



**POLITECNICO**  
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE  
E DELL'INFORMAZIONE



life.augmented

# Real-Time Power Conversion: Development and Validation of a Hardware-in-the-Loop Platform for EV Charging System

TESI DI LAUREA MAGISTRALE IN  
ELECTRICAL ENGINEERING - INGEGNERIA ELETTRICA

Author: **Francesco Covelli**

Student ID: 215032

Advisor: Prof. Luigi Piegari

Co-advisors: Ing. Daniele Caltabiano

Academic Year: 2023-24



# Abstract

This thesis explores the implementation and experimental validation of a Hardware-in-the-Loop (HIL) system for real-time digital power conversion simulation. The primary objective was to develop a HIL environment capable of simulating a high-power system within the constraints of a 5V microcontroller. The system emulates an on-board charger for electric vehicles, converting three-phase AC power to DC. To achieve this, averaged models of power converters were used, and control algorithms were implemented to ensure both stability and performance.

A comprehensive state machine was designed to manage all phases of the converter's operation, including grid synchronization, startup sequences, and fault detection. The results demonstrate that the developed system can emulate the dynamics of a real circuit effectively, providing a reliable tool for testing control algorithms in realistic, real-time conditions. This thesis also details the hardware configurations, validation methodologies, and experimental test scenarios.

**Keywords:** HIL simulation, digital power conversion, microcontroller, real-time, on-board charger, electric vehicles.



# Abstract in lingua italiana

Questa tesi esplora l'implementazione e la validazione sperimentale di un sistema Hardware-in-the-Loop (HIL) per la simulazione in tempo reale della conversione di potenza digitale. L'obiettivo principale è stato lo sviluppo di un ambiente HIL capace di simulare un sistema ad alta potenza all'interno delle limitazioni di un microcontrollore a 5V. Il sistema emula un caricatore di bordo per veicoli elettrici, convertendo potenza trifase da AC a DC. A tal fine, sono stati utilizzati modelli mediati di convertitori di potenza e implementati algoritmi di controllo per garantire stabilità e performance.

È stata progettata una macchina a stati completa per gestire tutte le fasi operative del convertitore, inclusi la sincronizzazione alla rete, le sequenze di avvio e la rilevazione di guasti. I risultati ottenuti dimostrano che il sistema sviluppato è in grado di emulare efficacemente le dinamiche di un circuito reale, rappresentando uno strumento affidabile per testare algoritmi di controllo in condizioni realistiche e in tempo reale. Questa tesi include inoltre una descrizione delle configurazioni hardware, delle metodologie di validazione e degli scenari di test sperimentale.

**Parole chiave:** simulazione HIL, conversione di potenza digitale, microcontrollore, tempo reale, caricatore di bordo, veicoli elettrici.



# Contents

<b>Abstract</b>	<b>i</b>
<b>Abstract in lingua italiana</b>	<b>iii</b>
<b>Contents</b>	<b>v</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Purpose of the Job . . . . .	1
1.2 State of Art . . . . .	2
1.2.1 Hardware in the Loop . . . . .	2
1.2.2 On Board Charger . . . . .	3
1.3 Definition of the problem . . . . .	5
1.4 Proposed Solution . . . . .	5
<b>2 Real Time Simulator</b>	<b>7</b>
2.1 Modeling of the system . . . . .	7
2.1.1 Development Environment . . . . .	8
2.1.2 Hypothesis of the real time simulation . . . . .	10
2.1.3 Validation of the model . . . . .	10
2.2 Discrete model and real time implementation . . . . .	12
2.2.1 PWM input . . . . .	12
2.2.2 Dual core implementation . . . . .	17
2.2.3 Emulate DAC output . . . . .	18
<b>3 Real Time Controller</b>	<b>21</b>
3.1 Modeling of the system . . . . .	21
3.1.1 Development Environment and Real time implementation . . . . .	21
3.1.2 Control algorithm . . . . .	23
3.1.3 State machine . . . . .	29

3.1.4	Validation of the control . . . . .	31
3.2	Real Time implementation . . . . .	31
<b>4</b>	<b>Hardware Description</b>	<b>33</b>
4.1	Control board . . . . .	33
4.1.1	Input filters of ADC's . . . . .	35
4.2	Simulation board . . . . .	35
4.3	Connection between simulation board and control board . . . . .	37
4.4	SR5E1 Adapter . . . . .	40
<b>5</b>	<b>Application of Three Phase Full Bridge Converter</b>	<b>43</b>
5.1	Model of the system . . . . .	43
5.2	Definition of the control . . . . .	45
5.2.1	ST Evaluation board with power mosfet . . . . .	46
5.2.2	IP's configuration . . . . .	47
5.2.3	HF, LF and Main task . . . . .	56
5.2.4	Burst mode . . . . .	60
5.2.5	Data dictionary . . . . .	60
5.3	Simulator of Three phase full bridge . . . . .	61
5.3.1	Average model . . . . .	61
5.3.2	Hardware implementation . . . . .	66
5.3.3	GUI with fault injection . . . . .	66
5.4	Experimental tests . . . . .	68
5.4.1	Desktop Simulation . . . . .	72
5.4.2	Real time simulation . . . . .	78
<b>6</b>	<b>Application of Three Level T-Type Converter</b>	<b>87</b>
6.1	Model of the system . . . . .	87
6.2	Definition of the control . . . . .	88
6.2.1	ST Evaluation board with power mosfet . . . . .	88
6.2.2	Burst mode . . . . .	90
6.2.3	DC side Capacitor Balance . . . . .	91
6.3	Simulator of Three Level T-Type . . . . .	92
6.4	Simulation and real-time tests . . . . .	92
6.4.1	Desktop simulation . . . . .	95
6.4.2	Real time simulation . . . . .	98
6.5	Lab tests with ST power board . . . . .	100
6.5.1	Setup of the lab . . . . .	101

6.5.2	PLL and PWM pattern . . . . .	102
6.5.3	Three phase diode bridge and burst mode . . . . .	103
6.5.4	PFC tests and PI tuning . . . . .	104
<b>7</b>	<b>Conclusions and future developments</b>	<b>107</b>
	<b>Bibliography</b>	<b>109</b>
	<b>List of Figures</b>	<b>111</b>
	<b>List of Tables</b>	<b>115</b>
	<b>Acknowledgements</b>	<b>117</b>



# 1 | Introduction

## 1.1. Purpose of the Job

The purpose of this work is to implement and validate a Hardware-in-the-Loop (HIL) platform capable of simulating devices with power ratings in the order of kW without the need for special equipments. The system to be simulated is based on existing boards from STMicroelectronics, specifically the "STDES-BCBIDIR" [8] and "STDES-PFCBIDIR" [5]. The development of the simulator starts from the characteristics of these boards, which are used for charging electric vehicles (EV) and on-board chargers (OBC). To complete the system, the control of these boards is implemented and tested with the simulator. The control is a power factor correction (PFC). The system can be represented as shown in Fig. 1.1.

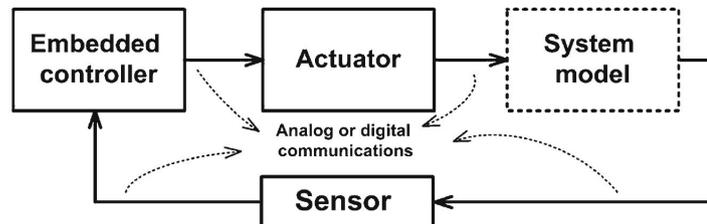


Figure 1.1: Closed loop control system block diagram for an ideal HIL [2].

Fig. 1.1 shows the setup for a Hardware-in-the-Loop test. The embedded controller is the CPU that performs the system control (in this case, the OBC) and communicates with the system through sensors and actuators. The actuators are the PWM signals that the control sends based on the inputs (sensors) and the control algorithm within the embedded controller. The system model can be the real system or the system simulated by a HIL platform that is capable of receiving and sending out signals and internally simulating the system via equations.

## 1.2. State of Art

The thesis explores the simulation of OBC using HIL techniques. For this reason, the state of the art is divided into two parts: one dedicated to HIL and the other dedicated to OBC.

### 1.2.1. Hardware in the Loop

A basic part of any HIL simulation system is a real-time simulation model of the plant which is executed on a real-time system. In an ideal HIL lab, the system is substituted with its simulator and other hardware and software are exactly implemented [2]. It is not always possible to use the hardware, and sometimes other parts of the hardware are replaced with software simulation to reduce costs.

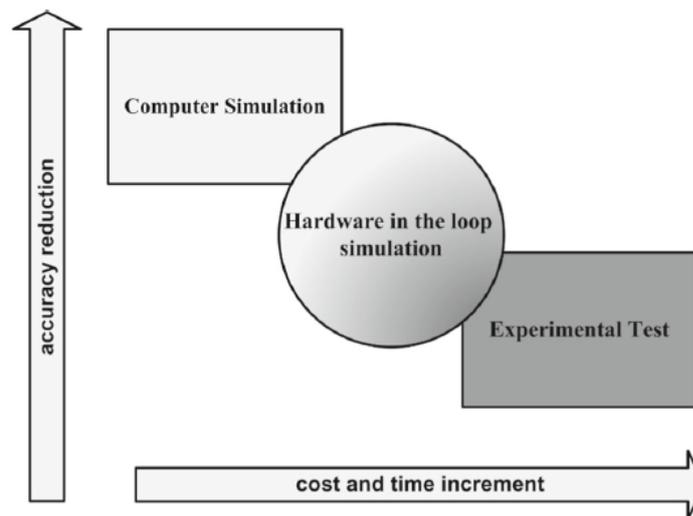


Figure 1.2: Precision, time and cost trade-off in computer simulation, HIL simulation, and actual situation [2].

Fig. 1.2 shows a graph that relates the costs and accuracy from desktop simulation to experimental testing with hardware. HIL is in the middle and allows real-time simulation, reducing costs while permitting the validation of some parts of the hardware (for example, the control) with good accuracy.

The field of HIL simulation for power conversion has advanced significantly in recent years, especially with the development of high-performance HIL platforms based on Field-Programmable Gate Arrays (FPGAs), such as those offered by Speedgoat. These FPGA-based solutions provide very high processing power and ultra-fast real-time capabilities,

making them particularly suited for applications requiring high-frequency switching and extremely precise timing. Additionally, these solutions allow for integration with software like Simulink, further speeding up the modeling process of the simulator. However, FPGA simulators are expensive and require a significant amount of time to transition from the Simulink model to the model ready to be loaded onto the FPGA.

In contrast, the HIL simulation system proposed in this thesis prioritizes portability and cost-effectiveness by using a standard dual-core microcontroller instead of an FPGA. Although microcontroller-based HIL systems have limitations in processing speed compared to FPGA-based solutions, they offer an affordable and flexible alternative that retains sufficient accuracy for testing a wide range of control strategies in power conversion applications.

### 1.2.2. On Board Charger

An OBC is a device integrated into EV that converts AC power from the grid into DC power to charge the vehicle's battery. It manages the charging process, ensuring safe and efficient energy transfer while optimizing battery life and performance. Fig. 1.3 shows main components of an EV.

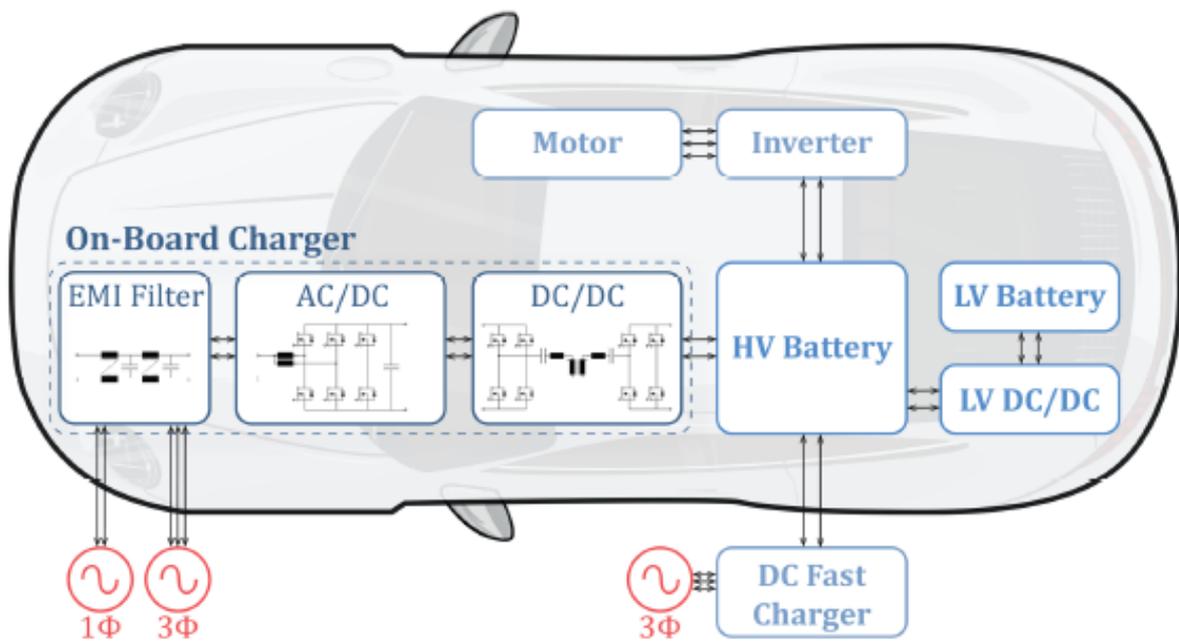


Figure 1.3: EV main components [9].

The core of an EV is the High Voltage (HV) battery, which powers the motor and the auxiliary service battery. The battery can be charged from either an AC or a DC voltage

source. Charging with DC is faster because the power is drawn from the charging station and directly absorbed by the battery. However, this type of charging requires a charging station capable of converting AC to DC.

The second type of charging is through a station with an AC voltage source, which requires onboard conversion of this energy into a DC voltage source: this is where the role of the OBC comes into play.

The OBC is mainly composed by three parts:

- *EMI filter*: by definition, this is a filter for Electromagnetic Compatibility.
- *PFC stage*: transforms the AC source into DC while ensuring a high power factor and a stable output voltage. This is the part analyzed in this thesis.
- *DC-DC stage*: adjusts the DC voltage to the battery level and provides isolation between the battery and the charging station (for example, through a transformer).

Currently, most EV's use a 400 V battery system but in 2019 Porsche has set a new standard and 800 V powertrains have emerged [9]. 800 V allow to have transmit the same amount of power of a 400 V system with half current. On the other hand all system must withstand 800 V and this sets the requirements for the entire high-voltage powertrain, including the OBC, low-voltage dc-dc converter, electric motor, and inverter [9].

In principle, in order to charge the battery, unidirectional OBC are sufficient (power goes from AC to DC) but today bidirectional OBC are implemented since they are more flexible and can provide other services. In particular, bidirectional OBC can provide the following features [9]:

- *Vehicle-to-Load (V2L)*: enables the use of an EV as a grid to connect external electrical devices. This feature applies in various practical cases, such as remote locations like campsites or construction sites.
- *Vehicle-to-Vehicle (V2V)*: enables energy exchange between vehicles in a local grid.
- *Vehicle-to-Home (V2H)*: connects the vehicle to a local home grid or building. Here, it can be interconnected with other assets like solar panels, loads, and stationary energy.
- *Vehicle-to-Grid (V2G)*: Connect the vehicle to the electric grid as a flexible storage unit.

### 1.3. Definition of the problem

With the rise of electric machines and the need to build an adequate infrastructure, combined with the evolution of electronic components, the simulation of power converters of tens of kilowatts has become increasingly necessary, and the controls have become more advanced and complex. Testing such converters (for example, OBC for EVs) requires expensive laboratory instruments and a lab with sufficient installed power. This is where HIL simulation comes into play.

One of the primary challenges in achieving effective HIL simulation in a compact, low-power system is the accurate modeling of power electronics components, such as MOSFETs and capacitors, on a microcontroller with limited processing power and memory. Additionally, to maintain real-time operation at the required frequencies, the system must employ efficient modeling to avoid computational overload. This thesis addresses these issues by implementing an averaged model of the power converter, reducing computational complexity while retaining essential system dynamics. This portable HIL setup must also manage fault conditions and synchronize signals to replicate real-world power system behavior, providing a reliable and valid alternative to traditional hardware-based testing.

### 1.4. Proposed Solution

This thesis proposes an HIL simulation system based on a dual-core microcontroller, emphasizing portability and cost-efficiency without compromising functionality or accuracy. One core is dedicated to generating three-phase signals and handling input filtering, while the other core manages control signals and computes duty cycles. This configuration allows the system to perform complex calculations and maintain accurate timing within the microcontroller's limitations. Moreover, the control algorithm implemented in this setup maintains the DC bus voltage, ensures sinusoidal current absorption, and achieves unity power factor, while incorporating robust protection mechanisms to manage faults effectively.



# 2 | Real Time Simulator

## 2.1. Modeling of the system

The system to be modeled is a power converter that can be used in an OBC of an EV to convert three-phase power from AC to DC and vice versa. The system is represented in Fig. 2.3 and is primarily composed of the following components:

1. **Three-phase AC grid:** Each of the three generators is an ideal voltage source with infinite power, perfectly sinusoidal. The entire grid has only a direct component, and the frequency is fixed.
2. **Relays:** These are necessary to connect or disconnect the grid from the power converter, and additional relays are required to prevent inrush current.
3. **Boost inductor:** These inductors are used to boost the output voltage and also function as filters since power converters inject harmonics into the grid. The inductor is modeled with a resistance and an inductance, as shown in Fig. 2.1.



Figure 2.1: Equivalent circuit of boost inductor.

4. **Power Converter:** It is composed of switches (MOSFETs) and capacitors. There are different topologies depending on how the switches are arranged, and the same applies to capacitors. The switch is modeled as an ideal switch with an anti-parallel diode, as shown in Fig. 2.2.

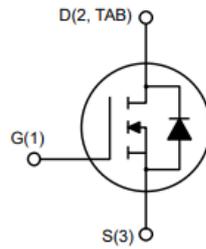


Figure 2.2: MOSFET schematic diagram.

5. **DC Load:** This is a generic DC load that can be a resistor or an ideal current source.

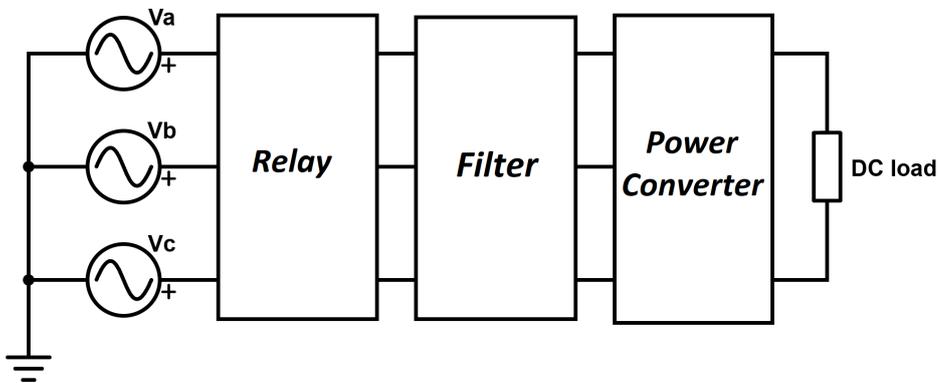


Figure 2.3: Full System.

The goal is to simulate an existing real board, so the data regarding nominal power, MOSFET characteristics, filters, and capacitors will be taken directly from datasheets.

### 2.1.1. Development Environment

The first approach is to simulate the previously explained circuit using Simulink and Simscape Electrical. Fig. 2.4 shows the Simulink model.

