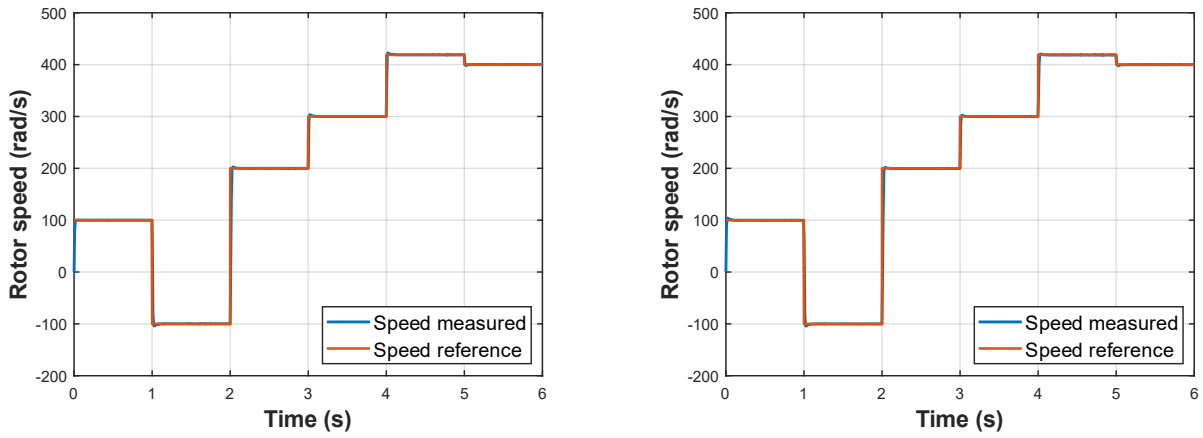


Figure 6.12 Zoomed i_{sq} transient around the representative speed step (a) constant load equal to 10% of rated torque and (b) ramp load with slope equal to one-tenth of rated torque per second

6.1.3 Deadbeat Current Control

The deadbeat current controller is evaluated in floating-point to assess the behavior of the predictive control law under ideal numerical conditions. Since the deadbeat formulation is inherently discrete time, these results primarily serve as a floating-point baseline before introducing hardware related effects.

Figure 6.13 shows the rotor speed response under floating-point deadbeat current control for the constant-load and ramp-load scenarios. In both cases, the measured speed accurately tracks the reference across all commanded speed steps, exhibiting short transients at each transition. The response remains very similar between the two load profiles, indicating that the outer-loop speed dynamics are robust to the gradually increasing disturbance torque introduced in the ramp-load case. Moreover, the speed tracking is comparable to the floating-point PI results, confirming that the outer-loop speed response is largely insensitive to the choice of inner current controller for the operating conditions considered.

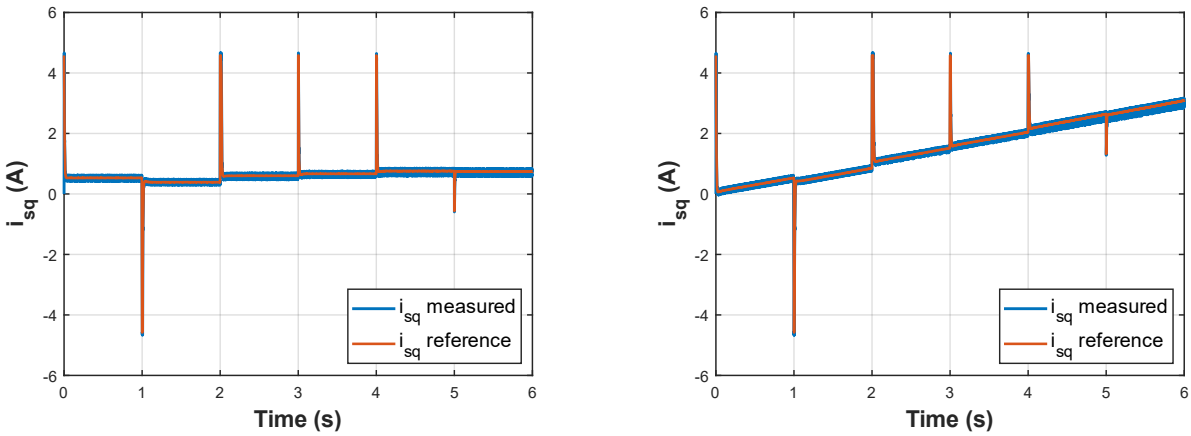


(a) (b)

Figure 6.13 Speed response with deadbeat current control under constant load (a) constant load equal to 10% of rated torque and (b) ramp load with slope equal to one-tenth of rated torque per second

Figure 6.14 reports the quadrature-axis current i_{sq} under floating-point deadbeat current control for the constant-load and ramp-load scenarios. As expected, i_{sq} exhibits sharp peaks at the instants of speed changes to provide the required acceleration/deceleration torque and then returns toward the operating-point level once the transient ends. Under constant load (Figure 6.14(a)), the mean i_{sq} remains approximately constant between speed steps, whereas under ramp load (Figure 6.14(b)) it increases progressively over time to supply the additional electromagnetic torque required by the ramping disturbance. The switching-related ripple is visible in both cases and appears slightly more pronounced overall than in the PI case, while the

step-related current peaks do not show a noticeable overshoot, consistent with the direct corrective action of deadbeat control.



(a) (b)

Figure 6.14 Quadrature axis current tracking with deadbeat current control under constant load (a) constant load equal to 10% of rated torque and (b) ramp load with slope equal to one-tenth of rated torque per second

Figure 6.15 presents the i_{sd} under floating-point deadbeat current control for the constant-load and ramp-load scenarios. In both cases, i_{sd} remains regulated around its zero reference. Unlike the PI current control results, the signal does not exhibit pronounced spikes at the speed-steps, suggesting reduced cross-coupling under deadbeat regulation. A high-frequency ripple component is still present; and it appears slightly more pronounced at lower speeds compared to PI current regulation. Overall, differences with respect to PI are mainly confined to the ripple band and brief transient intervals, while the i_{sd} remains centered around zero.

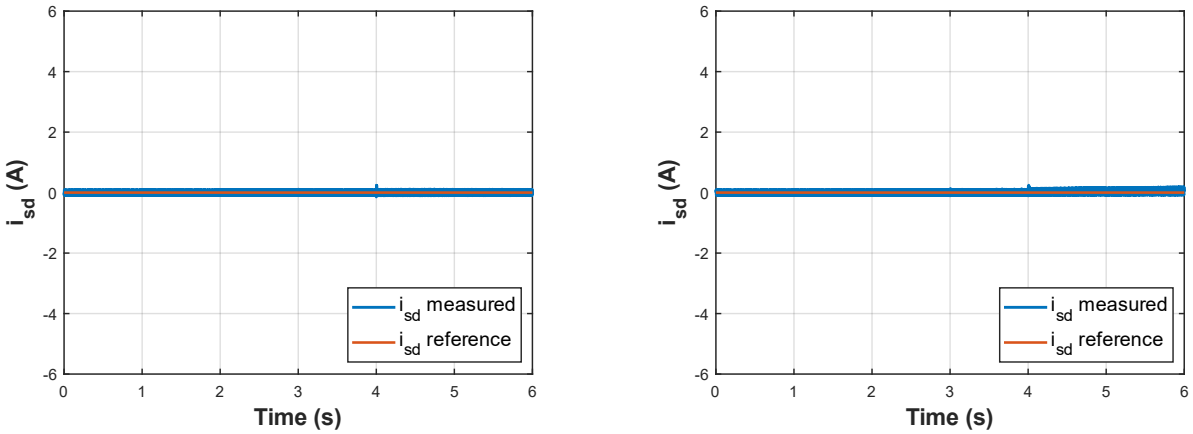


Figure 6.15 Direct-axis current tracking with deadbeat current control under (a) constant load equal to 10% of rated torque and (b) ramp load with slope equal to one-tenth of rated torque per second

The corresponding electromagnetic torque response is shown in Figure 6.16 for the constant-load and ramp-load scenarios. As expected, the torque waveform follows the same overall shape as the torque-producing current i_{sq}, since electromagnetic torque is directly proportional to i_{sq}. Accordingly, torque transients appear at the speed-step instants, reflecting the acceleration/deceleration demand. Under constant load (Figure 6.16 (a)), the torque settles to an approximately constant level between speed steps, whereas under ramp load (Figure 6.16 (b)) the torque increases progressively over time, reflecting the increasing disturbance torque; the same load-driven growth is also observed in the PI ramp-load results. Overall, the deadbeat torque response remains comparable to the floating-point PI baseline in terms of average torque production.

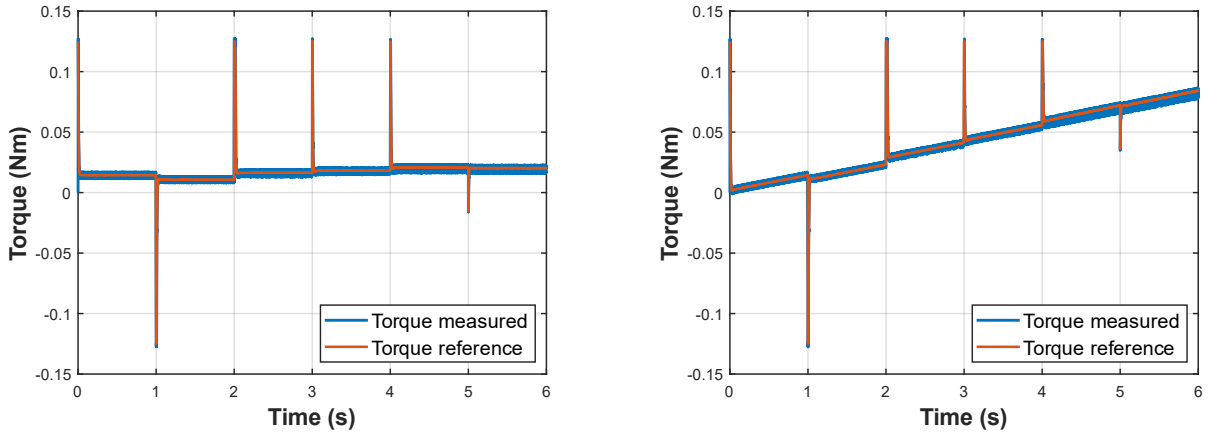


Figure 6.16 Electromagnetic torque with deadbeat control under constant load (a) constant load equal to 10% of rated torque and (b) ramp load with slope equal to one-tenth of rated torque per second

To highlight the transient behavior, Figure 6.17 shows a zoomed view of the rotor speed response around the representative speed step for the constant-load and ramp-load scenarios under floating-point deadbeat current control. In both cases, the response remains well damped with fast settling and closely matches the corresponding PI baseline, with any deviations being minor and confined to the transient interval. The similarity between the constant-load and ramp-load zoomed responses further indicates that the outer-loop speed transient quality is largely unaffected by the gradually increasing disturbance torque in the ramp-load test.

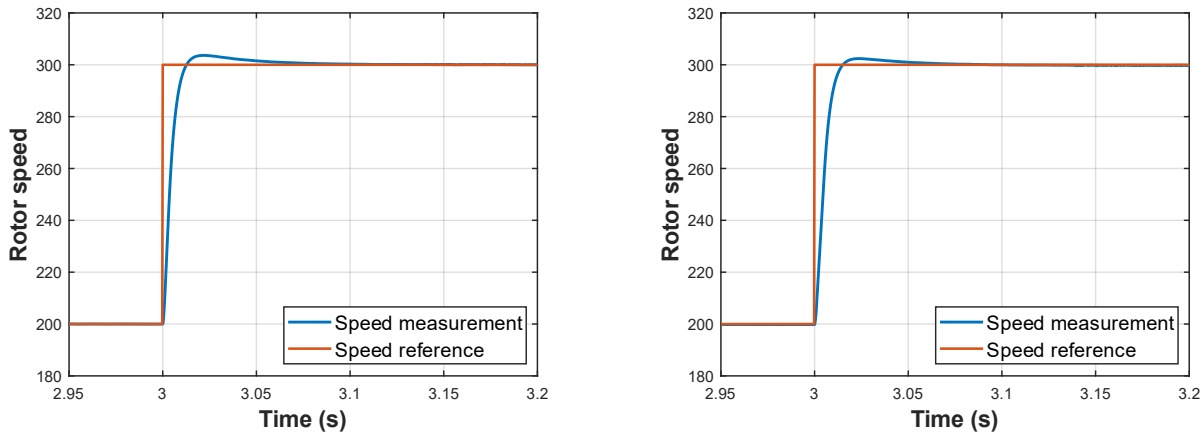


Figure 6.17 Zoomed speed response around the representative speed step (a) constant load equal to 10% of rated torque and (b) ramp load with slope equal to one-tenth of rated torque per second

Figure 6.18 shows the zoomed i_{sq} response around the representative speed step for the constant load and ramp-load scenarios under floating-point deadbeat current control. In both cases, the step-related current peak is followed by a rapid decay once the transient ends, and the response does not exhibit the small overshoot visible in the corresponding PI zoomed responses given in Figure 6.12. Under ramp load (Figure 6.18(b)), the mean current after the transient settles back onto the slowly increasing trajectory required to balance the increasing disturbance torque, while the transient shaping remains essentially unchanged. The remaining high-frequency ripple around the mean current is again dominated by PWM inverter switching effects rather than by the controller formulation.

