Master's Degree in Electronic Engineering

Restructuring Real-Time Hardware-in-the-Loop (RT HIL) Models Using a Multicore Approach

Enhancing Component Reusability and Intellectual Property Protection

Mina Samadian

Supervisors:
Prof. Luciano Lavagno
Ing. Antonio Vitale

Politecnico di Torino
2024

Abstract

This thesis aims to restructure a real-time model for a hardware-in-the-loop (RT HIL) system incorporating an inverter, electric motor, mechanical model, and bus simulation. Hardware-in-the-loop simulation develops and tests complex, real-time embedded systems. It allows for integrating physical and simulated components to create a comprehensive testing environment, enabling developers to evaluate control systems' performance and behavior under real-world conditions without needing a complete physical prototype. This method is precious in the automotive industry for safely and efficiently testing and validating electric motor control systems and other electronic control units (ECUs).

The restructuring utilizes a multi-core approach to enhance the reusability of individual components while preserving intellectual property. The resulting model is composed of two synchronized cores: the first core is a white box, providing full access to the code for complete calibration capabilities, the addition of new parts, and comprehensive bus simulation modeling. The second core is a black box containing all intellectual property using only the build results. This setup allows future development to use only the black-box core build while enabling full access to new parts development exclusively on the white-box core.

The multi-core architecture ensures correct data propagation between the processor's cores and FPGA, optimizing performance for complex simulations. The RT HIL system includes a controller developed on Simulink that runs on the HIL's real-time processor and is used for open-loop testing. The restructured model was validated using the open-loop integrated controller. The second step involved validation with the real device under test, composed of an actual control board (without power stage) for a high-speed PMSM motor connected by wire to the HIL. The system demonstrated robustness and high accuracy in motor simulations, maintaining performance in real-time scenarios and closing the loop. This new modular architecture allows

for a proper compromise of calibration, new development, and intellectual property policy compliance.

Contents

List of Figures

Chapter 1

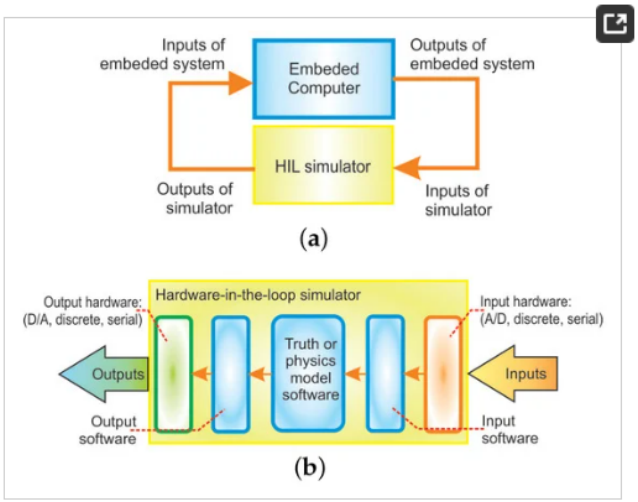
Theory and Background

1.1 HIL Theory

Theory and Technical Meaning of Hardware-in-the-Loop (HIL) Hardware-in-the-loop (HIL) testing is a simulation technique used to evaluate and validate the performance of control systems within a realistic simulated environment. HIL systems integrate a real-time environment simulation with either actual hardware components, known as the Device-Under-Test (DUT), or purely simulated models.

In HIL testing, the real-time simulation platform replicates the operational environment with which the control system or DUT normally interacts. This allows for detailed testing under conditions that closely mimic real-world scenarios. The platform provides real-time inputs to the control system or DUT and receives outputs, enabling dynamic testing and monitoring of the system's responses.

HIL systems use advanced computational tools to ensure simulations run in real time. Powerful processors and Field-Programmable Gate Arrays (FPGAs) handle complex calculations and data processing at high speeds. The simulation software models various physical phenomena and environmental conditions, creating a closed-loop system where the control system and the simulation platform continuously interact.



.

Fig. 1.1 HIL

1.1.1 Benefits and Applications of HIL Simulation

HIL simulation[1] offers numerous benefits across various industries, particularly in developing and testing complex systems. Some key advantages include:

- **Cost and Time Savings:** HIL simulation significantly reduces the need for expensive physical prototypes and real-world testing, leading to substantial cost and time savings in the development cycle.
- **Enhanced Safety:** By simulating hazardous scenarios and fault conditions in a controlled environment, HIL simulation allows for safe system testing without risking equipment or personnel damage.
- **Increased Test Coverage:** HIL simulation enables comprehensive testing of a wide range of operating conditions, including extreme cases and rare events that may be difficult to replicate in real-world testing.
- **Early Detection of Design Flaws:** By identifying and addressing design flaws early in development, HIL simulation helps prevent costly issues later on.
- **Improved Collaboration:** HIL simulation facilitates collaboration between different teams involved in system development, providing a common testing and validation platform.

The applications of HIL simulation span various industries, including:

- **Automotive:** HIL simulation is extensively used in the automotive industry for testing electronic control units (ECUs), powertrain systems, and advanced driver assistance systems (ADAS).
- **Aerospace:** HIL simulation plays a crucial role in developing and testing avionics systems, flight control systems, and other critical components in the aerospace industry.
- **Energy:** HIL simulation is employed in the energy sector for testing renewable energy systems, power electronics, and grid integration technologies.

1.1.2 DSpace HIL Systems

The SCALEXIO LabBox is a modular, high-performance, real-time processor specifically designed for HIL applications. It comes in various configurations, and in option, the 19-slot model offers a balance between scalability and functionality for diverse testing needs. Features of SCALEXIO labBox:

Fig. 1.2 LabBox

- **Real-time processor**: is based on an industrial PC with an Intel Core i7-6820EQ processor, a real-time operating system (RTOS), and a PCIe plug-in card designed by dSpace for communication between I/O and real-time processor.The Labbox DSpace connects to the host PC primarily through a networking interface, utilizing an IOCNET Router (DS6051) or similar components for flexible integration and scalability.

- **I/O boards**: Modular I/O refers to an adaptable system of input/output interfaces that can be customized and scaled according to specific project requirements. These systems allow for easy integration and configuration of various signal types to meet the diverse needs of simulation and testing environments. In the dSPACE SCALEXIO system context, the DS6101 board exemplifies this technology, offering versatile digital and analog channels to facilitate effective ECU testing and development.

- **Communication interfaces** the board used in this project. The DS6301 CAN/LIN Board by dSPACE supports various communication protocols, making it ideal for automotive projects where realistic communication between a hardware-in-the-loop (HIL) system and a device under test (DUT) is essential. The board is equipped to handle:

- CAN and CAN FD: The DS6301 provides four dedicated channels for CAN (Controller Area Network) and CAN FD (Flexible Data-rate) communications, extensively used in automotive networks for vehicle systems and components.

- LIN (Local Interconnect Network): Besides CAN, the board offers four dedicated channels for LIN, a simpler, cost-effective network protocol typically used for automotive sensors and actuaries.

- **Integration with DSpace software tools**: The LabBox integrates with DSpace software tools like ControlDesk and ConfigurationDesk, providing a user-friendly environment for configuring the HIL system, managing test cases, and analyzing results.

- **Scalability**: The 19-slot configuration allows adding additional I/O modules as needed, future-proofing your HIL system for potential expansion and adaptation to evolving test requirements.

In the Labbox, we utilize several FPGA boards, which will be described in detail in a later section.

1.2 Field-Programmable Gate Arrays (FPGAs)

A Field Programmable Gate Array (FPGA), a fundamental component in digital electronics, is a programmable logic device featuring reconfigurable logic circuits. These circuits allow for the implementation of both sequential (e.g., flip-flops or more complex designs) and combinational logic (such as AND/OR gates or more intricate configurations). FPGAs are programmed and modified using specific hardware description languages, notably Verilog or VHDL.

The core structure of an FPGA includes a grid of Configurable Logic Blocks (CLBs) linked by programmable interconnects. Located around the perimeter of this grid are the input and output blocks (I/O), which facilitate interactions with the external environment. CLBs execute the logic functions linked through a network of interconnections, while the I/O blocks handle external circuit interfacing.

Each CLB has several components, such as a lookup table (LUT), a multiplexer, and a register. These components are configurable to perform as needed. Typically, FPGAs utilize 4-input LUTs, which can be set to carry out any 4-input logic function. The LUT's output is connected directly to one of the outputs of the logic block and one input of the multiplexer. The multiplexer, which can select between inputs, sends its output to the register input. Registers within the FPGA can be configured to operate as a flip-flop or a latch, with clocks set to active high or low.

The FPGA's interconnects are flexible, enabling any output from a CLB to be linked to any input of another CLB. Additionally, the primary inputs of the FPGA can interface with any logic block's inputs, and the outputs from any logic block can drive the FPGA's primary outputs. Advanced peripheral interface functionalities, such as CAN, I2C, SPI, UART, and USB, are often integrated directly into the chip as hard cores.

Advantages of FPGAs:

- **Flexibility:** FPGAs can be reprogrammed post-deployment to adapt to new standards or update functionalities, providing significant adaptability.

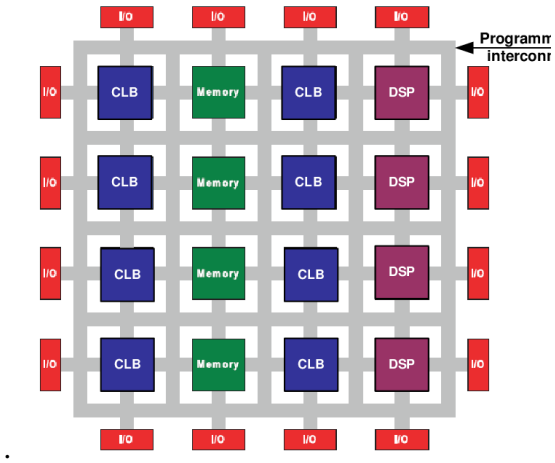


Fig. 1.3 FPGA

- **Performance:** Excelling in parallel processing, FPGAs are ideal for high-speed operations and real-time data processing.
- **Speed of Development:** The reconfigurability of FPGAs allows for rapid prototyping and faster iterations during design, reducing development time.
- **Cost-Effectiveness for Low to Medium Volume:** FPGAs may offer a more economical approach compared to custom ASICs for small to medium production volumes due to lower initial costs.
- **Long-term Maintenance:** FPGAs can be updated or reconfigured without physical replacements, extending the lifespan of the systems they are used in.
- **Specialized Integration:** Advanced peripheral functionalities such as CAN, I2C, SPI, UART, and USB can be integrated directly into the chip as hard cores, simplifying design.

The Labbox framework includes advanced FPGA boards designed to enhance simulation capabilities and interface management. Two keyboards, the DS6601, and the DS6651, serve distinct yet complementary purposes in the project setup.

- DS6601 FPGA Board The DS6601 FPGA Board is a cornerstone of our hardware setup, offering robust computational power that is essential for handling complex algorithms and simulation tasks. Its FPGA is particularly well-suited for protocols, third-party interfaces, and processor-based electric drive simulations. This board plays a pivotal role in our rapid control prototyping (RCP)

endeavors, facilitating the development of electric drive and power electronic controllers. The versatility and high performance of the DS6601 make it invaluable for advancing our project's objectives in real-time simulation and controller design.

Fig. 1.4 DS6601

- DS6651 Multi-I/O Module Complementing the DS6601, the DS6651 Multi-I/O Module extends the system's interfacing capabilities with various sensors and actuators. This module provides flexible interfaces for different position sensors, supports voltage measurement and generation, and includes multi-purpose digital I/O channels. Its diverse set of channels is ideally suited for highly dynamic control applications and the high-fidelity simulation of electric drives and power electronics components. The DS6651's compatibility with all SCALEXIO FPGA boards ensures seamless integration into our existing system architecture, enhancing the project's scalability and responsiveness to complex simulation requirements.

1.2.1 Xilinx System Generator

Xilinx System Generator is a powerful design tool that simplifies the development of digital signal processing (DSP) algorithms on FPGAs. It provides a high-level graphical interface within the MATLAB/Simulink environment[2], allowing engi-

Fig. 1.5 DS6651

neers to design and simulate DSP systems without requiring extensive knowledge of hardware description languages (HDLs).

With Xilinx System Generator, engineers can seamlessly integrate their Simulink models with FPGA hardware. The tool automatically generates HDL code from the Simulink model, which can then be synthesized and implemented on the FPGA. This streamlined workflow significantly reduces development time and effort, enabling faster prototyping and testing of HIL simulations.

1.2.2 Real-Time Interface (RTI) Programming

Real-Time Interface (RTI) programming is essential for establishing communication and data exchange between the FPGA and the HIL simulation software. It involves defining the interface protocols, data structures, and timing requirements to ensure seamless interaction between the two domains.

RTI programming typically involves developing software drivers and libraries that handle the communication between the FPGA and the simulation software. These drivers manage the transfer of data, synchronization of events, and control of the FPGA's operation. Effective RTI programming is crucial for achieving accurate and reliable HIL simulations, as it ensures that the FPGA's processing is synchronized with the simulation's timing requirements.

Chapter 2

Implementation and Development

2.1 Implementation Tools:

Implementing a Hardware-in-the-Loop (HIL) simulation for this thesis also requires some software tools. These tools assist in designing, developing, and running the simulation model and help configure and interface the hardware components of the model. Some of these tools are:

2.1.1 MATLAB

MATLAB, an acronym for Matrix Laboratory, is a multi-paradigm numerical computing environment and proprietary programming language developed by MathWorks. In the late 1970s, MATLAB was created by Cleve Moler, then chair of the computer science department at the University of New Mexico. Moler designed MATLAB to give his students access to LINPACK and EISPACK without requiring them to learn Fortran. It has since evolved into a powerful tool for matrix manipulations, functions, data visualization, algorithm implementation, user interface creation, and interfacing with programs written in other languages, including C, C++, Java, and Python.

The main and most widely used windows are the following four:

- Command window: is a window of the main MATLAB interface in which it is possible to type supported commands and view the results on the screen in real time;

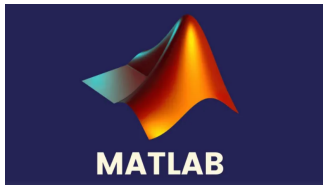


Fig. 2.1 Matlab

- Workspace: the workspace (or memory space) containing the declared variables;
- Current directory: it allows us to explore the contents of the folders on your memory medium;
- Command history: all recently typed commands are listed, divided by time and date

2.1.2 Simulink

Simulink is software for modeling, simulation, and analysis of dynamic systems. It was developed by the US company MathWorks and is tightly integrated with MATLAB. It is a block diagram environment used to design systems with multidomain models, perform simulations before moving on to hardware, and proceed with distribution without writing code. Simulink is used for Model-Based design, simulation, Model-Based System Engineering, and agile software development. The advantages of Simulink are that it requires neither the formulation of differential equations nor the knowledge of a particular language. It uses a graphical interface that allows you to build models such as block diagrams. It provides users with a wide range of predefined functional blocks so that almost all projects are reduced in practice to the appropriate interconnection of these blocks. Simulink contains a library of blocks that describes elementary static and dynamic elements. The user composes the block diagram of the system to be simulated by interconnecting the elementary blocks. Simulink automatically generates the equations and solves the desired numerical simulation problem. Models built in Simulink can be hierarchical models: each system block can be a complex subsystem. Simulink interacts with MATLAB through the Workspace (Simulink models can contain Workspace variables). Similarly, the results of the simulations can be exported to the Workspace and analyzed with MATLAB.

dSPACE Toolchain: dSPACE offers HIL simulation systems and tools. Their toolchain also features a software tool known as ConfigurationDesk, which allows configuring the hardware and software components of an HIL system, as well as another tool named ControlDesk that enables monitoring and interacting with the simulation—occurring in real time on an HIL setup. The ConfigurationDesk supports the engineer in defining I/O interfaces, signal conditioning, and various parameters involved within the HIL system. The easy-to-use interface on the ControlDesk enables one to view the simulation data, make changes in the parameters, and inject faults into the system to test its efficacy.