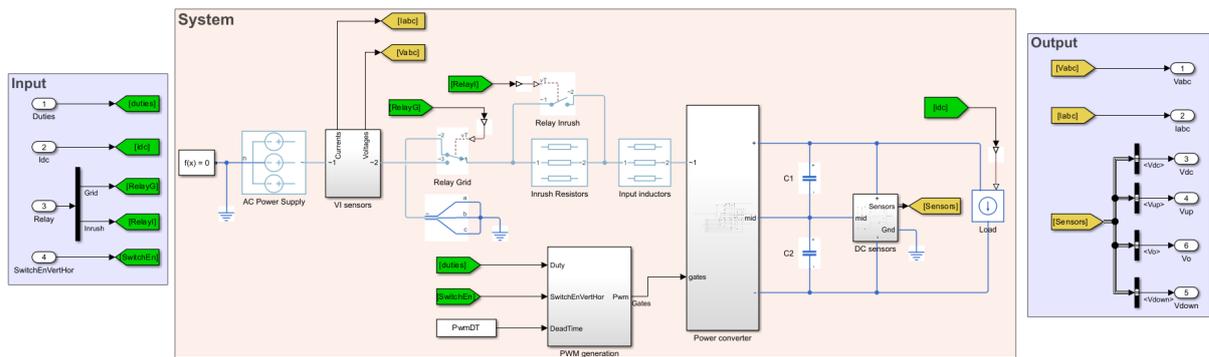


Figure 2.4: Simulink model.

In Fig. 2.4, it is possible to recognize three parts: Input, System, and Outputs. The inputs are parameters that can be sent to the system. They are explained in detail below.

1. **Duties:** Numbers from 0 to 1 that come from the control or an external source. These numbers are then input to the subsystem "PWM generation" to generate gate signals for the switches. After that, the switches are directly controlled by the gate signals without any driver since they are ideal MOSFETs.
2. **Relays:** There are two relays. The first one connects the grid to the converter, while the other short-circuits the inrush resistor.
3. **SwitchEn:** Allows enabling or disabling the switches independently from the gate signals. It can be a single signal or divided into multiple parts to enable only some switches.

The system is the one already explained in Fig. 2.3. The three-phase blocks from Simscape Electrical are used, making the Simulink model clearer. Starting from the left, there is the three-phase AC voltage source, followed by the "VI sensors" block, which is necessary to measure currents and voltages (phase voltages).

The purpose of the Relay Grid and Inrush was already explained; an inrush resistor is necessary to avoid damaging components during the startup phase. In that phase, all capacitors are discharged, and as soon as the Relay Grid closes, the capacitors immediately see a voltage.

Next, there is the input inductor, modeled as in Fig. 2.1. The subsystem "power converter" has a three-phase AC input and a DC output. Furthermore, it has gate signals to control the switches. These signals are created starting from the duty cycle, and it is also possible to add a dead-time.

After that block, there are DC side capacitors. They can be one or more depending on the topology. A midpoint can be present or not, depending on the power converter. The output "sensors" of the "DC sensors" subsystem are measurements that depend on the topology of the converter. The last part is a DC current source on the DC side.

The output area of the Simulink model consists of all measurements taken from the circuit.

### 2.1.2. Hypothesis of the real time simulation

Since the objective is to run the model in real-time on a microcontroller, the previously explained model is not suitable because it requires too much computational effort. Additionally, the sample time must be in the order of  $\mu s$ , which is not suitable for a microcontroller based on a CPU.

The solution for this is to use the so-called **averaged model** of the power converter. In the averaged model, the switching elements (in our case, MOSFETs as shown in Fig. 2.2) are replaced, and their behavior is represented by a number (the duty cycle) that indicates the time the switch is on during a switching period. The duty cycle is defined as:

$$d = \frac{t_{on}}{T_s} \quad (2.1)$$

where  $t_{on}$  is the time when the switch is on and  $T_s$  is the full period of switching. With this assumption, the model of the converter can run at the order of  $ms$  and is compatible with real-time implementation on a microcontroller. The drawback is that it is not possible to appreciate the current ripple caused by the switching of devices.

### 2.1.3. Validation of the model

In order to validate the average model, it is necessary to first use Simulink to test it. To do so, an equivalent system as shown in Fig. 2.4 is used. Equivalent means that the input and output must be the same, so with a variant subsystem, both models can be tested. The average model subsystem is shown in Fig. 2.5.

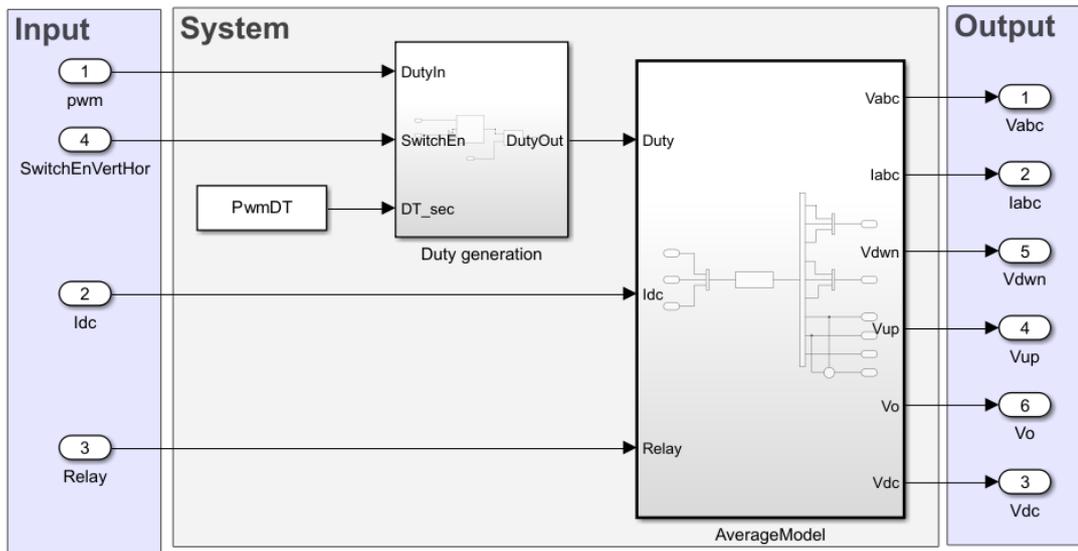


Figure 2.5: Simulink average model.

As mentioned before, the average model works with the duty cycle and not with a PWM signal. Therefore, the "Duty generation" block is necessary only to eventually add a dead time and enable or disable switches. Going into "AverageModel" (Fig. 2.6) allows us to observe the "Interpreted MATLAB Fcn" block, which permits calling a MATLAB function.

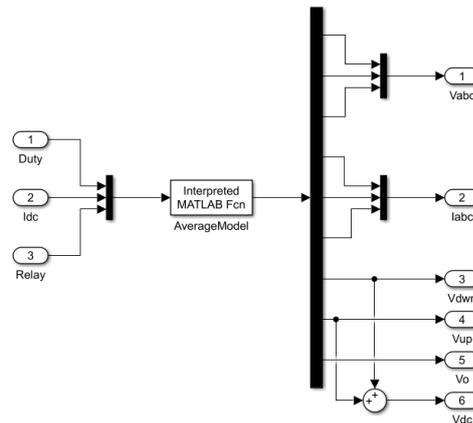


Figure 2.6: Simulink average model - Interpreted function.

In this case, the function contains the average model and manages both relays (grid relay and inrush relay). Outputs depend on the relays (whether they are open or closed) and, of course, on the duty cycles of the switches.

The final validation of the model is done with the full system Simulator + Controller. In

this way, it is feasible to run the Simscape model and the average model and check that the results are compatible (within the limits of the average model).

## 2.2. Discrete model and real time implementation

A real-time implementation requires the discretization of the system. The averaged model consists of differential equations that are already discretized within the MATLAB function. Consider the equation of an inductor:

$$v_L = L \frac{di}{dt} \quad (2.2)$$

In this scenario, the state variable is the current, which is computed at each step of the average model. The term  $dt$  becomes  $\Delta T$ , representing the step size. In other words, it is the time required to perform all the calculations of the average model, thereby determining the model's frequency. The term  $di$  needs to be discretized, and at a specific instant  $t_k$ , the current is:

$$i(t_k) = i(t_{k-1}) + \frac{\Delta T}{L} v_L \quad (2.3)$$

### 2.2.1. PWM input

There is a substantial difference between simulation and real implementation: during simulation, it is straightforward to directly provide duty cycles to the simulated plant (bypassing gate signals), whereas in real implementation, the control sends gate signals to the simulator. For this reason, it is necessary to implement an algorithm (directly in C code) that can receive gate signals and compute the duty cycle required for the average model.

To achieve this, the timer peripherals of the microcontroller are utilized. In particular, there is a function of the timer that can detect edges on the input signal. It is essential that all timers are synchronized and share the same clock (this can be done by adjusting the prescaler of each timer). This synchronization is useful because it provides a unique time reference, ensuring that all edges are captured in the same interrupt. One timer has a periodic interrupt (Application timer), and its trigger output is sent to all other timers (the number of other timers depends on the number of gate signals and the number of channels each timer has).

In any case, every gate signal is characterized by a rising edge, a falling edge, and a period,

as shown in Fig. 2.7.

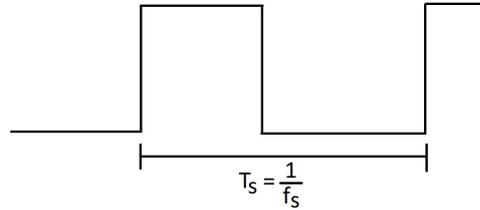


Figure 2.7: Gate signal.

The objective is to calculate the duty cycle as in equation 2.1, so we need the period of the signal and  $t_{on}$ . Since every gate signal has a rising and a falling edge, two channels of the timer are necessary for each signal: one to capture the rising edge and the other to capture the falling edge. Rising and falling values are given in ticks with respect to the timer counter.

For the calculation of the duty cycle, the period in ticks is necessary. The period of the signal is known a priori and is fixed. The number of ticks for the period is calculated as:

$$\text{Tick}_{\text{Period}} = \frac{\text{Frequency of the timer}}{f_s} \quad (2.4)$$

The aim is to have a real-time implementation that runs at a frequency close to the switching frequency. This is related to the definition of the duty cycle (eq. 2.1), which is defined with respect to the switching period. For this reason, the definition of the duty cycle is effectively respected if the sample time is close to the switching period.

One interrupt may not be sufficient to capture all the information necessary to compute the duty cycle. The solution is to configure the application timer with two interrupts: the prescaler of the timer is set in a way that the period of the timer is double with respect to sample time of the average model as detailed in fig. 2.8.

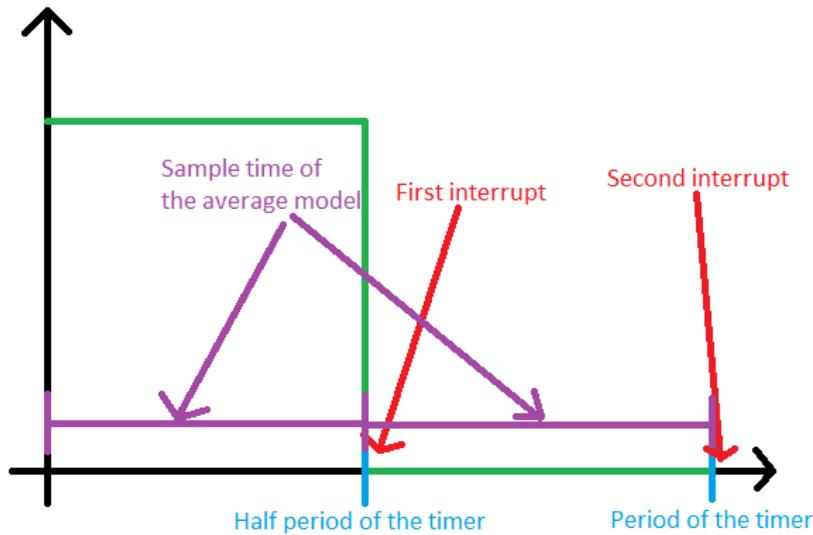


Figure 2.8: Application timer configuration.

The average model is executed at both interrupts but this configuration allows to have a buffer and avoid corrupting the computation in case of interrupt delay.

The entire procedure is divided into two main functions:

- StorePwmInfo:** This function is called immediately after the interrupt and, as the name suggests, stores all rising and falling edges into a global array (one array for rising edges and one array for falling edges). Since there are two interrupts, the array has double the dimensions with respect to the number of gate signals to be captured. When the interrupt is called, information about the interrupt (half or reset) is sent to the function StorePwmInfo so that the value of the rising edge can be stored in the correct position of the array. The same occurs for the falling event. There is the possibility that neither falling nor rising events are present; in this case, the status of the pin is also stored in an array. Rising and falling events are saved into a dedicated register and are read only if the flag register is on.
- ComputePwm:** This function effectively computes the duty cycle once events are stored and is called with the same interrupt as "StorePwmInfo". First, events of the current interrupt are checked (if the interrupt is the half, information about this interrupt is checked), and then, if rising or falling events are missing, events of the other interrupt are checked. The C code for this part is shown in Fig. 2.9.

```

if (Offset == 0) { // first use buffer0 since callback is on buffer0
    UsedRising = StoredRising[num];
    UsedFalling = StoredFalling[num];

    if (UsedRising == 0xFFFF) {
        UsedRising = StoredRising[num+nBuffer];
    }
    if (UsedFalling == 0xFFFF) {
        UsedFalling = StoredFalling[num+nBuffer];
    }
}

} else { // first use buffer1 since callback is on buffer1
    UsedRising = StoredRising[num+nBuffer];
    UsedFalling = StoredFalling[num+nBuffer];

    if (UsedRising == 0xFFFF) {
        UsedRising = StoredRising[num];
    }
    if (UsedFalling == 0xFFFF) {
        UsedFalling = StoredFalling[num];
    }
}
}

```

Figure 2.9: Check events.

The variable "offset" is passed to the function and indicates which of the two interrupts is running. At every interrupt, before saving events, they are set to the hexadecimal number 0xFFFF. This number cannot be read from the register, so if that number is found, it means no events are stored. "nbuffer" is a variable equal to the number of gate signals to be stored.

The second part of the function is dedicated to the computation of the duty cycle, and three possible cases need to be managed:

1. **Both events**
2. **Only one event**
3. **No events**

The part of the code that manages the cases just listed is shown in Fig. 2.10.

```

if ( (UsedRising != 0xFFFF) && (UsedFalling != 0xFFFF) ) { // Both edges are available

    delta = UsedFalling - UsedRising;
    if (delta > TICK_NUM_70) {
        delta -= TIM16_PERIOD;
    } else if (delta < -TICK_NUM_70) {
        delta += TIM16_PERIOD;
    }
    if (delta < 0) {
        delta += TICK_NUM_70;
    }
    pShared_core2[num] = (float)delta / TICK_NUM_70;
} else if (UsedRising != 0xFFFF) { //rising is available

    pShared_core2[num]=prev_duty[num];
} else if (UsedFalling != 0xFFFF) { //falling is available
    pShared_core2[num]=prev_duty[num];
} else {

    // else NO EDGES: 0% or 100%
    pShared_core2[num] = StoredPinStatus[num];
}
prev_duty[num]=pShared_core2[num]; //save duty in case in the next interrupt there is only one event

```

Figure 2.10: Duty cycle computation.

If both events are present, "delta" is used to compute the difference between the falling and rising edges. In an ideal situation, a gate signal is as shown in Fig. 2.7, and it is easy to compute the duty cycle as  $\delta/\text{Tick}_{\text{Period}}$ . However, this is not always true since the interrupt of the simulator is not at the same frequency as the input PWM, and the clock is not synchronized with the clock of the control board. Another difference from the ideal situation is that the event can be captured after the interrupt. An example is shown in Fig. 2.11.

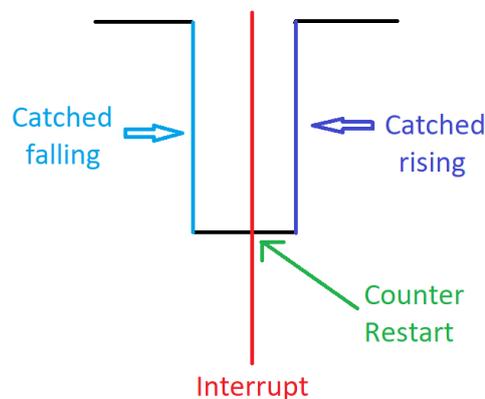


Figure 2.11: Interrupt delay.

In the case shown in Fig. 2.11, the time reference for the two events is different,

and since the timer counter restarts after the interrupt, the variable "delta" will be higher than  $\text{Tick}_{\text{Period}}$ , and the duty cycle would be higher than 1, which is impossible by definition. This is managed in the first "if" statement in Fig. 2.10, where  $\text{TICK\_NUM\_70} = \text{Tick}_{\text{Period}}$  and "TIM16\_PERIOD" is the count of the timer (considering the reset interrupt and not the half one). After managing that, it is checked that "delta" is greater than zero: it can happen that the rising edge is after the falling edge, as shown in Fig. 2.11, leading to a negative duty cycle without this check. "pShared\_core2" is the array where all duty cycles are stored. If only one event is present, the duty cycle computed in the previous interrupt is considered, while if there are no events, the duty cycle is the pin status (so the duty will be 0 or 1).

### 2.2.2. Dual core implementation

The average model, as mentioned earlier, uses differential equations that require significant computational effort for the CPU. Moreover, all variables of the average model are floats, which increases the CPU load.

To overcome these limitations while maintaining a good model frequency, a dual-core implementation is realized.

The first step is to decide which parts of the model will run on each core. Specifically:

1. **Core 1:** Three-phase generation, Input filter, Converter, Capacitor, DC load.
2. **Core 2:** Receive gate signals and compute duty cycles.

Core 1 initializes and runs Core 2, while on Core 2, every timer peripheral is set up. The two cores must communicate with each other, and the duty cycles computed by Core 2 must be passed to Core 1. To achieve this, shared memory (accessible by both cores) is used. Specifically, Core 2 initializes a pointer to the shared memory, and the same is done by Core 1. This part of the memory is also used by the microcontroller for its functions, which can create interference if the same part of the memory is used to pass data from Core 2 to Core 1. For this reason, the available shared memory for the microcontroller is reduced so that this specific part of the memory can be exclusively used to pass our data. Another important aspect is that Core 1 and Core 2 must read and write at the appropriate times. Core 2 computes all duty cycles as explained in Chapter 2.2.1, and when the procedure is completed, it sends an interrupt to Core 1. Fig. 2.12 shows the C code where the PWM computation function is called and an interrupt is sent to Core 1 (SEV\_).

```

void DPC_App1_TIM_Callback_reset(TIM_DRIVER_t *tdp)
{
    (void)tdp;

    USER_LED_0_ON();
    DPC_PWM_INPUTS_Compute(2500);
    USER_LED_0_OFF();

    __SEV(); //send interrupt on core 1
}

void DPC_App1_TIM_Callback_half(TIM_DRIVER_t *tdp)
{
    (void)tdp;
    USER_LED_0_ON();
    DPC_PWM_INPUTS_Compute(0);
    USER_LED_0_OFF();

    __SEV(); //send interrupt on core 1
}

```

Figure 2.12: Core 2 application callbacks.

As mentioned earlier, the application timer has two interrupts (half and reset), which can be seen in Fig. 2.12. Core 1 now executes the average model, three-phase grid, and emulates the DC load.

This ensures that Core 1 effectively takes the latest computation of the duty cycle.

### 2.2.3. Emulate DAC output

A microcontroller works with signals ranging from 0 V to 3.3 V. This means the simulator should emulate the same analog signals that a real power board sends to the controller. A DAC can achieve this, but there are many signals that the simulator needs to send, as shown in Fig. 2.6. The number of signals can be high, and this can be a limitation since microcontrollers usually do not have a large number of DACs.

For this reason, these signals are sent out in a different way to emulate a DAC. Specifically, timer peripherals are used.

The idea is to output a PWM signal from the microcontroller of the simulator and then filter it to obtain the analog signal as if it were sent from a DAC.