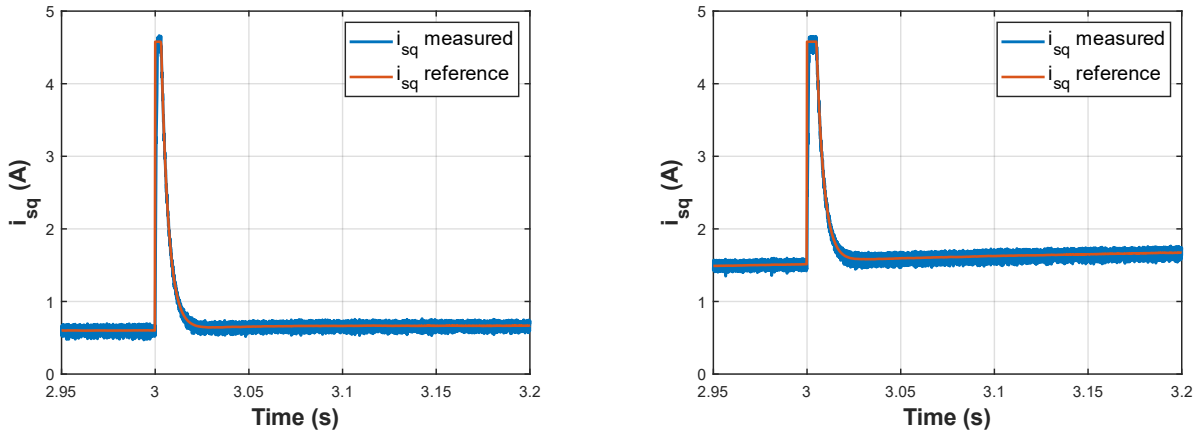


Figure 6.18 Zoomed i_{sq} transient around the representative speed step (a) constant load equal to 10% of rated torque and (b) ramp load with slope equal to one-tenth of rated torque per second

6.1.4 Control Comparison Under Ideal Numerical Conditions

Overall, the rotor speed response is largely unaffected by the choice of current controller. For both constant-load and ramp-load scenarios, the measured speed follows the reference with comparable transient behavior and without a steady-state error. The main differences emerge in the current waveforms. This is also reflected in the step-response metrics for the representative constant-load transient. The i_{sq} rise time decreases from 2.258 ms with PI control to 1.306 ms with deadbeat control, confirming the tighter current transient observed in the zoomed i_{sq} plots. In addition, i_{sd} remains regulated around zero without showing significant peaks during speed transient. On the other hand, the switching related ripple becomes more evident when deadbeat control is used. This behavior is consistent with the more aggressive, one-step-ahead nature of deadbeat control, which tends to be more sensitive to inverter switching ripple and measurement noise than PI control. Therefore, the floating-point

results already indicate a trade-off: deadbeat improves current transient shaping at the cost of increased visible ripple in the regulated currents.

To quantify the differences observed in the ripple content, Table 6.1 reports representative metrics extracted from the constant-load test. The evaluation window is selected as $t \in [3,4]$ s, corresponding to the same transient interval highlighted in the zoomed plots.

Table 6.1 Comparison of steady-state current ripple for PI and deadbeat current controllers under constant-load condition

Controller	i_{sq}^{ref}	$\Delta i_{sq,pp}\ (A)$	$\Delta i_{sq,pp}$~(\%)	$\Delta i_{sd,pp}(A)$
PI	0.6582	0.1376	20.9%	0.103
Deadbeat	0.6656	0.1494	22.4%	0.138

6.2 FPGA-in-the-Loop Results

The FPGA-in-the-Loop (FIL) validation of the discrete-time PI current controllers is presented in this subsection. The same cascaded control structure used in the floating-point discrete-time simulations is retained; however, in FIL the controller executes in hardware. Therefore, any deviation from the floating-point waveforms can be attributed to implementation realism rather than to changes in the control structure.

6.2.1 Resource Utilization and Timing

Table 6.2 summarizes the post-implementation resource utilization and timing results for the FIL controller mapped to the ZedBoard (Xilinx Zynq-7000, part: XC7Z020CLG484-1). The FIL–PI implementation meets the 25 MHz clock constraint with positive slack (WNS = +0.383 ns), confirming timing closure for the current build. In terms of resources, the design is primarily DSP-dominated while LUT/FF and BRAM consumption remain comparatively low, leaving margin for additional instrumentation and future extensions. The deadbeat implementation satisfies the same 25 MHz clock constraint with a larger positive timing margin (WNS = +1.257 ns). In addition, it results in a small but consistent reduction in resource usage compared to the PI baseline, most notably in DSP blocks (107 vs 109) and flip-flops (6661 vs 6903), while LUT and BRAM utilization remain essentially unchanged. Overall, this indicates that the deadbeat datapath is not more demanding on the target FPGA and, for the current build, even provides improved implementation margin in both timing and hardware resources.

Table 6.2 Post-implementation WNS and FPGA resource utilization for the FIL implementations

Metric	FIL–PI	FIL–Deadbeat
Clock constraint (MHz)	25	25
WNS (ns)	+0.383	+1.257
LUT	7165 / 53200 (13.47%)	7125 / 53200 (13.39%)
LUTRAM	226 / 17400 (1.30%)	226 / 17400 (1.30%)
FF	6903 / 106400 (6.49%)	6661 / 106400 (6.26%)
BRAM	5.5 / 140 (3.93%)	5.5 / 140 (3.93%)
DSP	109 / 220 (49.55%)	107 / 220 (48.64%)
IO	2 / 200 (1.00%)	2 / 200 (1%)
BUFG	3 / 32 (9.38%)	3 / 32 (9.38%)
MMCM	1 / 4 (25.00%)	1 / 4 (25%)

6.2.2 FPGA-in-the-Loop Performance of PI Current Control

Figure 6.19 shows the rotor speed response obtained in the FIL configuration with PI current control for the constant-load and ramp-load scenarios. In both cases, the measured speed accurately tracks the reference profile across all commanded steps, including the sign reversal, confirming that stable outer-loop speed regulation is preserved after switching to hardware implementation. Compared to the floating-point discrete-time baseline, the overall transient shape remains consistent; however, small differences become visible during the transition intervals, where the FIL response exhibits a slightly modified peak and settling profile. These deviations are expected and originate from deterministic computation, fixed-point arithmetic, and modulation timing in the FIL loop rather than from changes in the control structure. Under ramp-load conditions (Figure 6.19(b)), the speed remains tightly regulated despite the continuously increasing disturbance torque, indicating that the observed deviations are implementation-related and not load-driven.

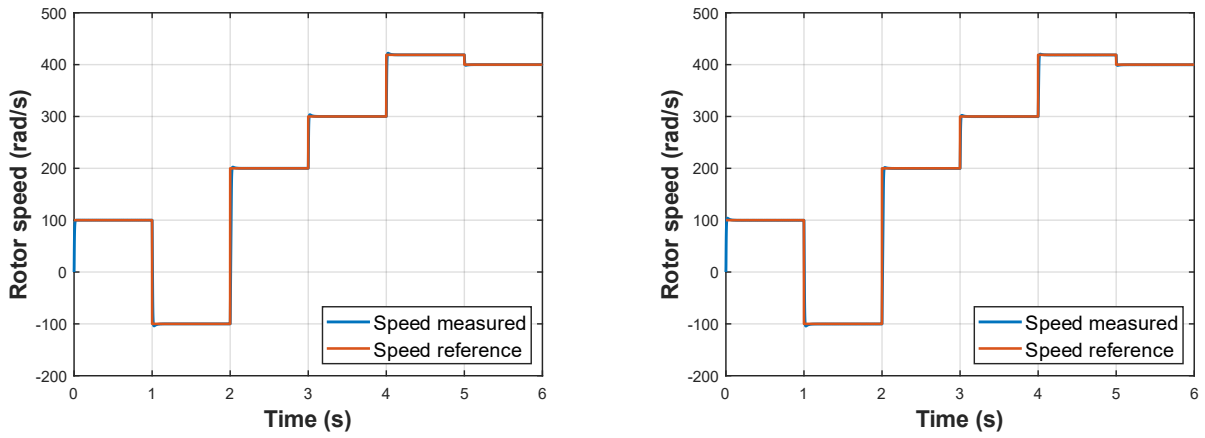
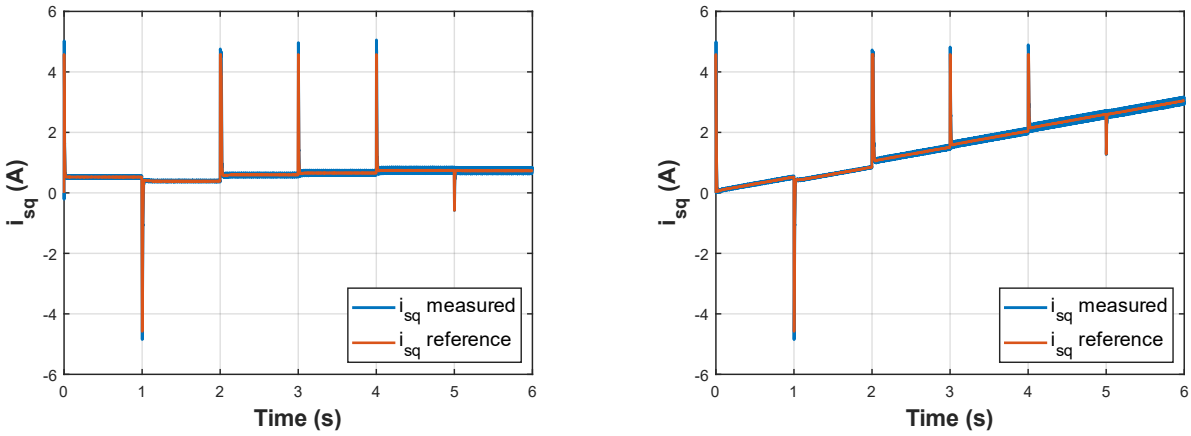


Figure 6.19 Speed response in FIL with PI current control under constant load (a) constant load equal to 10% of rated torque and (b) ramp load with slope equal to one-tenth of rated torque per second

Figure 6.20 reports the torque-producing current component i_{sq} in the FIL configuration with PI current control for the constant-load and ramp-load scenarios. As expected in the cascaded structure, sharp current demands occur at each speed step to generate the acceleration/deceleration torque, after which i_{sq} returns toward the operating-point level. Under ramp load (Figure 6.20(b)), the mean i_{sq} trajectory increases progressively between speed steps to supply the rising electromagnetic torque required to reject the increasing mechanical load, while the step-related transient peaks remain qualitatively similar to the constant-load case. A visible high-frequency ripple band is present in both scenarios, consistent with inverter PWM switching and the finite stator inductance. A visible high-frequency ripple band is

present in both scenarios, consistent with inverter PWM switching and the finite stator inductance. In the FIL implementation, additional implementation effects such as fixed-point arithmetic and finite numerical resolution, can make the ripple component more pronounced and visually enlarge the ripple envelope. Overall, the measured i_{sq} follows the reference trend, while the ripple visibility tends to increase at higher operating current levels.



(a) (b)

Figure 6.20 Quadrature-axis current in FIL with PI current control under constant load (a) constant load equal to 10% of rated torque and (b) ramp load with slope equal to one-tenth of rated torque per second

Figure 6.21 presents the flux-producing current component i_{sd} obtained in the FPGA-in-the-loop (FIL) configuration with PI current control for the constant-load and ramp-load scenarios. In both cases, the controller maintains i_{sd} close to its zero reference throughout the full sequence, confirming that correct field-oriented operation and d–q decoupling are preserved after hardware mapping. Deviations are primarily limited to short transients at the speed-step instants, while no sustained d-axis component is introduced even in the presence of a gradually increasing load torque. A small high-frequency ripple component is visible around the mean current, consistent with PWM switching and finite machine inductance; in the FIL implementation, fixed-point arithmetic and deterministic execution can further affect the apparent ripple envelope compared to idealized floating-point simulations.

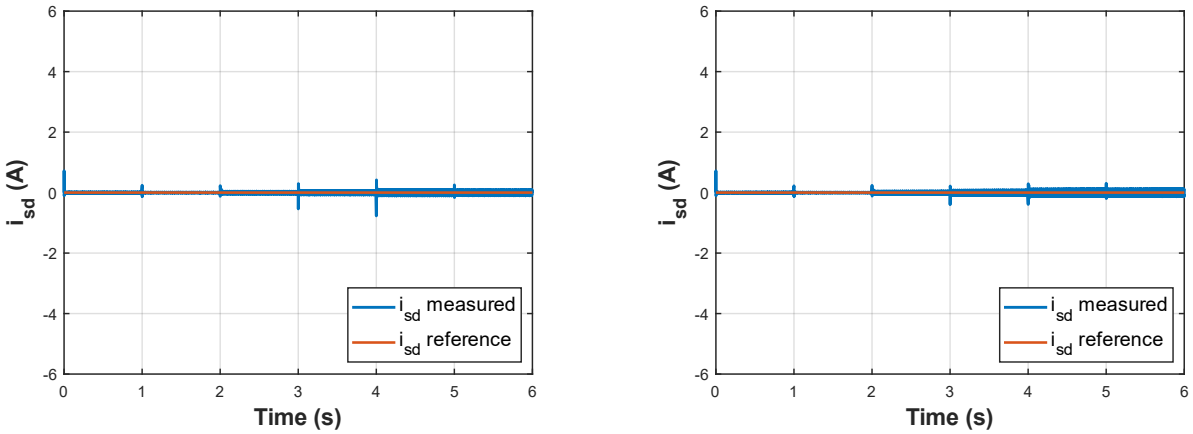


Figure 6.21 Direct-axis current FIL with PI current control under constant load (a) constant load equal to 10% of rated torque and (b) ramp load with slope equal to one-tenth of rated torque per second

The corresponding electromagnetic torque responses are shown in Figure 6.22, where Figure 6.22 (a) corresponds to the constant-load case and Figure 6.22 (b) corresponds to the ramp-load case. In both cases, torque spikes coincide with the speed reference transitions, consistent with the transient i_{sq}peaks required to accelerate the rotor. After each transient, the torque settles to the value required by the operating point. In the constant-load case (Figure 6.22 (a)), the torque baseline remains approximately constant between speed steps, reflecting the steady torque demand under fixed load. In the ramp-load case (Figure 6.22 (b)), the torque baseline increases progressively during the ramp interval, matching the upward trend in $i_{sq}$and the increasing load demand, while maintaining the same step-induced transient peaks. The measured torque closely follows the reference in both subfigures, with a small ripple component that is expected from current ripple and the discrete-time actuation of the inverter.