2.1.3 Configuration Desk

Configuration Desk [3]is a highly intuitive graphical configuration and implementation software for extensive hardware-in-the-loop (HIL) testing and management of RCP (Rapid Control Prototyping) applications. This software is integrated with dSpace real-time hardware platforms, such as SCALEXIO, and is utilized in our thesis to implement behavioral models and I/O function code. Configuration Desk facilitates the configuration of real-time hardware and associated behavioral models from a structured setup of external devices. The software offers numerous advantages for the development and testing of controllers:

.

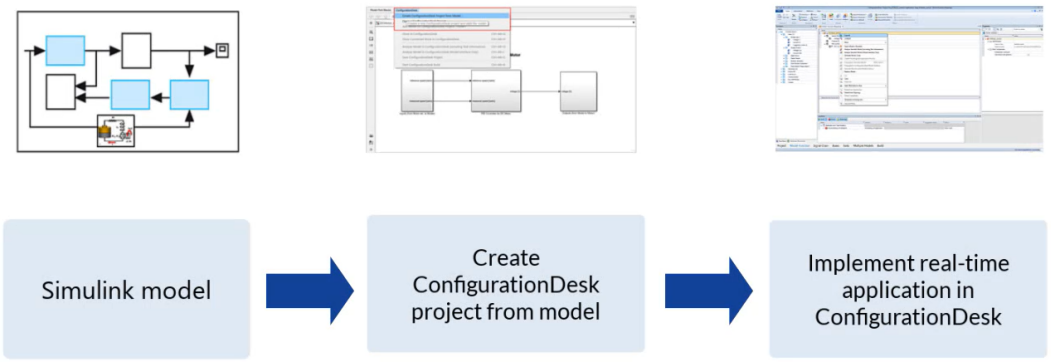


Fig. 2.2 Configuration Desk

- Provides comprehensive management of the signal pathway from the external device to the model interface, ensuring a clear overview of the entire application.
- Decouples the behavioral model from the I/O port configuration, enhancing the model's flexibility and reusability.
- Supports simulation of I/O with preset values, enabling testing even when the actual I/O ports are not yet operational.
- Facilitates the automatic deployment of applications on Real-Time dSpace hardware.
- An intuitive graphical interface guides the workflow for CPR and HIL applications, enhancing user experience.
- Allows for integrating multiple models, supporting the development of extensive modular applications.

With tools like MIPS and Simulink Coder, it is possible to generate Simulink Implementation Container (SIC) files from Simulink models. These SIC files contain all necessary codes to execute the models across various projects and on different dSPACE platforms, including VEOS, MicroAutoBox III, and SCALEXIO. This code is then loaded onto a real-time application via Configuration Desk, streamlining the deployment process.

2.1.4 Control Desk

Control Desk is a comprehensive dSPACE software tailored to develop control units. It provides a unified work environment for managing all stages of an experiment, from initiation to completion. Primarily utilized for Rapid Control Prototyping (RCP), Hardware in the Loop (HIL) simulations, ECU measurements, calibration, and diagnostics, it also facilitates access to system buses like CAN, CAN FD, LIN, and Ethernet. Additionally, it supports virtual validation with tools such as VEOS and SCALEXIO.

With ControlDesk, you can organize project or experiment data that can later be accessed in operator mode. The software enables the management of real-time

applications created in the Simulink environment and set up through ConfigurationDesk. Furthermore, it allows for the creation of a graphical user interface that helps monitor and control all relevant variables within the mode. Key Benefits for ECU Development ControlDesk offers you various tools to create the best possible conditions for your specific use cases in ECU development.

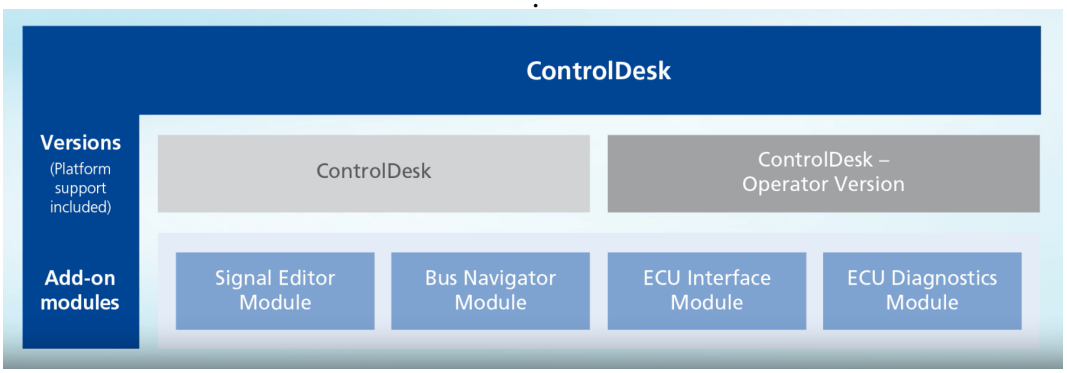


Fig. 2.3 Control Desk

Chapter 3

RT-Model architecture

3.1 eDrive Model

In this section, the modeling approach is based on the enhanced Demo Drive model dSPACE, which has been further developed by Kineton. This eDrive model, executed within the MATLAB-Simulink environment and established a robust framework for detailed analysis and simulation. My thesis builds upon Kineton's improvements, introducing additional refinements to the model to meet the specific requirements of this project. This is the general layout for the initial model structure. Each subsystem includes different functions, all summarized in a structure array where every field is associated with the corresponding subsystem. The general system model is composed of the following subsystems:

- **ENVIRONMENT CONTROL**
- **DC LINK**
- **INVERTER**
- **MOTOR**
- **MECHANIC**
- **NETWORK**
- **SoftECU**

- **I/O**
- **ENCODER**

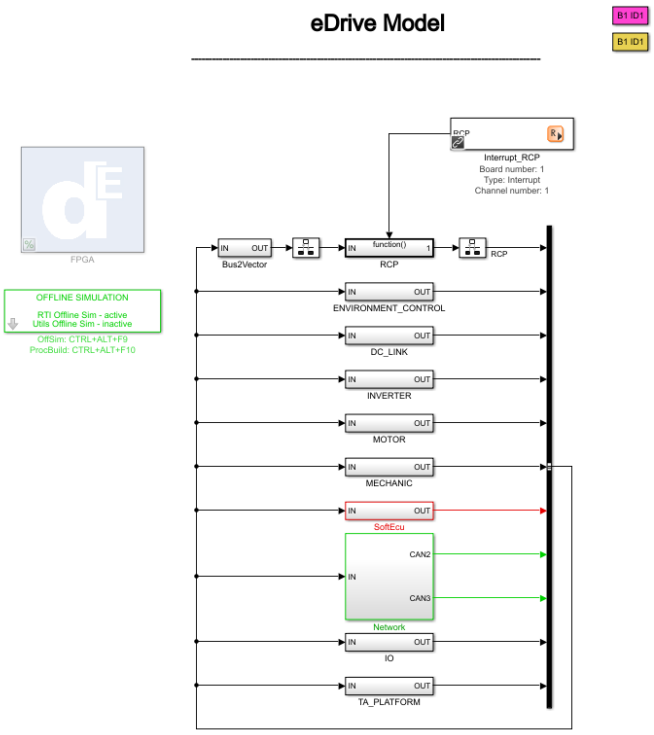


Fig. 3.1 eDrive Model

3.1.1 ENVIRONMENT_CONTROL

The **ENVIRONMENT_CONTROL** subsystem is crucial for setting and managing the operational environment within the simulation. It includes controls for:

- **V_DC_Link [V]**: Sets the supply voltage level.
- **V_DC_Link Mode [0|1]**: Chooses between a dynamic voltage simulation using a capacitor and a static voltage.
- **Reset [0|1]**: Resets the FPGA.
- **External Torque Switch [0|1]**: Selects constant torque or rotational speed dependency.

- **Control Mode [1|2]**: Manages logic for open-loop operations.
- **KL15 [0|1]**: Activates or deactivates the system.
- **n_Set [rpm]** and **Trq_set [Nm]**: Specify rotation speed and mechanical torque, respectively.
- **Watchdog Features**: Allows software resets directly from ControlDesk, facilitating real-time monitoring and adjustments.

This configuration ensures comprehensive control over simulation parameters, enhancing testing accuracy and flexibility.

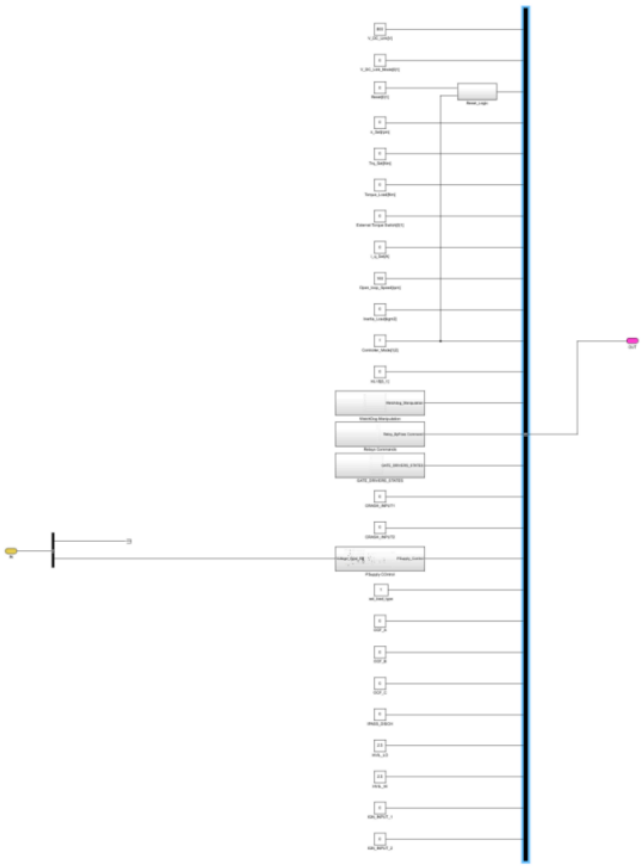


Fig. 3.2 ENVIRONMENT_CONTROL Model

The **ENVIRONMENT _CONTROL** subsystem includes several constants used for setting specific operational parameters and fault conditions, such as:

- **CRASH_INPUT1** and **CRASH_INPUT2**: Trigger specific crash scenarios.
- **OCF_A**, **OCF_B**, **OCF_C**: Set commands for overcurrent faults.
- **HVIL_LO** and **HVIL_HI**: High Voltage Interlock settings.
- **IGN_INPUT1** and **IGN_INPUT2**: Ignition input states.
- **Various other constants**: Control relays and gate driver states.

Power Supply Control Subsystem

Manages the power supply to the system through **Voltage_Input_FB**, controlling variables such as **PowerSupply_PowerEnable1**, **PSBoard**, and **PS Current Limit** to adjust power delivery dynamically based on simulation needs.

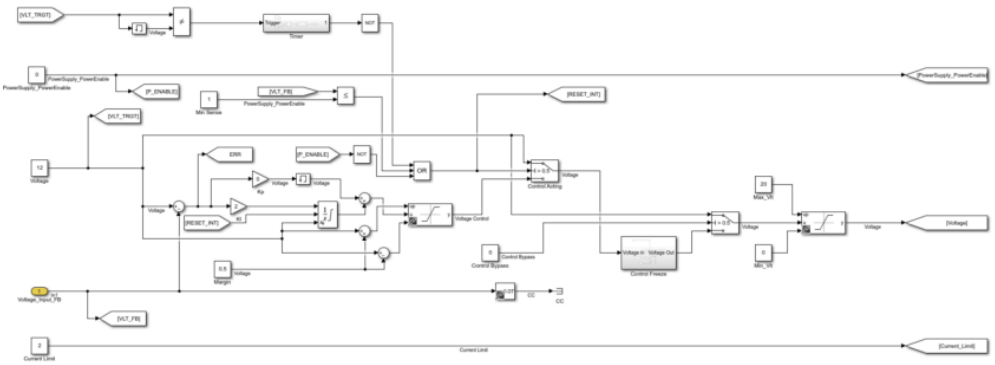


Fig. 3.3 Power Supply Control Subsystem

This integration ensures precise control over the simulation environment, enhancing the system's response to different test conditions.

3.1.2 DC_LINK

This block simulates the power supply to the inverter/motor system. It offers flexibility in how the voltage dynamics are modeled, allowing users to simulate realistic voltage behaviors or apply a fixed, predefined voltage level depending on the simulation's requirements. The `DC_LINK` block includes an option to select the mode of operation through the `V_DC_Link_Mode [0|1]` signal:

Mode 0: Directly imposes a specified voltage value, bypassing any dynamic voltage simulation. This mode is helpful for tests requiring constant voltage levels for stability or specific testing conditions.

Mode 1: Engages a more complex and realistic simulation of voltage dynamics. This is achieved using the “Capacitor” block, which calculates the effective voltage based on parameterizable capacitance and resistance values and the current flowing through the DC link.

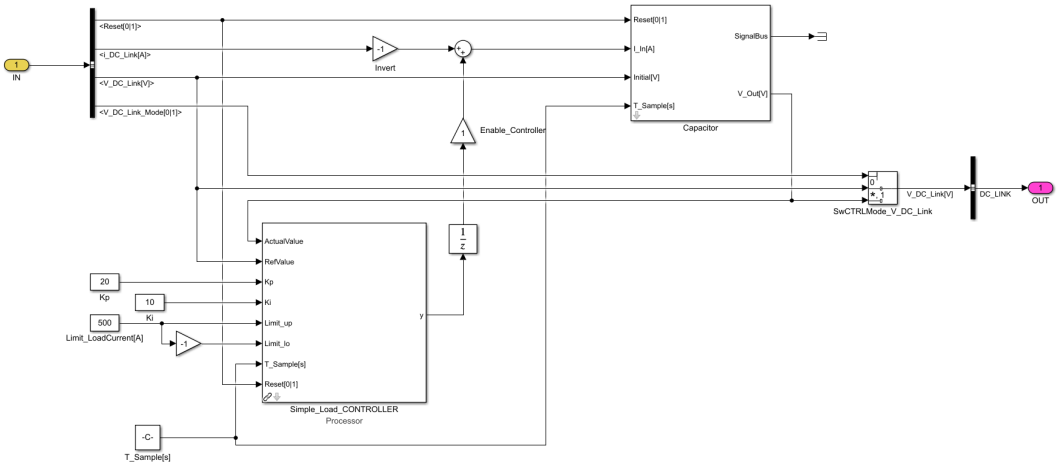


Fig. 3.4 DC_LINK Model

The core component for realistic voltage dynamics is the "Capacitor" block. This block represents how real-world capacitors would behave under varying electrical loads. It simulates the charge and discharge cycles that affect voltage levels based on the electrical current demands of the connected systems, such as motors and inverters. The capacitance and resistance values can be adjusted to match specific hardware specifications or experimental conditions, offering a versatile tool for engineers to replicate and study different scenarios accurately.

3.1.3 INVERTER

The DCM Inverter plays a pivotal role in the power electronics unit of our model, specifically tailored to process and convert DC electrical power into AC. The DCM Inverter, identified in the model as THREE_PHASE_DCM_INVERTER, encompasses six power switches of Insulated Gate Bipolar Transistors (IGBT) and body

diodes. This configuration ensures effective conversion while managing natural switching effects like free-wheeling diodes and passive energy recovery back to the battery.