To achieve this, the counter is set to a maximum value that matches the DAC's resolution (4095 in the case of a 12-bit DAC). In this way, the timer's comparator can be set directly using the bias and offset as in a DAC. The signal must be as continuous as possible, and while a PWM is not continuous, the higher the PWM frequency, the smaller the error compared to a DAC signal. After the PWM output, an active second-order low-pass filter

with a gain different from 1 is introduced. The complete configuration is shown in Fig. 2.13.

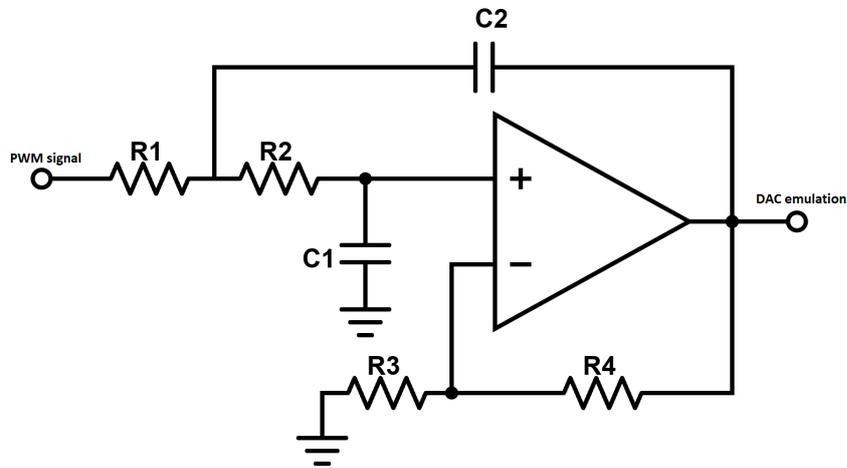


Figure 2.13: Second order low pass filter.

A gain may be necessary depending on the configuration and characteristics of the timer peripheral to fully exploit the 0-3.3 V range.



# 3 | Real Time Controller

## 3.1. Modeling of the system

The purposes of the controller are [3]:

- Control the DC bus voltage, maintaining it at a constant desired value
- Ensure sinusoidal current absorption
- Achieve unity power factor operation

The converter is bidirectional and can either absorb current on the DC side (rectifier mode) or inject current on the AC side (inverter mode).

To implement the control block, the following requirements are necessary [3]:

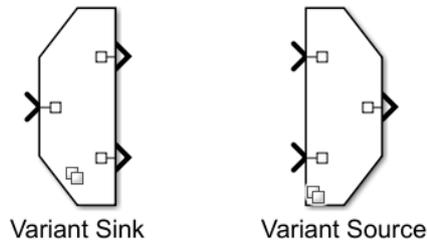
- Measure the AC grid voltage and line current
- Measure the DC side voltage
- Estimate the phase of the AC grid voltage and synchronize the line current phase (Phase Locked Loop)
- Compute and set the desired reference values for the controlled variables
- Generate gate signals for the converter switches (PWM)
- Adjust the error between the reference and the actual values (Regulators)

Additionally, a state machine is implemented to manage all control operations and prevent system damage by checking for faults on both the AC and DC sides.

### 3.1.1. Development Environment and Real time implementation

The development environment used is Simulink, as described in Chapter 2. The control is developed and then tested using the previously implemented simulator. The control is implemented using Simulink *variant sink* and *variant source* blocks, as illustrated in

Figure 3.1.

Figure 3.1: *Variant source* and *variant sink* blocks.

In this way, thanks to model-based design (if the microcontroller is compatible), it is possible to directly generate the C code of the same control algorithm just tested in desktop simulation.

The controller communicates with inputs from the simulator through sensors and actuators, as described in Figure 1.1. The controller subsystem is shown in Figure 3.2.

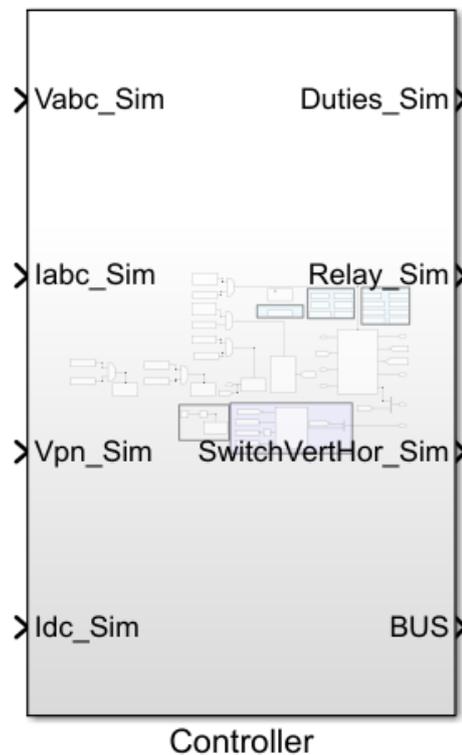


Figure 3.2: Controller subsystem.

The inputs are the same outputs of the simulator and are taken directly from Simulink in the case of a desktop simulation, while inputs are taken from an ADC buffer in the case of C code generation. The same applies to the outputs: in the case of desktop simulation, they go to the simulator, while in the case of C code generation, they go to GPIO and timer peripherals.

The *BUS* is a bus that contains all debug variables coming from the control algorithm.

### 3.1.2. Control algorithm

As mentioned in section 3.1, the controller operates in a d-q reference frame. To achieve this, Park and Clarke transforms are implemented. The Clarke transform allows the transition to an orthogonal reference frame (alpha, beta, zero) as follows:

$$\begin{cases} U_\alpha = \frac{2}{3}u_a - \frac{1}{3}u_b - \frac{1}{3}u_c \\ U_\beta = \frac{\sqrt{3}}{3}(u_b - u_c) \end{cases} \quad (3.1)$$

The zero component does not appear since it is assumed to be a balanced system where the sum of a, b, c is equal to 0.

The Park transform allows the rotation of the reference frame according to a specific angle, which in this case is provided by the Phase Locked Loop (PLL). The Park transform equations are:

$$\begin{cases} U_d = U_\alpha \sin(\omega t) - U_\beta \cos(\omega t) \\ U_q = U_\alpha \cos(\omega t) + U_\beta \sin(\omega t) \end{cases} \quad (3.2)$$

The angle  $\omega t$  depends on the phased-locked-loop (PLL). The purpose of using a PLL is to have a reference frame in which  $V_q = 0$ . To achieve this, a PLL block is implemented with a PI regulator that acts on  $V_q$ . The Park transformation requires the computation of sin and cos of the angle  $\omega t$ , but in the PLL block, these two functions are already computed. Consequently, sin and cos are directly provided for use in the Park transformation, reducing the computational effort for Simulink and especially for the microcontroller. The PLL Simulink block is shown in Fig. 3.3.

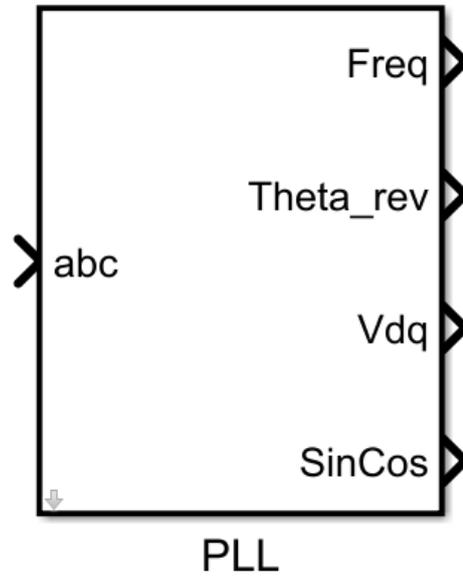


Figure 3.3: Phase Locked Loop.

The "abc" inputs are three-phase AC side voltages. The "freq" and "Theta\_rev" outputs are for debugging purposes (to check if the PLL is working properly), while "Vdq" and "SinCos" are used in the control.

The purpose of the control is already described in 3.1. An equivalent single-phase circuit of the system is represented in Fig. 3.4.

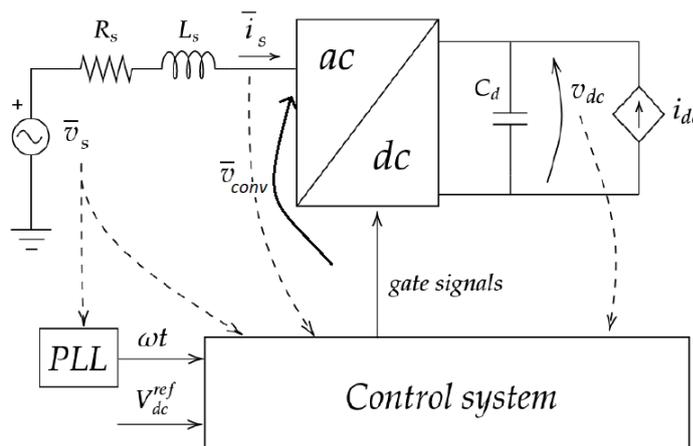


Figure 3.4: Equivalent single-phase circuit [3].

In particular, two control loops are carried out: one loop for DC voltage regulation and the other for unity power factor operation. PI regulators are used to compensate for the

error.

The equivalent single-phase circuit is useful for understanding the dynamics of the circuit in order to tune the PI regulators. The purpose of the inner loop (current loop) is to ensure the absorption of a sinusoidal current in phase with the voltage. The dynamics of that loop depend on the following equation:

$$\bar{v}_{conv} = \bar{v}_s + \left( R_s + L \frac{di}{dt} \right) \bar{i}_s \quad (3.3)$$

Moving to a dq reference frame, the equation becomes:

$$v_{conv-d} = v_{sd} - R_s i_d - L \frac{di_d}{dt} + \omega L i_q \quad (3.4)$$

$$v_{conv-q} = v_{sq} - R_s i_q - L \frac{di_q}{dt} - \omega L i_d \quad (3.5)$$

where the term  $u = R_s i_q - L \frac{di_q}{dt}$  is the output of the PI, and the others are compensating terms. The same happens for both d and q axes. The compensating term is divided into two parts:

- $v_{sd}$  and  $v_{sq}$  are the AC feedforward terms
- $\omega L i_q$  and  $\omega L i_d$  are decoupling terms

The final Simulink block scheme of the inner loop is shown in Fig. 3.2.

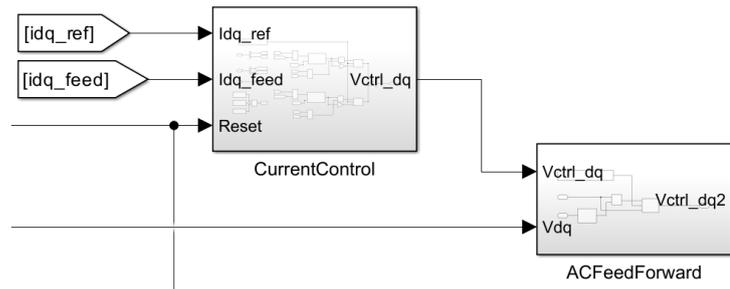


Figure 3.5: Inner control loop.

The "CurrentControl" subsystem is shown in Fig. 3.6, while the "ACFeedforward" subsystem is shown in Fig. 3.7. To guarantee unity power factor,  $i_{qref}$ , which refers to reactive power, is set to 0; nonetheless,  $i_{dref}$  comes from the outer control loop.

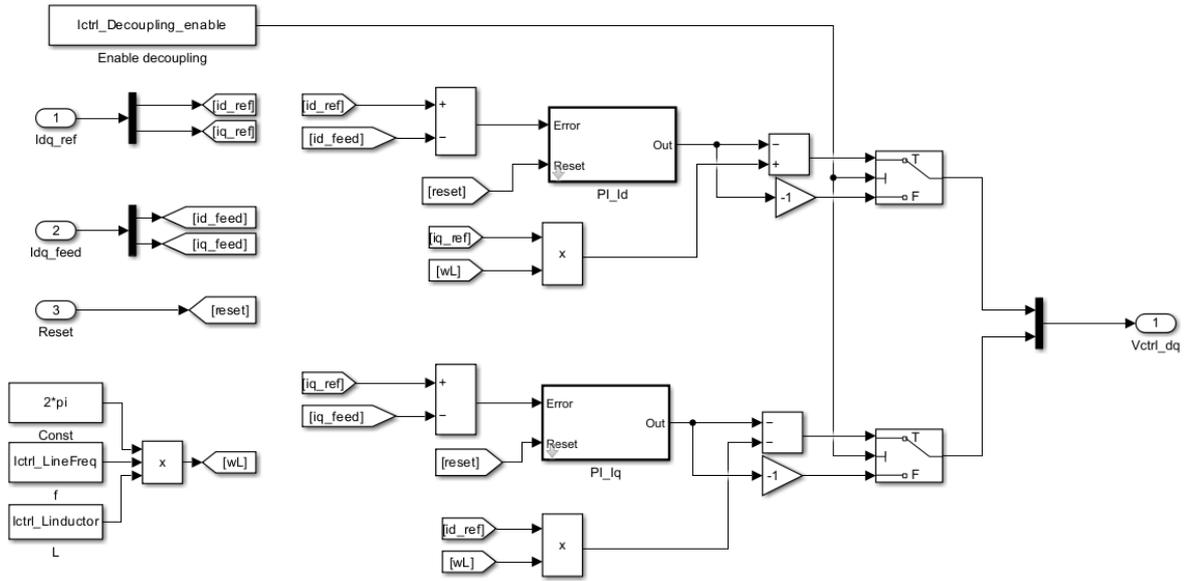


Figure 3.6: CurrentControl subsystem.

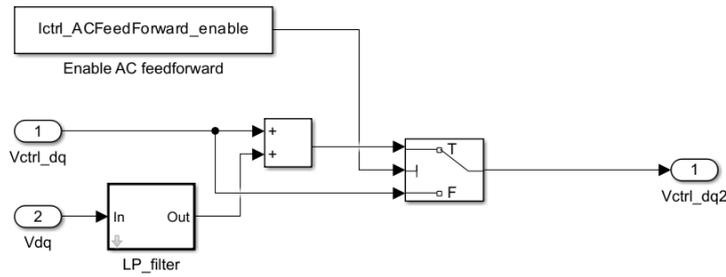


Figure 3.7: ACFeedforward subsystem.

It is possible to observe that these subsystems represent equations 3.4 and 3.5. Additionally, an enable block is added so that feedforward and compensating terms can be disabled.

The PI controller is implemented as shown in Fig. 3.8.

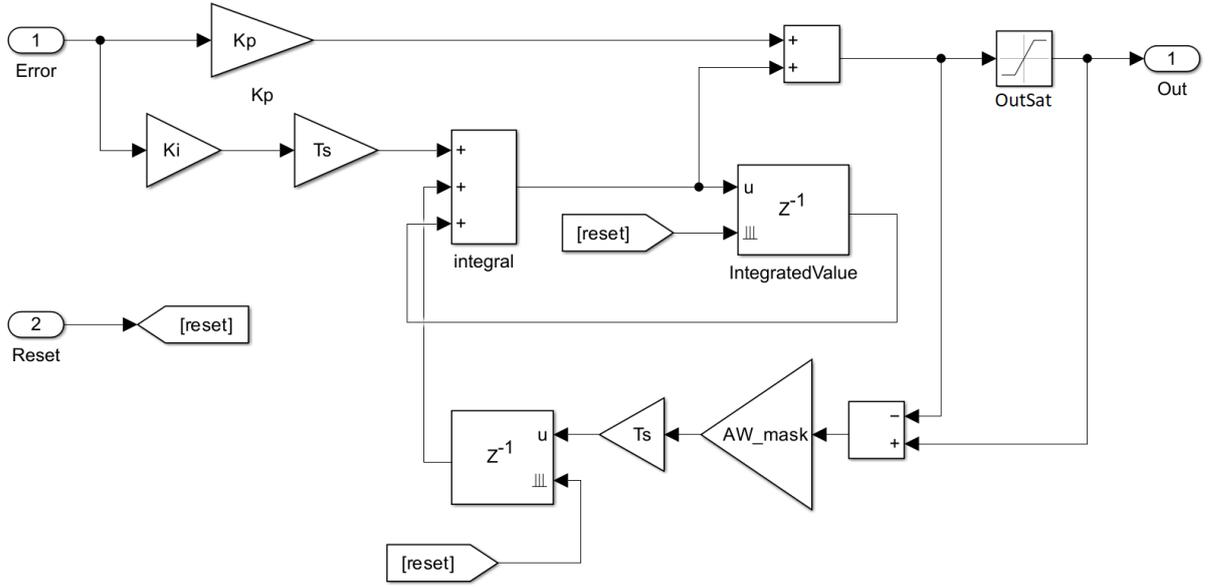


Figure 3.8: PI controller.

$T_s$  is the sample time of the control where the PI is placed, and an anti-windup technique is carried out: based on the difference across the "OutSat" block, the integral is reduced (or increased depending on the sign).

The DC side of the system is now analyzed. The aim of the outer loop is to maintain the DC voltage bus ( $v_{dc}$  in Fig. 3.4) at a given reference value and provide the reference for the d-axis current to the inner loop.

Let us analyze the dc-side capacitor power [3]:

$$p_{cap}(t) = \frac{dE_{cap}}{dt} = \frac{d}{dt} \left( \frac{1}{2} C_d v_{dc}^2(t) \right) = \frac{1}{2} C_d \frac{d}{dt} v_{dc}^2(t) \quad (3.6)$$

In the Laplace domain, assuming  $v_{dc}^2 = x$  [3]:

$$P_{cap}(S) = \frac{1}{2} C_d S X(S) \quad (3.7)$$

Moving to the square of DC voltage, it is found a relation with the power that is linear. As a result, the tuning of the PI is easier and does not depend on the working point. After the PI, a feedforward term for the power is considered: assuming a lossless converter, the DC power is:

$$p_{ac} \approx p_{dc} = v_{dc} \dot{i}_{dc} = p_{cap} + p_{load} \quad (3.8)$$

where  $p_{load} = v_{dc} i_{load}$  is known and  $p_{cap}$  is the output of the PI regulator. Based on the

dq reference frame, the power on the AC side is:

$$\begin{cases} P = \frac{3}{2}v_d i_d \\ Q = \frac{3}{2}v_d i_q \end{cases} \quad (3.9)$$

It has already been pointed out that the q-axis current is equal to 0 in order to have unity power factor, while from the active power equation, the d-axis reference current is found and then sent to the inner loop.

The outer loop implemented in Simulink is shown in Fig. 3.9. The output of the block "id\_ref\_computation" is in p.u. so the inner control loop works in directly in p.u.

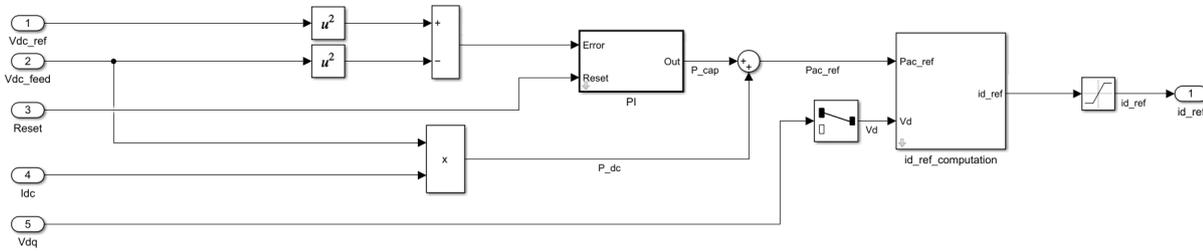


Figure 3.9: Voltage loop.

The last part of the control algorithm is the modulation to control the switches of the converter. Specifically, after computing  $v_d$  and  $v_q$  from equations 3.4 and 3.5, Inverse Park and Clarke transformations are used to pass from dq reference to abc in order to have modulation waveforms (which must be between -1 and 1). From modulation waveforms, duty cycles are obtained that are instead between 0 and 1. Fig. 3.10 represents the Simulink block just described.