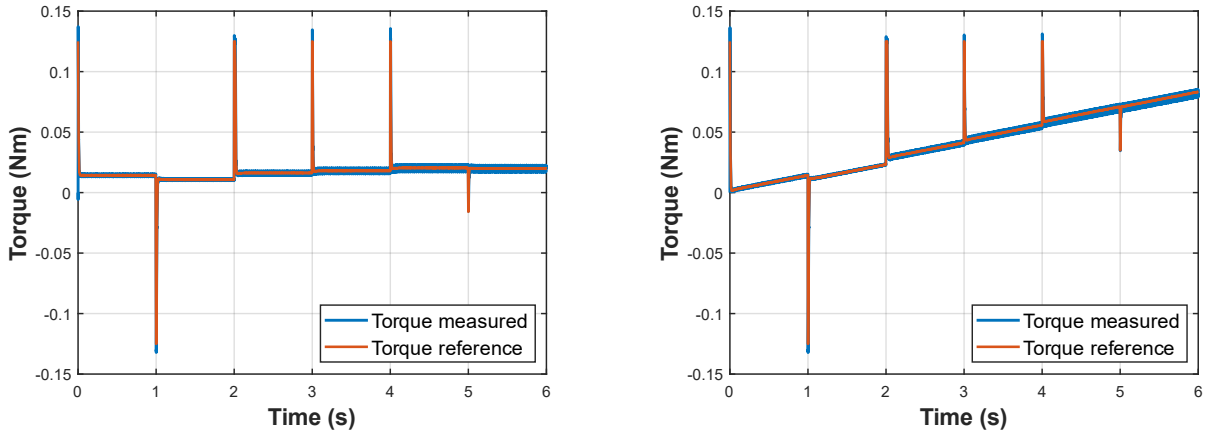


Figure 6.22 Electromagnetic torque response in FIL with PI current control under constant load (a) constant load equal to 10% of rated torque and (b) ramp load with slope equal to one-tenth of rated torque per second

To emphasize the transient behavior, Figure 6.23 shows a zoomed view of the speed response around a representative step, where Figure 6.23 (a) corresponds to the constant-load case and Figure 6.23 (b) corresponds to the ramp-load case. In both subfigures, the FIL response remains well damped and settles rapidly to the reference value. A small overshoot is visible immediately after the step, followed by a smooth decay to the steady-state trajectory. The transient is short and the measured speed converges quickly to the reference, which is consistent with a finite-bandwidth outer speed loop combined with discrete-time updates and practical actuator limits.

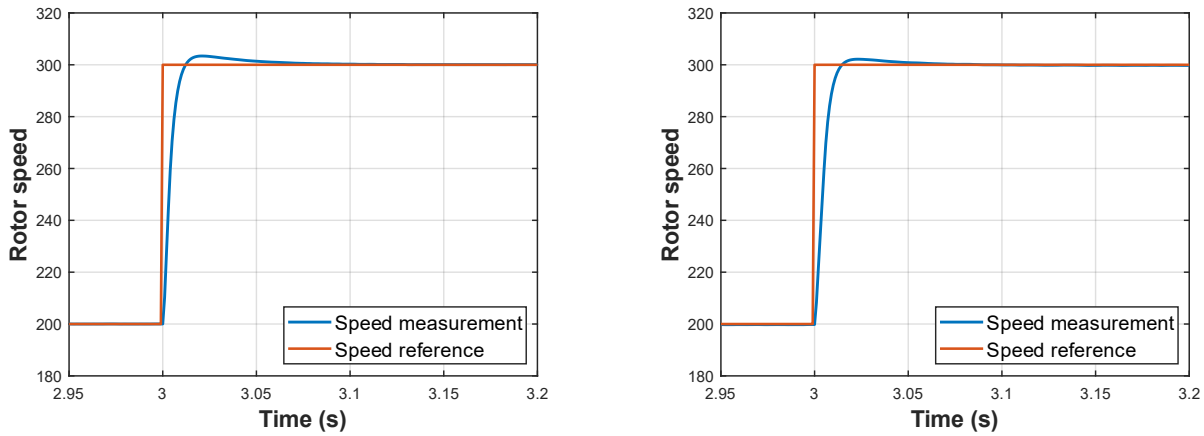
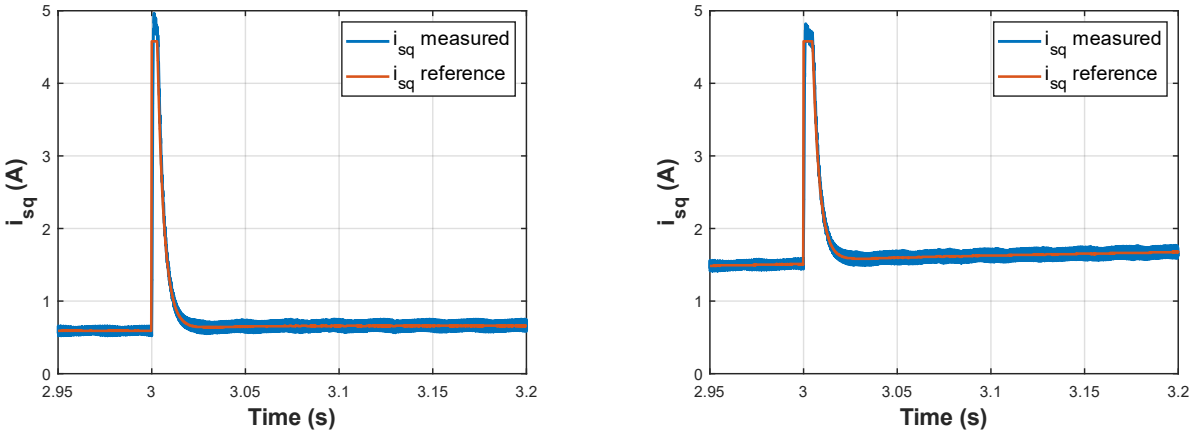


Figure 6.23 Zoomed view of the speed transient in FIL with PI current control under constant load (a) constant load equal to 10% of rated torque and (b) ramp load with slope equal to one-tenth of rated torque per second

Figure 6.24 shows the corresponding zoomed i_{sq} response over the same interval, where Figure 6.24 (a) corresponds to the constant-load case and Figure 6.24 (b) corresponds to the ramp-load case. In both subfigures, the current rises rapidly to meet the acceleration demand and then decays from the transient peak toward the steady-state level (constant-load) or the ramp-dependent operating value (ramp-load). The measured i_{sq} trajectory remains close to the reference, while the ripple band becomes clearly visible in the zoomed view. In addition, the reference exhibits a slight "stutter" pattern, which is most likely caused by finite numerical resolution in the discrete-time FIL signal chain rather than by the motor dynamics. This effect becomes visible only at high zoom levels and does not affect the overall tracking conclusions; however, it is useful for highlighting implementation effects that are not apparent in the ideal floating-point baseline.



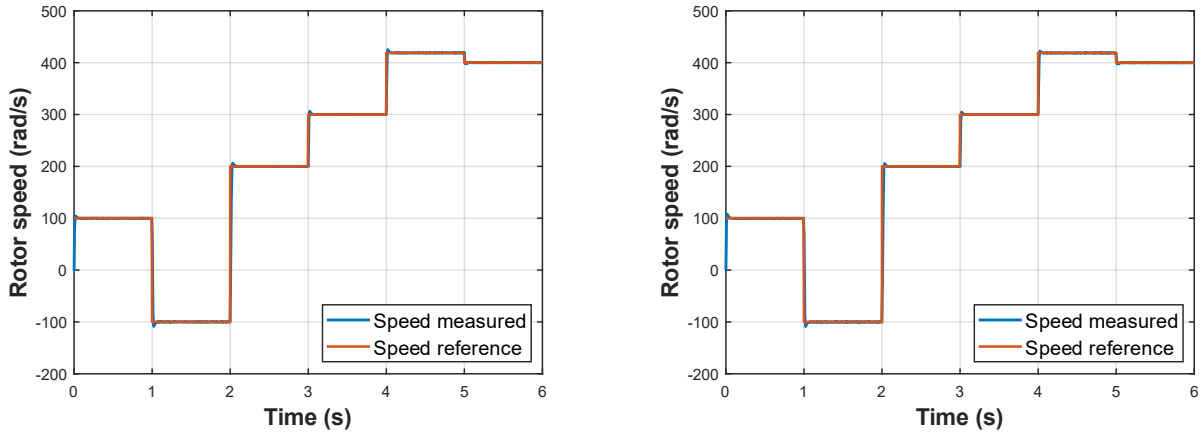
(a) (b)

Figure 6.24 Zoomed view of the i_{sq} transient in FIL with PI current control under constant load (a) constant load equal to 10% of rated torque and (b) ramp load with slope equal to one-tenth of rated torque per second

6.2.3 FPGA-in-the-Loop Performance of Deadbeat Current Control

Figure 6.25 shows the rotor speed response obtained with the deadbeat current controller in the FPGA in the Loop configuration, where Figure 6.25(a) corresponds to the constant load case and Figure 6.25(b) corresponds to the ramp load case. In both scenarios, the measured speed accurately follows the reference profile over all commanded steps, including the speed reversal, with short transients at each transition. The overall behavior remains comparable to the FIL PI case, confirming that the outer speed loop dynamics are preserved when replacing the PI current controllers with the deadbeat current controllers in hardware. Under ramp load (Figure 6.25(b)), the progressive increase of load torque does not introduce visible speed instability; instead, the controller compensates by adjusting the torque demand, keeping speed

regulation essentially unchanged. Any minor deviations around the transients remain comparable to the PI ramp load case, indicating that the outer speed response is not significantly affected by the choice of inner current controller, provided the current loop remains stable and sufficiently fast.



(a) (b)

Figure 6.25 Speed response in FIL with deadbeat current control under constant load (a) constant load equal to 10% of rated torque and (b) ramp load with slope equal to one-tenth of rated torque per second

Figure 6.26 reports the quadrature axis current i_{sq} in the FIL configuration with deadbeat current control, where Figure 6.26 (a) corresponds to the constant load case and Figure 6.26 (b) corresponds to the ramp load case. At each speed transition, the deadbeat controller generates sharp current peaks to provide the required acceleration or deceleration torque. Compared to the FIL PI case, the i_{sq} transient response is more abrupt and settles more rapidly to the steady state level demanded by the operating point, which is consistent with the predictive nature of deadbeat control that computes the control action to enforce the desired current value within one sampling period under nominal conditions. Under ramp load the i_{sq} reference increases progressively to supply the additional electromagnetic torque required to balance the rising load, and the measured i_{sq} follows this trend closely across the interval. In both subfigures, the deadbeat FIL traces exhibit a visibly stronger ripple band around the mean trajectory than the corresponding FIL PI results and the ideal floating point deadbeat baseline, which is consistent with the combination of inverter PWM switching and the more aggressive discrete time voltage action of the deadbeat law under fixed point arithmetic

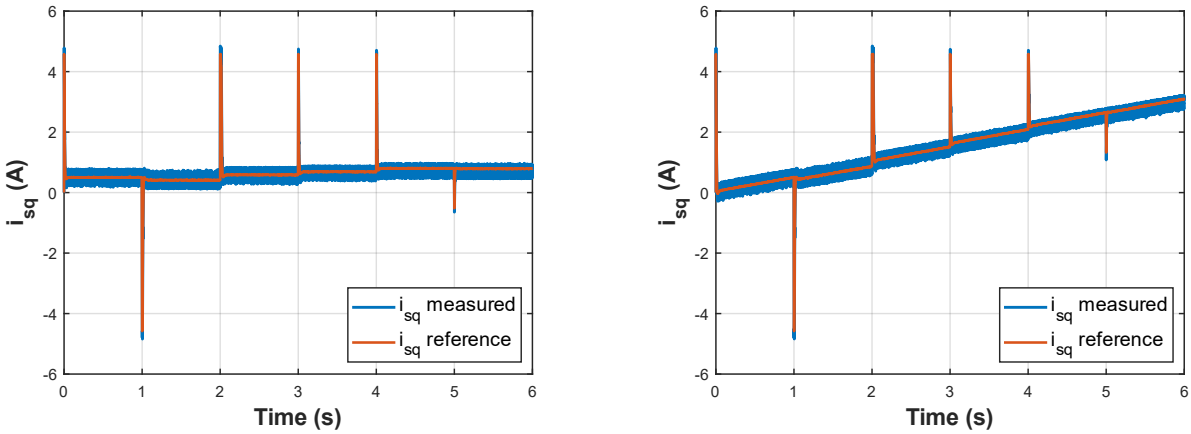


Figure 6.26 Quadrature-axis current in FIL with deadbeat current control under constant load (a) constant load equal to 10% of rated torque and (b) ramp load with slope equal to one-tenth of rated torque per second

Figure 6.27 presents the direct axis current i_{sd} in the FIL configuration with deadbeat current control, where Figure 6.27(a) corresponds to the constant load case and Figure 6.27(b) corresponds to the ramp load case. In both scenarios, the controller maintains i_{sd} tightly regulated around its zero reference throughout the sequence, confirming correct field-oriented operation. No pronounced i_{sd} excursions are observed at the speed step instants, indicating reduced transient cross coupling between the current axes compared to the PI case, while the mean value remains close to zero across the entire interval. As in the i_{sq} case, a switching related ripple band is clearly visible and appears more pronounced than in the corresponding FIL PI results and the ideal floating-point baseline, especially under ramp load as the operating point shifts, but without affecting the average regulation.