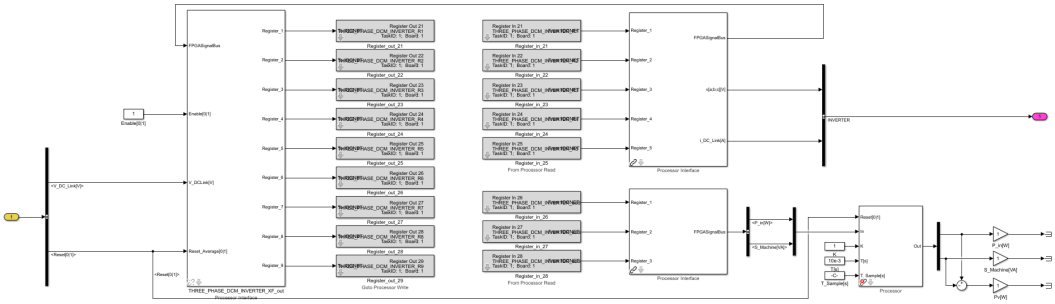


Fig. 3.5 Inverter Model

Functionality and Interaction

The inverter receives input from the DC Link, which provides the necessary power supply. It then processes this power through PWM (Pulse Width Modulation) signals generated by the control unit. These signals are crucial as they determine the operation of the switches that control the flow of electricity through the motor's phases, thus directly affecting the motor's behavior.

Processor Synchronous Average Calculation

A standout feature of this inverter is its ability to enable detailed and high-precision simulations of motor behavior. It incorporates a processor-synchronous average calculation of the motor torque and stator currents. This mechanism works by toggling a bit with every sample step (PROC_SYNC_IMP) on the processor side and synchronously on the FPGA side, capturing and calculating the time between the toggle's two edges. This sophisticated calculation allows for the precise control and adjustment of the motor's torque output based on real-time data.

FPGA Integration

On the FPGA side[4], the inverter is part of a complex system that ensures the accuracy of simulations by reducing the latency and increasing the computation speed, thanks to its parallel processing capability. This integration is crucial for real-time applications with critical timing and response speed.

Parameter Adjustments and Monitoring

The inverter model within the processor setup allows for online adjustments of characteristic parameters through the Controldesk interface, providing flexibility and ease in testing different scenarios. This functionality is essential for the dynamic testing environment of HIL simulations, where parameters often need quick adjustments based on the evolving conditions of the test.

3.1.4 MOTOR

The MOTOR model represents a high-fidelity simulation of a Permanent Magnet Synchronous Motor implemented within the MOTOR block of the simulation environment. This model is crucial for examining the dynamic responses of PMSMs to various control strategies and operating conditions in real-time simulations.[5] The PMSM_XF model is intricately linked to several other simulation blocks, forming a comprehensive system that reflects the complexities of real-world motor operation. It receives three-phase voltages from the INVERTER block, speed measurements from the MECHANIC block, and configurable motor parameters and settings from the MOTOR interface block within the processor model. All these parameters are adjustable in real-time using the Controldesk interface.

Input and Output Dynamics

Upon receiving its inputs, the PMSM_XF model processes the three-phase voltages through the `INPUT_ParkClark_v[a;b;c]` block, which applies Clarke and Park transformations. These transformations convert the voltages from the stationary abc reference frame to the rotating dq reference frame. The transformed voltages in the dq frame are then input into the `dq_MDL` block.

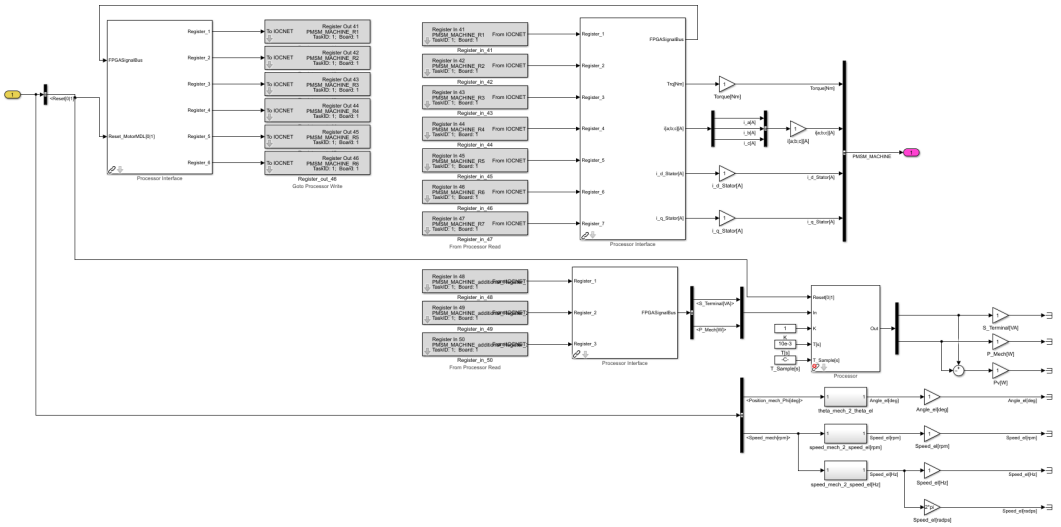


Fig. 3.6 Motor Model

Within the `dq_MDL` block, the core equations of the PMSM are employed to compute the corresponding currents in the dq frame and the motor torque, denoted as $Trq[Nm]$. These equations are given by:

$$\frac{di_d}{dt} = \frac{v_d - R_s i_d + L_q \omega i_q}{L_d},$$
$$\frac{di_q}{dt} = \frac{v_q - R_s i_q - L_d \omega i_d - \psi\omega}{L_q},$$
$$T_{rq} = \frac{3}{2} p \left(\psi i_q + (L_d - L_q) i_d i_q\right),$$

where i_d and i_q are the direct and quadrature axis currents, v_d and v_q are the direct and quadrature axis voltages, R_s is the stator winding resistance, L_d and L_q are the direct and quadrature axis inductances, ω is the angular velocity, ψ is the flux induced by the magnet, and p is the number of pole pairs.

The results from these calculations are then converted back from the dq frame to the abc frame using the `OUTPUT_ParkClark_i_HD[a;b;c]` block, yielding the three-phase currents essential for feedback to the control systems and further analysis.

3.1.5 MECHANIC

The MECHANIC block is a fundamental component within the simulation environment. It is designed to interact with the MOTOR block to simulate mechanical dynamics and responses in real-time. This block integrates mechanical parameters and settings from the processor-FPGA interface and handles the complex interactions between mechanical forces and motor outputs.

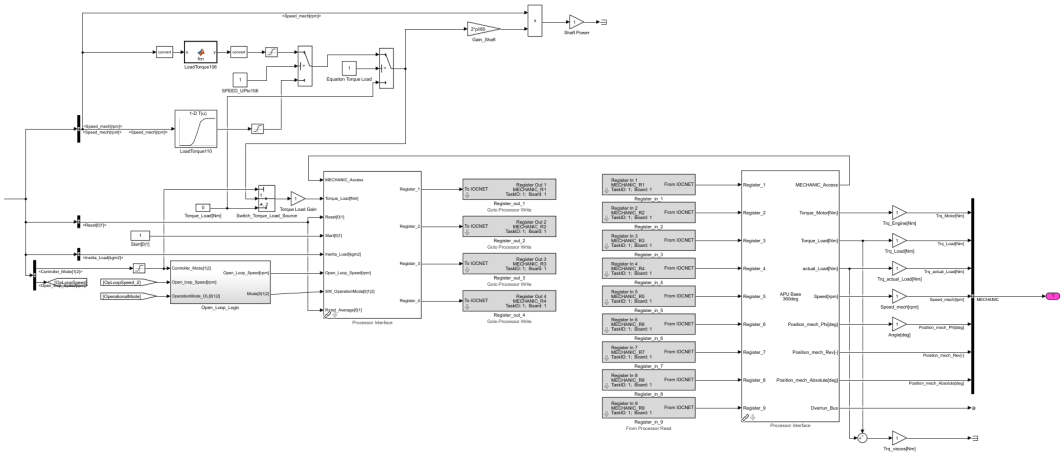


Fig. 3.7 Mechanic Model

Functionality and Interaction

The Mechanic block plays a critical role by receiving torque from the MOTOR block and mechanical parameters and settings from the processor-FPGA interface. This setup enables precise control over mechanical responses such as speed and rotation, which are fundamental for replicating real-world mechanical systems. The received torque is modulated by friction torque (`Friction_Trq`), which depends on the motor's speed and is vital in stimulating the motor's stopping behavior.

Outputs and Calculations

At the output, the Mechanic block yields essential metrics such as speed (`APU_Speed`) and angle of rotation (`APU_Angle`). These outputs are derived from the following equations, which are part of the integrated Permanent Magnet Synchronous Machine

(PMSM) framework, ensuring that the mechanical behavior accurately corresponds to the electrical inputs from the motor:

$$\omega_{\text{Motor}} = \frac{1}{J_{\text{Motor}} + J_{\text{Load}}} \int (Trq_{\text{Motor}} - Trq_{\text{Load}})\, dt$$

$$\varepsilon_{\text{Motor}} = \int \omega_{\text{Motor}}\, dt$$

Where:

- ω_{Motor} and $\varepsilon_{\text{Motor}}$ represent the angular velocity and position of the motor, respectively.
- Trq_{Motor} and Trq_{Load} denote the torques from the motor and load.
- J_{Motor} and J_{Load} are the inertias of the motor and load.

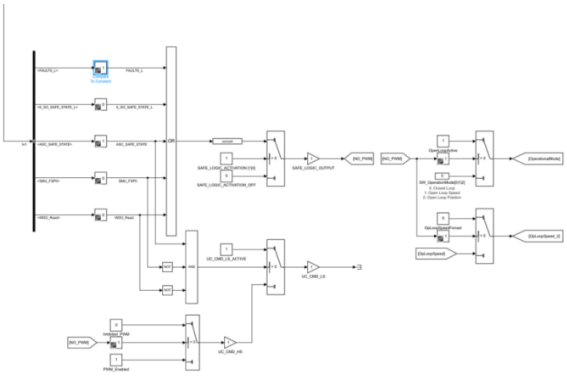


Fig. 3.8 Security and Fault Managementl

Security and Fault Management

Furthermore, the Mechanic block incorporates a security system capable of initiating an 'off state' similar to deactivating an inverter under specific conditions. This feature is critical for managing faults and ensuring safety within the simulation environment, responding to particular fault signals like `FAULTS_L` and `6_SO_SAFE_STATE_L`, thus preserving the system's integrity during anomalies.

Integration with FPGA

Regarding integration, the Mechanic block is intricately linked with FPGA functionalities to deliver real-time computation and response capabilities. This connection permits dynamic adjustments to mechanical parameters, essential for conducting tests involving variable mechanical loads and conditions.

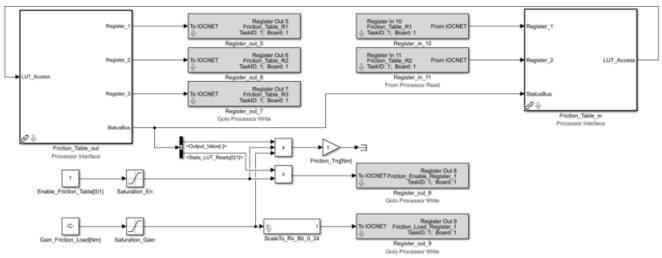


Fig. 3.9 Mechanic Model

3.1.6 NETWORK

The NETWORK block is essential for communication within the simulation environment, particularly handling CAN frames from various sources. It processes and directs frames generated by the soft ECU block and the Device Under Test (DUT) within the simulation loop. Specifically, it manages frames originating directly from the real Electronic Control Unit (ECU) as RX (Receive) frames, while transmitting (TX) frames are those generated by the soft ECU block.

The output functionality of the NETWORK block primarily focuses on dispatching `CAN RX` frames. These frames, received from the actual `ECU`, are fully compiled by the block, incorporating comprehensive details such as the frame's `ID`, signal names, message format, transmission status (`TX_Status`), reception status (`RX_Status`), and raw data bytes.

This capability ensures that each CAN frame on the network is thoroughly assembled and enriched with all necessary metadata, allowing the simulation model to control and manipulate the data precisely. This feature is particularly beneficial for testing and validating the interactions within vehicular networks, ensuring accurate representation and functionality testing in a controlled simulation environment.

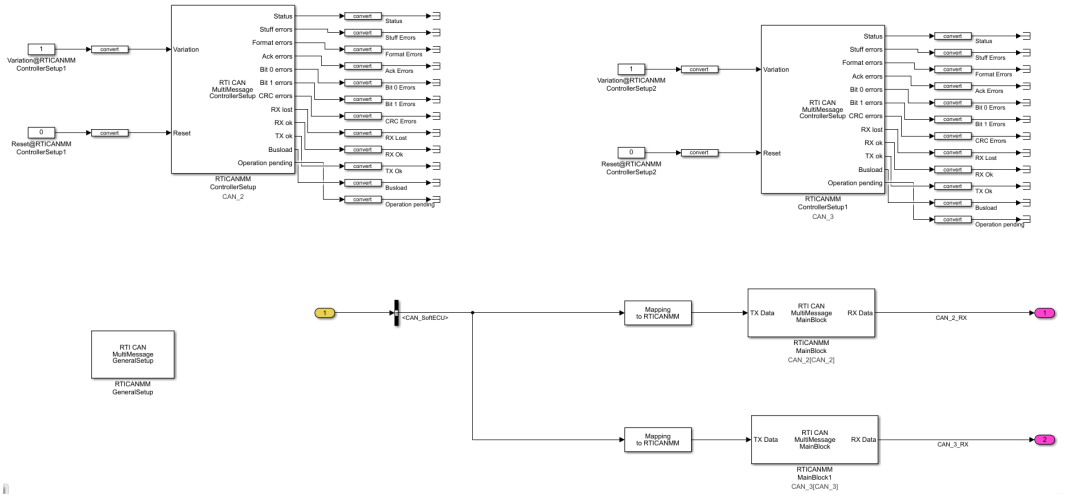


Fig. 3.10 Network

Controller Area Network (CAN)

Controller Area Network (CAN) is a robust vehicle bus standard designed to facilitate communication among various automotive or industrial system components. Developed by Bosch in the 1980s, CAN allows microcontrollers and devices to communicate with each other without the need for a host computer, providing a reliable and real-time communication system.

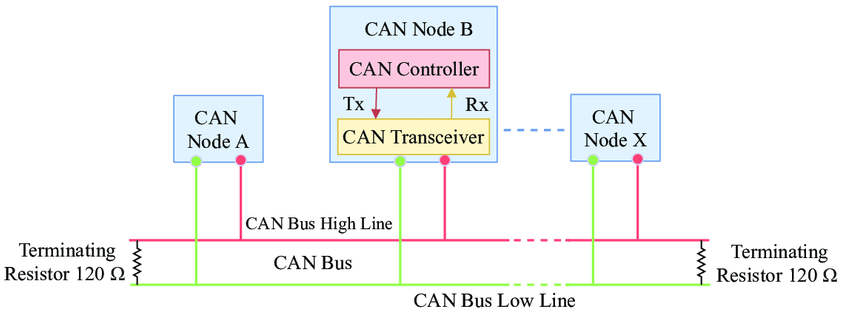


Fig. 3.11 CAN

Features of CAN CAN has several key features that make it suitable for critical applications:

- **Multi-Master Capability:** CAN allows multiple devices to take control of the bus, enabling a decentralized network without a central master device.