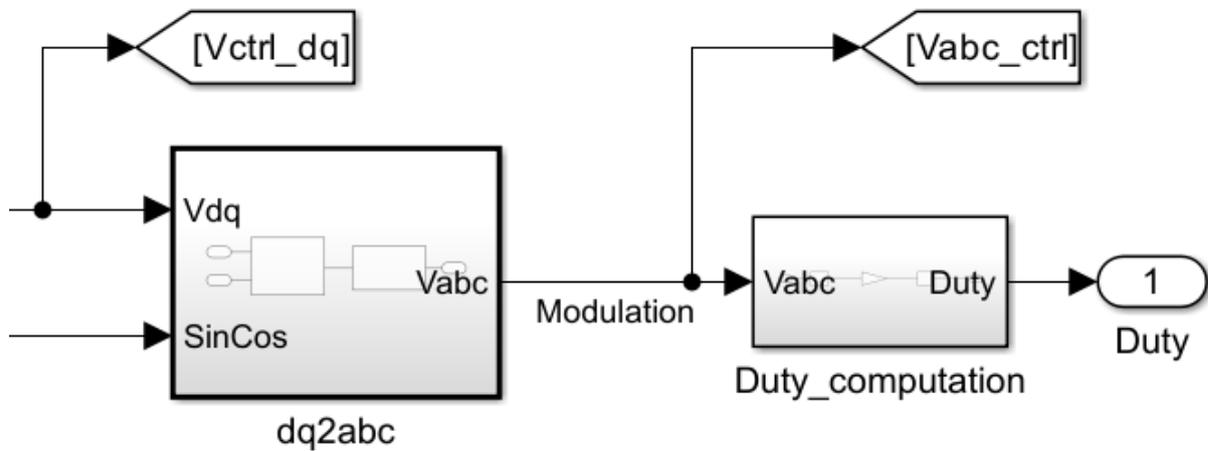


Figure 3.10: Duty computation.

The subsystem "Duty\_computation" depends on the topology of the converter, and also the number of outputs changes depending on how many switches there are in the converter.

### 3.1.3. State machine

The Simulator has elements to be controlled (Relay, SwitchEn). Additionally, the Controller must be able to detect any type of faults and act to prevent damaging components. For these reasons, a State Machine is deployed using Stateflow, which is integrated into Simulink.

The state machine is divided into two main parts:

- **IO FAULTS SM:** Receives all measurements and checks that every measurement is under the maximum limit (Fig. 3.11) and sends a signal to the main state machine.

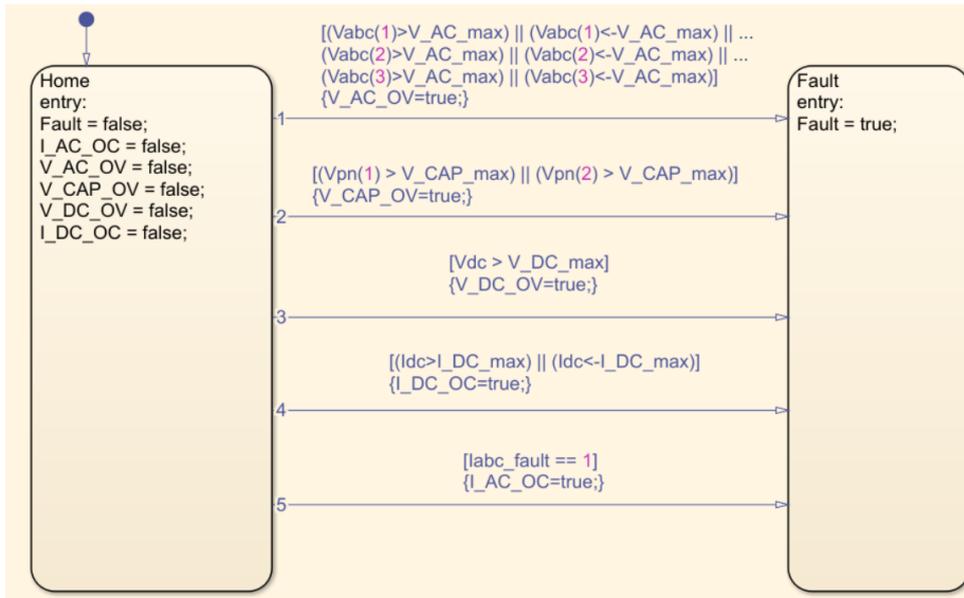


Figure 3.11: IO FAULTS SM.

- **Main State Machine:** Starting from the situation where everything is off, it manages the inrush, the grid relay, activates PWM, and in case of a fault, switches off PWM and opens the grid relay. The states of the state machine are:
  - *Wait:* The state machine is waiting for the AC source, for the PLL to be synchronous with the grid, and for the DC current to be zero.
  - *Idle:* Everything is ready to close, and this state waits for a certain interval before proceeding to the next step.
  - *Init:* The grid relay is closed. An inrush current charges the capacitors, and now the circuit works as a diode rectifier.
  - *Burst (start):* After the voltage has reached a certain limit (comparable with the DC voltage for a diode bridge rectifier) and a certain time has passed, a burst mode is implemented to increment the DC voltage up to the reference value (some switches may be off). The inrush relay can be closed.
  - *PFC:* After reaching a certain voltage, the PFC starts, all switches are turned on, and the control begins.
  - *Fault:* In any part of the machine, if a fault is detected, all relays are opened and all switches are turned off.

### 3.1.4. Validation of the control

The control algorithm is tested in Simulink with the Simscape model of the simulator to ensure that all requirements have been met. After this, the average model is used to verify that the results of the Simulink simulation are comparable with the results obtained using the Simscape model.

## 3.2. Real Time implementation

The real-time implementation requires C code to be built for the microcontroller. This is done directly by Simulink using the Embedded Coder. The Simulink model must meet certain requirements to be compatible with the Embedded Coder: first, all blocks used must have a discrete time implementation, and the solver type must be Fixed-Step with the solver set to "discrete (no continuous states)". The control algorithm is the same for both desktop simulation and the microcontroller. The main difference lies in how inputs and outputs are managed.

Inputs for the controller are the outputs of the simulator and vice versa, as already defined in Chapter 2. Inputs of the control are managed by the ADC peripheral, while outputs are PWM signals managed by the timer peripheral and other GPIOs for the relay.



# 4 | Hardware Description

In this chapter, the hardware used is described. The hardware is divided into the control board and the simulator board. The control board is the one that is also used with the real board, while the simulator board emulates the real board. Therefore, the connectors and the function of each pin on the connector must be the same.

## 4.1. Control board

The control board used is based on an STMicroelectronics Stellar E MCU (SR5E1E7). The SR5E1 family consists of MCUs with two 32-bit ARM Cortex-M7 cores, running at 300 MHz. The two cores can be used either in parallel or with one core in a lockstep configuration. It has 12 timers and 2 High-Resolution Timers (HRTIM) that allow the creation of complex waveforms with a resolution of 102 ps. Specifically, the high-resolution timer can generate up to 12 digital signals with highly accurate timings. It is primarily intended to drive power conversion systems such as switch-mode power supplies or lighting systems, but it can also be used for general purposes whenever very fine timing resolution is required. Its modular architecture allows for the generation of either independent or coupled waveforms. Furthermore, it is capable of handling various fault schemes for safe shutdown purposes [7]. The control board is shown in Fig. 4.1.

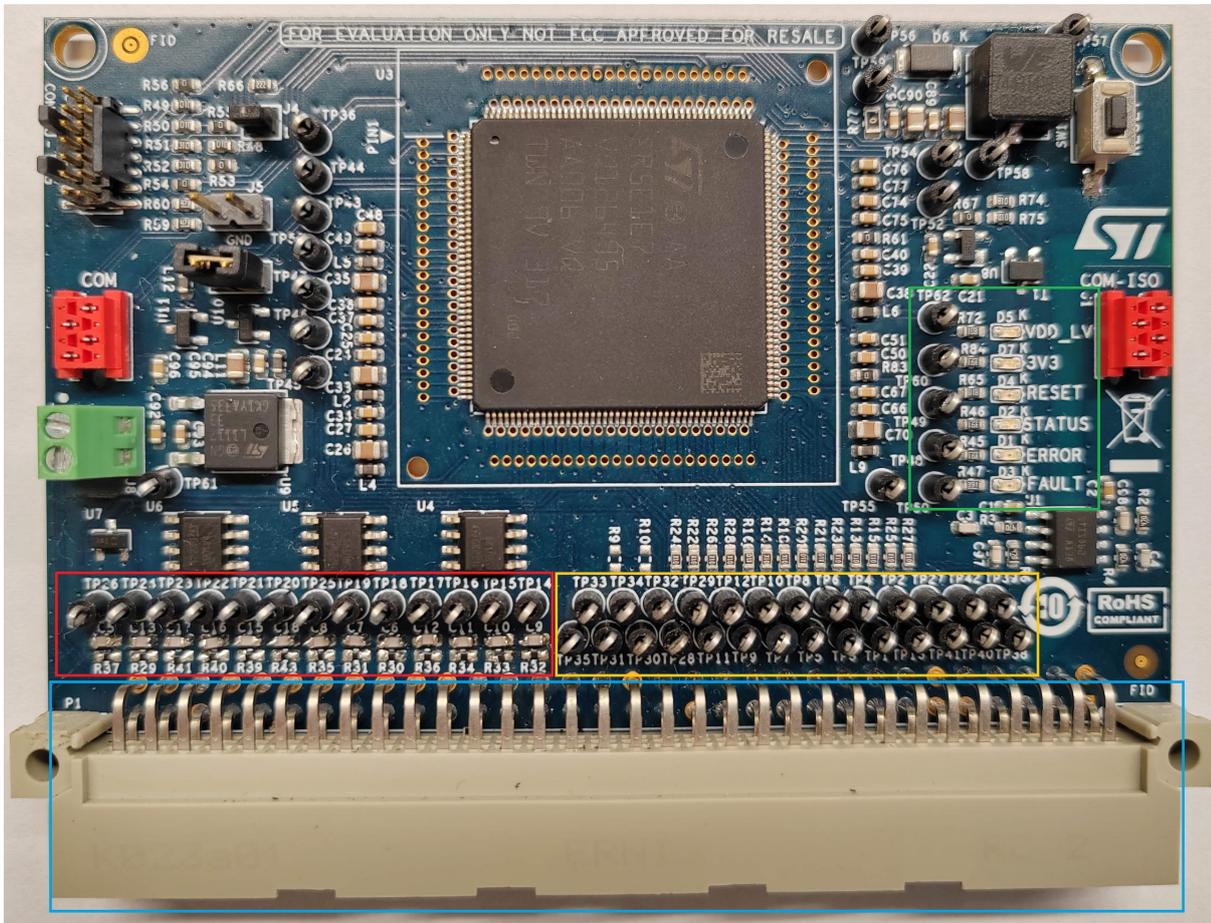


Figure 4.1: SR5E1 Control board.

**The blue rectangle** is the connector to communicate with the simulator board.

**The red rectangle** highlights all ADC inputs; in detail, there is an RC filter and a test point to check the signal before entering the microcontroller.

**The green rectangle** points out some LEDs with their respective test points. LEDs D5 and D7 are useful to check if the power supply is functioning correctly; LED D4 is used when the microcontroller is reset. The other three LEDs below can be programmed by the user.

**The yellow rectangle** shows other test points related to timers, HRTIM, and GPIOs. The board also has 2 DACs that do not have dedicated test points: one is in the red rectangle (where the ADCs are) and the other one is in the yellow rectangle.

The output of the connector must be compatible with both the simulator and the power board: each pin must have the correct function. The connector is divided into two columns (A and B), each with 32 pins. The purpose of each pin is described together with the simulation board since they must be matched.

### 4.1.1. Input filters of ADC's

The ADCs on the connector (Red rectangle in Fig. 4.1) of the microcontroller have an RC filter. The control board is equipped with an RC filter as shown in Fig. 4.2.

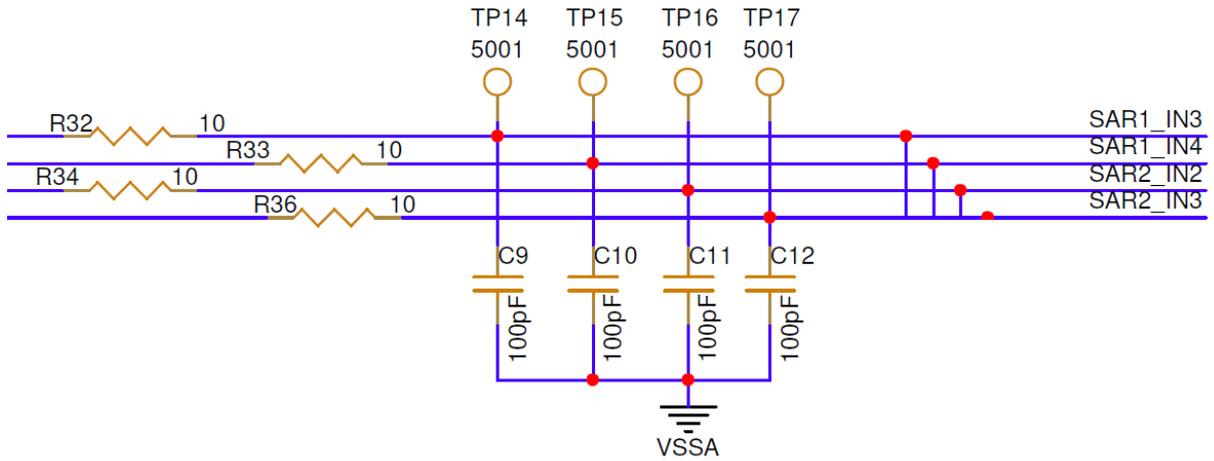


Figure 4.2: Control board input filter.

The cutoff frequency of this filter is:

$$f_c = \frac{1}{2\pi RC} = \frac{1}{2\pi \cdot 10\Omega \cdot 100\text{pF}} = 1.6 \text{ GHz} \quad (4.1)$$

This cutoff frequency does not match the requirements of this application. Therefore, this filter has been modified (by changing R and C) as follows:

$$f_c = \frac{1}{2\pi RC} = \frac{1}{2\pi \cdot 10\text{k}\Omega \cdot 2.2\text{nF}} = 7.2 \text{ kHz} \quad (4.2)$$

## 4.2. Simulation board

The simulator board is also based on an STMicroelectronics Stellar E MCU (SR5E1E7). The simulation board is shown in Fig. 4.3.

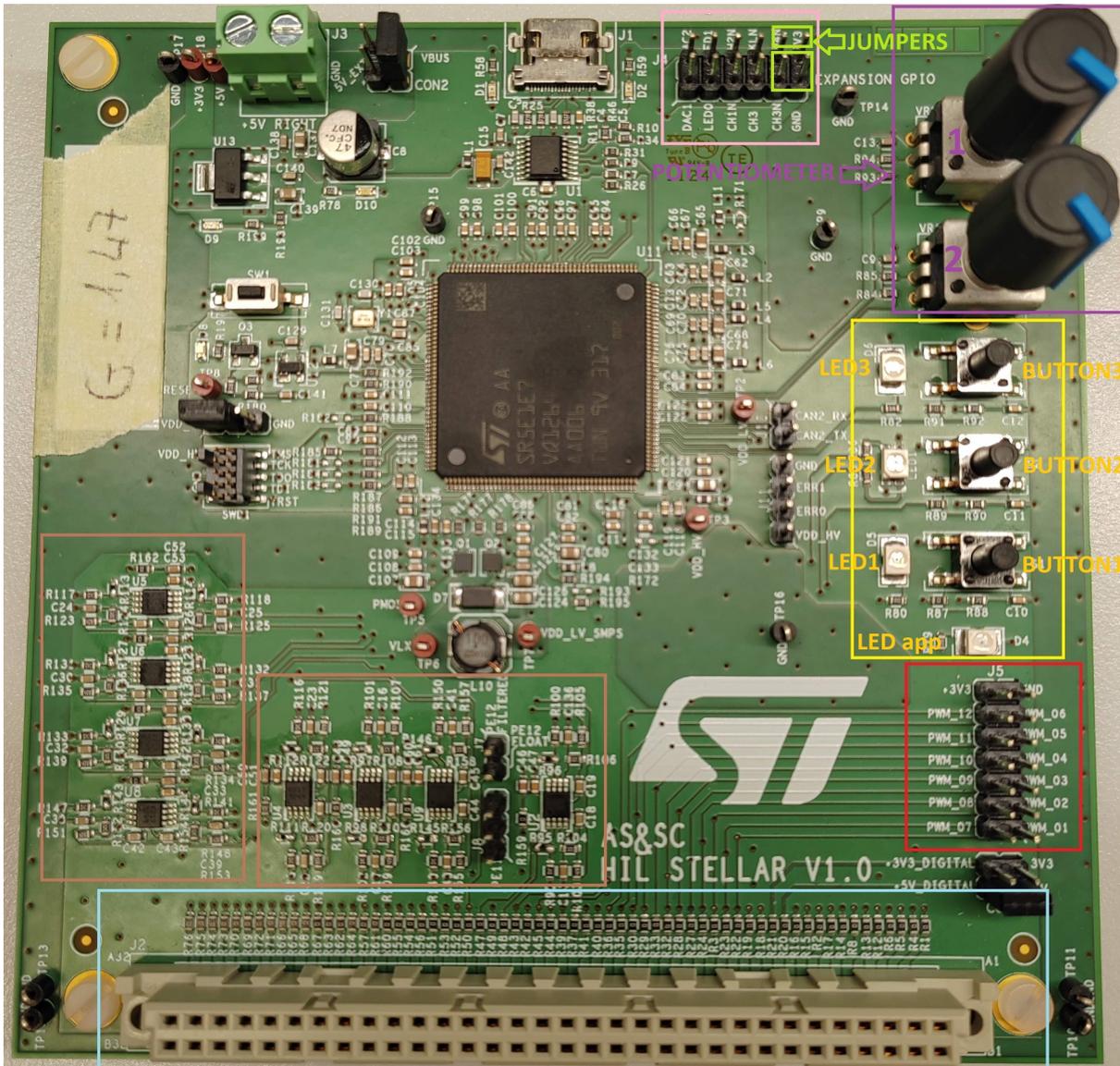


Figure 4.3: Simulation board.

The details of each rectangle are explained below:

- **Blue:** Female connector compatible with the male connector of the control board.
- **Red:** Debug pins for PWM signals that come from the controller. PWM signals go from the HRTIM peripheral of the controller board to the TIM (timer) peripherals of the microcontroller on the simulator board.
- **Purple:** Potentiometers 1 and 2 are used to regulate the DC current (load current).
- **Yellow:** LEDs and buttons are described in detail below.

1. *LED app:* A red LED that flashes every second to indicate that the system is

running and ready.

2. *LED1*: A red LED used to indicate if the grid relay is closed. The command comes from the control board.
  3. *BUTTON 1*: Not used.
  4. *LED 2*: A bicolor LED (red and green). If it is on (either red or green), it means that the inrush relay is closed. As with the grid relay, the command comes from the control board.
  5. *BUTTON 2*: Allows switching between potentiometer 1 and 2. If LED2 is red, potentiometer 2 is used; if LED2 is green, potentiometer 1 is used.
  6. *LED 3*: A red LED that indicates whether the DC load is connected.
  7. *BUTTON 3*: Connects or disconnects the DC load. LED 3 represents its status.
- **Pink**: Debug pins, including two DACs.
  - **Yellow**: In the same section as the debug pins, there are two jumpers that can be used, for example, to decide the polarity of the DC current from the first potentiometer.
  - **Brown**: This section contains opamps for the DAC emulation described in 2.2.3. There are 8 opamps, each with two inputs, so the total number of emulated DACs is 16.

### 4.3. Connection between simulation board and control board

The connection between the two boards is achieved with a 64-pin connector. The function of each pin must be compatible for both boards. The idea is that the simulation board must represent the real system, so the connections are based on that. The purpose of each pin is described in Table 4.1. The column **A function** indicates the general function of the A side of the connector with respect to the simulator board (HIL). This function must be matched with the control board. The column **B function** serves the same purpose. The column **Sim Board** indicates the peripheral of the simulation board that connects to that pin. For example, where the function is "PWM," a timer is configured and sent to that pin to manage PWM inputs.