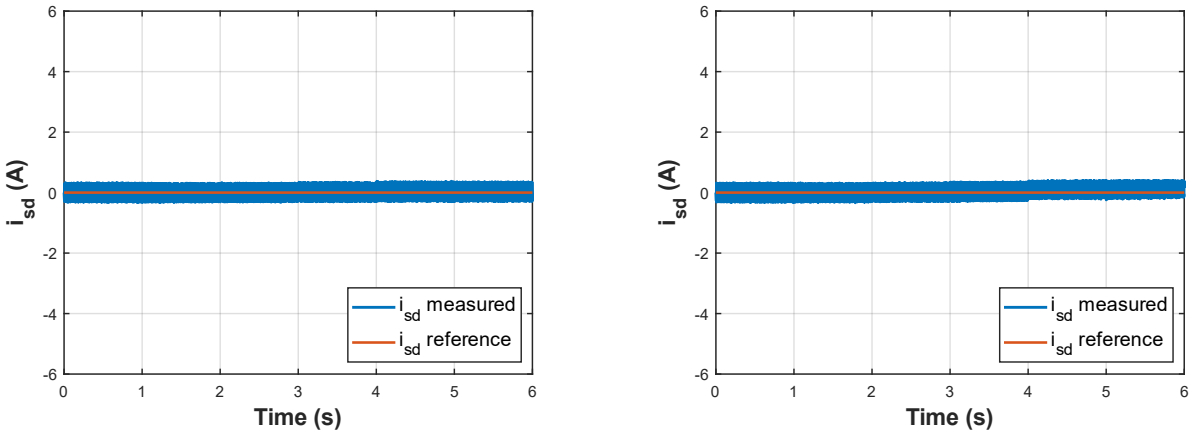


Figure 6.27 Direct-axis current in FIL with deadbeat current control under constant load (a) constant load equal to 10% of rated torque and (b) ramp load with slope equal to one-tenth of rated torque per second

The corresponding electromagnetic torque responses are shown in Figure 6.28, where Figure 6.28(a) corresponds to the constant-load case and Figure 6.28(b) corresponds to the ramp-load case. In both scenarios, torque transients coincide with the speed reference steps, reflecting the acceleration and deceleration demands imposed by the outer speed loop. After each transient, the torque settles to the level required by the current operating point. In the constant-load case, the torque baseline remains approximately constant between speed steps, while in the ramp-load case the mean torque increases progressively in response to the rising mechanical load. The measured torque closely follows the reference in both subfigures, and the overall torque profile remains consistent with the FIL PI baseline. Despite the more aggressive current action of the deadbeat controller, no instability or low-frequency oscillations are observed, confirming that the deadbeat formulation achieves the required torque production while preserving stable speed regulation under both loading conditions.

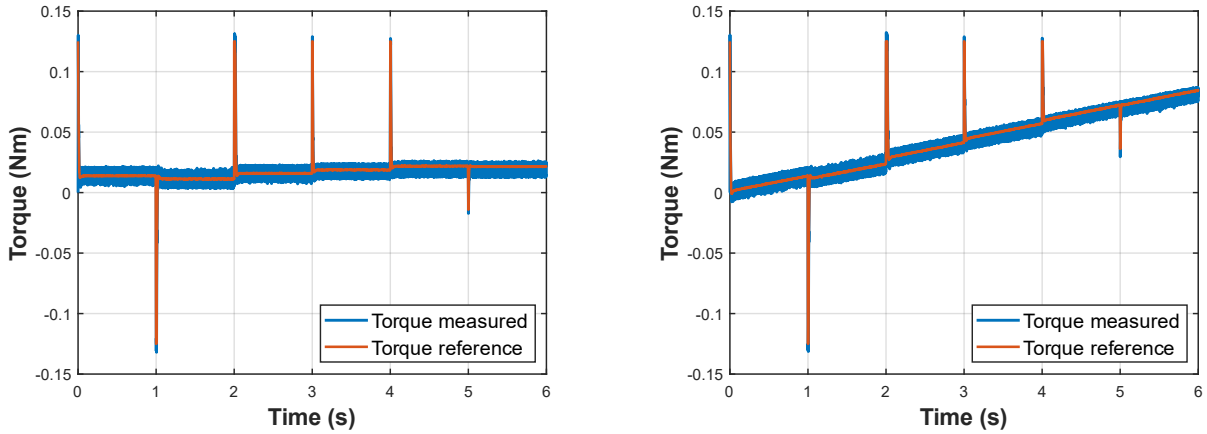
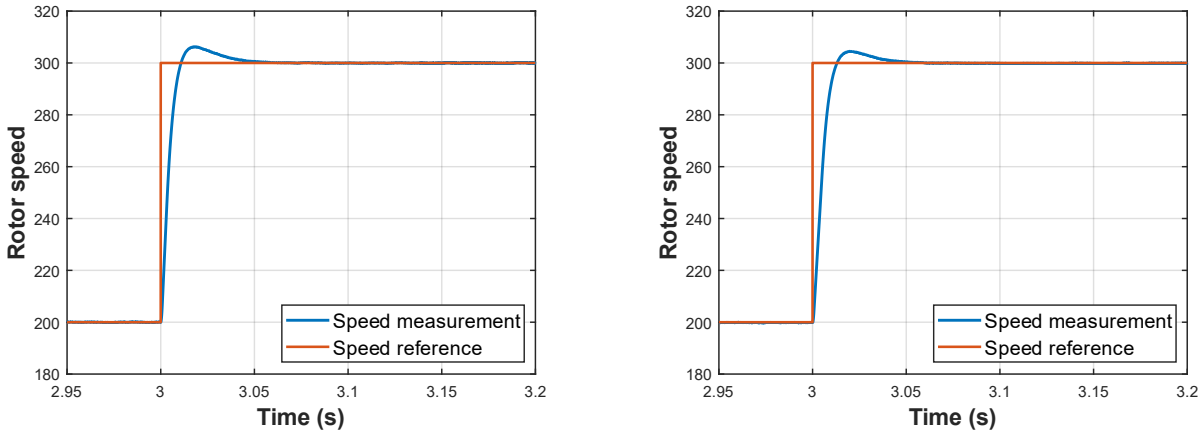


Figure 6.28 Electromagnetic torque response in FIL with deadbeat current control under constant load (a) constant load equal to 10% of rated torque and (b) ramp load with slope equal to one-tenth of rated torque per second

To emphasize the transient behavior, Figure 6.29 shows a zoomed view of the rotor speed response around the representative speed step used throughout this chapter, where Figure 6.29(a) corresponds to the constant-load case and Figure 6.29(b) corresponds to the ramp-load case. In both subfigures, the response remains well damped and settles rapidly to the reference value, with a transient shape comparable to the FIL–PI case. A small overshoot is visible immediately after the step, followed by a smooth decay to the steady-state trajectory. Under ramp-load operation (Figure 6.29(b)), the transient behavior remains very similar to the constant-load case, and no additional oscillations or degradation in damping are observed. Any minor differences are confined to the immediate transient interval and are consistent with differences in the inner-loop current response under deterministic FPGA execution. Overall, the zoomed response indicates that the outer speed-loop behavior is essentially unchanged across the two loading conditions, provided the inner current loop remains stable and sufficiently fast.



(a) (b)

Figure 6.29 Zoomed view of the speed transient in FIL with deadbeat current control under constant load (a) constant load equal to 10% of rated torque and (b) ramp load with slope equal to one-tenth of rated torque per second

Figure 6.30 shows the corresponding zoomed view of the quadrature axis current i_{sq}, where Figure 6.30(a) corresponds to the constant-load case and Figure 6.30(b) corresponds to the ramp-load case. In both subfigures, the current rises sharply to the peak value required by the speed step and then decays rapidly toward the operating point level. In the ramp-load case, after the transient the current continues to follow the ramped reference. Compared to the FIL PI case, the transient duration is shorter, and the high frequency ripple band is significantly more visible, which is expected since the PI current controller provides an inherent low pass filtering effect through its closed loop dynamics, whereas the deadbeat law applies a more direct and aggressive discrete time voltage action. In addition, the ripple envelope is more pronounced than in the ideal floating point deadbeat baseline. This is consistent with FIL implementation effects such as fixed-point arithmetic and limited numerical resolution, including division approximations in the FPGA implementation, which reduce overall control accuracy and tend to amplify switching ripple and quantization induced components. Overall, the measured i_{sq} tracks the reference closely in both load cases, while exhibiting the expected trade off in deadbeat current control.

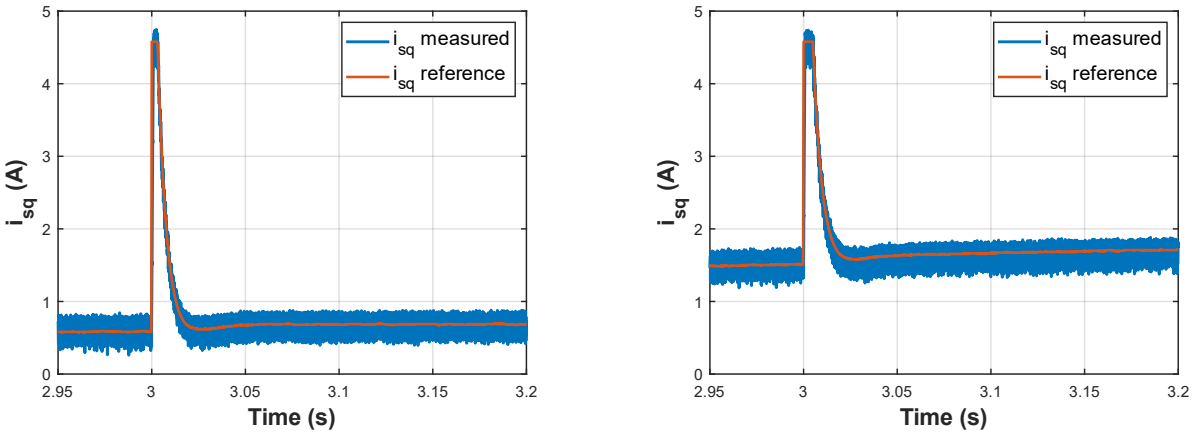


Figure 6.30 Zoomed view of the quadrature-axis current transient in FIL with deadbeat current control (a) constant load equal to 10% of rated torque and (b) ramp load with slope equal to one-tenth of rated torque per second under constant load

6.2.4 Effects of Fixed-Point and Deterministic FPGA Execution

Under FPGA-in-the-loop execution, the rotor speed response remains largely unaffected by the choice of current controller. For the constant-load condition, the measured speed follows the reference without steady-state error for both PI and deadbeat control. The transient behavior is comparable, confirming that the outer-loop dynamics are not significantly influenced by the fixed-point implementation of the inner current controller.

The main differences again appear in the current waveforms, but with a stronger impact than in the floating-point simulation case. The i_{sq} rise time for PI current controller was 2.258 ms for floating point case. In FIL simulation this value is 2.153 ms. Similarly rise time for deadbeat current controller was 1.306 ms with deadbeat control. On the other hand, rise time was calculated as 1.288ms for the FIL deadbeat current controller. This confirms that the tighter current transient is still observed while both controllers maintaining similar rise time compared to their ideal cases. The small differences between floating-point and FIL rise times can be attributed to fixed-point quantization. Coefficient rounding slightly modifies the effective loop gain and prediction accuracy, which affects PI and deadbeat controllers differently due to their distinct control structures. Nevertheless, the variations remain marginal, and the relative transient characteristics of the two controllers are preserved.

The steady-state ripple metrics under FIL execution are reported in Table 6.3, allowing direct comparison with the floating-point simulation results presented in For the PI controller, the peak-to-peak ripple of i_{sq} increases to 0.1533 A (23.2%), compared to 0.1376 A (20.9%) in simulation. The i_{sd} ripple rises to 0.1391 A. This moderate increase

is attributed to fixed-point quantization, ADC resolution limits, and the deterministic multi-cycle latency introduced by the FPGA pipeline. Despite these non-idealities, the PI controller preserves stable and smooth current regulation.

In contrast, the deadbeat controller exhibits a more pronounced increase in ripple under FIL conditions. The peak-to-peak i_{sq} ripple reaches 0.2276 A (33.1%), compared to 0.1494 A (22.4%) in the floating-point case. Similarly, the i_{sd} ripple increases to 0.2432 A. This behavior reflects the higher sensitivity of deadbeat control to implementation-induced effects. Because the control action is computed from a one-step prediction of the discrete-time model, any mismatch caused by fixed-point truncation, quantization noise, or pipeline delay directly affects the voltage command. As a result, switching-related ripple and measurement noise become more visible in the regulated currents.

Compared to simulation, the FIL results therefore amplify the trade-off already identified, deadbeat control maintains sharper transient current shaping but exhibits significantly higher steady-state ripple under realistic hardware constraints. The PI controller, although slightly slower in transient shaping, demonstrates greater robustness to quantization and deterministic execution delay.