- **Collision Detection and Arbitration:** CAN uses a non-destructive bitwise arbitration method to handle collisions. When two devices transmit simultaneously, the one with the higher priority message continues, while the other waits, ensuring smooth communication without data loss.
- **Error Detection and Handling:** CAN includes several error detection and handling mechanisms, such as CRC checks, acknowledgment slots, and error counters. Faulty nodes are automatically removed from the network to maintain communication integrity.
- **High Speed and Reliability:** CAN operates at speeds up to 1 Mbps, with some advanced versions like CAN FD (Flexible Data-rate) offering higher speeds and larger data payloads. Its robust error-checking mechanisms make it highly reliable for critical applications.

Applications of CAN: CAN is widely used in various fields due to its reliability and efficiency:

- **Automotive:** In vehicles, CAN networks connect various electronic control units (ECUs), sensors, and actuators, handling everything from engine management to climate control and safety systems.
- **Industrial Automation:** CAN is used in factory automation systems, connecting programmable logic controllers (PLCs), sensors, and actuators.
- **Medical Equipment:** CAN networks ensure reliable communication in medical devices, such as patient monitoring systems and laboratory equipment.

Structure of CAN Frames CAN frames are the fundamental units of data transmitted over a CAN network. Each frame consists of several fields, including:

- **Identifier (ID):** Uniquely identifies the message type and its priority.
- **Control Field:** Contains information about the frame, such as its length and type.
- **Data Field:** Carries the actual data being transmitted, up to 8 bytes in standard CAN and more in CAN FD.

- **CRC Field:** Ensures data integrity by allowing error detection.
- **ACK Field:** Allows receivers to acknowledge the successful reception of the frame.
- **End of Frame:** Marks the end of the frame.

3.1.7 SOFT ECU

The SOFT ECU module plays a pivotal role in simulating the operations of a Control Unit within a CAN network, utilizing sophisticated artificial logic to manage and execute network communications. This simulation module is tailored to handle the ECM mode with its related frames, adhering closely to the predefined CAN dbc information. The design allows for the input of both model variables and measured variables, enhancing its utility in diverse testing scenarios. ECU's response capabilities under simulated service conditions.

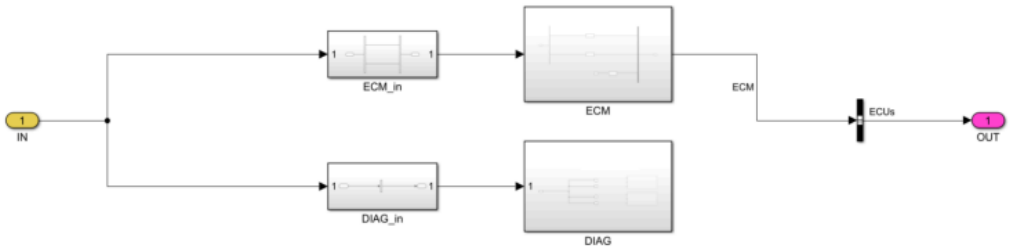


Fig. 3.12 Soft ECU

Within its operational framework, the SOFT ECU is adept at managing a complex array of data, encompassing two principal message types and three key variables. This capability ensures that the module can effectively mimic real-world ECU functionalities, providing a realistic environment for network communication testing.

A critical function of the SOFT ECU involves processing the message texttttECmprCtl_01_XIX_Motor_SUBCAN. This task includes monitoring the state of the KL15 switch to check if the ignition key is turned ON or OFF, determining the required revolutions for the eMotor, and computing the maximum allowable power output. These computations are vital for accurately controlling the eMotor's performance within the network.

Additionally, the module incorporates a DIAG block, a Multi-caching Frame Machine that operates independently of the network. This block is particularly proficient in handling service requests, such as when service 30 hex is activated by a HiL User, and it can manage up to 60 frames per session. This function is crucial for assessing the ECU's response capabilities under simulated service conditions.

Furthermore, the SOFT ECU module is designed to handle various fault conditions and scenarios that might occur during operation. It can simulate a range of faults, including sensor malfunctions, communication errors, and actuator failures, providing a comprehensive testing environment. The ability to inject faults and monitor the ECU's response helps validate the robustness and reliability of the control algorithms under adverse conditions.

The SOFT ECU also supports real-time data logging and monitoring, enabling engineers to capture and analyze the performance metrics and operational data. This feature is essential for fine-tuning the control strategies and ensuring that the ECU operates optimally in all scenarios.

The SOFT ECU module is a powerful tool for developing and validating automotive control systems. Simulating real-world conditions and providing detailed insights into the ECU's behavior plays a vital role in the design and testing process, ensuring that the final product is reliable and efficient.

3.1.8 I/O

The I/O block in the model comprises five subsystems essential for interfacing and data conversion between the Simulink models and the hardware:

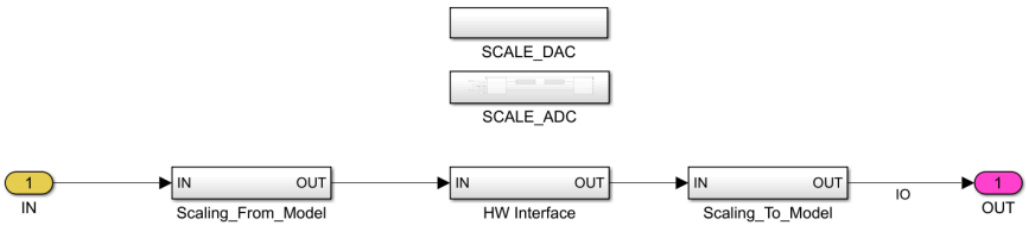


Fig. 3.13 I/O

- **SCALE_DAC**: Manages digital-to-analog conversions, enabling direct control over phase voltages and currents through user inputs in ControlDesk.

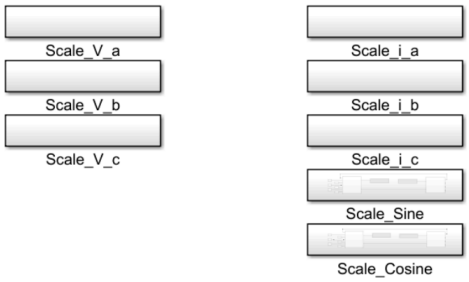


Fig. 3.14 DAC

- **SCALE_ADC**: Handles analog-to-digital conversions, capturing essential physical inputs like voltage and current from the Device Under Test (DUT).
- **Scaling_From_Model**: Translates model outputs to hardware-compatible signals, ensuring they meet the specifications for real-world application.
- **HW_Interface**: Acts as the central hub for all input and output operations, integrating signals from environmental and behavioral models.
- **Scaling_To_Model**: Converts incoming hardware signals back into a format usable by the Simulink models, maintaining the fidelity of simulation data.

Operational Flow and Integration

- Inputs to the **SCALE_DAC** include three-phase voltages and currents, which, after processing, are output directly to the I/O interface.
- The **HW_Interface** subsystem serves as the nexus for all data exchange, managing both digital and analog signals and coordinating with power supply management features for optimal performance.
- ADC inputs, such as voltage and current feedback from the DUT, are crucial for the system's ability to monitor and respond accurately to changes.

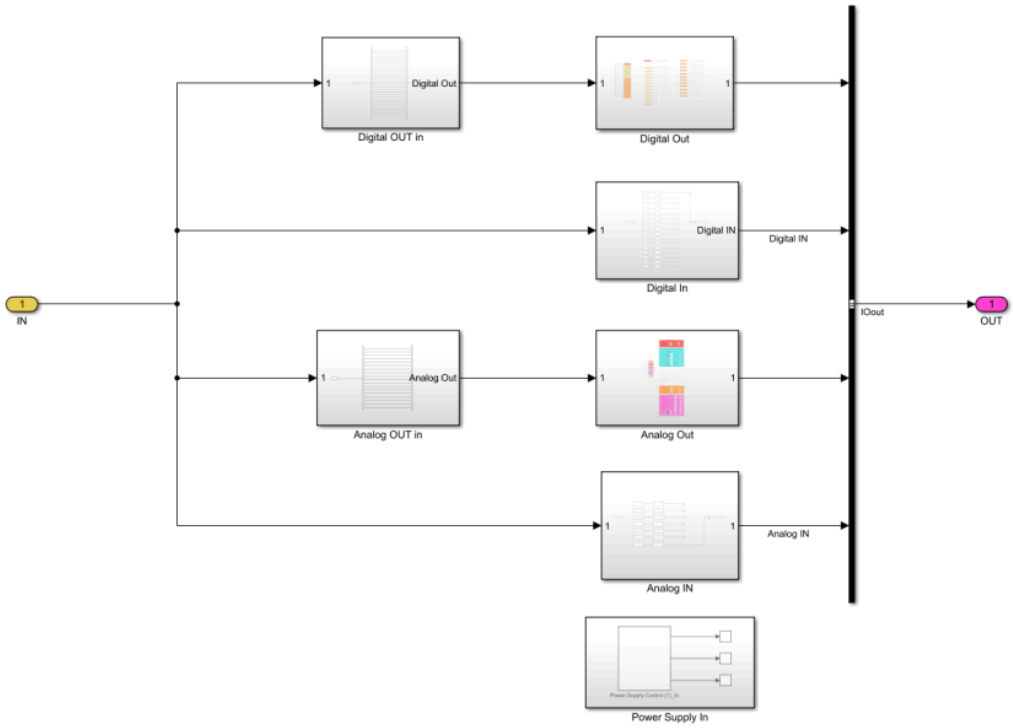


Fig. 3.15 HW-Interface

3.2 FPGA Model

Field-Programmable Gate Arrays (FPGAs) [6]are pivotal in high-intensity testing environments due to their flexibility and high-speed computational capabilities. This discussion elaborates on the specific roles and advantages of using an FPGA in Hardware in the Loop (HIL) systems, particularly focusing on the simulation of electromechanical systems like Permanent Magnet Synchronous Motors (PMSM).

Parallel Processing and System Efficiency

FPGAs enhance simulation environments through their inherent parallel processing capabilities, significantly outpacing traditional microcontrollers that execute instructions sequentially. This feature is critical in simulations where rapid processing and response times are essential.

- **Hardware-level Configuration:** The architecture of FPGAs allows for customized configurations that optimize computational efficiency. Each operation

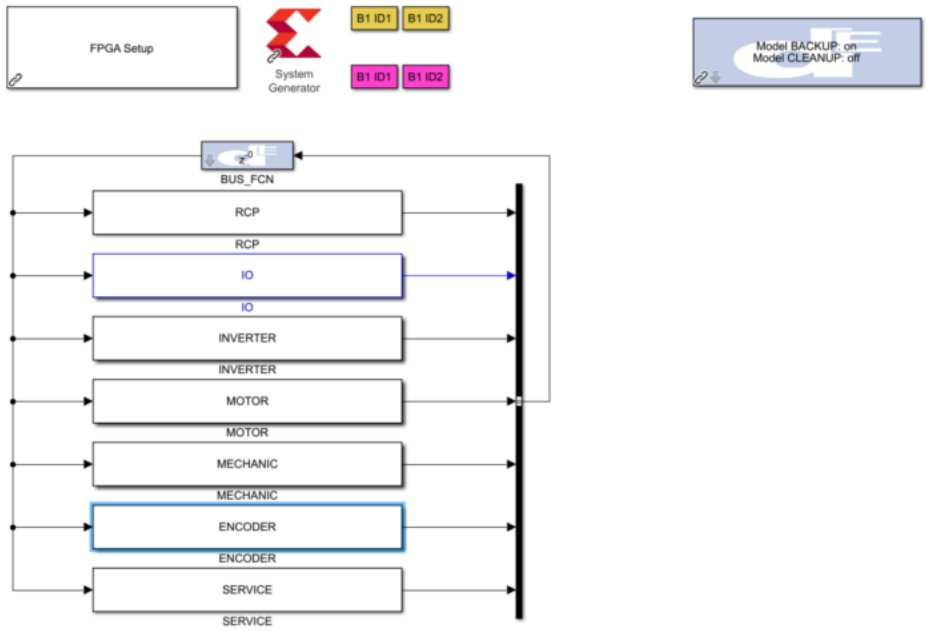


Fig. 3.16 FPGA

within the FPGA is implemented via logic gates directly on the hardware level, enabling faster execution speeds.

- **DS6602 FPGA:** The chosen FPGA for our application is the DS6602, provided by dSPACE. It is specifically tailored for dynamic simulations of complex models, particularly in the automotive industry. Detailed specifications and performance metrics can be found in the referenced hardware documentation [7].

Integration with XSG Electric Components Library

The integration of the FPGA with the XSG Electric Components library demonstrates its capability to handle sophisticated simulations that accurately reflect real-world behaviors. This integration is crucial for the PMSM simulation, utilizing components such as the APU, Resolver, and a custom-designed inverter.

- **PWM Measurement and Generation:** The FPGA is equipped with functions for PWM control, crucial for the precise manipulation of motor speeds and

torques. This capability ensures that the motor control algorithms can be tested under various conditions and configurations.

- **Signal Capturing with Multi-Scope Function:** The FPGA's Multi-Scope function allows for the capturing and monitoring of various signals across the system, providing a comprehensive view of the system's operational status.
- **Real-Time Signal Scaling:** The MultiScale-DAC function implemented within the FPGA facilitates the real-time scaling of DAC signals, enhancing the adaptability of tests and simulations.

Comprehensive FPGA Model Structure

The FPGA model encompasses several functional blocks including RCP, I/O, INVERTER, MOTOR, and MECHANIC. These components are orchestrated to simulate and control every aspect of the motor's operation, from electrical inputs to mechanical outputs, ensuring high fidelity in the simulations.

3.2.1 RCP

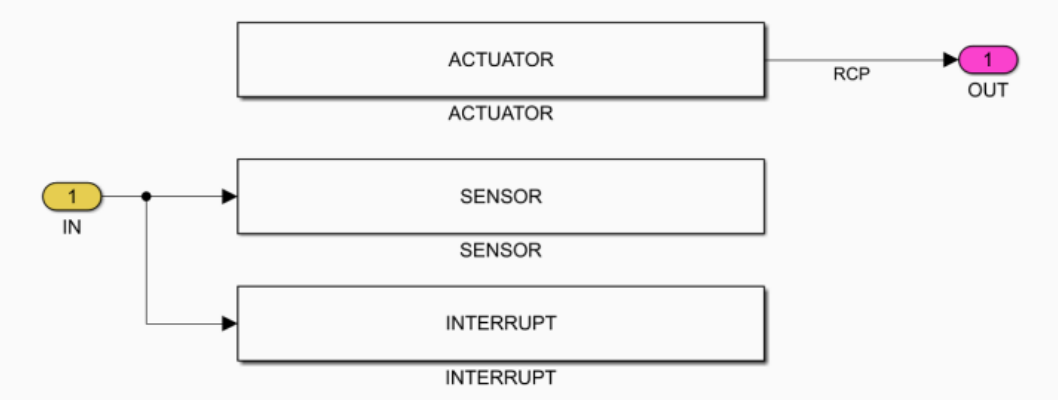


Fig. 3.17 RCP

The PMSM Controller task, RCP (Rapid Control Prototyping), includes the PMSM controller of the motor. In this subsystem, a simulation of a current measurement and a simulated sensor feedback for speed and position are included. The Controller measures with these inputs the actual motor current, the actual speed and position of the motor and provides the necessary duty cycles of each motor phase.

To generate a PWM signal from this duty cycle the PWM-Generator (located in the subsystem THREE_PHASE_PWM_GENERATOR) of the XSG Utils[8] library is used.

ACTUATOR: On the ACTUATOR block there are no direct inputs but only inputs coming from the FPGA interface, the output is the PWM generated to command the inverter gates.