The column **Control Board** shows the corresponding function for the control board. For

instance, where there is a DAC on the simulator side, there is an ADC on the control side.

An important feature of the HRTIM, relevant to this thesis, is the high-resolution dead-time insertion applied to the same timing unit. In Table 4.1, HR1.A1 means timing unit A output 1, and the dead-time insertion is done on output 2. For this reason, the order of the channels must match the real board. This is not an issue for the simulation board, where PWM inputs can be managed and switched without any problem. However, the real board has a fixed pin mapping for PWMs, so an adapter is implemented to ensure compatibility.

Control Board	Sim Board	A function (HIL)	#	B function (HIL)	Sim Board	Control Board
VDD iso			1			GND iso
USART TX iso			2			USART RX iso
SAR5.1/GPIO			3			SAR5.1/GPIO
CAN TX			4			CAN RX
USART RX			5			SPI MOSI
USART TX			6	GND	GND	GND
+ 5 V	+ 5 V	+ 5 V	7	+ 3 V	+ 3 V	+ 3 V
GPIO			8			GPIO
HR1.A1	TIM1.1	PWM1	9	PWM7	TIM4.1	HR1.A2
HR1.B1	TIM1.3	PWM2	10	PWM8	TIM4.3	HR1.B2
HR1.C1	TIM2.1	PWM3	11	PWM9	TIM5.1	HR1.C2
HR1.D1	TIM2.3	PWM4	12	PWM10	TIM5.3	HR1.D2
HR1.E1	TIM3.1	PWM5	13	PWM11	TIM8.1	HR1.E2
HR1.F1	TIM3.3	PWM6	14	PWM12	TIM8.3	HR1.F2
GPIO	GPIO	GPIO	15	GPIO	GPIO	GPIO
GPIO	GPIO	GPIO	16	GND	GND	GND
GPIO	GPIO	GPIO	17	GPIO	GPIO	GPIO
GPIO/DAC2	GPIO	GPIO	18	GPIO	GPIO	GPIO
GPIO	HR1.C2	DAC01	19	DAC15/GPIO	GPIO	GPIO
SAR1.3	HR1.B2	DAC02	20	DAC16/GPIO	GPIO	VDD
SAR1.4	HR1.A2	DAC03	21	GND	GND	GND
SAR2.2	HR1.C1	DAC04	22	GND	GND	GND
SAR2.3	HR1.A1	DAC05	23	GND	GND	GND
SAR4.4	HR1.B1	DAC06	24	GND	GND	GND
SAR2.7	HR1.E1	DAC07	25	GND	GND	GND
SAR1.1	HR1.D1	DAC08	26	GND	GND	GND
SAR3.3	HR1.E2	DAC09	27	GND	GND	GND
SAR3.2	HR1.D2	DAC10	28	GND	GND	GND
SAR3.1	HR1.F1	DAC11	29	GND	GND	GND
SAR4.5/DAC1	HR1.F2	DAC12	30	GND	GND	GND
SAR1.2	HR2.A1	DAC13	31	GND	GND	GND
SAR2.1	HR2.A2	DAC14	32	GND	GND	GND

Table 4.1: 64-pin connector.

## 4.4. SR5E1 Adapter

The objective of the adapter is to make the control board compatible with the real board. Despite this, it is useful to test the control board with the same configuration that will be used in the final test with the real board.

Starting from the following PCB (Fig. 4.4), a female and a male 64-pin connector are soldered.

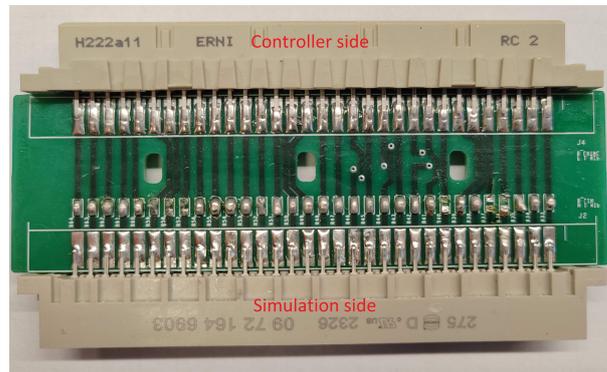


Figure 4.4: SR5E1 adapter.

The PWM configuration at the output of the adapter is now different, as shown in Table 4.2.

Adapter Output	Sim Board	A function (HIL)	#	B function (HIL)	Sim Board	Adapter Output
HR1.B2	TIM1.1	PWM1	9	PWM7	TIM4.1	HR1.A2
HR1.B1	TIM1.3	PWM2	10	PWM8	TIM4.3	HR1.A1
HR1.D2	TIM2.1	PWM3	11	PWM9	TIM5.1	HR1.C2
HR1.D1	TIM2.3	PWM4	12	PWM10	TIM5.3	HR1.C1
HR1.F2	TIM3.1	PWM5	13	PWM11	TIM8.1	HR1.E2
HR1.F1	TIM3.3	PWM6	14	PWM12	TIM8.3	HR1.E1

Table 4.2: New PWM configuration.

Additionally, Fig. 4.4 shows that the adapter needs resistors to connect control board signals with the simulation board. This permits the use of some test points for debug pins without connecting them to the control board. As already mentioned, the DACs do not have dedicated test points, but their pins go to the 64-pin connector. The adapter

permits the crossing of outputs where DACs are present. Remaining pins are connected to the simulation side connector with a  $0\ \Omega$  resistor.

Finally, the control board has dedicated test points for DACs, debug pins, and is compatible with the real board. Fig. 4.5 represents the control board with the adapter.

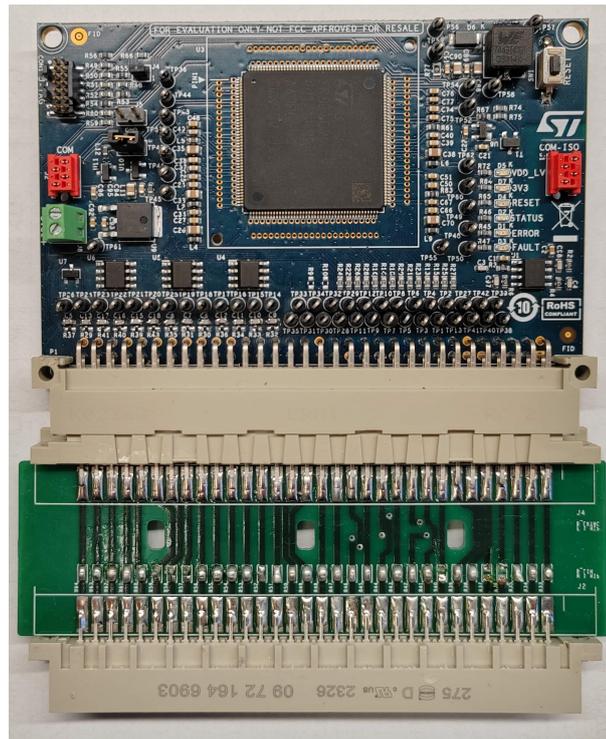


Figure 4.5: Final control board.

The final hardware setup with both control and simulator board is in Fig. 4.6.



Figure 4.6: Final hardware setup.

# 5 | Application of Three Phase Full Bridge Converter

## 5.1. Model of the system

The three-phase full bridge is a bidirectional two-level converter consisting of 6 MOS-FETs. The detailed model of the converter is implemented in Simulink (with Simscape Electrical) and shown in Fig. 5.1.

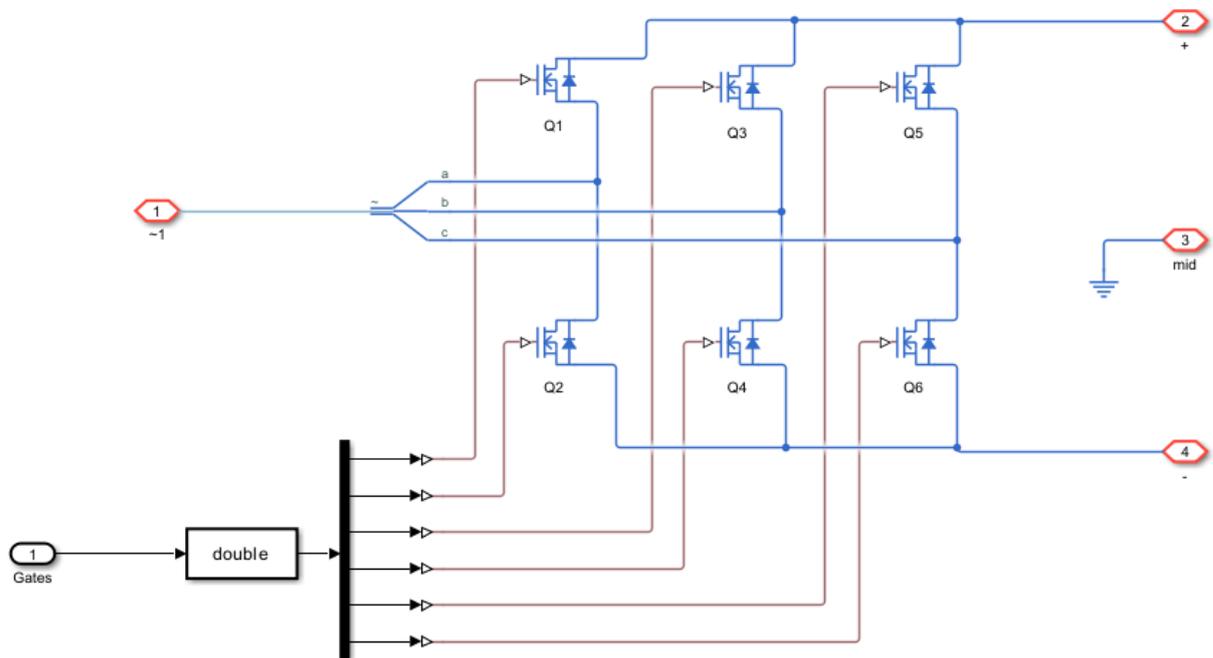


Figure 5.1: Three-phase full bridge Simulink model.

Input 1 of this subsystem is the AC side, while outputs 2 and 3 are the positive and negative sides of the DC side, respectively. The midpoint is not used in this configuration. Considering one leg of the converter (Q1 and Q2), it is important to note that these two switches cannot be closed simultaneously; otherwise, the leg would be short-circuited.

Specifically, when the upper switch is on, the lower one must be off, and vice versa. The complete Simulink model to manage both the control and the plant is shown in Fig. 5.2.

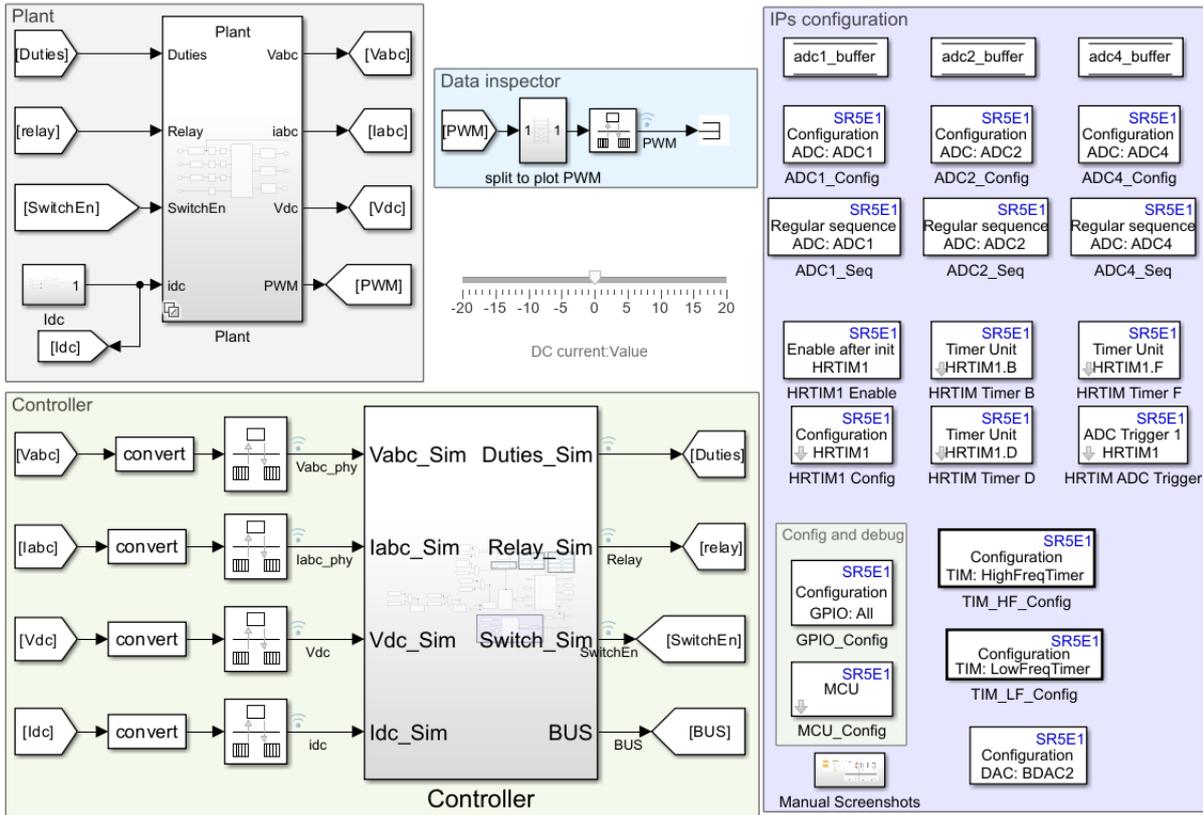


Figure 5.2: Full Simulink model of the three-phase full bridge.

The model is divided into three main parts:

- **Plant:** Contains the model of the plant (simulator), either the Simscape model or the average model. Inputs and outputs are already described in Chapter 2. The plant is a variant subsystem that automatically changes to a "Fake plant" when the Embedded Coder is used. Opening the subsystem reveals the following (Fig. 5.3).

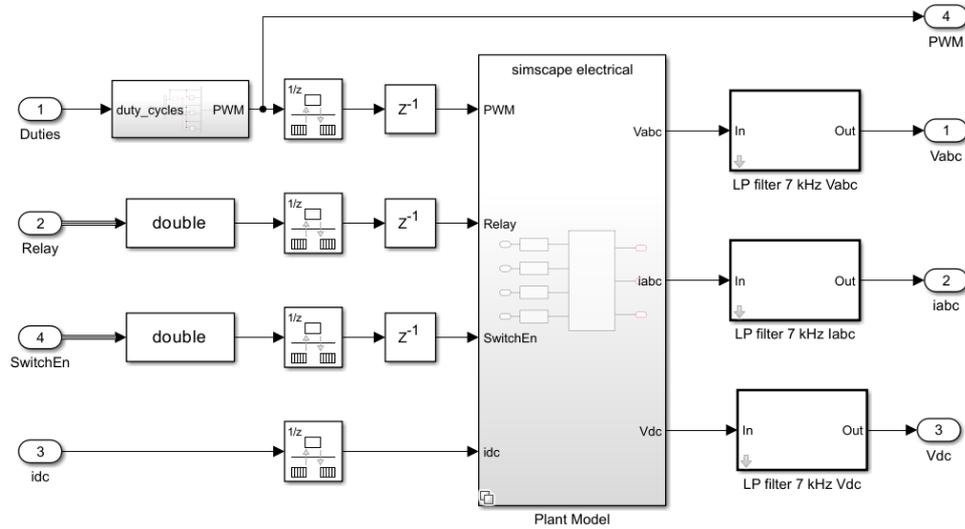


Figure 5.3: Plant subsystem.

On the output side, the first-order RC filters present on the control board are simulated to make the simulation as similar as possible to the hardware setup. On the input side, the duty cycles are received, which in this case are 3 (upper switches), while the other 3 duty cycles are defined as  $duty_{lower} = 1 - duty_{upper}$  (this is valid for each leg). This part is handled directly by the HRTIM peripheral when using the microcontroller, which is why this block is on the plant side. The "Plant model" is a variant subsystem and internally contains either the Simscape model or the average model, as explained in Fig. 2.4 and Fig. 2.5. In this specific case, the only DC voltage to be sensed is the total voltage across the DC bus since the midpoint is not present.

- **Controller:** Manages the control of the converter. Inputs and outputs are the same as for the plant.
- **IP's configuration:** Contains blocks to configure all peripherals of the control board. These blocks are specific to the ST Stellar E microcontroller. The peripherals used for control are: ADC, DAC, HRTIM, TIMER.

## 5.2. Definition of the control

The principles of the control have already been defined. Now let's delve into the details of the control application for the three-phase full bridge.

### 5.2.1. ST Evaluation board with power mosfet

The control board should work with the simulation board as if it were the real board: the pin mapping must be taken directly from the real board, which is not yet available. The real board is from STMicroelectronics, model "STDES-BCBIDIR". According to the ST website: "STDES-BCBIDIR is an 11 kW bidirectional battery charger based on a three-phase two-level PFC and isolated DC-DC converter" [8]. In this specific case, the control should manage the first stage (PFC) of the system. The details of the 64-pin connector for the PFC stage are taken from the schematic (Fig. 5.4).

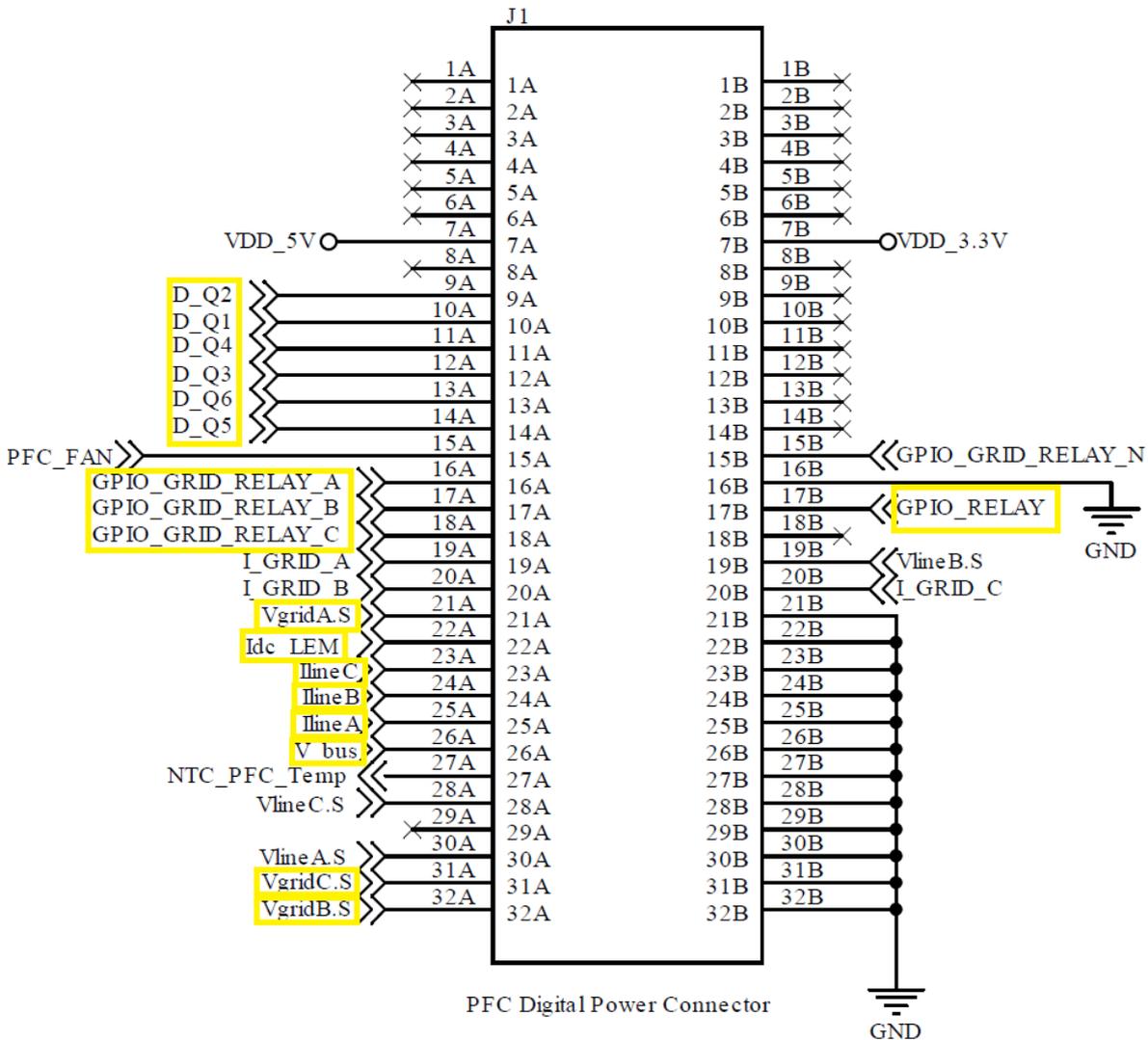


Figure 5.4: PFC connector of STDES-BCBIDIR.