Overall, the FIL experiments confirm that fixed-point arithmetic and pipeline latency are critical factors in predictive current control implementation. While deterministic FPGA execution ensures repeatable timing, predictive strategies such as deadbeat control require careful latency-aware modeling and scaling optimization to maintain their theoretical advantages in practical hardware realization.

Table 6.3 Comparison of steady-state current ripple for PI and deadbeat current controllers in FIL under constant-load condition

Controller	i_{sq}^{*}	$\Delta i_{sq,pp}\ (A)$	$\Delta i_{sq,pp}\sim(\backslash\%)$	$\Delta i_{sd,pp}(A)$
PI	0.6616	0.1533	23.2%	0.1391
Deadbeat	0.6885	0.2276	33.1%	0.2432

6.3 Experimental Results of VHDL Implementation

In this chapter, the performance of the implemented PI and deadbeat current controllers is experimentally evaluated on the FPGA platform. Both controllers are realized in fixed-point arithmetic and executed within the programmable logic (PL) of the FPGA. The evaluation focuses on current tracking behavior, transient response, and steady-state ripple characteristics under representative operating conditions. The experimental validation was carried out using the laboratory test bench shown in Figure 6.31.

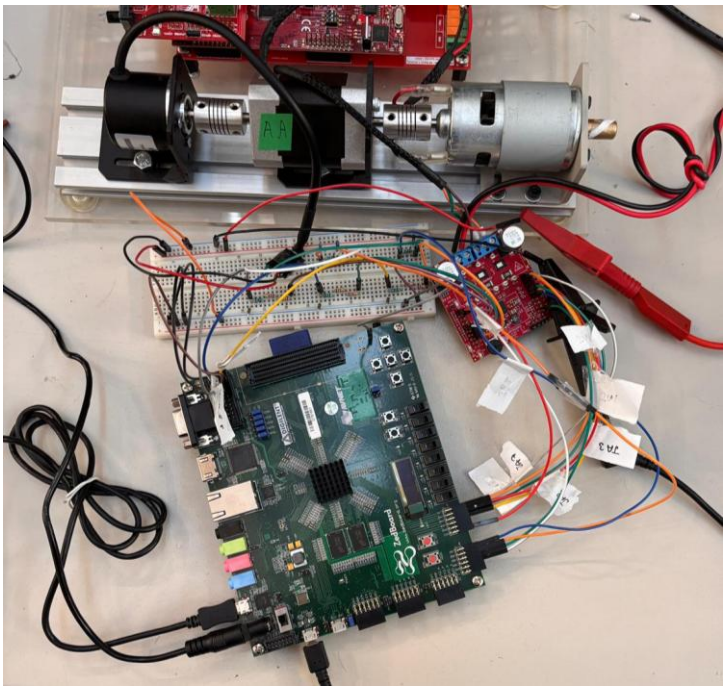


Figure 6.31 Experimental test setup used for validation of the control algorithms

It should be noted that, in this section, only the inner current control loops are implemented and evaluated in hardware. The outer speed control loop is intentionally excluded from the VHDL-based experimental analysis. This choice is motivated by the fact that the primary objective of this work is to investigate the dynamic behavior, latency sensitivity, and hardware efficiency of discrete-time current control strategies. Since the speed controller operates at a significantly lower bandwidth than the current loop, its influence on high-frequency ripple, pipeline latency effects, and predictive current behavior is negligible. By isolating the current controllers, the impact of fixed-point arithmetic, deterministic FPGA execution, and pipeline delay can be evaluated directly without the additional dynamic filtering introduced by the outer loop. This separation enables a clearer and more focused comparison between PI and deadbeat current control implementations.

For data acquisition and visualization, a UART-based communication interface is employed to transfer measured signals from the programmable logic (PL) to the

processing system (PS), and subsequently to the host computer. Although UART provides a simple and robust solution for debugging and data logging, it operates at a significantly lower bandwidth compared to dedicated high-speed communication interfaces. Therefore, the sampling rate available for visualization on the host side is limited. Fine switching-frequency ripple components cannot be faithfully reconstructed in real time, and the displayed waveforms may appear as discrete or irregular steps rather than continuous high-resolution signals.

Nevertheless, the transmitted data clearly demonstrate that the implemented controllers can track the current reference accurately, confirming the correct functional behavior of the VHDL implementation. The absence of visibly resolved ripple in the plotted signals should therefore be interpreted as a limitation of the communication bandwidth and visualization process rather than an indication of reduced control performance. The internal operation of the controller within the PL, including its deterministic timing and high-frequency execution, remains unaffected by the UART interface.

This approach provides a practical compromise between implementation complexity and experimental observability, enabling reliable comparative analysis of PI and deadbeat current control performance under identical hardware conditions.

6.3.1 Resource Utilization and Timing

Table 6.4 summarizes the post-implementation timing and FPGA resource utilization results for the custom VHDL realization of the PI and deadbeat current controllers. Both designs successfully meet timing at a clock constraint of 100 MHz, with positive worst negative slack (WNS) values of +1.983 ns for the PI controller and +2.292 ns for the deadbeat controller. Achieving timing closure at 100 MHz represents a substantial improvement in operating frequency compared to the MATLAB-generated implementation, which is limited to 25 MHz. The fourfold increase in clock frequency provides significantly greater timing margin and scalability for higher control-loop execution rates.

At 100 MHz, each clock period corresponds to 10 ns. The longest data path between pipeline boundaries spans 34 clock cycles for the PI controller and 33 clock cycles for the deadbeat controller, resulting in deterministic computational latencies of approximately 340 ns and 330 ns, respectively. These latencies are fixed by design and remain fully repeatable due to the synchronous FPGA pipeline structure. The slightly shorter latency of the deadbeat implementation does not introduce additional hardware overhead and confirms that predictive computation has been integrated efficiently within the datapath.

Similar to MATLAB-generated design, deadbeat current controller showed slight improvement in terms of WNS and resource utilization compared to PI current

controller. In terms of logic utilization, the custom VHDL implementation exhibits significantly lower resource consumption compared to the MATLAB-generated design. The PI controller occupies 6.51% of LUTs and 3.62% of flip-flops, while the deadbeat controller uses 6.20% of LUTs and 3.52% of flip-flops. These values are approximately half of the logic utilization observed in the MATLAB-based implementation, indicating a more compact and structurally optimized design. The most pronounced difference appears in DSP utilization. Custom architecture requires only 18 DSP blocks (8.18%), whereas the MATLAB-generated implementation consumes a substantially larger portion of available DSP resources. This represents the largest relative improvement and highlights the effectiveness of manual arithmetic scheduling and resource sharing in the VHDL implementation.

Overall, the results demonstrate a considerable improvement in both timing performance and hardware efficiency. The custom VHDL design achieves a fourfold increase in maximum clock frequency while simultaneously reducing logic and DSP resource utilization. The particularly large reduction in DSP usage confirms that the arithmetic datapath has been carefully optimized, resulting in a high-performance, resource-efficient implementation suitable for scalable and high-bandwidth current control applications.

Table 6.4 Post-Implementation Timing and Resource Utilization of the VHDL Current Controllers

Metric	**PI**	**Deadbeat**
Clock constraint (MHz)	100	100
WNS (ns)	+1.983	+2.292
LUT	3462/ 53200 (6.51%)	3297/ 53200 (6.20%)
LUTRAM	136/ 17400 (0.78%)	136 / 17400 (0.78%)
FF	3854 / 106400 (3.62%)	3748 / 106400 (3.52%)
BRAM	0 / 140 (0%)	0 / 140 (0%)
DSP	18/ 220 (8.18%)	18 / 220 (8.18%)
IO	18 / 200 (9.00%)	18 / 200 (9.00%)
BUFG	2 / 32 (6.25%)	2 / 32 (6.25%)
MMCM	1 / 4 (25.00%)	1 / 4 (25%)

6.3.2 Performance Evaluation of the VHDL PI Current Controller

The PI current controllers implemented on both axis currents were validated on hardware, while key internal variables were streamed to the host PC through UART (COM4, 115200 baud) for monitoring. The UART link is used as a lightweight visualization interface. Therefore, the plots are intended to confirm correct closed-loop behavior and signal consistency rather than to provide a high-bandwidth measurement of switching ripple

Figure 6.32 shows the i_{sq} response under a sequence of stepped references with alternating sign. The alternating polarity transitions impose torque reversals, allowing the controller's response speed and recovery to be evaluated under consistent, worst-case step conditions. The measured i_{sq} follows the commanded levels with consistent polarity and repeatable behavior across the full capture, indicating stable regulation of the torque-producing current component. A visible ripple band is present around each plateau, which is expected in practical operation and is further discussed using a zoomed view.

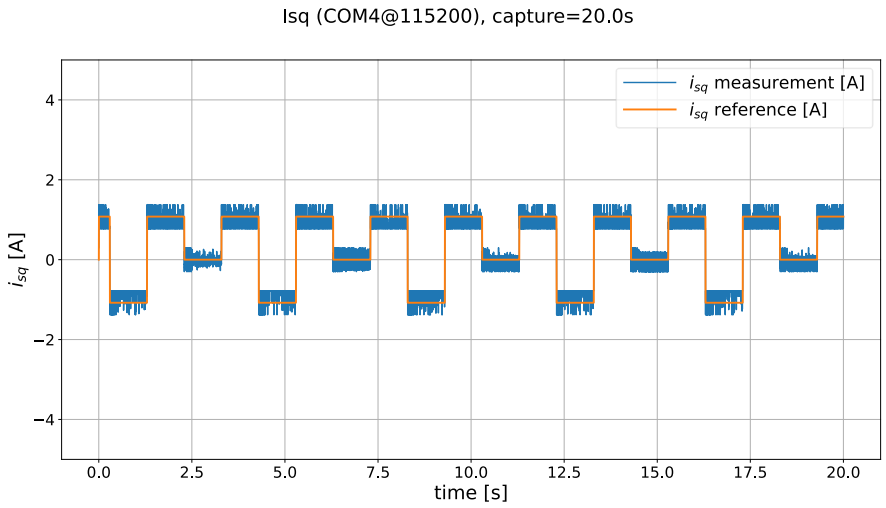


Figure 6.32 Quadrature axis current regulation with VHDL PI controller

To highlight the short-time dynamics, Figure 6.33 provides a zoom around a representative i_{sq} reference transition. This excitation highlights the loop's transient response and settling tendency under torque reversals, although the limited effective sampling rate of the UART stream prevents a high-confidence characterization of fine-scale dynamics and ripple.

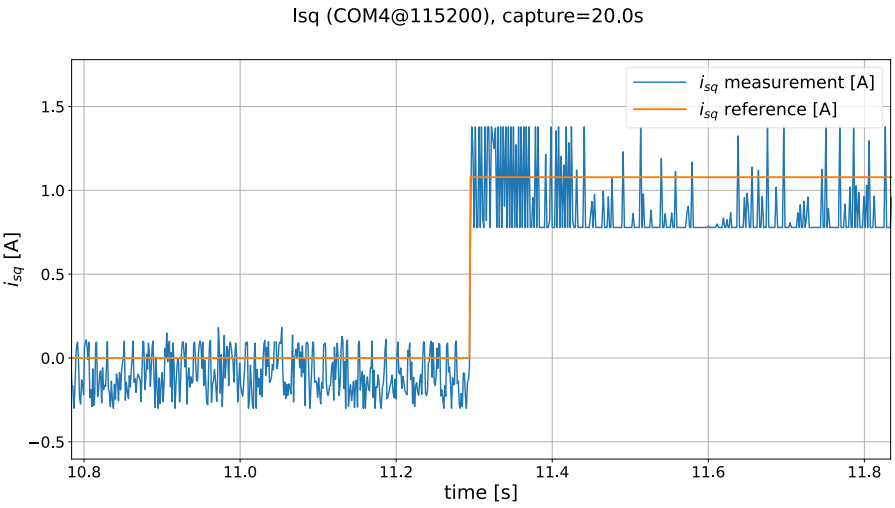


Figure 6.33 Quadrature axis current step response with VHDL PI controller

Figure 6.34 reports the i_{sd} component during the same test, with a reference set to zero. The measured i_{sd} remains centered close to zero for the entire duration, confirming that the controller maintains the flux-producing current component properly regulated and that the dq decoupling is effective under the applied i_{sq} steps.