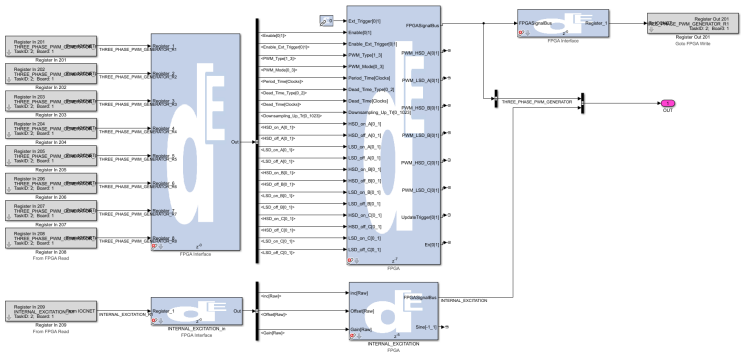


Fig. 3.18 ACTUATOR-FPGA Model

SENSOR: this interface block sends information from the ENCODER block to the FPGA interface.

INTERRUPT: This feature adjusts the computation parameters of the primary task, ensuring optimal synchronization and efficiency.

3.2.2 I/O Functional Blocks

Analog I/O

The Analog I/O block is designed to interface physical values with the FPGA's oscilloscope. It receives inputs such as the three-phase currents, the three-phase voltage EMFs, and the internal excitation from the RCP block, facilitating real-time monitoring and analysis.

Digital I/O

The Digital I/O block plays a critical role in controlling the PWM signals. It accepts enabling signals for the internal controller and the six excitation values necessary for

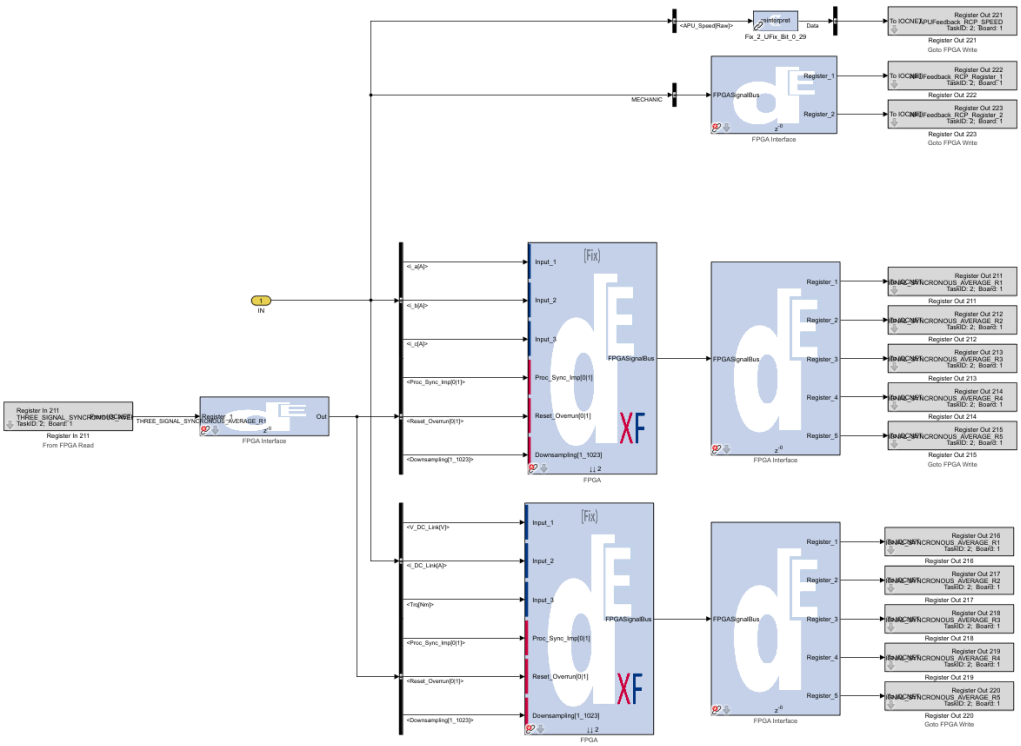


Fig. 3.19 SENSOR-FPGA Model

PWM generation. The outputs from this block are the six finalized PWM signals, which are crucial for the actuation of power devices within the system.

3.2.3 INVERTER

The **THREE_PHASE_DCM_INVERTER_XF** block constitutes a three-phase power converter, integrating six power switches (IGBTs and body diodes) in a bridge configuration. This block is engineered to perform highly precise simulations, accounting for natural switching phenomena such as the behavior of the free-wheeling diode and passive energy recovery to a battery system.

3.2.4 Motor Model

This block incorporates a synchronous averaging mechanism of motor torque and stator currents to facilitate the reusability of critical motor model parameters in advanced processor models. This feature, termed *processor synchronous impulse*

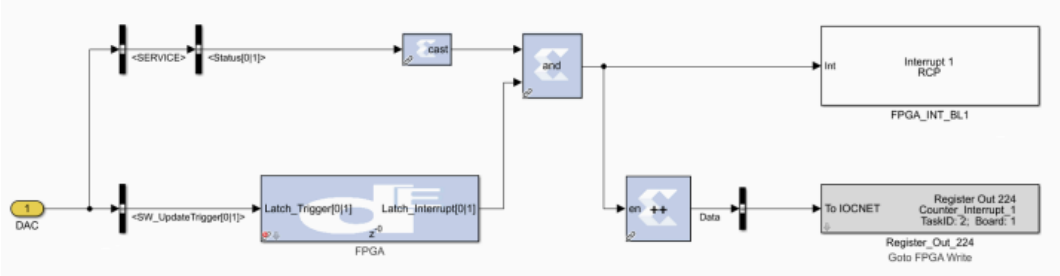


Fig. 3.20 INTERRUPT-FPGA Model

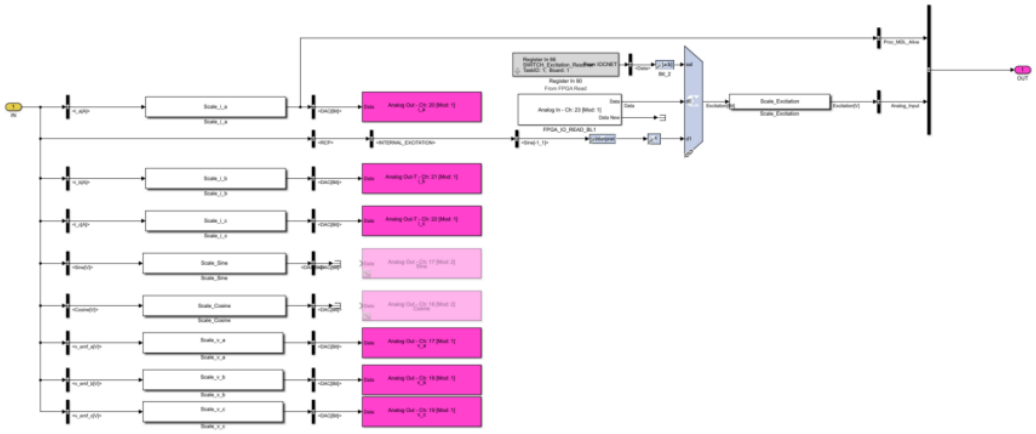


Fig. 3.21 Analog I/O-FPGA Model

(PROC_SYNC_IMP), operates by toggling a bit at every sample step. On the processor side, this bit triggers the accumulation of torque and current values, while on the FPGA side, it coordinates the timing and accumulation of these values between toggle events. This system also supports downsampling to extend the capture duration, although this may affect resolution. Additionally, a reset functionality for the averaging process is integrated to maintain accuracy and reliability in continuous operation.

MOTOR Block

The MOTOR block simulates a permanent magnet synchronous machine (PMSM) modeled in fixed-point arithmetic. The sinusoidal back electromotive force (EMF) generated by the PMSM dictates that the machine be modeled within dq coordinates, also known as the rotor reference frame. This approach ensures that all motor parameters, including phase currents and transformed currents, can be dynamically

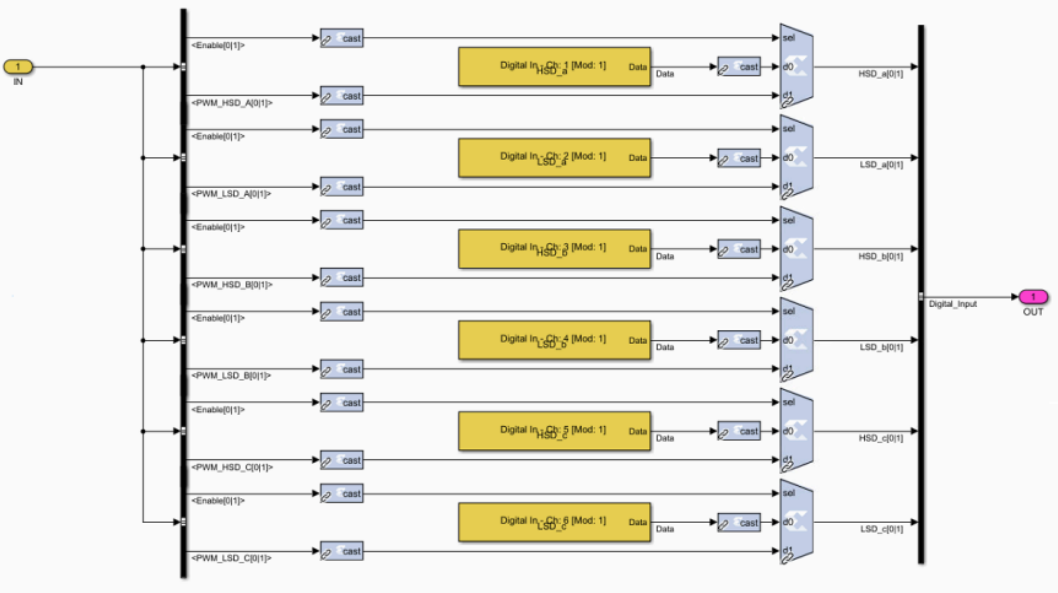


Fig. 3.22 Digital I/O-FPGA Model

adjusted by the processor in real time, corresponding to the processor's sampling rate.

Dynamic Parameter Adjustment

In addition to standard operation, the MOTOR block can be set into a stimulus mode, allowing for individual phase currents (a, b, c) or transformed currents (d, q) to be manually specified. This feature is particularly useful for verifying the current scaling against the electronic control unit (ECU) specifications, ensuring compatibility and performance tuning.

Integration with System Blocks

The MOTOR block integrates seamlessly with the INVERTER and MECHANIC blocks, receiving comprehensive input data that influences all operational aspects. The output from the MOTOR block encapsulates all relevant physical values such as phase voltages (Va, Vb, Vc) and currents (i, a, i, b, i, c), which are consolidated into a single array known as the FPGA Signal Bus. This array is crucial as it provides a uniform data structure transmitted to both the FPGA interface and the processor, en-

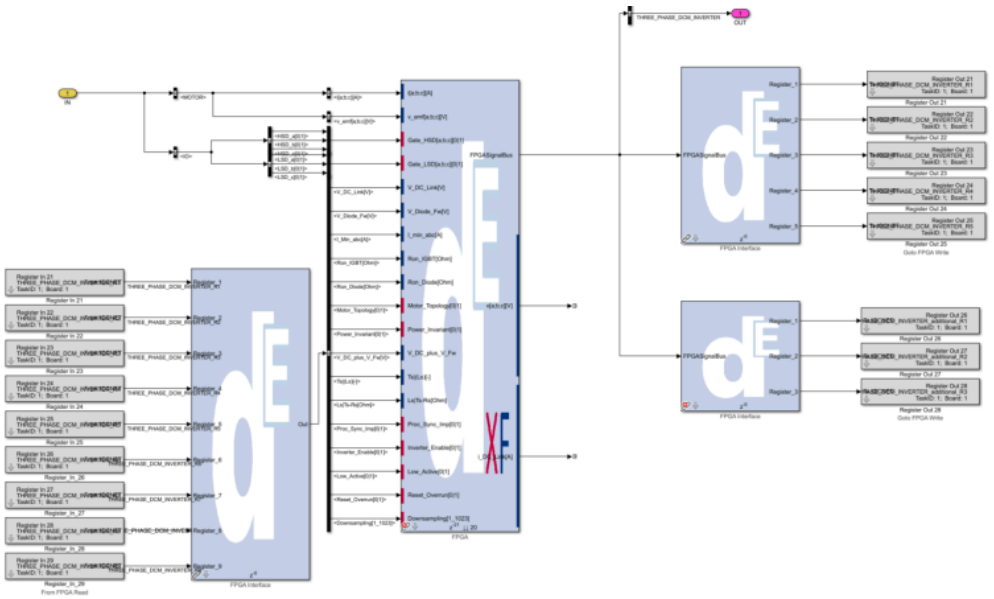


Fig. 3.23 INVERTER-FPGA Model

suring a consistent and accurate representation of the motor's operational parameters across the simulation platform.

Synchronization and Feedback

The structured data from the FPGA Signal Bus is also relayed to the Simulink model, facilitating a synchronized feedback loop that enhances the simulation's accuracy and responsiveness to parameter changes. This integration underscores the MOTOR block's pivotal role in accurately simulating electromechanical dynamics within the HIL system.

3.2.5 MECHANIC

The MECHANIC block in the FPGA framework facilitates the simulation of a mechanical model characterized by minimal inertia. This module is an integral part of the system, working in conjunction with the motor dynamics to provide a realistic simulation environment.

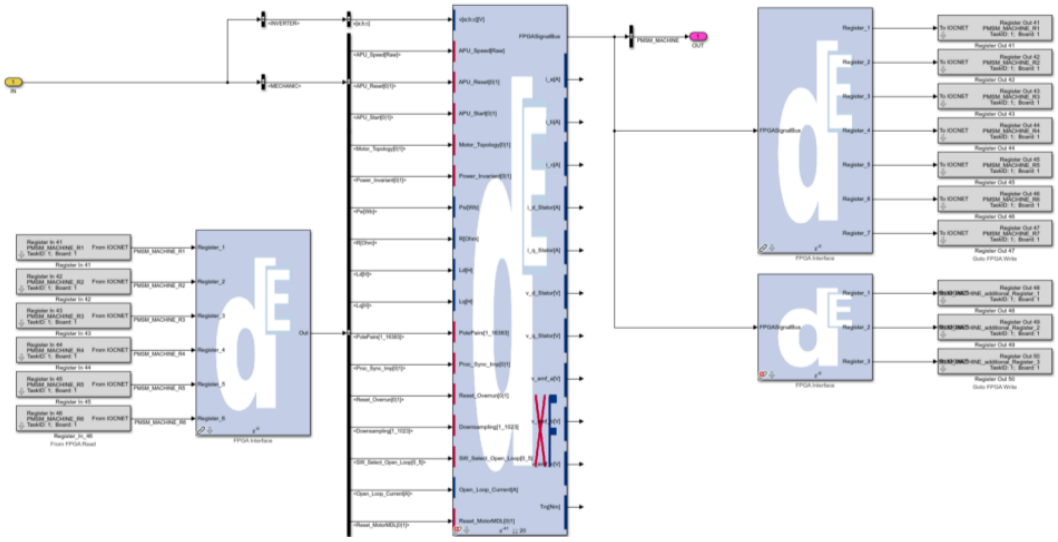


Fig. 3.24 MOTOR-FPGA Model

Functional Integration

The mechanical block set within the FPGA represents a crucial bundle of the processor model functions, specifically "Motor Inertia" and "Motor Position." These functions are detailed in the "Processor-based model parts" chapter, underscoring their importance in the overall simulation architecture.

Dynamic Parameter Adjustability

Crucially, this block allows for real-time adjustments to various parameters such as the damping coefficient and load torque and the implementation of open-loop control functionalities. These adjustments are made possible through direct inputs from the processor during runtime, ensuring that the model's mechanical response can be finely tuned to match the simulated conditions accurately.