The pins of interest for the control are highlighted in yellow.

From 9A to 14A, there are *PWM signals*. The names of the switches are the same as in Fig. 5.1 and are compatible with the "adapter output" column in Table 4.2. Some GPIOs are necessary to control the *Grid relay* (one for each phase), while 17B manages the *inrush relay* (one signal controls three inrush relays). The last important part is the sensing of current and voltage: *Vgrid* is the phase-to-neutral voltage of the grid (before the grid relay), while *Iline* is the current measured after the inrush relay and before the input inductor. Looking at the "control board" column on the left side of Table 4.1, it is observed that every sensing pin has an ADC channel as expected.

### 5.2.2. IP's configuration

Peripherals are directly configured in Simulink using dedicated Simulink blocks. Starting from the top of the "IP's configuration" area in Fig. 5.2, all blocks are analyzed in detail.

- **ADC:** A buffer for each ADC is necessary. In this case, three ADCs are needed, so three buffers are used (data stores from Simulink blocks). The pin mapping of the ADCs is shown in Table 4.1. The first block is the configuration of the ADC (Fig. 5.5).

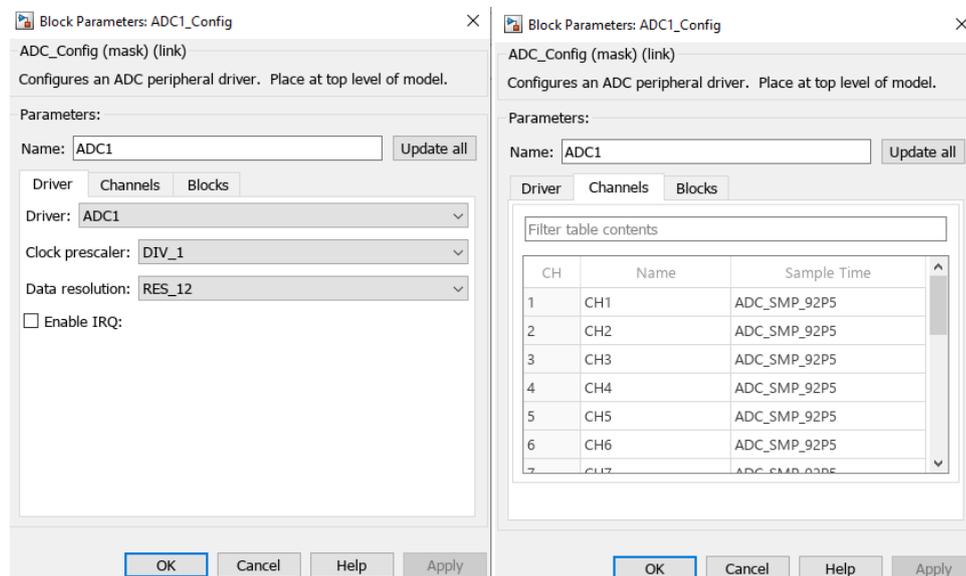


Figure 5.5: ADC driver configuration.

From this block, it is possible to enable the interrupt for the ADC. This feature is used for the analog watchdog, which allows the application to detect if the input voltage goes outside the user-defined high or low thresholds [7]. This will be used in the control for AC overcurrent protection, so the "enable irq" flag is selected for

ADCs 2 and 3 (where  $I_{abc}$  sensing occurs). The conversion of the ADC is a regular one, and the configuration is shown in Fig. 5.6.

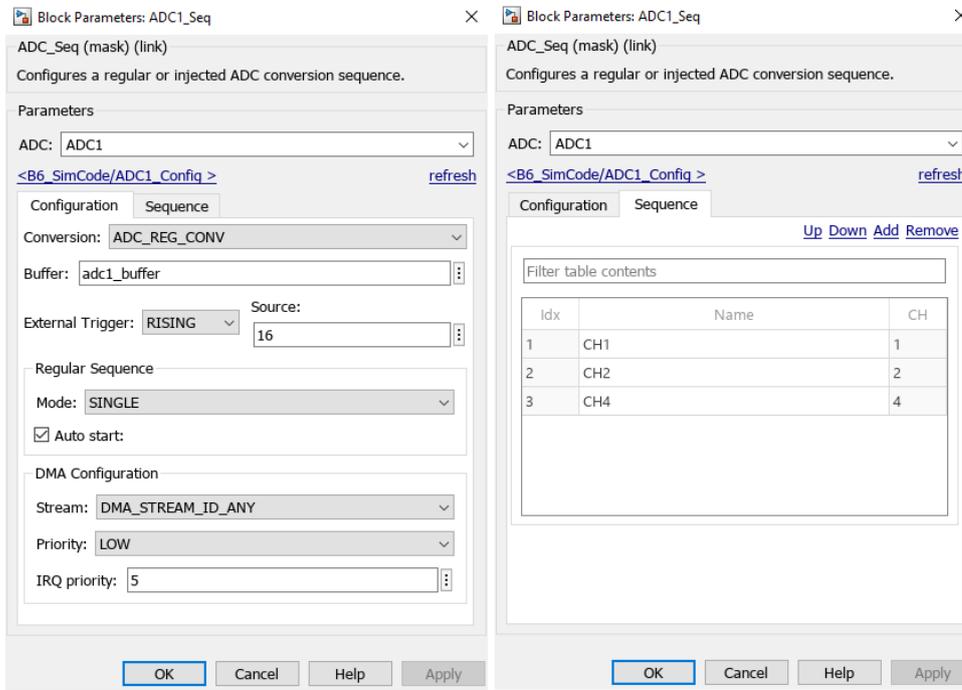


Figure 5.6: ADC regular sequence configuration.

In the "Configuration" tab of this block, the buffer is associated with the ADC. The ADC can be triggered by different sources, selected in Fig. 5.7.

ADC trigger selection EXTSEL[4:0] or JEXTSEL[4:0]	ADC triggers signals assignment			
	ADC 1/2		ADC3/4/5	
	Regular	Injected	Regular	Injected
0	tim1_cc1	tim1_trgo	tim3_cc1	tim1_trgo
1	tim1_cc2	tim1_cc4	tim2_cc3	tim1_cc4
2	tim1_cc3	tim2_trgo	tim1_cc3	tim2_trgo
3	tim2_cc2	tim2_cc1	tim8_cc1	tim8_cc2
4	tim3_trgo	tim3_cc4	tim3_trgo	tim4_cc3
5	tim4_cc4	tim4_trgo	exti2	tim4_trgo
6	exti11	exti15	tim4_cc1	tim4_cc4
7	tim8_trgo	tim8_cc4	tim8_trgo	tim8_cc4
8	tim8_trgo2	tim1_trgo2	tim8_trgo2	tim1_trgo2
9	tim1_trgo	tim8_trgo	tim1_trgo	tim8_trgo
10	tim1_trgo2	tim8_trgo2	tim1_trgo2	tim8_trgo2
11	tim2_trgo	tim3_cc3	tim2_trgo	tim1_cc3
12	tim4_trgo	tim3_trgo	tim4_trgo	tim3_trgo
13	tim6_trgo	tim3_cc1	tim6_trgo	exti3
14	tim15_trgo	tim6_trgo	tim15_trgo	tim6_trgo
15	tim3_cc4	tim15_trgo	tim2_cc1	tim15_trgo
16	hrtim1_adc_trg1	hrtim1_adc_trg2	hrtim1_adc_trg1	hrtim1_adc_trg2
17	hrtim1_adc_trg3	hrtim1_adc_trg4	hrtim1_adc_trg3	hrtim1_adc_trg4
18	hrtim1_adc_trg5	hrtim1_adc_trg5	hrtim1_adc_trg5	hrtim1_adc_trg5
19	hrtim1_adc_trg6	hrtim1_adc_trg6	hrtim1_adc_trg6	hrtim1_adc_trg6
20	hrtim1_adc_trg7	hrtim1_adc_trg7	hrtim1_adc_trg7	hrtim1_adc_trg7
21	hrtim1_adc_trg8	hrtim1_adc_trg8	hrtim1_adc_trg8	hrtim1_adc_trg8
22	hrtim1_adc_trg9	hrtim1_adc_trg9	hrtim1_adc_trg9	hrtim1_adc_trg9
23	hrtim1_adc_trg10	hrtim1_adc_trg10	hrtim1_adc_trg10	hrtim1_adc_trg10
24	hrtim2_adc_trg1	hrtim2_adc_trg2	hrtim2_adc_trg1	hrtim2_adc_trg2
25	hrtim2_adc_trg3	hrtim2_adc_trg4	hrtim2_adc_trg3	hrtim2_adc_trg4
26	hrtim2_adc_trg5	hrtim2_adc_trg5	hrtim2_adc_trg5	hrtim2_adc_trg5
27	hrtim2_adc_trg6	hrtim2_adc_trg6	hrtim2_adc_trg6	hrtim2_adc_trg6
28	hrtim2_adc_trg7	hrtim2_adc_trg7	hrtim2_adc_trg7	hrtim2_adc_trg7
29	hrtim2_adc_trg8	hrtim2_adc_trg8	hrtim2_adc_trg8	hrtim2_adc_trg8
30	hrtim2_adc_trg9	hrtim2_adc_trg9	hrtim2_adc_trg9	hrtim2_adc_trg9
31	hrtim2_adc_trg10	hrtim2_adc_trg10	hrtim2_adc_trg10	hrtim2_adc_trg10

Figure 5.7: ADC trigger source [7].

In this specific case, we want the HRTIM to trigger the ADC, so the source for regular conversion is set to 16 (hrtim\_adc\_trg1).

The final section is for DMA configuration and priority. The right side of Fig. 5.6 configures the channels of the ADC used. This depends on the pin mapping. The ADC buffer must be an array with a number of elements equal to the number of channels used in the specific ADC.

- **HRTIM:** The high-resolution timer is used to generate gate signals for switches. First, the HRTIM is initialized as shown in Fig. 5.8.

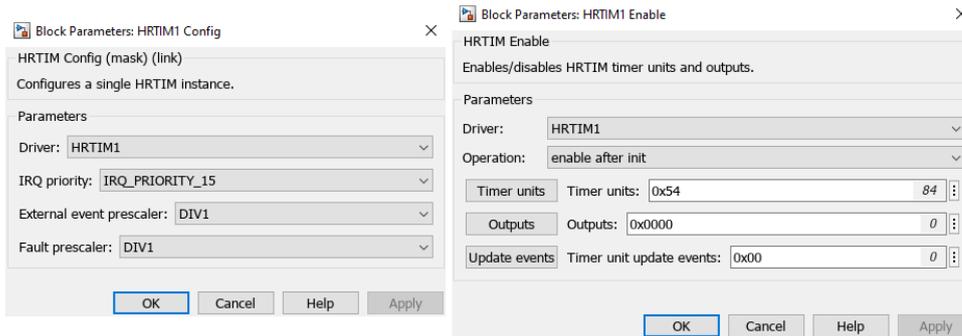


Figure 5.8: HRTIM configuration.

The left block configures the HRTIM1 driver, where the prescaler is set to 1, so the clock of the HRTIM1 is the same as the CPU clock (300 MHz). On the right side, the timer units that will be used are enabled. In this specific case, TIMER B, D, and F are used according to Fig. 5.4 and Table 4.2. Only the timer units are initialized on startup since the outputs are managed by the state machine. Each timer unit is configured similarly and has two outputs with dead-time implementation, so each unit will manage one leg of the converter. The details of the configuration are shown in Fig. 5.9 and Fig. 5.10.

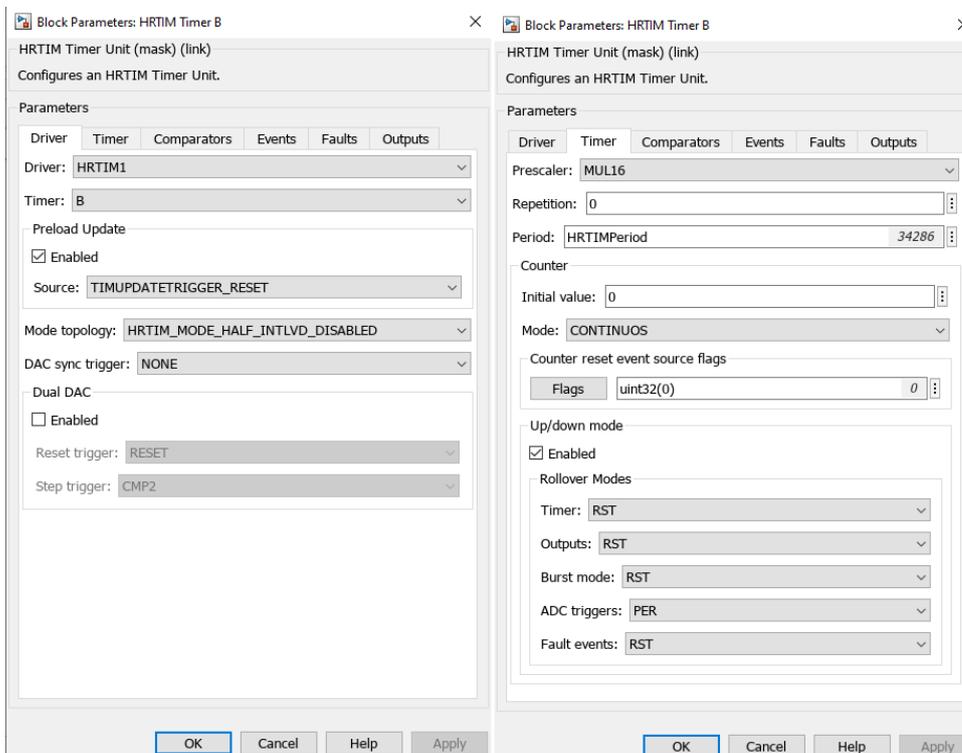


Figure 5.9: HRTIM timer unit.

Except for the selection of the "timer" line, the remaining configuration is the same for each timer unit.

The preload update is selected, which typically prevents waveforms from being altered by a register update not synchronized with active events. When preload mode is enabled, accessed registers are shadow registers, and their content is transferred into the active register after an update request, either software or synchronized with an event [7].

The second tab, "Timer," configures the period and the counting mode. The counting mode is set to up/down, ensuring output waveforms are center-aligned and symmetric. The PWM frequency is set to 70 kHz. Specifically, the unit's prescaler is set to "MUL16," meaning the frequency of the unit is  $300 \text{ MHz} \cdot 16 = 4.8 \text{ GHz}$ . Based on this, the timer period is:

$$HRTIM_{period} = \left( \frac{\text{timer clock}}{\text{PWM frequency}} \right) / 2 = \left( \frac{4.8 \text{ GHz}}{70 \text{ kHz}} \right) / 2 = 34286 \quad (5.1)$$

The last division by 2 is necessary for the up/down mode, which counts twice during one switching period (first up and then down).

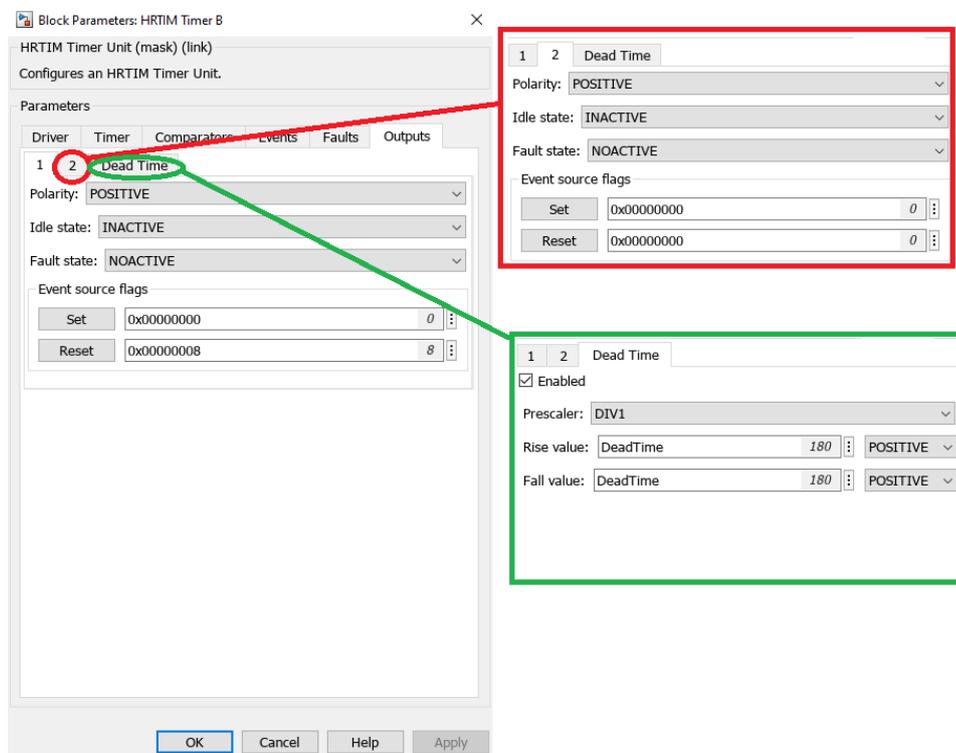


Figure 5.10: HRTIM outputs configuration.

The configuration of outputs is done in the "output" tab. To better understand this part, it is necessary to delve into the details of up/down counting. Output 1 goes to the upper side of the leg of the converter, so we want the output duty cycle to match the control signal. The different results on output waveforms depending on the configuration are shown in Fig. 5.11.

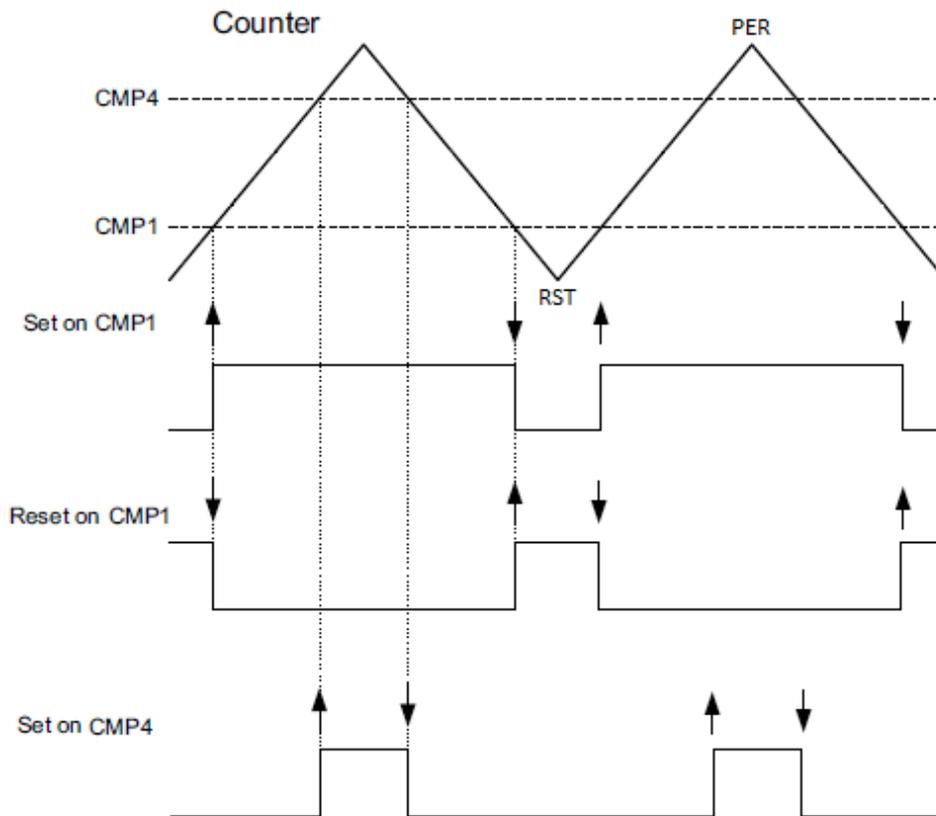


Figure 5.11: Output waveforms.