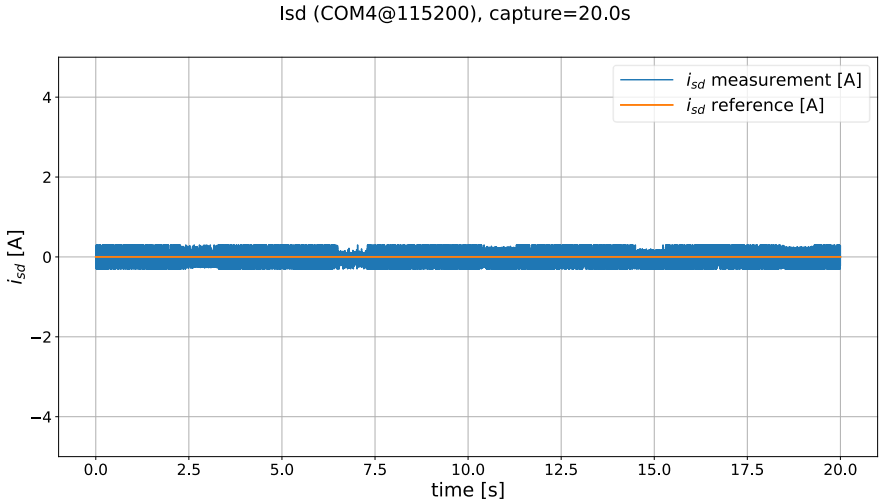


Figure 6.34 Direct axis current regulation with VHDL PI controller

Figure 6.35 shows the measured speed during the same sequence. The speed profile changes consistently with the sign and magnitude of i_{sq}, confirming that the mechanical response correlates with the applied torque-producing current. This agreement supports the correctness of the current-to-torque direction and the overall closed-loop operation during the experiment

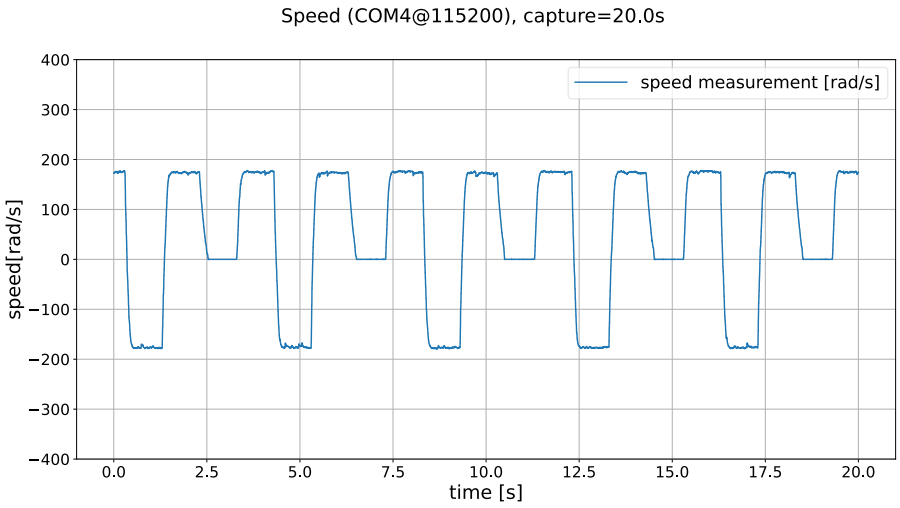


Figure 6.35 Measured rotor speed response under i_{sq} torque steps

6.3.3 Performance Evaluation of the VHDL Deadbeat Current Controller

Figure 6.36 shows the i_{sq} response under a sequence of stepped references with alternating sign. The overall tracking behavior is comparable to the PI-UART case in the sense that the measured current follows the commanded plateaus with the correct polarity and repeatable response across the full capture. A qualitative difference that is clearly visible is the ripple band around the zero-current level, which appears more pronounced and better defined in this deadbeat implementation.

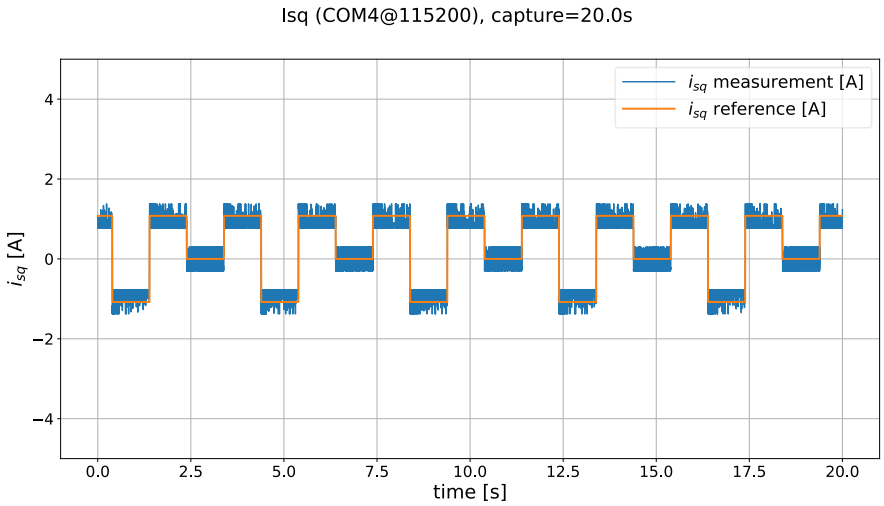


Figure 6.36 Quadrature axis current regulation with VHDL deadbeat controller

Figure 6.37 provides a zoomed view of a representative i_{sq} transition. In this view, the most evident difference relative to the PI-UART result is the more clearly defined ripple/variation when the reference is near 0 A, which makes the zero-reference regulation behavior easier to distinguish. Apart from this, the measured response appears broadly similar to the PI case within the time resolution and bandwidth

limitations of the UART monitoring stream, and therefore no definitive conclusions on fine transient metrics can be drawn from the zoomed trace alone.

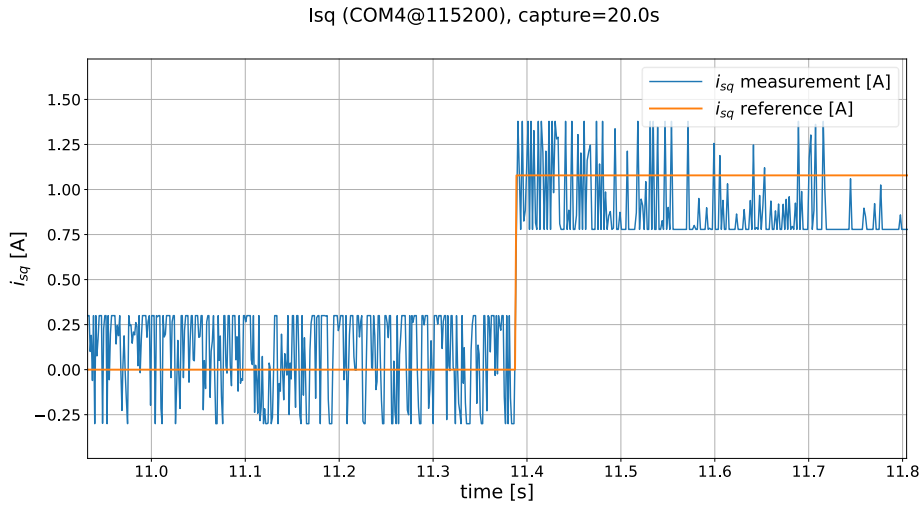


Figure 6.37 Quadrature axis current step response with VHDL deadbeat controller

Figure 6.38 shows the i_{sd} response with the reference maintained at 0 A. The measured i_{sd} remains centered around zero throughout the capture, indicating that the d-axis current is properly regulated and that the controller preserves the intended flux component while i_{sq} is stepped. Overall, the tracking is stable and well-behaved.

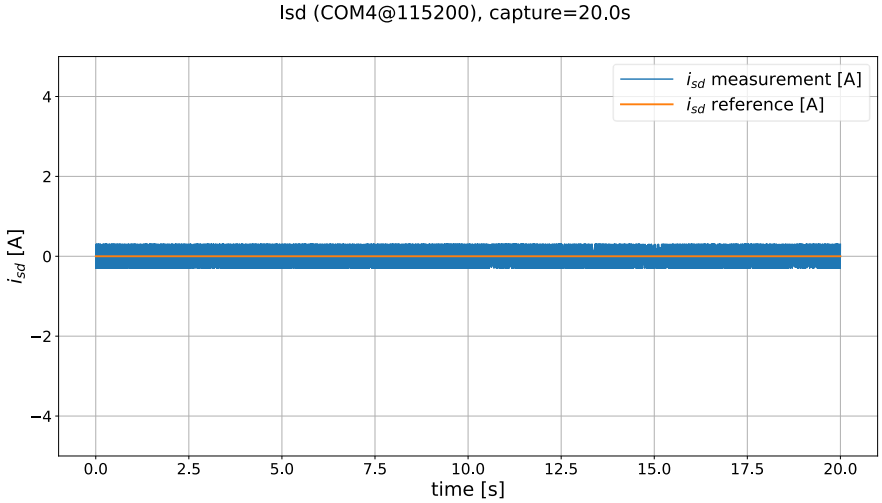


Figure 6.38 Direct axis current regulation with VHDL deadbeat controller

Figure 6.39 reports the measured motor speed during the same alternating-sign i_{sq}excitation. The speed reverses consistently with the sign of the torque-producing current and reaches repeatable plateaus for each commanded level, confirming that the mechanical response correlates with the applied i_{sq} steps. Therefore, the waveform is mainly used to validate correct torque direction and overall stability.

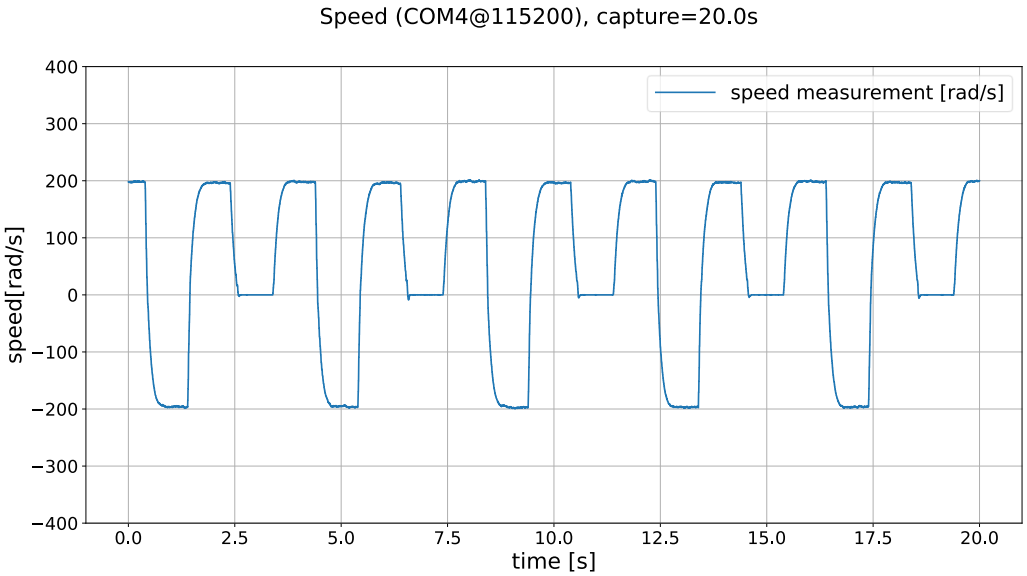


Figure 6.39 Measured rotor speed response under i_{sq} torque steps

6.3.4 Effects of VHDL Implementation

This subsection highlights the practical effects observed after transitioning to the custom VHDL implementation, focusing on the measured current ripple behavior in the UART domain. In both PI and deadbeat cases, the controller maintains the expected closed-loop operation. i_{sq} tracks the alternating-sign stepped reference while i_{sd} remains regulated around its 0 A command, and the measured speed response remains consistent with the applied torque-producing current.

A noticeable difference between the two control laws appears around the $i_{sq} = 0$ operating point, where the ripple band becomes more pronounced in the deadbeat implementation. From the zoomed UART traces, the peak-to-peak variation increases from approximately 0.52A for the PI controller to 0.60A for the deadbeat controller. While the UART stream does not provide sufficient bandwidth for a detailed spectral characterization, the same qualitative trend was also observed in the floating-point simulations and the FIL results, supporting the conclusion that this difference is repeatable and inherent to the control response near zero current.

Finally, it should be noted that transient metrics such as rise time and settling time cannot be stated conclusively from the UART captures, since the effective streaming rate and quantization limit the time resolution of the recorded traces.

6.4 Discussion

This chapter was structured to separate controller level effects from implementation-level effects, progressing from floating-point simulations to FIL and finally to experimental validation on the custom VHDL design. In the stages where the cascaded structure is available (floating-point simulation and FIL), the speed response is primarily governed by the outer loop tuning and the mechanical dynamics, while the choice of inner-loop current law (PI vs. deadbeat) has a secondary impact as long as stable and sufficiently fast current regulation is maintained.

Under floating point conditions, the results follow the theoretical expectations. Deadbeat achieves a tighter transient current response, reflected by the step-response metrics where the rise time decreases from 2.258 ms (PI) to 1.306 ms (deadbeat). At the same time, the deadbeat strategy exhibits a modest increase in ripple visibility, consistent with its more direct, model-based voltage computation. This trade-off is already measurable in the steady-state ripple metrics, where the peak-to-peak ripple for quadrature axis current increases from 0.1376 A (PI) to 0.1494 A (deadbeat) in the evaluated window.

When transitioning to FIL execution, the same qualitative trade-off remains, but the differences become more pronounced due to fixed-point arithmetic, limited numerical resolution, and deterministic multi-cycle pipeline delay. In particular, the quadrature axis current ripple for PI controller increases moderately from 0.1376 A in floating point to 0.1533 A in FIL, whereas the quadrature axis current ripple for deadbeat controller increases substantially from 0.1494 A to 0.2276 A. This widening gap is consistent with the inherent sensitivity of deadbeat control. Any mismatch introduced by quantization, coefficient approximation, measurement noise, and delay directly perturbs the one-step prediction used to compute the voltage reference, making switching-related ripple more visible in the regulated current. However, this increase in ripple does not eliminate the transient-response advantage of the deadbeat approach in the FIL environment. The step-response metrics remain consistent with the floating-point case, with the rise time decreasing from 2.153 ms for the FIL PI controller to 1.288 ms for the FIL deadbeat controller, confirming faster current-loop reaction despite the higher ripple envelope.

Finally, the experimental VHDL validation confirms correct real-time operation and preserves the qualitative trends observed in simulation and FIL. However, the UART stream is primarily intended for monitoring and does not provide sufficient bandwidth to draw definitive conclusions on fine transient metrics or the high-frequency ripple spectrum. Within this limitation, a repeatable difference is still observable around $i_{sq}^{*} = 0$: the peak-to-peak ripple band increases from approximately 0.52 A for PI to 0.60 A for deadbeat, which is consistent with the same direction of change previously identified in floating-point and FIL results.