Synchronization and Control

The MECHANIC block enables parameter adjustments during runtime to ensure the mechanical simulation remains synchronized with other system components. This synchronization is vital for maintaining the simulation's fidelity, particularly in scenarios where dynamic changes to the mechanical properties are required.

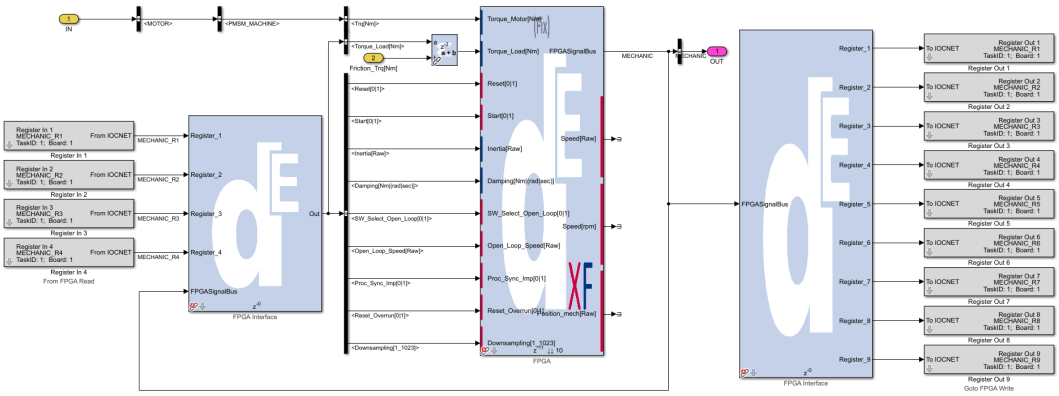


Fig. 3.25 MECHANIC-FPGA Model

Chapter 4

Restructuring model implementation and new model architectures

4.1 Methodology and Approach

4.1.1 Multi-Core Approach Rationale

The SALEXIO Multi-core architecture is specifically designed to meet the demands of modern Hardware-in-the-Loop (HIL) simulations, which require high computational power and efficient data handling. This architecture leverages multiple processing cores within a single unit or across multiple units to enhance performance and scalability.

SALEXIO Multi-core Architecture

The SALEXIO system utilizes two synchronized cores: a white-box core and a black-box core. The white-box core offers full access to the code, allowing developers to perform complete calibration, add new components, and carry out comprehensive bus simulation modeling. This core is essential for in-depth testing and development activities. Conversely, the black-box core contains proprietary intellectual property and uses only the build results, ensuring the protection of sensitive information. This dual-core setup provides a balanced approach, enabling the reuse of the black-box core while offering full development capabilities on the white-box core.

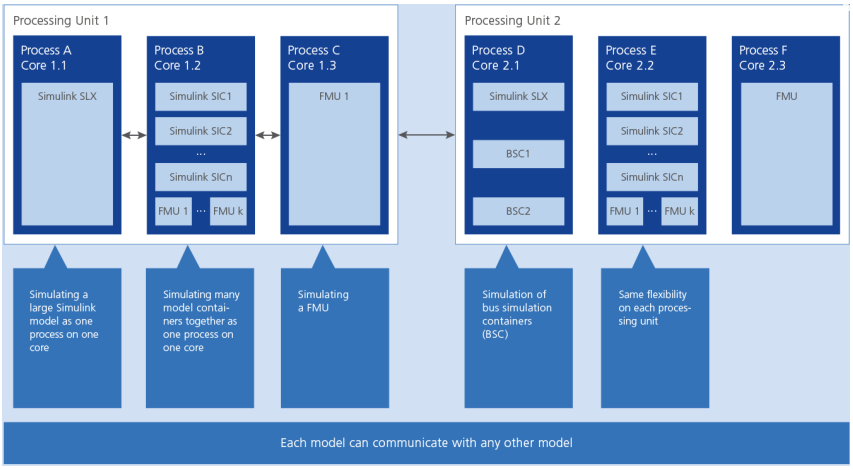


Fig. 4.1 SALEXIO Multi-core Architecture

Task Management:

Effective task management is crucial in multi-core systems to optimize the performance of real-time applications. Tasks are pieces of code controlled by a real-time operating system (RTOS) and are executed based on their assigned priorities. High-priority tasks can preempt low-priority tasks, ensuring that critical operations are completed promptly. Each task can execute one or more runnable functions, which are the functional blocks of code that perform specific computations. By managing tasks efficiently, the system ensures that real-time performance requirements are met and computational resources are optimally utilized (DSpace) (DSpace).

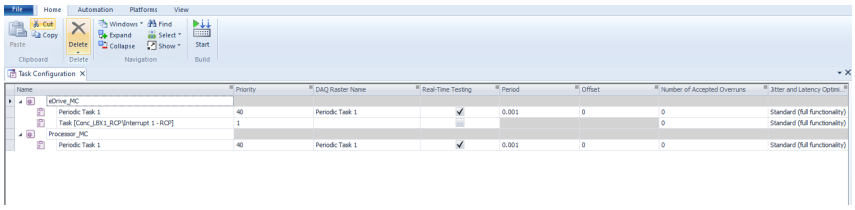


Fig. 4.2 Task management

Interrupt Techniques

Interrupt techniques are used to manage the execution of tasks by allowing higher-priority tasks to interrupt lower-priority tasks. This preemptive multitasking ensures that urgent tasks are addressed immediately, maintaining the application's real-time

performance. The turnaround time for a task includes the execution time and any delays caused by higher-priority tasks. By monitoring the task turnaround time and the number of task overruns, developers can optimize task execution and avoid performance bottlenecks (DSpace) (DSpace).

Multi-PU System

A multi-processing-unit (multi-PU) system consists of several coupled processing units, increasing the available real-time processing power and the number of possible I/O connections. This setup is proper for large real-time applications and testing the interaction between multiple electronic control units (ECUs) of a target vehicle without integrating them into a real vehicle. Multi-PU applications must be partitioned into several parts, each executed on a separate processing unit, enabling parallel processing and efficient resource utilization (DSpace) (DSpace) (DSpace).

The multicore approach ensures correct data propagation between the processor's cores and FPGA, optimizing performance for complex simulations. This architecture is particularly effective in maintaining the integrity and performance of real-time scenarios, enhancing the accuracy and reliability of HIL simulations (DSpace) (DSpace).

By leveraging the capabilities of the SALEXIO Multi-core architecture, the system can handle complex real-time simulations efficiently, ensuring accurate and reliable performance in various testing scenarios. This architecture facilitates the protection and reuse of intellectual property while providing the flexibility needed for comprehensive development and testing.

4.1.2 Model Reusability and Calibration

The concepts of model reusability and calibration are integral to enhancing the efficiency and effectiveness of Hardware-in-the-Loop (HIL) simulations. This section thoroughly explores these concepts, focusing on the strategies and tools that facilitate model reuse and the calibration processes that ensure model accuracy and reliability.

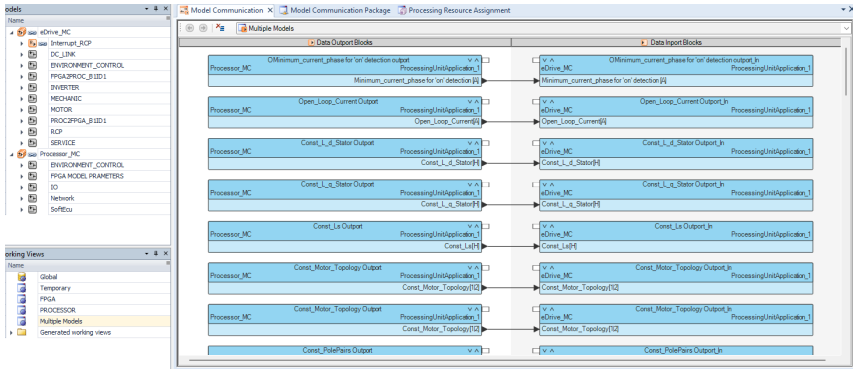


Fig. 4.3 Model Interfaces

Decoupling Behavioral Models from I/O Configuration

One primary method to enhance model reusability is decoupling behavioral models from their I/O configurations. Tools like ConfigurationDesk facilitate this by allowing developers to define and manage the I/O interfaces separately from the behavioral models. This separation means the same model can be reused with different I/O configurations, making it adaptable to various hardware setups and project requirements.

Use of Simulink Implementation Container (SIC) Files

With tools like Simulink Coder, SIC files can be generated from Simulink models. These SIC files encapsulate all necessary code to execute the models across different projects and dSPACE platforms, including VEOS, MicroAutoBox III, and SCALEXIO. This encapsulation ensures that models can be easily transferred and reused in different simulation environments without the need for recoding or extensive modifications.

Automatic Deployment and Model Portability

ConfigurationDesk supports the automatic deployment of applications on dSPACE real-time hardware. It ensures a clear overview of the entire application and enhances the model's flexibility and reusability by managing the signal pathway from the external device to the model interface. The graphical interface of ConfigurationDesk

simplifies the process of integrating multiple models, supporting the development of extensive modular applications that can be easily modified and reused as needed.

4.1.3 Calibration

Calibration is the process of fine-tuning model parameters to ensure that the simulated results closely match real-world behaviors. Accurate calibration is essential for the reliability and validity of HIL simulations.

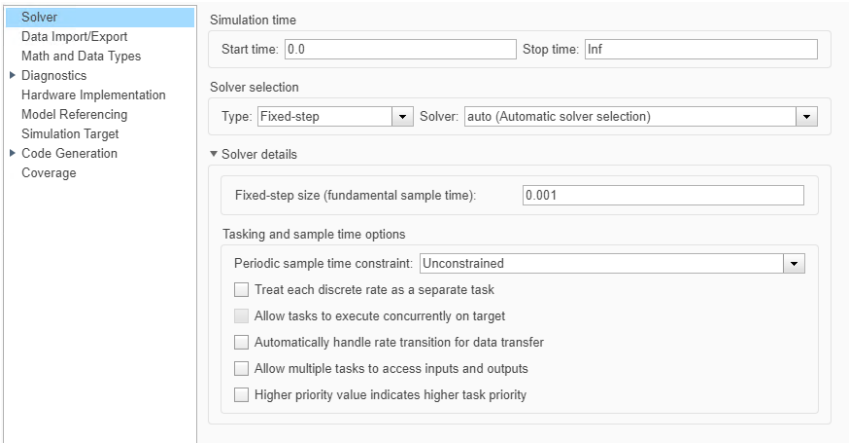


Fig. 4.4 Calibration

Parameter Management

Effective calibration involves managing a wide range of parameters that influence the behavior of the simulation models. ConfigurationDesk provides tools for setting and adjusting these parameters, ensuring the models behave as expected under various conditions. This includes handling both digital and analog signals and coordinating with power supply management features for optimal performance.

Real-Time Monitoring and Adjustment

Tools like ControlDesk allow for real-time monitoring and adjustment of simulation parameters. This capability is crucial during the calibration phase, where immediate feedback and iterative adjustments are necessary to fine-tune the model. ControlDesk

enables users to view simulation data, make real-time parameter changes, and inject faults to test the system's resilience and accuracy.

Integration with MATLAB and Simulink

Integrating ConfigurationDesk and ControlDesk with MATLAB and Simulink further enhances the calibration process. This integration allows for the seamless transfer of model parameters and results between the simulation environment and the development tools, facilitating a more efficient and accurate calibration process. Users can leverage MATLAB's analytical capabilities to analyze simulation results and refine model parameters accordingly.

4.2 New Models Structure and Components

Restructuring the initial model led to the creation of two distinct parts: a black-box model and a white-box model. This separation optimizes performance, enhances flexibility, and facilitates efficient development and testing processes. The models are classified into a black-box model, which handles high-speed FPGA-related tasks, and a white-box model, which focuses on processor tasks, emphasizing parameter calibration and adjustment.

4.2.1 Black-Box Model (eDrive-MC)

The black-box model, eDrive-MC, integrates the FPGA model and related high-speed processor components. This model handles real-time control tasks and signal processing, taking full advantage of the FPGA's capabilities. Key components of this model include:

- **Bus2Vector:** Converts bus signals into vector signals for processing within the FPGA.
- **RCP (Rapid Control Prototyping):** Includes several instances that manage various real-time control tasks, such as:

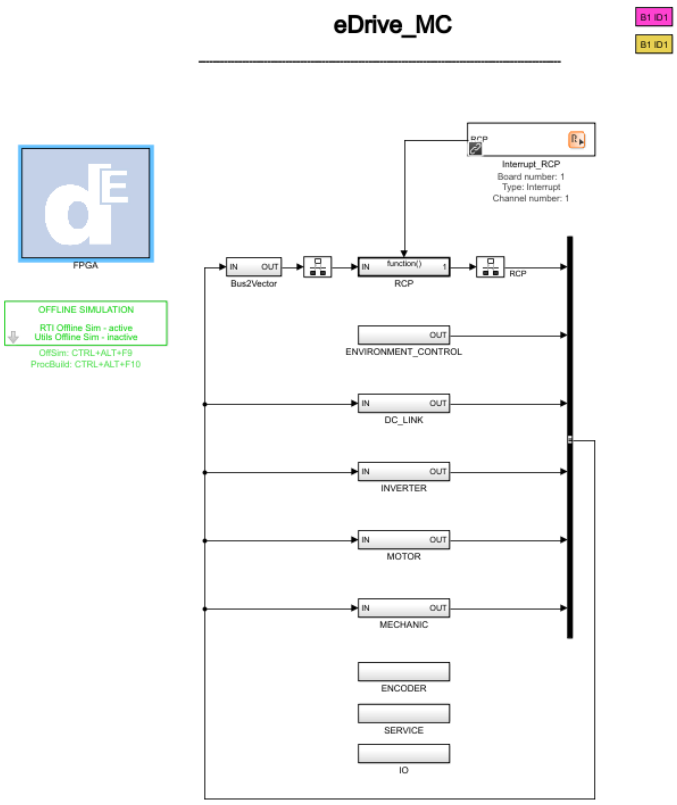


Fig. 4.5 Black-Box Model

- **Environment Control:** Manages environmental parameters affecting the simulation.
- **DC Link:** Manages the link between the power sources and the inverter.
- **Inverter:** Controls the conversion process from DC to AC.
- **Motor:** Manages motor control algorithms.
- **Mechanic:** Simulates mechanical aspects like load and torque.
- **Interrupt Handling:** A dedicated block for managing real-time interrupts, ensuring timely processing of high-priority tasks.

4.2.2 White-Box Model (eDrive-processor-MC)

The white-box model, eDrive-processor-MC, handles processor tasks that do not require the FPGA's high-speed capabilities. This model is designed for tasks requiring more flexibility and accessibility for development and calibration. It includes:

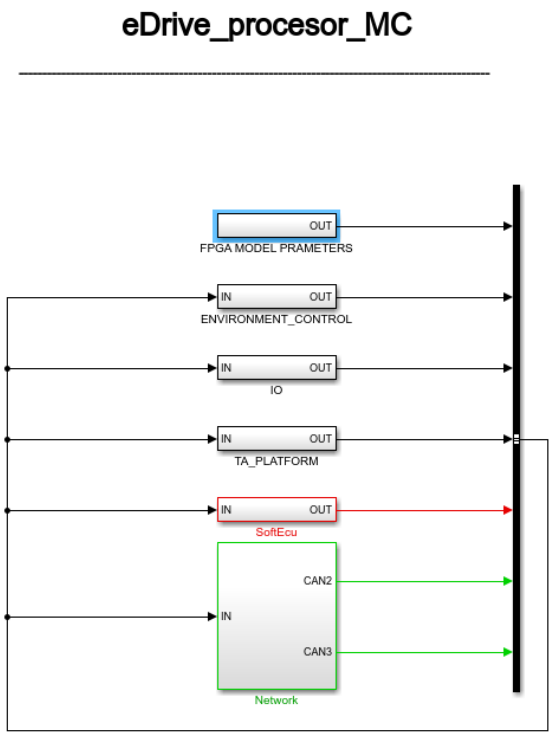


Fig. 4.6 White-Box Model

- **FPGA Model Parameters:** This block outputs parameters the FPGA model requires, ensuring synchronization and data consistency. It allows for calibrating and adjusting input parameters, making the system adaptable to various testing scenarios.
- **Environment Control:** Manages environmental aspects that do not require real-time processing.
- **IO:** Manages general input and output operations.