Fig. 5.11 demonstrates that regardless of the compare value, the output signal is center-aligned. The waveform "Set on CMP1" and "Set on CMP4" shows that CMP4 has a higher duty cycle. However, the  $t_{on}$  on the output waveforms is lower for CMP4, indicating this configuration is not suitable for our purpose. The waveform "Reset on CMP1" is the correct choice since it is the opposite of "Set on CMP1". Finally, the event source flag in Fig. 5.10 is set to reset. CMP1 is used for each timer unit. The output polarity is positive since we want 0 as the output when the duty cycle is 0.

Output 2 is automatically generated when dead-time is active. The event source flags in the red rectangle are set to 0 for both set and reset. Dead-time calculation

is done using a factor that multiplies the clock of the timer with a prescaler. The prescaler is set to "DIV1," so the dead-time value in seconds is:

$$Dead - time[s] = \frac{1}{300 \text{ MHz}} \cdot t_{DTG} \quad (5.2)$$

where  $t_{DTG}$  is the factor to be set in the Simulink block. A conservative value of 180 is set, leading to a dead-time value of:

$$Dead - time[s] = \frac{1}{300 \text{ MHz}} \cdot 180 = 600 \text{ ns} \quad (5.3)$$

The HRTIM is also used to trigger and start ADC conversion. A dedicated block in Simulink is used for this purpose (Fig. 5.12).

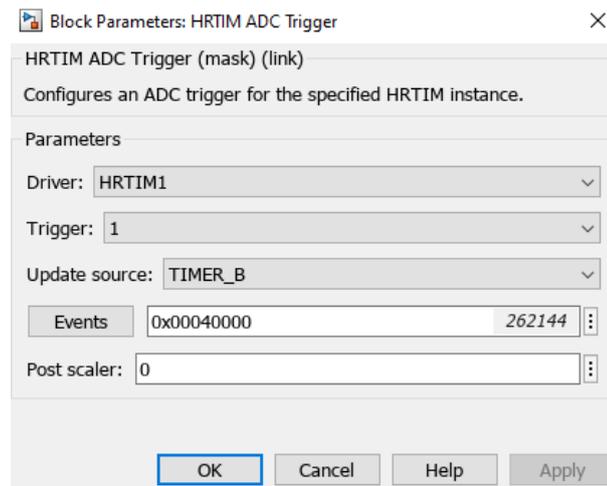


Figure 5.12: ADC trigger configuration.

It has already been mentioned that the ADC is triggered by the HRTIM. In this specific block, the timer unit and the event that sends the trigger are selected. Specifically, the event is the period of timer B. As a result, when the ADC starts the conversion, it ensures there are no edges in any PWM.

- **TIMER:** The interrupt is used to trigger the control subsystem in Simulink. The control is divided into two tasks, which is why there are two timer configurations in Fig. 5.2: HF stands for High Frequency and LF stands for Low Frequency. Fig. 5.13 shows the configuration of the High Frequency tasks.

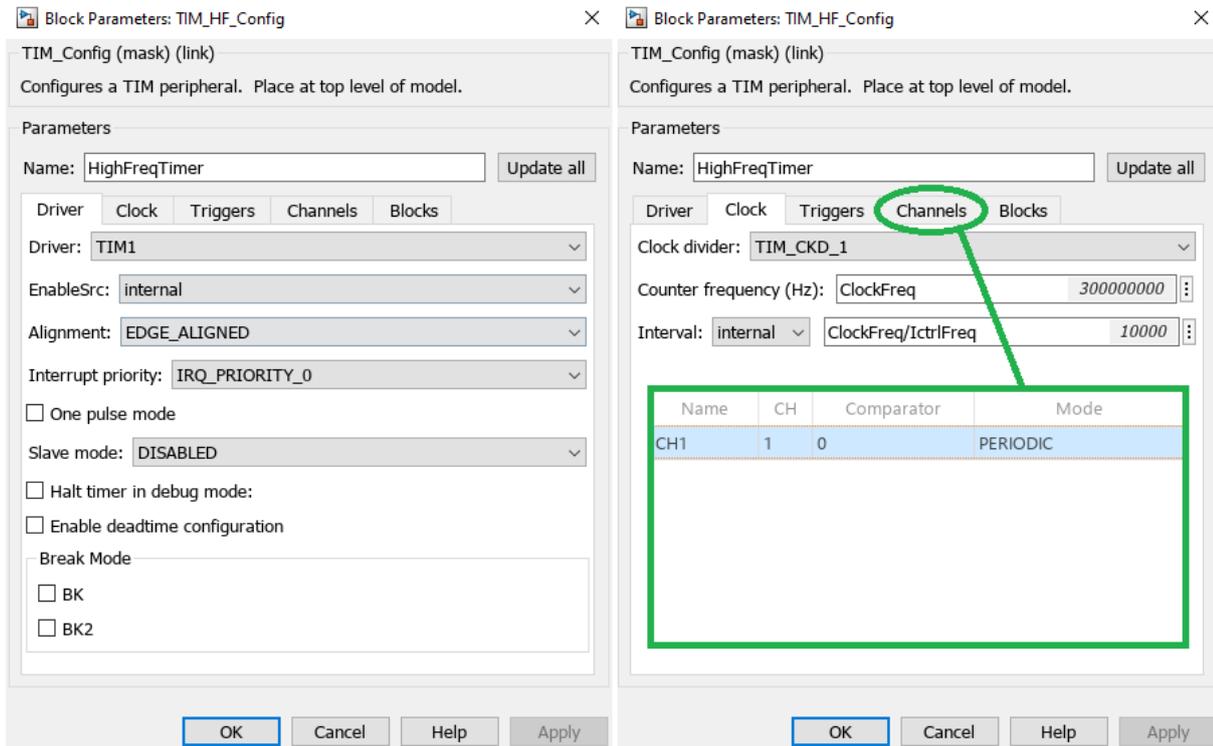


Figure 5.13: TIMER configuration.

The important parameter is the priority, set to 0 for HF and 1 for LF. The "clock" tab configures the counter for the timer. In this specific case, the clock of the timer is the CPU clock, and the timer counter is calculated as:

$$Timer\ counter = \frac{Frequency\ of\ the\ timer}{Desired\ control\ frequency} \quad (5.4)$$

The "channel" tab (in the green rectangle) shows the configured timer channel. In this case, only one period channel is needed for the periodic interrupt.

The same procedure is followed for the Low Frequency timer.

- **DAC:** As previously mentioned, the control board has only two DAC channels. Both of them are channels of the same peripheral BDAC1, and the configuration is shown in Fig. 5.14.

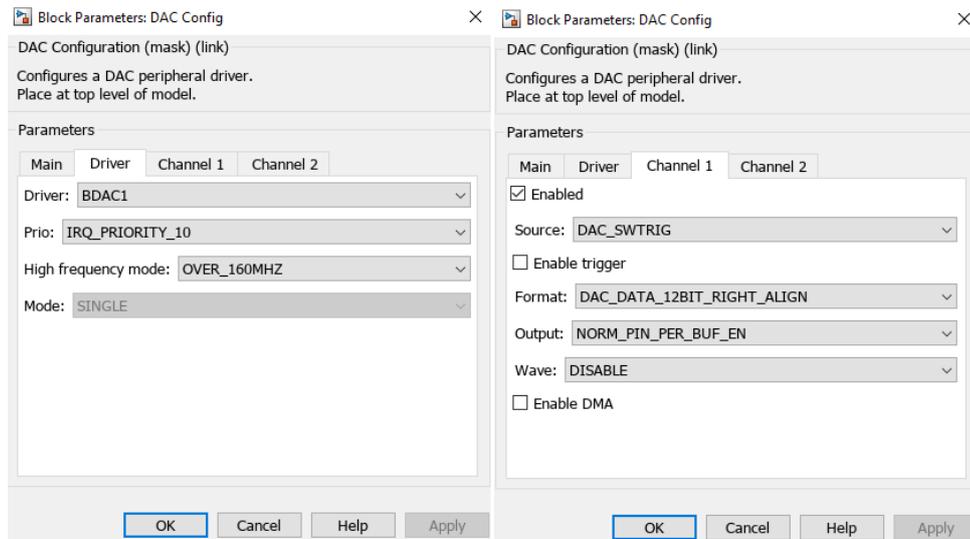


Figure 5.14: DAC configuration.

Channel 1 and channel 2 are configured in the same way.

- **GPIO:** All pins are configured in the GPIO configuration block (Fig. 5.15).

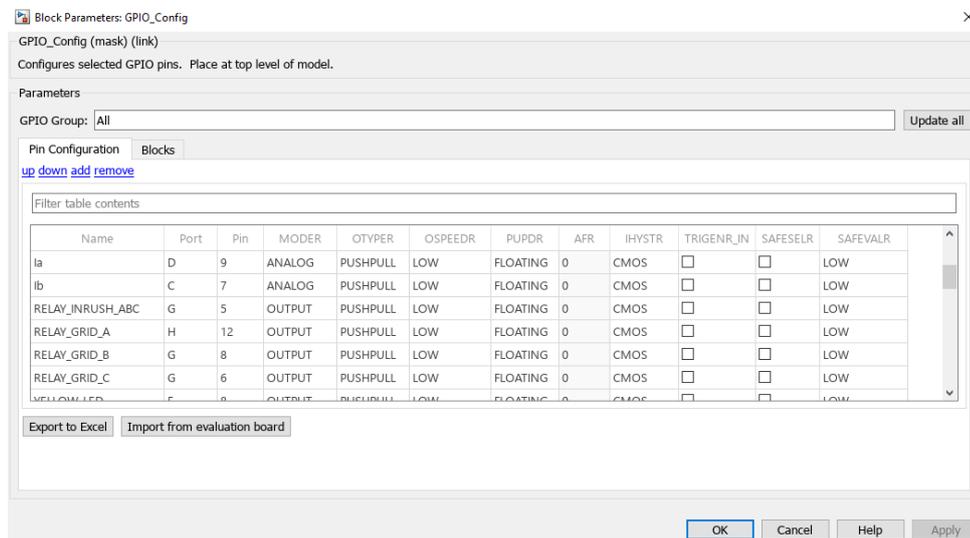


Figure 5.15: GPIO configuration.

In Fig. 5.15, some lines of pin configuration are shown. Ia and Ib are pins for the ADC, while relays are GPIO outputs.

### 5.2.3. HF, LF and Main task

Going into the "Controller" subsystem of Fig. 5.2, the situation is detailed in Fig. 5.16.

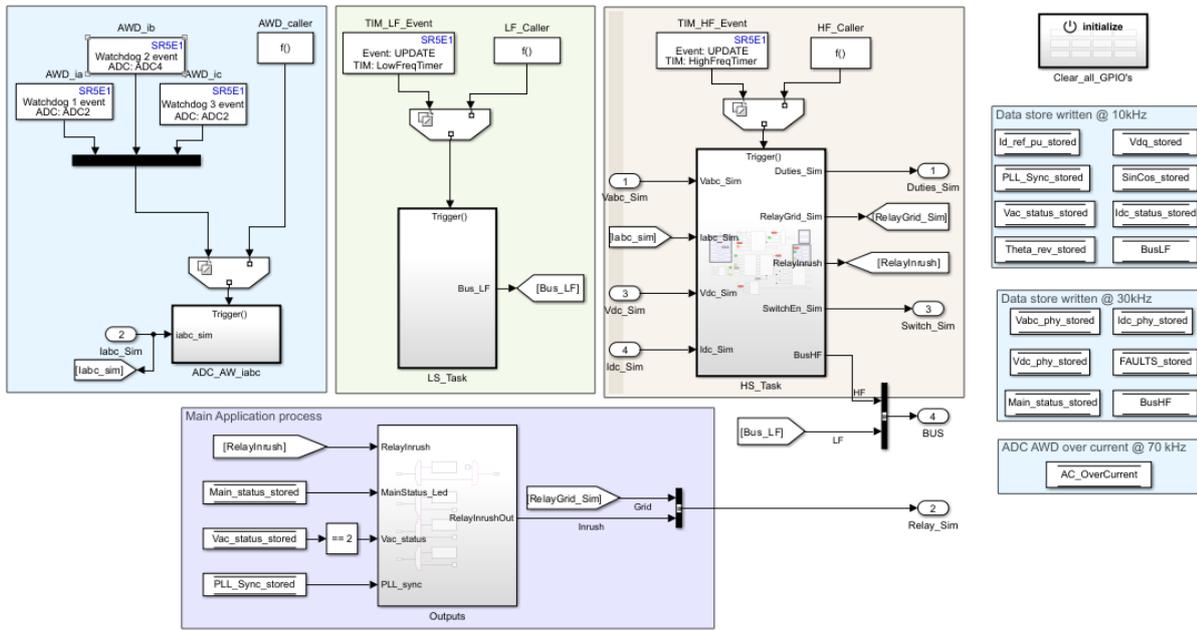


Figure 5.16: Controller tasks.

There are four main areas, plus areas where data stores are initialized and an initialization block. Starting from the top left, there's the *Analog Watchdog* used for AC protection. Each current has a dedicated channel; it is feasible that three StellarE blocks are connected together with a mux to trigger a subsystem. The analog watchdog event block is shown in Fig. 5.17.

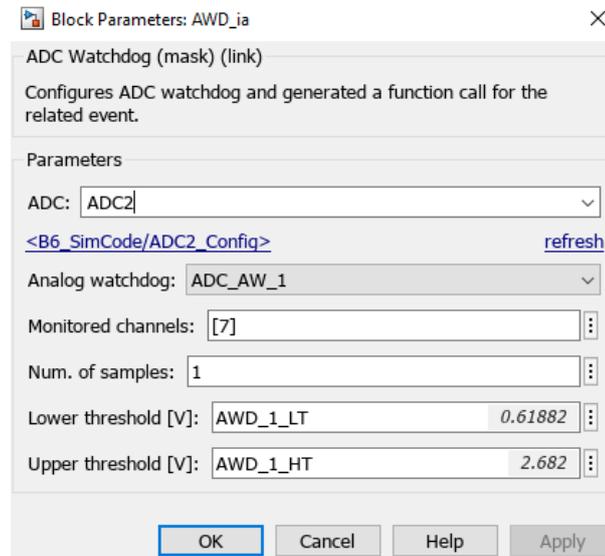


Figure 5.17: Analog watchdog event configuration.

The event configuration allows the selection of the channel, ADC number, and low and high thresholds (in V, ranging from 0 to 3.3 V) over which the interrupt is sent and the subsystem is triggered. The event is simulated for desktop simulation with a function call generator (simulating a periodic interrupt) at PWM frequency.

If the subsystem is triggered in hardware implementation, all grid relays are opened, PWM outputs are stopped, and the "AC\_OverCurrent" data store is set to one. This data store will then be read by the main State Machine. In desktop simulation, the subsystem is triggered periodically, and it checks if the current is out of range; if so, "AC\_OverCurrent" is set to one.

Moving to the right in Fig. 5.16, there is the *LF task*, triggered by the Low Frequency timer. The low-frequency application is triggered every 10 kHz. Inside the LF\_Tasks subsystem, you will find the PLL and voltage controller. The DC voltage reference and feed are given in physical values, while the output of the voltage loop (reference current of the d-axis) is in p.u. (per unit) with respect to the maximum AC current that can be sensed by the ADC. Similarly, AC voltages in the d-q reference frame are computed in p.u. Additionally, an output of the PLL block (Fig. 3.3) is the frequency, which must match the grid frequency. This is checked before the control starts and is done in the LF task. The output of the "LF\_Tasks" subsystem in Fig. 5.16 is the LF bus, which contains debug variables.

The last triggered subsystem is the *HF task*, where several functions are carried out:

- **Manage ADC buffer data:** As mentioned earlier (Fig. 5.7), the ADC is triggered by the HRTIM. After that, all data are saved in the ADC buffer and processed in

the HF task. Each signal has a bias and a gain, and starting from the ADC value, the physical value of that signal is:

$$Physical\ value = \frac{ADC\ value - Bias}{Gain} \quad (5.5)$$

- **Inner loop** of the control (current loop): It receives  $I_{dref}$  from the voltage loop, while  $I_{qref}$  is set to 0. The final part is shown in Fig. 3.10. For the three-phase full bridge converter, the Duty computation shifts modulation  $V_{abc\_ctrl}$  from  $[-1,1]$  to  $[0,1]$ ; in this way, they are ready to be managed by the HRTIM in hardware and compared with the carrier signal in desktop simulation.
- **Outputs:** Actuators, grid relay, and switches are managed. Duty cycles must be processed before sending them to the HRTIM comparator. Since the HRTIM clock is higher than the system clock, the least significant bits of the counter and capture registers are not significant. The least significant bits cannot be written (counter register only) and return 0 when read. For instance, if the prescaler of the HRTIM is set to 4 (leading to an equivalent timer frequency of  $300\text{ MHz} \times 8 = 2.4\text{ GHz}$ ), writing  $0xFFFF$  into the counter register yields an effective value of  $0xFFFF8$ . Conversely, any counter value between  $0xFFFF$  and  $0xFFFF8$  is read as  $0xFFFF8$  [7]. Fig. 5.18 shows how many bits are not significant with respect to the timer prescaler.

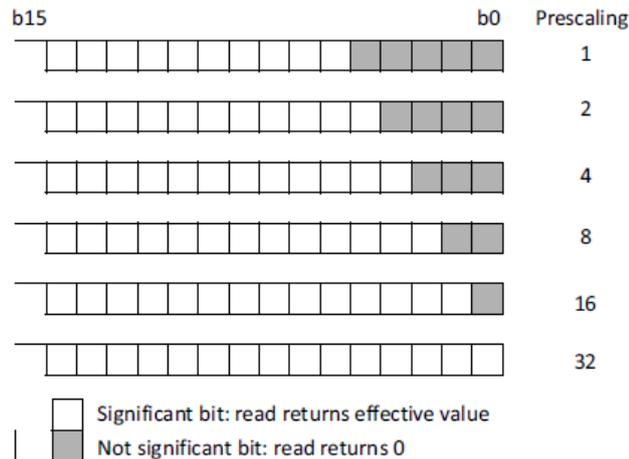


Figure 5.18: Counter and capture register format vs clock prescaling factor [7].

In this specific case, the minimum value of the comparator is  $dutyMin = 48$  and the maximum value is  $dutyMax = Period - 48 = 34286 - 48 = 34238$ . The computation starting from the duty cycle is as follows:

$$- Cmp\_value = duty \cdot HRTIM_{period}$$

- Check if the compare value is within limits (Fig. 5.19)

```

if(uint16(cmp1)>=uint16(dutyMax)) % cmp1
    cmp1Out=uint16(1);
elseif(uint16(cmp1)==uint16(0))
    cmp1Out=uint16(0);
elseif(uint16(cmp1)<uint16(dutyMin))
    cmp1Out=uint16(dutyMin);
else
    cmp1Out=uint16(cmp1);
end

```

Figure 5.19: Check limit of comparator.