Overall, Chapter 6 supports a clear interpretation aligned with theory: deadbeat improves current transient shaping under ideal conditions, but its advantage is progressively reduced in practical hardware by quantization, coefficient approximation, saturation, and delay, which primarily appear as increased ripple and sensitivity near critical operating regions. In contrast, the PI controller exhibits more robust behavior under the same implementation constraints, at the cost of slower (bandwidth-limited) transient dynamics.

7 Conclusion and future developments

7.1 Conclusion

This thesis began with a rapid prototyping stage based on FPGA-in-the-Loop (FIL) validation of controllers generated from MATLAB. In this initial phase, both current and speed control structures were functionally verified within the FPGA environment. While the automatically generated VHDL allowed fast deployment and algorithmic validation, the resulting hardware implementation was not optimized for high-performance operation. The synthesized design exhibited high resource utilization and limited timing performance, with a maximum achievable clock frequency of approximately 25 MHz. This constraint directly limited the attainable sampling rate and reduced the practical bandwidth of the control system.

Despite these hardware limitations, the FIL implementation successfully demonstrated closed-loop operation of both the inner current control loop and the cascaded speed control loop. Stable regulation of the dq currents was achieved, and the speed controller was able to generate appropriate torque-producing current references, confirming the correctness of the discrete-time formulations and fixed-point scaling strategy. The FIL stage therefore served as an important functional validation step, ensuring that the control algorithms behaved as expected before transitioning to a fully optimized hardware design.

To overcome the performance limitations of the automatically generated design, the control architecture was subsequently redesigned using custom VHDL implementations. Both the discrete-time PI controller and the deadbeat current controller were rewritten with explicit fixed-point scaling, pipelined arithmetic structures, and resource-aware scheduling. Special attention was given to deterministic data flow, register balancing, and latency characterization throughout the control pipeline. This hardware-oriented redesign significantly reduced logic and DSP resource utilization while improving timing closure margins.

As a result, the optimized architecture achieved stable operation at 100 MHz, representing a substantial improvement over the initial 25 MHz limitation of the FIL-generated implementation. The increased clock frequency enables shorter control cycle execution and greater flexibility in selecting sampling and switching frequencies. Furthermore, the reduced resource footprint enhances scalability, allowing integration of additional modules such as advanced modulation strategies, communication interfaces, and higher-level control loops.

Experimental validation of the optimized architecture confirmed correct real-time operation of both current control strategies. The PI controller demonstrated robust steady-state tracking and stable dynamic behavior, while the deadbeat controller achieved faster transient response and reduced current tracking error, consistent with its predictive nature. The results also emphasized the importance of explicitly accounting for pipeline latency in predictive control implementation, as hardware delay directly influences one-step prediction accuracy.

In summary, the work demonstrates that while FPGA-in-the-Loop and automatic code generation provide an effective platform for rapid functional validation of motor control algorithms, high-performance and high-frequency operation require dedicated hardware-aware VHDL design. The transition from FIL-generated controllers to custom optimized implementations resulted in significant improvements in timing performance, resource efficiency, and architectural determinism. The final system establishes a scalable and high-frequency FPGA-based motor control framework suitable for advanced electric drive applications.

7.2 Future Developments

Although the optimized FPGA architecture achieves high-frequency deterministic current control, several developments are required to complete the system as a fully mature electric drive platform.

A primary extension concerns the full experimental validation of the cascaded speed control loop within the optimized 100 MHz architecture. While speed regulation was successfully demonstrated during the FPGA-in-the-Loop stage, including testing under varying load profiles in a controlled environment, equivalent real-time validation on the fully optimized hardware remains necessary. Future work should therefore focus on systematic speed-loop tuning, bandwidth separation verification, and experimental testing under realistic load conditions comparable to those applied during FIL validation. This includes step-load disturbances, dynamic torque variations, and steady-state operation across the torque–speed plane. Such testing would finalize the drive-level hierarchy and provide comprehensive performance characterization under practical operating scenarios.

A second important development is the replacement of the sinusoidal PWM strategy with a Space Vector Modulation scheme. Implementing SVM directly in FPGA fabric would improve DC-link voltage utilization and better reflect industrial practice in high-performance motor drives. Furthermore, the interaction between modulation delay and predictive current control could be examined more rigorously under an SVM-based switching framework.

Communication infrastructure also represents a critical development area. The integration of a robust Ethernet-based interface would enable high-bandwidth data

acquisition, structured logging of internal variables, and real-time parameter adjustment. Compared to low-bandwidth serial communication methods, Ethernet would significantly enhance system observability and support more detailed experimental analysis during controller tuning and validation.

Finally, measured pipeline latency indicates that the computational delay is considerably shorter than the current sampling interval. This suggests that higher switching frequencies and increased sampling rates may be achievable without structural modification of the control datapath. Operating at reduced sampling periods would allow further bandwidth expansion and deeper investigation of predictive controller behavior under faster update rates, fully exploiting the deterministic execution capability of the FPGA architecture.

Together, these developments represent the natural continuation of the presented work toward a complete, experimentally validated, and industrially aligned FPGA-based motor drive system.

Bibliography

[1] N. Hoffmann, F. W. Fuchs, M. P. Kazmierkowski and D. Schröder, "Digital Current Control in a Rotating Reference Frame—Part I: System Modeling and the Discrete Time-Domain Current Controller With Improved Decoupling Capabilities," *IEEE Transactions on Power Electronics,* vol. 31, no. 7, p. 5290–5303, 2016.

[2] K. Ito, R. Suzuki, K. Yoshimoto and T. Yokoyama, "A Study of Multisampling Deadbeat Control for Low Carrier Frequency PMSM Drive System Used in EVs and HEVs," in *2021 IEEE International Conference on Mechatronics (ICM)*, 2021.

[3] Z. Chen, M. Yan, H. Fang, X. Zhang, D. Wu and G. Luo, "Deadbeat Current Predictive Control for PMSM Drives Based on a Single FPGA with Digital Implementation Optimization," *2021 IEEE International Conference on Predictive Control of Electrical Drives and Power Electronics (PRECEDE),* 2021.

[4] L. Rovere, A. Formentini and P. Zanchetta, "FPGA Implementation of a Novel Oversampling Deadbeat Controller for PMSM Drives," *IEEE Transactions on Industrial Electronics,* p. 3731–3741, 2019.

[5] W. Tu, G. Luo, Z. Chen, C. Liu and L. Cui, "FPGA Implementation of Predictive Cascaded Speed and Current Control of PMSM Drives With Two-Time-Scale Optimization," *IEEE Transactions on Industrial Informatics,* vol. 15, no. 9, p. 5276–5288, 2019.

[6] J. Shi, B. Wang, P. Du and D. Xu, "FPGA Implementation of High-Bandwidth Vector Control for PMSM Based on Parallel Processing Technology," in *26th International Conference on Electrical Machines and Systems (ICEMS)*, 2023.

[7] MathWorks, "PMSM Field-Oriented Control," [Online]. Available: https://www.mathworks.com/help/sps/ref/pmsmfieldorientedcontrol.html. [Accessed 22 February 2026].

[8] T.-T. Nguyen, "The Multilevel Inverter with Clamped Dİode," *Indonesian Journal of Electrical Engineering and Computer Science,* vol. 14, pp. 1189-1195, 2019.

[9] MathWorks, "PWM Generator," [Online]. Available: https://www.mathworks.com/help/sps/ref/pwmgeneratorthreephasethreelevel.html. [Accessed 22 February 2026].

[10] AMD, "CORDIC v6.0 LogiCORE IP Product Guide (DS249)," AMD, [Online]. Available: https://docs.amd.com/v/u/en-US/cordic_ds249. [Accessed 22 February 2026].

A Appendix A

A.1. VHDL Code for Q-axis PI Current Controller

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;


entity isq_PI_module is
Generic (
        TDATA_WIDTH : positive := 16;
        KP_F20_14 : integer := 55670; --in fix20_14 format
        KI_FP : integer := 458; --in fix16_14 format KI*Ts
        I_LIM_F16_14 : integer := 6769;
        U_LIM_F16_14 : integer := 13538;
        TS : positive := 10
    );
Port (
        aclk : in  std_logic;
        aresetn : in  std_logic;
        start_PI : in std_logic;
        isq_measurement : in signed(TDATA_WIDTH - 1 downto 0);
        isq_reference : in signed(TDATA_WIDTH - 1 downto 0);
        vq_ref_pi_out : out signed(TDATA_WIDTH - 1 downto 0)
    );
end isq_PI_module;

architecture Behavioral of isq_PI_module is
    type state_type is (ERROR_CALC, P_STATE, I_STATE, SUM_AND_SAT_STATE);
    signal state : state_type := ERROR_CALC;

    signal old_error : signed (TDATA_WIDTH - 1 downto 0) := (others => '0');
    signal current_error : signed (TDATA_WIDTH - 1 downto 0) := (others => '0');
    signal p_state_out : signed (TDATA_WIDTH - 1 downto 0) := (others => '0');
    signal i_state_out : signed (TDATA_WIDTH - 1 downto 0) := (others => '0');
    signal new_control_command : signed (TDATA_WIDTH - 1 downto 0) := (others => '0');
    signal old_control_command : signed (TDATA_WIDTH - 1 downto 0) := (others => '0');
begin
    process(aclk)
        variable var_p_out : signed(2 * TDATA_WIDTH + 3 downto 0) := (others => '0'); -
-36 bits just to be safe
        variable var_i_out : signed(2 * TDATA_WIDTH - 1 downto 0) := (others => '0');
        variable var_control_command : signed(2 * TDATA_WIDTH - 1 downto 0) := (others
=> '0');
    begin
        if rising_edge(aclk) then
            if aresetn = '0' then
                old_error <= (others => '0');
                current_error <= (others => '0');

                p_state_out <= (others => '0');
                i_state_out <= (others => '0');

                old_control_command <= (others => '0');
                new_control_command <= (others => '0');

                state <= ERROR_CALC;
            else
                case state is
```

```
                        when ERROR_CALC =>
                            if start_PI = '1' then
                                current_error <= isq_reference - isq_measurement;
                                state <= P_STATE;
                            end if;
                        when P_STATE =>
                            var_p_out := to_signed(KP_F20_14, 20) * (current_error -
    old_error);
                            p_state_out <= resize(shift_right(var_p_out, 14),
    p_state_out'length);
                            state <= I_STATE;

                        when I_STATE =>
                            var_i_out := resize(to_signed(KI_FP, current_error'length) *
    current_error, var_i_out'length);
                            i_state_out <= resize(shift_right(var_i_out, 14),
    i_state_out'length);
                            state <= SUM_AND_SAT_STATE;

                        when SUM_AND_SAT_STATE =>
                            var_control_command := resize(p_state_out,
    var_control_command'length) + resize(i_state_out, var_control_command'length) +
    resize(old_control_command, var_control_command'length);

                            if var_control_command > to_signed(U_LIM_F16_14,
    var_control_command'length) then
                                var_control_command := to_signed(U_LIM_F16_14,
    var_control_command'length);
                            elsif var_control_command < to_signed(-U_LIM_F16_14,
    var_control_command'length) then
                                var_control_command := to_signed(-U_LIM_F16_14,
    var_control_command'length);
                            end if;

                            old_error <= current_error;
                            old_control_command <= resize(var_control_command,
    TDATA_WIDTH);
                            new_control_command <= resize(var_control_command,
    TDATA_WIDTH);
                            state <= ERROR_CALC;
                    end case;
                end if;
            end if;
        end process;