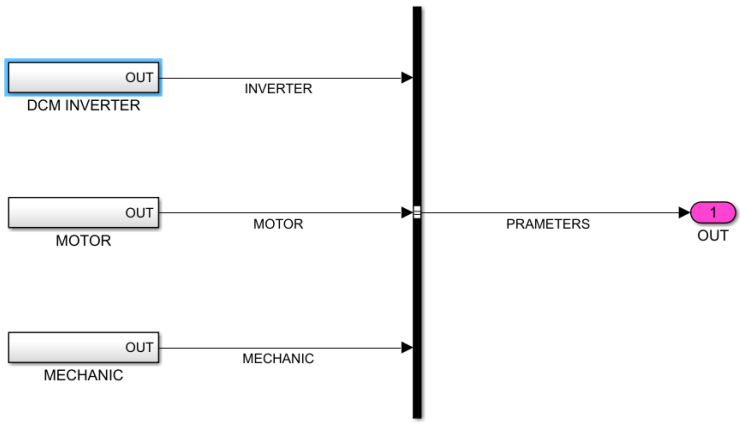


Fig. 4.7 FPGA Model parameter

- **TA-PLATFORM:** Manages platform-specific tasks.
- **SoftECU:** Simulates electronic control units (ECUs) and manages communication with CAN buses (CAN2, CAN3) (Industrial Partner).

FPGA Parameters Model

A unique feature of the new structure is the FPGA PARAMETERS block within the eDrive-processor-MC model. This block consolidates input parameters for the inverter, motor, and mechanic components, ensuring that all relevant parameters are correctly configured and fed into the FPGA model. The components and parameters within this block are:

1. **DCM Inverter:**
    - Const-Inductance-Ls: Inductance (H)
    - Const-Resistance-Rs: Resistance (Ω)
    - Const-Resistance-RonIGBT: IGBT Resistance (Ω)
    - Const-Resistance-RonDiode: Diode Resistance(Ω)
    - Const-VfDiode: Diode Forward Voltage (V)
    - Const-Minimum-Phase-Current: Minimum Phase Current (A)

- Const-Downsampling-of-Average-Unit: Downsampling Unit.

2. **Motor:**

   - Const-Ld-Stator: Stator Inductance (H)
   - Const-Lq-Stator: Stator Inductance (H)
   - Const-NominalCurrent: Nominal Current (A)
   - Const-PolePairs: Number of Pole Pairs
   - Const-R-Stator: Stator Resistance (Ω)
   - Map-Factor-Id-Stator: Mapping Factor
   - Map-Factor-Iq-Stator: Mapping Factor
   - Map-Factor-Psi: Mapping Factor MotorTopology: Motor Topology.

3. **Mechanic:**

   - Const-Damping: Damping Coefficient (Ns/m)
   - Const-Inertia:Moment of Inertia (kg·m²).

This structure allows the white-box model to interact dynamically with the black-box model through parameter adjustments, enabling efficient calibration and real-time simulation tuning. By segregating tasks based on their processing requirements and accessibility needs, the system leverages the FPGA for high-speed tasks while managing less critical tasks within the processor model. This approach optimizes performance, simplifies development and maintenance, and enhances model calibration and testing flexibility.

The new model structure enhances resource utilization, improves real-time performance, and provides greater model calibration and testing flexibility. This modular approach aligns with best practices in system design, ensuring that each component is developed and tested in an environment best suited to its requirements.

4.3 Final Model Implementation

4.3.1 Building the Completed Model in ConfigurationDesk

The following steps outline the process completed to build the final model in ConfigurationDesk:

1. **Creating a ConfigurationDesk Project:** A new ConfigurationDesk project was created from the Simulink Editor. The Simulink model representing the desired FPGA application, which includes components such as the RCP, IO, INVERTER, MOTOR, MECHANIC, ENCODER, SERVICE, and DEBUG blocks, was imported.

2. **Model Integration:** The Simulink model was added to the ConfigurationDesk application. This step involved importing the model into the ConfigurationDesk environment and mapping the ports and signals to the FPGA hardware resources.

3. **Defining Hardware Topology:** The hardware topology, including the FPGA and other essential components, was defined. This topology outlines the physical connections and configurations needed for the application.

4. **Adding Function Blocks:** Necessary function blocks, such as CAN, were added to the working view in ConfigurationDesk. Each function block corresponds to specific hardware resources and communication protocols.

5. **Port Mapping:** The function blocks' configuration ports were manually mapped to their corresponding ports in the Simulink model. This step ensured the signals and data paths were correctly established between the model and the FPGA hardware.

6. **Configuring Function Blocks:** Function blocks were configured based on the application's requirements. Parameters for communication protocols were set, hardware resources were assigned, and signal properties were defined.

7. **Optimizing Configuration:** ConfigurationDesk's optimization tools were used to ensure the configuration is efficient and free of conflicts. The tool provides features for automatic configuration optimization and conflict resolution.

8. **Saving the Configuration:** Once the model was fully configured and optimized, the configuration was saved in ConfigurationDesk. This saved configuration was then used for the next step of building the FPGA application.

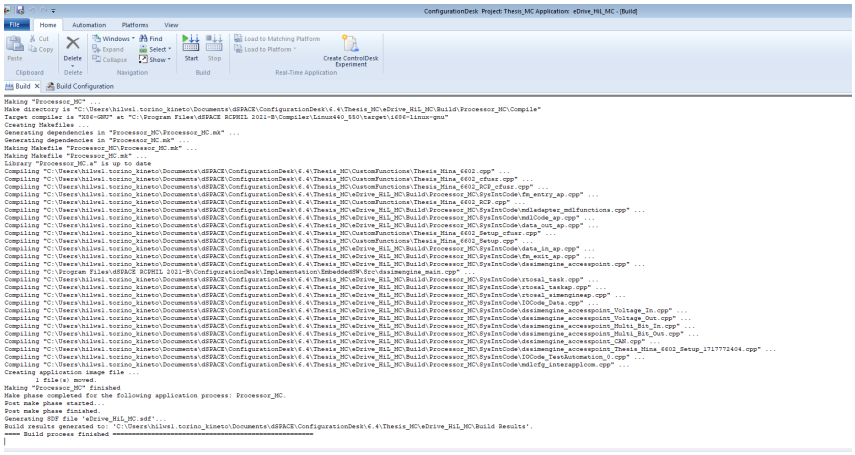


Fig. 4.8 Building the Model

4.3.2 Building the FPGA

After developing the model to run on the FPGA, critical steps must be followed to successfully configure and test the FPGA using the dSPACE ConfigurationDesk. These steps ensure that the model is properly prepared for real-time applications.

Timing Analysis

The timing analysis step is crucial to verify that each block and operation within the implemented model adheres to the set latencies. This test identifies any operations that require more time than allocated, ensuring the model will run efficiently on the FPGA.

Building the Model

The build process translates the Simulink model into HDL code, essential for FPGA implementation. Once the build process is complete, an `.ini` file is generated within a specific folder. This file is necessary to configure the FPGA part within the ConfigurationDesk.

Configuring the FPGA in ConfigurationDesk

Upon completion of the timing analysis and build process, the next step involves configuring the model in ConfigurationDesk. The FPGA setup block in ConfigurationDesk automatically generates a project with all necessary connections for communication between the FPGA and the processor.

The Model Separation Setup block from the dSPACE libraries is utilized to create this project. This block indicates which subsystems will run on the processor, streamlining the configuration process.

Final Configuration

After the initial configuration steps, ConfigurationDesk presents a comprehensive interface displaying the connections and settings for the model. Here, the communication between the models running on the processor and the FPGA can be monitored and adjusted as needed.

Once the configuration is complete, the simulator hardware can be imported, and the new model can be built. This process generates a `.sdf` file is inserted into the simulator to enable real-time application execution.

In summary, the steps outlined ensure that the FPGA is correctly configured and ready for real-time simulation, enabling efficient testing and deployment of the developed models.

Chapter 5

Conclusion and Results

5.1 GUI

This section presented those that were graphical interfaces developed in such a way that you can control the application in real time but above all you can make it as intuitive and accessible as possible

5.1.1 GUI overview

- **ON/OFF Control:** Located in the upper left corner of the interface, the ON/OFF button is crucial for starting or stopping the Device Under Test (DUT). This toggle also enables users to monitor the operational status of the motor, including its speed, which is essential for system testing and adjustments.
- **Dynamic Parameter Monitoring:**
  - **Supply Voltage (V_{dc}):** Displayed in real-time, this parameter is vital for tracking the power supplied to the motor, essential for operational assessments.
  - **Current (I_{abc}) and Voltage (V_{abc}):** These textboxes update dynamically as system conditions change, providing critical insights into the electrical performance of the motor.
  - **Mechanical Speed (ω_m):** This displays the motor's speed, offering insights into its mechanical output during operation.

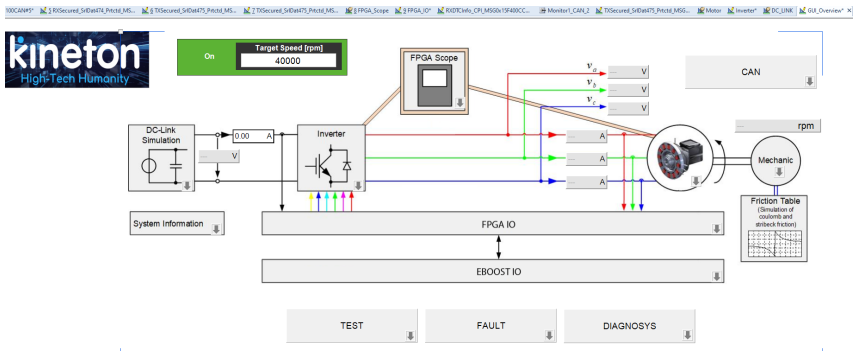


Fig. 5.1 GUI overview

- **Interface Usability:**
  - **Quick Access Links:** The GUI includes easy-to-navigate links to additional layouts and control panels, enhancing the interface's usability without overwhelming the main display.
  - **Layout and Design:** Aimed at maximizing user experience, the GUI is logically arranged and user-friendly, with all controls and indicators easily accessible and clearly labeled.

5.1.2 Controller

The controller interface for the PMSM is detailed here. It includes real-time control over three PI controllers' proportional and integral coefficients. The layout includes familiar elements such as ON/OFF, Reset, and speed control. Graphical outputs provide instant trends of significant control signals.

5.1.3 Electric Model

it represents the Permanent Magnet Synchronous Motor (PMSM) electric model. The user interface includes controls such as an ON/OFF switch, a speed control text box, and links to the general overview and reset options. Users can adjust real-time electrical characteristics such as stator resistance, inductance, and flow on the left side of the interface. The interface also allows for selection between triangle or star connection configurations. Graphs displayed on the right side show trends in phase currents, torque, and rotating frame correction factors.

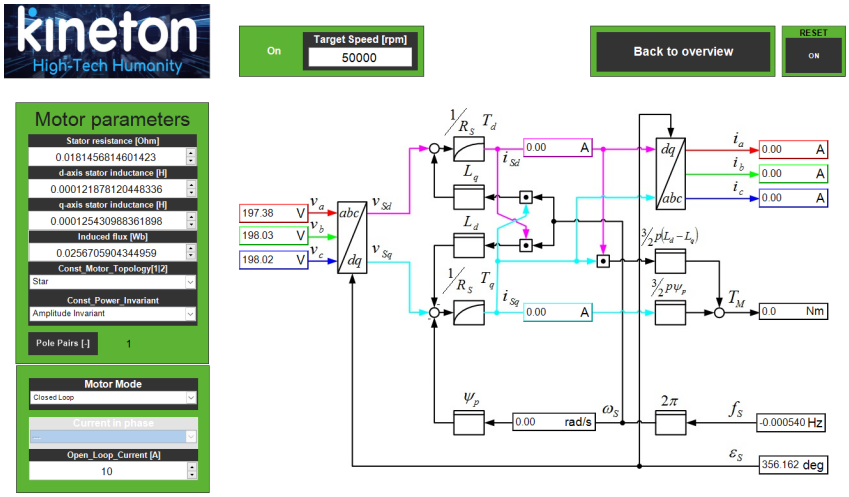


Fig. 5.2 Electric Graphical Interfaces

5.1.4 Mechanic model

This interface gives the complete vision of what is our mechanic model of our Permanent Magnet Synchronous motor (PMSM):

Also in this layout, we can find the same buttons present in the previous interfaces: ON/OFF, Reset, Target Speed and 'Back to Overview'. On the left side, we find textboxes that allow us to change the engine's mechanical parameters, such as the inertia or the damping coefficient, in real-time.

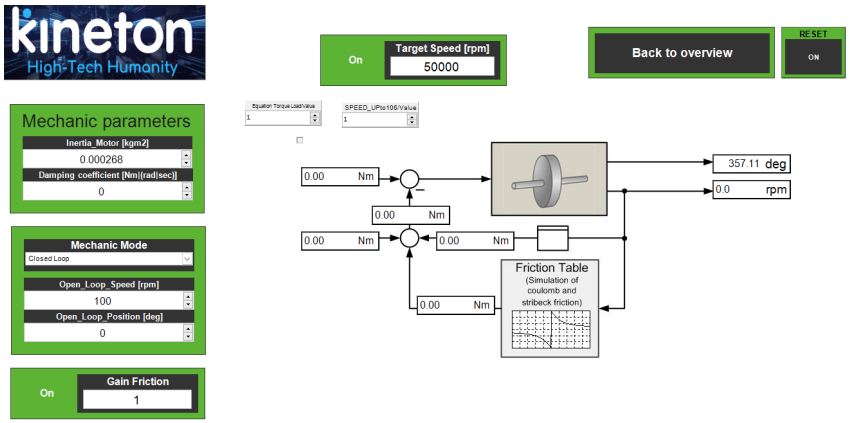


Fig. 5.3 Mechanic Graphical Interfaces

In the center, we can see the change in the model's values from the appropriate display, while on the right side, we find three graphs that represent, from the top, the

engine's speed and the mechanical position. Finally, you can activate gain friction to insert a torque that depends on the speed in the model. However, it has not been implemented in the model.