        vq_ref_pi_out <= new_control_command;
    end Behavioral;
```

A.2. VHDL Code for D-axis PI Current Controller

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;


entity isd_PI_module is
Generic (
        TDATA_WIDTH : positive := 16;
        KP_F20_14 : integer := 55670; --in fix20_14 format
        KI_FP : integer := 458; --in fix16_14 format KI*Ts
        I_LIM_F16_14 : integer := 6769;
        U_LIM_F16_14 : integer := 13538;
        TS : positive := 10
    );
Port (
        aclk : in  std_logic;
        aresetn : in  std_logic;
        start_PI : in std_logic;
        isd_measurement : in signed(TDATA_WIDTH - 1 downto 0);
        vd_ref_pi_out : out signed(TDATA_WIDTH - 1 downto 0)
    );
end isd_PI_module;

architecture Behavioral of isd_PI_module is
    type state_type is (ERROR_CALC, P_STATE, I_STATE, SUM_AND_SAT_STATE);
    signal state : state_type := ERROR_CALC;

    constant isd_reference : signed(TDATA_WIDTH - 1 downto 0) := (others => '0');
    --constant isd_reference : signed(TDATA_WIDTH - 1 downto 0) := to_signed(1500,16);

    signal old_error : signed (TDATA_WIDTH - 1 downto 0) := (others => '0');
    signal current_error : signed (TDATA_WIDTH - 1 downto 0) := (others => '0');
    signal p_state_out : signed (TDATA_WIDTH - 1 downto 0) := (others => '0');
    signal i_state_out : signed (TDATA_WIDTH - 1 downto 0) := (others => '0');
    signal new_control_command : signed (TDATA_WIDTH - 1 downto 0) := (others => '0');
    signal old_control_command : signed (TDATA_WIDTH - 1 downto 0) := (others => '0');
begin
    process(aclk)
        variable var_p_out : signed(2 * TDATA_WIDTH + 3 downto 0) := (others => '0'); -
-36 bits just to be safe
        variable var_i_out : signed(2 * TDATA_WIDTH - 1 downto 0) := (others => '0');
        variable var_control_command : signed(2 * TDATA_WIDTH - 1 downto 0) := (others
=> '0');
    begin
        if rising_edge(aclk) then
            if aresetn = '0' then
                old_error <= (others => '0');
                current_error <= (others => '0');

                p_state_out <= (others => '0');
                i_state_out <= (others => '0');

                old_control_command <= (others => '0');
                new_control_command <= (others => '0');

                state <= ERROR_CALC;
            else
                case state is
                    when ERROR_CALC =>
                        if start_PI = '1' then
                            current_error <= isd_reference - isd_measurement;
                            state <= P_STATE;
                        end if;

                    when P_STATE =>
                        var_p_out := to_signed(KP_F20_14, 20) * (current_error -
old_error);
                        p_state_out <= resize(shift_right(var_p_out, 14),
p_state_out'length);
```

```
                        state <= I_STATE;

                    when I_STATE =>
                        var_i_out := resize(to_signed(KI_FP, current_error'length) *
current_error, var_i_out'length);
                        i_state_out <= resize(shift_right(var_i_out, 14),
i_state_out'length);
                        state <= SUM_AND_SAT_STATE;

                    when SUM_AND_SAT_STATE =>
                        var_control_command := resize(p_state_out,
var_control_command'length) + resize(i_state_out, var_control_command'length) +
resize(old_control_command, var_control_command'length);

                        if var_control_command > to_signed(U_LIM_F16_14,
var_control_command'length) then
                            var_control_command := to_signed(U_LIM_F16_14,
var_control_command'length);
                        elsif var_control_command < to_signed(-U_LIM_F16_14,
var_control_command'length) then
                            var_control_command := to_signed(-U_LIM_F16_14,
var_control_command'length);
                        end if;

                        old_error <= current_error;
                        old_control_command <= resize(var_control_command,
TDATA_WIDTH);
                        new_control_command <= resize(var_control_command,
TDATA_WIDTH);
                        state <= ERROR_CALC;
                end case;
            end if;
        end if;
    end process;

    vd_ref_pi_out <= new_control_command;
end Behavioral
```

A.3. VHDL Code for Q-axis Deadbeat Current Controller

```
library IEEE;
    use IEEE.STD_LOGIC_1164.ALL;
    use IEEE.NUMERIC_STD.ALL;


    entity isq_deadbeat_module is
    Generic (
            TDATA_WIDTH : positive := 16;
            I_LIM_F16_14 : integer := 6769;
            U_LIM_F16_14 : integer := 13538;
            TS : positive := 10
        );
    Port (
            aclk : in  std_logic;
            aresetn : in  std_logic;
            start deadbeat : in std logic;
            isq_measurement : in signed(TDATA_WIDTH - 1 downto 0);
            isq_reference : in signed(TDATA_WIDTH - 1 downto 0);
            vq_ref_pi_out : out signed(TDATA_WIDTH - 1 downto 0)
        );
    end isq_deadbeat_module;

    architecture Behavioral of isq_deadbeat_module is
        type state_type is (IDLE, DEADBEAT, SAT_STATE);
        signal state : state_type := IDLE;

        signal new_control_command : signed (TDATA_WIDTH - 1 downto 0) := (others =>
'0');

    begin
        process(aclk)
            variable var_control_command : signed(2 * TDATA_WIDTH - 1 downto 0) :=
(others => '0');
        begin
            if rising_edge(aclk) then
                if aresetn = '0' then
                    new_control_command <= (others => '0');
                    state <= IDLE;
                else
                    case state is
                        when IDLE =>
                            if start_deadbeat = '1' then
                                state <= DEADBEAT;
                            end if;

                        when DEADBEAT =>
                            var_control_command := shift_right(to_signed(965, 16) *
isq_reference, 3) - to_signed(120, 16) * isq_measurement;
                            state <= SAT_STATE;

                        when SAT_STATE =>
                            if var_control_command > to_signed(U_LIM_F16_14,
var_control_command'length) then
                                new_control_command <= to_signed(U_LIM_F16_14,
new_control_command'length);
                            elsif var_control_command < to_signed(-U_LIM_F16_14,
var_control_command'length) then
                                new_control_command <= to_signed(-U_LIM_F16_14,
new_control_command'length);
                            end if;
                            state <= IDLE;
                    end case;
                end if;
            end if;
        end process;

        vq_ref_pi_out <= new_control_command;
    end Behavioral;
```

A.4. VHDL Code for D-axis Deadbeat Current Controller

```
        library IEEE;
        use IEEE.STD_LOGIC_1164.ALL;
        use IEEE.NUMERIC_STD.ALL;


        entity isd_deadbeat_module is
        Generic (
                TDATA_WIDTH : positive := 16;
                U_LIM_F16_14 : integer := 13538;
                TS : positive := 10
            );
        Port (
                aclk : in  std_logic;
                aresetn : in  std_logic;
                start_deadbeat : in std_logic;
                isd measurement : in signed(TDATA WIDTH - 1 downto 0);
                vd_ref_pi_out : out signed(TDATA_WIDTH - 1 downto 0)
            );
        end isd_deadbeat_module;

        architecture Behavioral of isd_deadbeat_module is
            type state_type is (IDLE, DEADBEAT, SAT_STATE);
            signal state : state_type := IDLE;

            signal new_control_command : signed (TDATA_WIDTH - 1 downto 0) := (others =>
    '0');

        begin
            process(aclk)
                variable var_control_command : signed(2 * TDATA_WIDTH - 1 downto 0) :=
    (others => '0');
            begin
                if rising_edge(aclk) then
                    if aresetn = '0' then
                        new_control_command <= (others => '0');
                        state <= IDLE;
                    else

                        case state is
                            when IDLE =>
                                if start_deadbeat = '1' then
                                    state <= DEADBEAT;
                                end if;

                            when DEADBEAT =>
                                var_control_command :=  - to_signed(120, 16) *
    isq_measurement;
                                state <= SAT_STATE;

                            when SAT_STATE =>
                                if var_control_command > to_signed(U_LIM_F16_14,
    var_control_command'length) then
                                    new_control_command <= to_signed(U_LIM_F16_14,
    new_control_command'length);
                                elsif var_control_command < to_signed(-U_LIM_F16_14,
    var_control_command'length) then
                                    new_control_command <= to_signed(-U_LIM_F16_14,
    new_control_command'length);
                                end if;
                                                                    state <= IDLE;
                      end case;
                    end if;
                end if;
            end process;

            vd_ref_pi_out <= new_control_command;
        end Behavioral;
```

A.5. VHDL Code for PWM Generation for Phase A

```
    process (aclk)
        begin -- PHASE A PWM GENERATION
            if rising_edge (aclk) then
                if aresetn = '0' then
                        state_pwm_1         <= IDLE_A;
                        pwm_1_reg           <= '0';
                        pwm_1_old           <= '0';
                        pwm_1_upper_reg     <= '0';
                        pwm_1_lower_reg     <= '0';
                        deadtime_counter_1  <= 0;
                else
                    case state_pwm_1 is
                        when IDLE_A =>
                            if enable_pulse = '1' then
                                state_pwm_1 <= COMPARE_A;
                            end if;
                        when COMPARE_A =>
                            if phase_A_ref > triangular_counter then
                                pwm_1_reg <= '1';
                                if pwm_1_old = '1' then
                                    state_pwm_1 <= UPPER_ON_A;
                                else
                                    state_pwm_1 <= TRANS_UPPER_A;
                                end if;
                             else
                                pwm_1_reg <= '0';
                                if pwm_1_old = '0' then
                                    state_pwm_1 <= LOWER_ON_A;
                                else
                                    state_pwm_1 <= TRANS_LOWER_A;
                                end if;
                            end if;
                        when UPPER_ON_A =>
                            pwm_1_upper_reg <= '1';
                            pwm_1_lower_reg <= '0';
                            pwm_1_old   <= pwm_1_reg;
                            state_pwm_1 <= IDLE_A;
                        when LOWER_ON_A =>
                            pwm_1_upper_reg <= '0';
                            pwm_1_lower_reg <= '1';
                            pwm_1_old   <= pwm_1_reg;
                            state_pwm_1 <= IDLE_A;
                        when TRANS_UPPER_A =>
                            pwm 1 upper reg <= '0';
                            pwm 1 lower reg <= '0';
                            if deadtime_counter_1 = DEADTIME_CYCLES then
                                deadtime_counter_1 <= 0;
                                state_pwm_1 <= UPPER_ON_A;
                            else
                                deadtime_counter_1 <= deadtime_counter_1 + 1;
                            end if;
                        when TRANS_LOWER_A =>
                            pwm_1_upper_reg <= '0';
                            pwm_1_lower_reg <= '0';
                            if deadtime_counter_1 = DEADTIME_CYCLES then
                                deadtime_counter_1 <= 0;
                                state_pwm_1 <= LOWER_ON_A;
                            else
                                deadtime_counter_1 <= deadtime_counter_1 + 1;
                            end if;
                    end case;
                end if;
            end if;
        end process;
```

A.6. VHDL Code for Clark Transformation

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;


entity ClarkeTransformation is
    Port ( clk : in STD_LOGIC;
           rstn : in STD_LOGIC;
           i_a : in signed (15 downto 0);
           i_b : in signed (15 downto 0);
           i_c : in signed (15 downto 0);
           i_alpha : out signed (15 downto 0);
           i_beta : out signed (15 downto 0));
end ClarkeTransformation;

architecture Behavioral of ClarkeTransformation is

begin
    process (clk)
       variable temp_val : signed(31 downto 0);
    begin
        if rising_edge (clk) then
            if rstn = '0' then
                i_alpha <= (others => '0');
                i_beta  <= (others => '0');
            else
                i_alpha <= i_a;
                temp val := (i b - i c) * to signed(74, 16);
                i_beta <= resize(shift_right(temp_val , 7), 16);
            end if;
        end if;

    end process;
end Behavioral;
```

A.7. VHDL Code for Inverse Clark Transformation

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;


entity InvClarkeTransform is
Port ( clk : in STD_LOGIC;
           rstn : in STD_LOGIC;
           v_alpha : in signed (15 downto 0);
           v_beta : in signed (15 downto 0);
           v_a : out signed (15 downto 0);
           v_b : out signed (15 downto 0);
           v_c : out signed (15 downto 0)
           );
end InvClarkeTransform;

architecture Behavioral of InvClarkeTransform is

begin
   process (clk)
   variable temp_val : signed(31 downto 0);
    begin
        if rising_edge (clk) then
            if rstn = '0' then
                v_a <= (others => '0');
                v_b  <= (others => '0');
                v_c  <= (others => '0');
            else
                temp_val := shift_right(v_beta * to_signed(110, 16), 7);
                v_a <= v_alpha;
                v_b <= -shift_right(v_alpha, 1) + resize(temp_val, 16);
                v_c <= -shift_right(v_alpha, 1) - resize(temp_val, 16);
            end if;
        end if;
    end process;

end Behavioral;
```

List of Figures

List of Tables

List of Symbols

Variable	Description	SI Unit
v_a, v_b, v_c	Stator phase voltages	V
i_a, i_b, i_c	Stator phase currents	A
Φ_a, Φ_b, Φ_c	Stator phase flux linkages	Wb
v_s	Stator voltage space vector	V
i_s	Stator current space vector	A
ϕ_s	Stator flux-linkage space vector	Wb
v_α, v_β	Stator voltages in $\alpha\beta$ frame	V
i_α, i_β	Stator currents in $\alpha\beta$ frame	A
v_{sd}, v_{sq}	Stator voltages in synchronous dq frame	V
i_{sd}, i_{sq}	Stator currents in synchronous dqframe	A
R_s	Stator resistance	Ω
L_s	Stator inductance	H
L_d, L_q	d- and q-axis inductances	H
φ_n	Permanent-magnet flux linkage	Wb
p	Number of pole pairs	–
θ_m	Mechanical rotor position (angle)	rad

Variable	Description	SI Unit
θ_e	Electrical rotor position (angle)	rad
ω_m	Mechanical rotor angular speed	rad/s
ω_e	Electrical angular speed	rad/s
T_e	Electromagnetic torque	N·m
T_L	Load torque	N·m
J	Rotor inertia	kg·m²
B	Viscous friction coefficient	N·m·s/rad
e	Control error (reference − measured)	same as signal (A, rad/s, etc.)
u	Controller output (commanded voltage in axis frame)	V
K_p	PI proportional gain	(depends on loop; typically V/A for current loop)
K_i	PI integral gain	(depends on loop; typically V/(A·s) for current loop)
T_s	Sampling period	s
f_s	Sampling frequency	Hz
V_{dc}	DC-link voltage	V
f_{sw}	PWM switching frequency	Hz

Acknowledgments

I would like to express my sincere gratitude to Professor Luigi Piegari for his supervision, guidance, and valuable feedback throughout this thesis. I also thank Sarper Öztürk for his support, technical insight, and the time he dedicated to discussions that helped shape this work. I am grateful to the members of PeDS Lab for providing a constructive research environment and for their assistance during the implementation and experimental phases.

I am deeply thankful to my family, my friends and my girlfriend for their encouragement, support, and belief in me. I would not have been able to achieve this without them.

Finally, I dedicate this achievement to the memory of my grandfathers, Mesut Orhan and Uğur Kadri Kınoğlu, who passed away before they could see my graduation.

I would like to express my sincere gratitude to Mustafa Kemal Atatürk, whose legacy enabled the opportunities that shaped my education. I remain committed to following the path he paved for Turkish youth.