5.1.5 Inverter

This interface provides a comprehensive view of the inverter. The layout includes familiar buttons from previous interfaces: ON/OFF, Reset, Target Speed, and 'Back

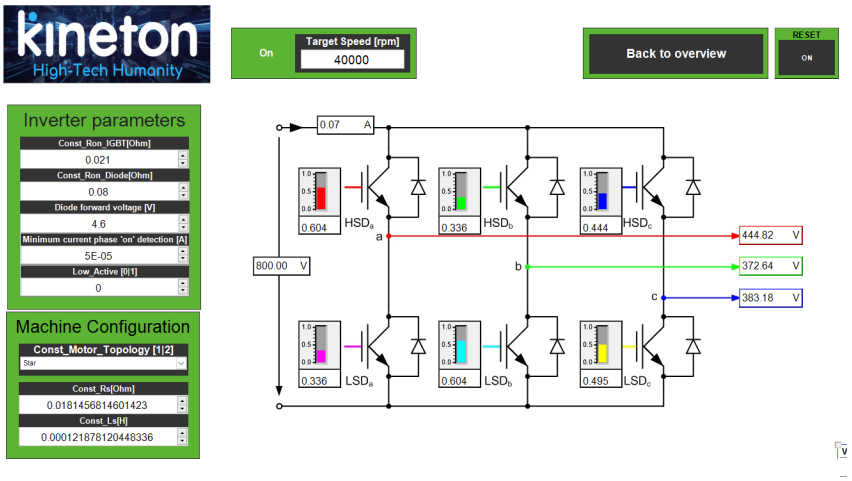


Fig. 5.4 Inverter Graphical Interfaces

to Overview'. Some textboxes on the left side allow real-time modification of the inverter parameters. The inverter diagram clearly displays the PWM trends on each transistor and the phase voltages for every branch. On the right side, graphs display the supply voltage and the duty cycles managing the transistors.

5.1.6 DC Link

This interface focuses on the DC Link component, designed to enable or disable the simulation of a real or ideal power phase. Although not yet functional, the layout is prepared to facilitate this feature, providing a schematic view and corresponding operational parameters for the DC Link.

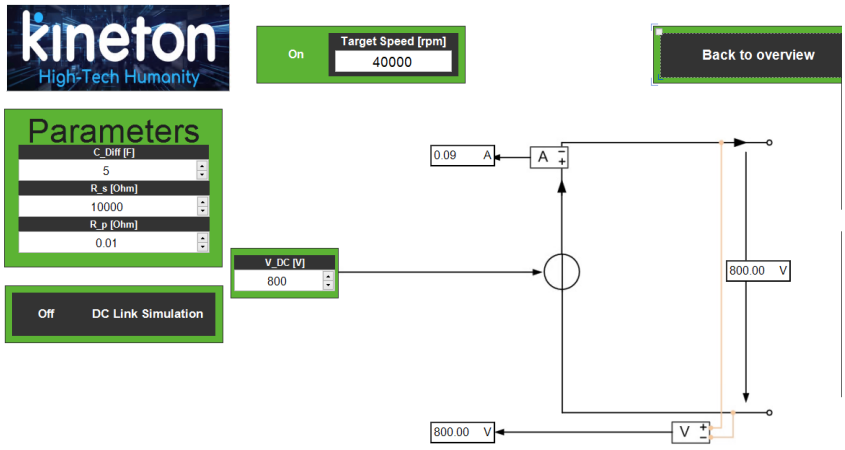


Fig. 5.5 DC Link Graphical Interfaces

5.2 Open-Loop Test

The open-loop test is a critical phase in validating the functionality and performance of the restructured Real-Time Hardware-in-the-Loop (RT HIL) model. This test primarily focuses on assessing the integrated controller's ability to manage and control the simulation environment without feedback from the system under control.

Setup and Methodology

In the open-loop test, the integrated controller developed in Simulink is executed on the RT HIL system's real-time processor. The controller operates in an environment that simulates real-world conditions as closely as possible, incorporating the inverter, electric motor, mechanical model, and bus simulation. The key objective of this phase is to ensure that the controller can handle various operational scenarios accurately and efficiently.

To set up the open-loop test, the following steps were undertaken:

- **Controller Development:** The controller was implemented in the demo Simulink model to ensure seamless integration with the RT HIL system.
- **Environment Simulation:** A comprehensive simulation environment was created, integrating the key components such as the inverter, electric motor, and mechanical model.

- **Execution on Real-Time Processor:** The Simulink-based controller was deployed on the RT HIL system's real-time processor, allowing for the execution of control algorithms in real time.

Test Scenarios

Several test scenarios were defined to evaluate the controller's performance under different conditions. These scenarios included:

- **Start-Up and Shutdown Sequences:** Verify the controller's ability to manage the system's initial and final states.
- **Steady-State Operation:** To assess the controller's performance in maintaining stable operation under nominal conditions.
- **Dynamic Response:** To evaluate the controller's response to changes in input conditions, such as sudden load variations or speed changes.

Results and Analysis

The open-loop test demonstrated that the integrated controller could effectively manage the simulation environment, providing accurate and stable control of the electric motor and other components. Key findings from the test include:

- **Accuracy:** The controller accurately followed the desired input commands, with minimal deviations observed in the output responses.
- **Stability:** The system maintained stable operation throughout the test scenarios, with no significant oscillations or instability.
- **Performance:** The controller demonstrated robust performance, handling dynamic changes efficiently and maintaining desired operational states.

The successful completion of the open-loop test validated the initial implementation of the controller and provided a solid foundation for subsequent closed-loop testing. The insights gained from this phase were crucial in identifying areas for further optimization and ensuring the overall reliability of the RT HIL system.

The open-loop test phase was instrumental in verifying the functionality and performance of the restructured RT HIL model. This phase ensured that the integrated controller could effectively manage the complex simulation environment by simulating real-world conditions and evaluating the controller's response to various scenarios. The positive results from the open-loop test provided confidence in the system's design and paved the way for more comprehensive closed-loop testing.

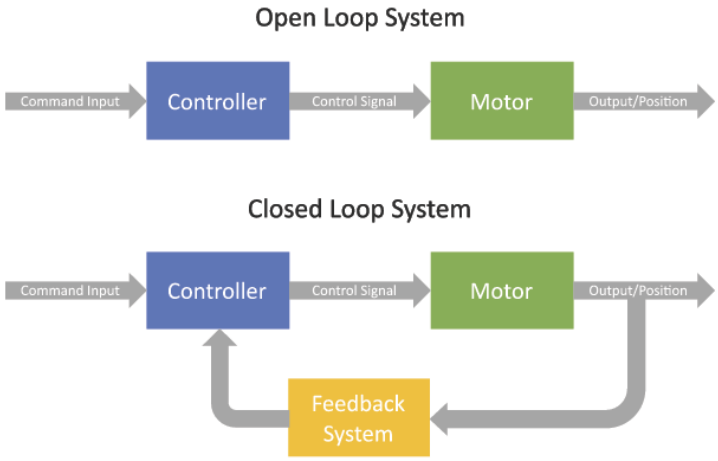


Fig. 5.6 Example of a Figure

5.3 Close-Loop Test

The closed-loop test[9] is the final and critical phase in validating the performance and robustness of the restructured Real-Time Hardware-in-the-Loop (RT HIL) model. This test ensures that the integrated controller can dynamically respond to real-time feedback from the system under control, maintaining optimal performance under various conditions.

Setup and Methodology

The closed-loop test setup involved integrating the RT HIL system with a physical control board designed for a high-speed Permanent Magnet Synchronous Motor (PMSM). This control board, excluding the power stage, was connected to the HIL

system, establishing a real-time interaction between the simulated environment and the actual hardware.

The following steps were undertaken to set up the closed-loop test:

Hardware Integration: The physical control board was connected to the RT HIL system, creating a closed-loop environment where the controller received continuous feedback from the motor. System Calibration: Calibration ensured alignment between the physical and simulated components, allowing accurate data exchange between the control board and the HIL system. Feedback Mechanism: Real-time feedback from the control board, including parameters like speed, torque, and electrical signals, was fed back into the controller to adjust its commands dynamically. Test Scenarios Several scenarios were defined to evaluate the controller's performance under different conditions:

- **Load Variations:** The controller's ability to maintain stability and performance under varying loads was tested.
- **Speed Control:** The accuracy and responsiveness of the controller in achieving and maintaining desired motor speeds were assessed.
- **Fault Injection:** Faults and disturbances were introduced to evaluate the controller's robustness and fault-handling capabilities.

Results and Analysis

The closed-loop test provided comprehensive insights into the controller's performance. Key findings include:

- **Dynamic Adjustment:** The controller effectively adjusted its commands based on real-time feedback, maintaining desired performance levels across various scenarios.
- **Stability Under Load Variations:** The system exhibited stable operation even with significant load variations, demonstrating the controller's efficiency in handling dynamic changes.
- **Accurate Speed Control:** The controller maintained precise motor speed control with minimal deviations from set points.

- **Robust Fault Handling:** The system successfully mitigated the impact of faults and disturbances, maintaining operational stability and reliability.

The closed-loop test phase validated the RT HIL system's capability to operate effectively in a feedback-driven environment. The integrated controller demonstrated robust performance, dynamically adjusting to real-time feedback and maintaining optimal performance across various test scenarios. These successful outcomes confirmed the system's design and readiness for real-world applications, highlighting areas for further optimization and ensuring reliable, high-performance control for complex embedded systems.

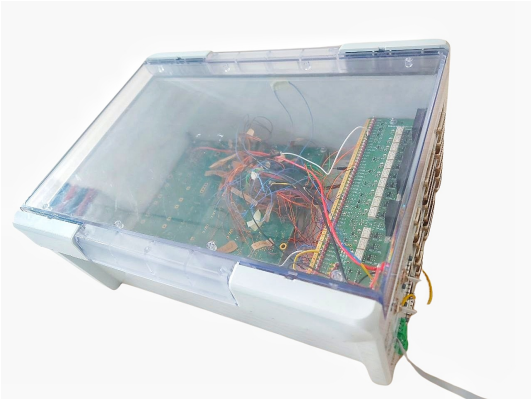


Fig. 5.7 FPGA Model parameter

5.4 Conclusion

The restructuring and implementation of the Real-Time Hardware-in-the-Loop (RT HIL) system in this thesis mark significant advancements in simulation accuracy, control performance, and system robustness. By leveraging a multi-core approach, the RT HIL model now exhibits enhanced computational efficiency and superior data handling capabilities, enabling more complex and high-fidelity simulations.

Key Findings

Graphical User Interface (GUI)

An intuitive and accessible GUI was developed, facilitating real-time control and monitoring of the simulation environment. The GUI enabled dynamic parameter monitoring, quick access links, and a user-friendly layout, significantly enhancing the overall user experience.

Open-Loop Test Validation

The controller reliably tracked desired input commands, displaying minimal deviations in output responses. The system maintained stable operation across various test scenarios, showing no significant oscillations or instability. The controller demonstrated robust performance, efficiently managing dynamic changes and maintaining desired operational states.

Closed-Loop Test Validation

The controller adeptly adjusted commands based on real-time feedback, consistently maintaining desired performance levels across various scenarios. The system demonstrated stable operation even with significant load variations, highlighting the controller's efficiency in handling dynamic changes. The controller achieved precise motor speed control with minimal deviations from set points. The system effectively mitigated the impact of faults and disturbances, maintaining operational stability and reliability.

5.5 Advantages of the New Architecture

The new architecture provides several notable advantages:

- **White Box for Easy Calibration and Model Update**: The white box approach enables straightforward calibration and model updates, significantly simplifying the process of adding or updating CAN/LIN emulation. This ensures the system remains adaptable to new requirements and technologies. The

ease of model updates and calibration enhances system flexibility, reducing the time and effort required for adjustments.

- **CAN/LIN Emulation**: The architecture supports comprehensive CAN/LIN emulation, facilitating the testing and validation of network communication protocols within the simulation environment. This capability is essential for developing and testing automotive and industrial applications where these protocols are extensively used.

- **Adding New Sensors Emulation on RTP**: The system's architecture allows for the addition of new sensor emulations on the Real-Time Processor (RTP), enhancing the simulation's realism and accuracy by incorporating various sensor inputs. This enables more comprehensive testing of control algorithms under different conditions.

- **Dynamic Parameter Adjustability**: The new architecture allows for real-time adjustments to various parameters, such as the damping coefficient and load torque, and the implementation of open-loop control functionalities. This flexibility is critical for fine-tuning the model to match simulated conditions accurately.

- **Task Management and Interrupt Techniques**: Effective task management in multi-core systems optimizes the performance of real-time applications. High-priority tasks can preempt low-priority tasks, ensuring critical operations are completed promptly. This capability is essential for maintaining real-time performance in complex simulations.

- **Integration with System Blocks**: The new model structure enables seamless integration of various system blocks, such as MOTOR, INVERTER, and MECHANIC blocks, ensuring comprehensive input data influences all operational aspects. This integration is crucial for simulating electromechanical dynamics accurately.

- **Model Reusability and Calibration**: Decoupling behavioral models from their I/O configurations enhances model reusability, making the same model adaptable to various hardware setups and project requirements. This approach, combined with tools like ConfigurationDesk and ControlDesk, ensures efficient model reuse and accurate calibration.

- **Parallel Processing Capability**: The architecture leverages the inherent parallel processing capabilities of FPGAs, significantly enhancing the speed and efficiency of simulations. This is particularly beneficial for real-time data processing and high-speed operations.
- **Flexibility and Reconfigurability**: FPGAs can be reprogrammed post-deployment to adapt to new standards or update functionalities. This flexibility ensures the system can evolve over time without requiring significant hardware changes.
- **Rapid Prototyping and Development**: The reconfigurability of FPGAs allows for rapid prototyping and faster iterations during design, reducing development time and enabling quicker transitions from concept to implementation.
- **Long-term Maintenance and Updates**: FPGAs can be updated or reconfigured without physical replacements, extending the lifespan of the systems and ensuring they remain up-to-date with the latest advancements and standards.

5.6 Future Work

The successful implementation and testing of the RT HIL system open several avenues for future research and development: Further research into advanced calibration techniques can optimize system performance, especially in handling complex and dynamic simulation scenarios. Integrating machine learning algorithms can enhance the system's predictive capabilities, enabling more accurate simulations and improved control strategies. Expanding the emulation capabilities to include a broader range of sensors and communication protocols will increase the system's applicability across different domains and use cases.

In conclusion, the restructured RT HIL model, with its advanced features and capabilities, represents a significant step forward in real-time simulation and control. The system's flexibility, robustness, and ease of calibration position it as a powerful tool for future developments in complex embedded systems. The outlined advantages, including easy calibration and model updates, comprehensive CAN/LIN emulation, and the addition of new sensor emulations, underscore the system's enhanced adaptability and effectiveness.

In conclusion, the restructured RT HIL model, with its advanced features and capabilities, represents a significant step forward in real-time simulation and control. The system's flexibility, robustness, and ease of calibration position it as a powerful tool for future developments in complex embedded systems. The outlined advantages, including easy calibration and model updates, comprehensive CAN/LIN emulation, and the addition of new sensor emulations, underscore the system's enhanced adaptability and effectiveness.

References

[1] Franc Mihalič, Mitja Truntič, and Alenka Hren. Hardware-in-the-loop simulations: A historical overview of engineering challenges. *Electronics*, 11(15), 2022.

[2] Enes Durbaba, Eyup Akpinar, Abdül Balıkcı, Buket Turan Azizoğlu, and Ali Kocamis. Fast prototyping of a photovoltaic system by using dsp in matlab simulation loop. 10 2019.

[3] dSPACE GmbH. *ConfigurationDesk Real-Time Implementation Guide*. dSPACE GmbH, Paderborn, Germany, November 2021. For ConfigurationDesk 6.8, Release 2021-B.

[4] MathWorks. Implement motor speed control, 2024.

[5] Fabio Brescia. Development of a behavioral model of a permanent magnet synchronous motor for hardware in the loop implementation. Master's thesis, Università degli Studi di Modena e Reggio Emilia, Modena, Italy, November 2021. In Italian: Sviluppo di un modello comportamentale di un motore sincrono a magneti permanenti per implementazione Hardware in the Loop.

[6] DigiKey. Fundamentals of fpgas: What are fpgas and why are they needed, 2021.

[7] dSPACE GmbH. Xsg electric library documentation, 2021.

[8] dSPACE GmbH. Xsg utils library documentation, 2021.

[9] Electronics Tutorials. Closed loop system, 2023.