If the duty is over *dutyMax*, the comparator value is set to 1. This is a special value for which the output duty cycle is 1.

The procedure is repeated for the other two duty cycles. Finally, the comparator value is sent to the HRTIM with a dedicated block in Simulink, as shown in Fig. 5.20.

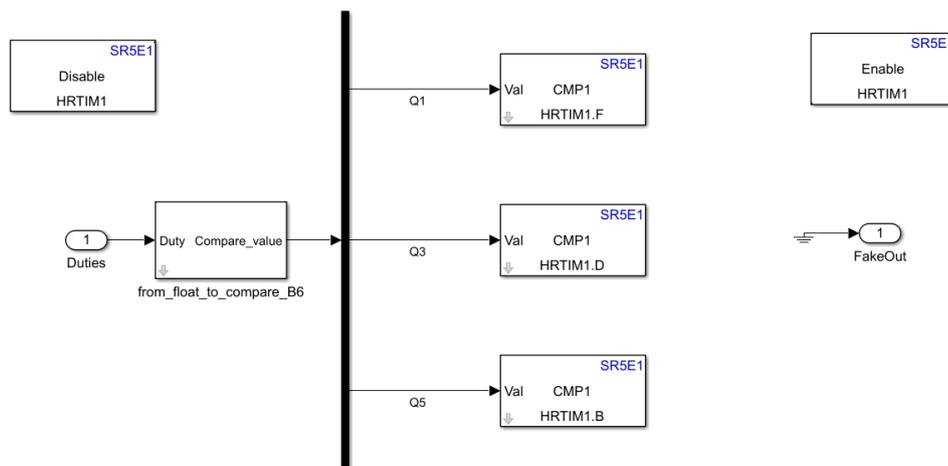


Figure 5.20: Actuator output.

Fig. 5.20 also shows "Disable" and "Enable" blocks. These blocks are important because the HF task and PWM have different frequencies, and it is crucial that all compare values are updated in the same interrupt. The disable block (executed first thanks to priority settings in Simulink) disables all event updates: if a new compare value is set, it is not updated even if the HRTIM is ready. After all computations, the update event is enabled again, ensuring all compare values are correct.

Looking at Fig. 5.16, the remaining part is the *Main Application*, where some LEDs are managed, and the inrush relay is controlled. Additionally, other debug GPIOs are used.

### 5.2.4. Burst mode

One purpose of the state machine is to increment the DC voltage up to the reference value. A dedicated control is implemented to avoid peak currents and undesired behavior. In this specific case, only the lower side switches are enabled with a fixed duty cycle, and only when the voltage of each leg is positive, as shown in Fig. 5.21.

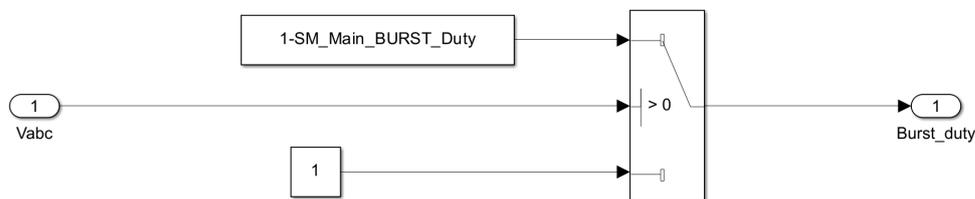


Figure 5.21: Burst mode for Three Phase Full Bridge.

### 5.2.5. Data dictionary

In order to save all parameters of the control, a data dictionary is used. This allows customization and control over how each parameter should appear in the generated code. In this way, Simulink creates a .h file in the C code where variables are defined using "Define". Additionally, some variables are generated in the C code as tunable, allowing them to be changed without rebuilding the project. Fig. 5.22 shows some lines of the automatically generated .h file.

```
/* Exported data define */
/* Definition for custom storage class: Define */
#define ADC_IAC_B          2050
#define ADC_IAC_G          58.438F
#define ADC_IDC_B          2050
#define ADC_IDC_G          58.404F
#define ADC_VAC_B          2048
#define ADC_VAC_G          5.4575F
#define ADC_VDC_B          0
#define ADC_VDC_G          4.2667F
#define DAC_CH1            1
#define DAC_CH1_B          2047.0F
#define DAC_CH1_G          2048.0F
#define DAC_CH2            3
#define DAC_CH2_B          2047.0F
#define DAC_CH2_G          2048.0F
```

Figure 5.22: .h file with defined variables.

## 5.3. Simulator of Three phase full bridge

Once the control has been defined and implemented, the focus shifts to the simulator side. The average model is first implemented in MATLAB, tested, and validated in Simulink. After that, the model is rewritten in C code.

### 5.3.1. Average model

The equivalent circuit of the three-phase full bridge with the three-phase grid and DC capacitor is shown in Fig. 5.23.

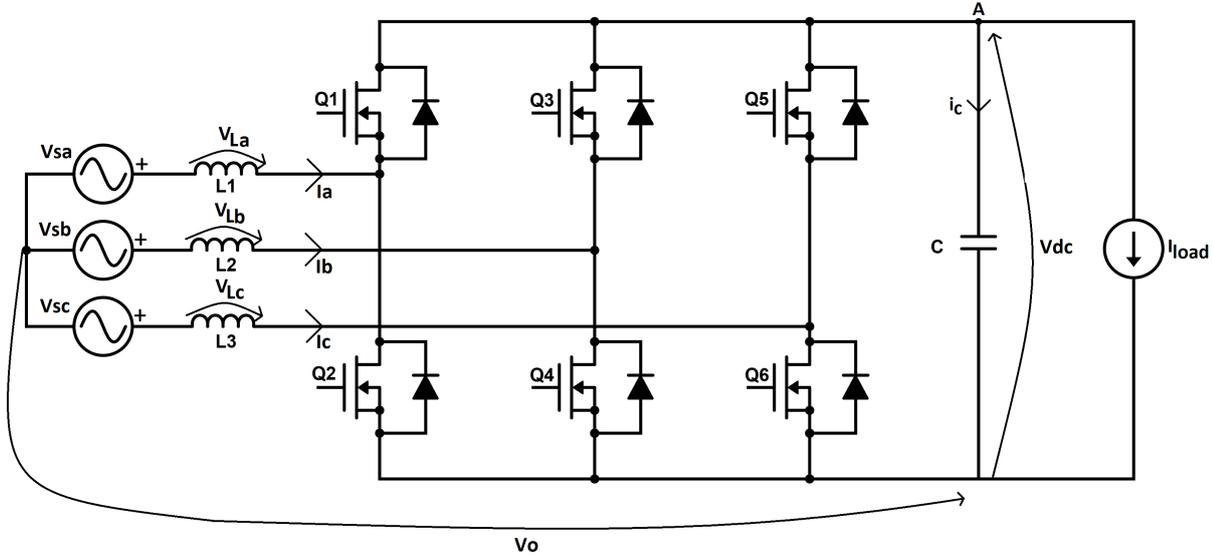


Figure 5.23: Scheme of Three phase full bridge system.

The system has as many differential equations as there are elements that accumulate energy: in this specific case, there are three inductors on the AC side and a capacitor on the DC side. Since the AC side is three-phase, the equation for the inductor is the same for each phase, so a general equation for an inductor is written. First, we define:

$$v_k = v_{sk} - R \cdot i_k - v_o \quad (5.6)$$

where:

- $k = a, b, c$ .
- $v_s$  is the source voltage.
- $R$  is the total resistance of the circuit, where  $R = R_{inductor} + R_{mos} + R_{inrush}$ , with  $R_{inrush}$  being present or not depending on the relay.
- $i_k$  is the phase current.

The average model is divided into four main parts:

1. *Force computation*: This part is necessary to manage the startup phase where all switches are off and the circuit behaves like a diode bridge. Given  $d_{k,PN} =$  duty cycle of  $Up$ , Down transistor for leg  $k$ , the step of the average model is divided into two parts: the first part when the upper switch is on and the second part when the lower switch is on. Equation 5.7 shows when the current flows in the

upper switch.

$$d_{kq} = \begin{cases} \max(d_{kP}, 1 - d_{kN}) & i_k > 0; \text{ or } \begin{cases} i_k = 0 \\ v_k > v_{dc} \end{cases} \\ d_{kP} & \text{otherwise, i.e. } i_k < 0; \text{ or } \begin{cases} i_k = 0 \\ v_k \leq v_{dc} \end{cases} \end{cases} \quad (5.7)$$

where all  $d$  variables are duty cycles. Let's take the example where all switches are off to detail equation 5.7. This is the situation where the converter is working as a three-phase diode bridge, so depending on the sign of the AC voltage, the diode must be closed, so the duty must be 1 even if the switches are off.

The second part is when the lower switch is closed, and in this case, equation 5.8 shows the result.

$$d_{km} = \begin{cases} \max(d_{kN}, 1 - d_{kP}) & i_k < 0; \text{ or } \begin{cases} i_k = 0 \\ V_k < 0 \end{cases} \\ d_{kN} & \text{otherwise, i.e. } i_k > 0; \text{ or } \begin{cases} i_k = 0 \\ V_k \geq 0 \end{cases} \end{cases} \quad (5.8)$$

After these considerations, the voltage across the inductor can be calculated as

$$v_{kL} = L \frac{di_k}{dt} = \begin{cases} V_{kP} = V_k - v_{dc} & \text{during } d_{kP} \\ V_{kN} = V_k & \text{during } d_{kN} \end{cases} \quad (5.9)$$

where  $V_{kP}$  is the voltage across the inductor when the current flows in the upper switch, and  $V_{kN}$  is the voltage when the current flows in the lower switch. The state variable is the current, which can now be calculated with a finite difference.

$$\frac{L}{\Delta T} \Delta i_k = V_{kP} \cdot d_{kq} + V_{kN} \cdot d_{km} \quad (5.10)$$

The current at a certain instant  $t$  is

$$i_{k1} = i_{k\_old} + \frac{\Delta T}{L} (V_{kP} \cdot d_{kq} + V_{kN} \cdot d_{km}) \quad (5.11)$$

where  $i_{k\_old}$  is the current computed at the previous step and  $(i)$  indicates that the procedure is repeated for each phase.  $T_s$  is the sample time of the average model.

Before proceeding, we need to consider that switches can work in continuous conduction mode (CCM) and discontinuous conduction mode (DCM). A detailed view is shown in Fig. 5.24.

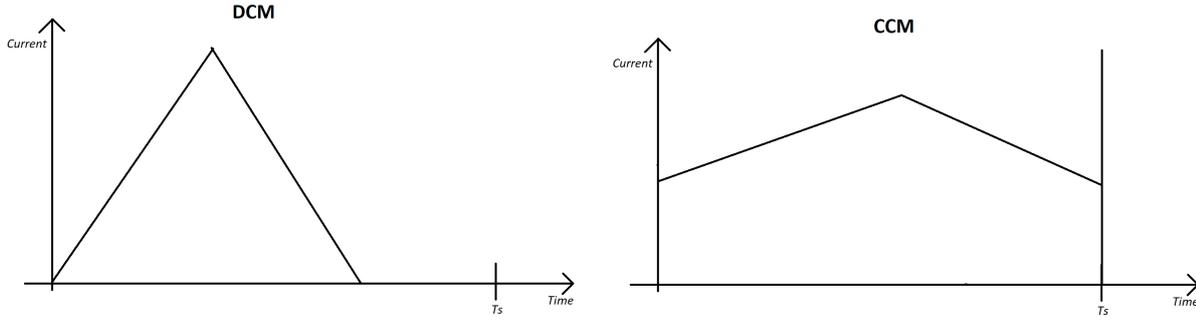


Figure 5.24: CCM and DCM.

In discontinuous conduction mode, the current goes to 0 in a period of switching  $T_s$ . This condition can be recognized by checking the zero crossing as in equation 5.12.

$$i_{k1} = \begin{cases} 0 & \text{if } \text{sign}(i_{k1}) \cdot \text{sign}(i_{k\_old}) < 0 \\ i_{k1} & \text{otherwise} \end{cases} \quad (5.12)$$

2. *Free-wheel*: The second phase occurs only if both switches are off for a certain time during the interval, so this condition is checked in equation 5.13. In principle, the control always turns on either the high switch or the low switch, with one being the negation of the other. However, in reality, the switching of the switches is not instantaneous, and therefore a deadtime is added. As a result, the sum of the duty cycles of the high switch and the low switch is no longer 1.

$$d_{k1} = d_{kq} + d_{km}$$

$$d_{k2} = 1 - d_{k1}$$

$$\text{If } i_{k1} \neq 0 \text{ and } d_{k2} > 0 \quad (5.13)$$

$$i_{k1} > 0 \rightarrow V_{k2} = V_{kP}$$

$$i_{k1} < 0 \rightarrow V_{k2} = V_{kN}$$

From  $V_{k2}$ , we calculate the current during the free-wheel phase  $i_{k2}$ .

$$i_{k2} = i_{k1} + \frac{T_s}{L} V_{k2} d_{k2} \quad (5.14)$$

Considerations about saturation of  $i_{k2}$  are the same as those done for  $i_{k1}$ .

It can happen that  $i_{k2} = 0$  due to the switches being in DCM. In this case, it is not true that the current is really zero, and the following approximation is made:

$$i_{k\_average} = \begin{cases} i_{k2} & \text{if } i_{k2} \neq 0 \\ \frac{i_{k\_old} + i_{k1}}{2} d_{k1} + \frac{i_{k1} + i_{k2}}{2} d_{k2} & \text{if } i_{k2} = 0 \end{cases} \quad (5.15)$$

3. *Updating  $V_o$* : The last step on the AC side is to calculate the voltage  $V_o$ . Starting from

$$\begin{aligned} i_{k1} &= i_{k\_old} + \frac{\Delta T}{L} (V_{kp} \cdot d_{kq} + V_{kN} \cdot d_{km} + V_k \cdot d_{kH}) \\ &= i_{k\_old} + \frac{\Delta T}{L} [(v_{sk} - R \cdot i_k - v_o) \cdot d_{k1} - v_p \cdot d_{kq} + v_n \cdot d_{km}] \\ &= i_{k\_old} + \frac{\Delta T}{L} [(v_{sk} - R \cdot i_k) \cdot d_{k1} - v_p \cdot d_{kq} + v_n \cdot d_{km}] - \frac{\Delta T}{L} \cdot d_{k1} \cdot v_o \end{aligned} \quad (5.16)$$

It is defined  $\alpha_k$  as

$$\alpha_k = i_{k\_old} + \frac{\Delta T}{L} [(v_{sk} - R \cdot i_k) \cdot d_{k1} - v_p \cdot d_{kq} + v_n \cdot d_{km}] \quad (5.17)$$

and consequently

$$\begin{cases} i_k = \alpha_k - \frac{\Delta T}{L} \cdot d_{k1} \cdot v_o \\ \sum i_k = 0 \end{cases} \rightarrow \sum \alpha_k - \sum \frac{\Delta T}{L} \cdot d_{k1} \cdot v_o = 0 \quad (5.18)$$

Up to now, all current computations have to be repeated for every phase. From now on, equations are no longer repeated.

The final computation of the voltage  $V_o$  is:

$$v_o = \frac{L}{\Delta T} \cdot \frac{\sum \alpha_k}{\sum d_{k1}} \quad (5.19)$$

4. *Voltage computation*: After considering the inductor, it's time to consider the capacitor on the DC side. The state variable of a capacitor is the voltage, and the equation is:

$$i_c = C \frac{dv_{dc}}{dt} \quad (5.20)$$

The current flowing into the capacitor is computed starting from KCL (Kirchhoff's current law) at node A:

$$i_c + i_{load} - i_{conv} = 0 \rightarrow i_c = i_{conv} - i_{load} \quad (5.21)$$

where

$$i_{conv} = \sum i_k \cdot d_{kq} \quad (5.22)$$

The final equation for the voltage across the capacitor is:

$$\frac{dv_{dc}}{dt} = \frac{1}{C} \left( \sum i_k \cdot d_{kq} - i_{load} \right) \rightarrow v_{dc} = v_{dc\_old} + \frac{\Delta T}{C} \left( \sum i_k \cdot d_{kq} - i_{load} \right) \quad (5.23)$$

### 5.3.2. Hardware implementation

The equations explained are implemented in C code and called periodically in an interrupt. The interrupt is divided into several parts:

1. *Grid Voltages*: These are sensed before the grid relay and must always be monitored by the controller, so they are continuously sent out.
2. *Relay Grid*: The relay grid is considered closed only if all three grid relays are closed, based on the signal from the controller.
3. *Relay Inrush*: Manages the inrush relay according to the controller's signals.
4. *Duty Cycles*: These are computed as explained in Chapter 2 and then saved in an array by core2. The duty cycles are taken from shared memory and passed through the function containing the differential equations of the model.
5. *DAC Output*: The last part of the interrupt manages the DAC output based on the same gain and bias used on the real board and, consequently, the same as the control board.

### 5.3.3. GUI with fault injection

The objective of the simulator is to test the performance of the control in all its aspects. One important aspect is the state machine and how faults and protections against them are managed.

A GUI for the Stellar family microcontroller already exists and is integrated into the simulation software. Specifically, the GUI allows visualization of variables running in the

differential model before they are sent to the control board. The GUI is presented in Fig. 5.25. Additionally, it is possible to change parameters of the simulator in real-time (e.g.,  $R$ ,  $L$ ,  $C$ , source voltage).

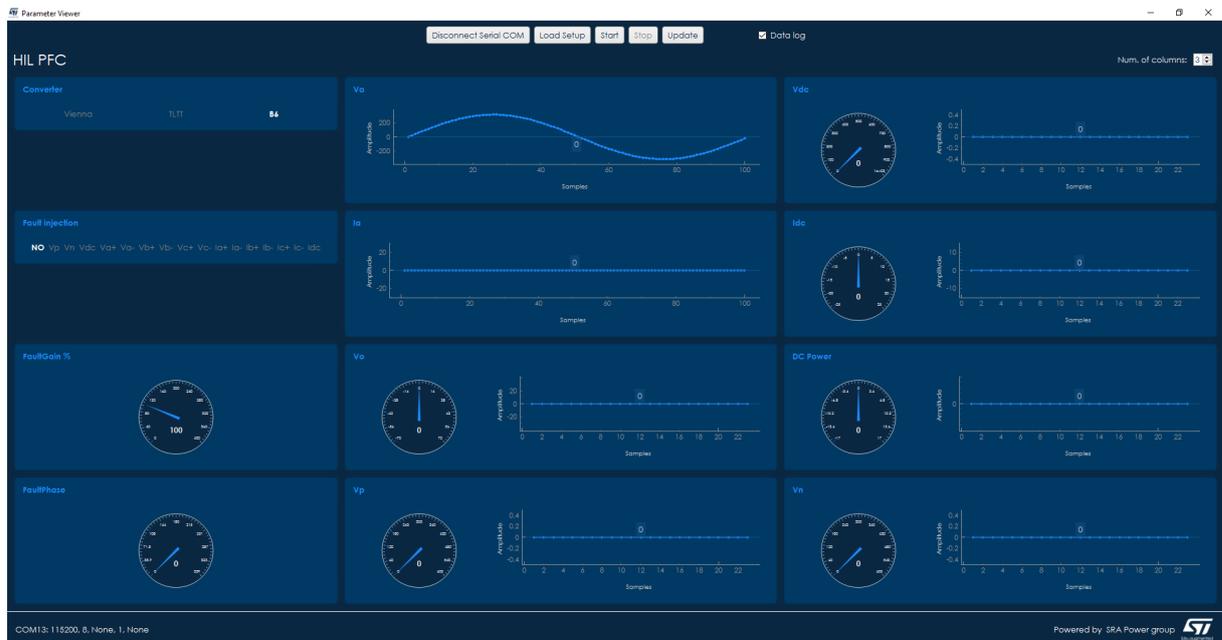


Figure 5.25: Simulator GUI.

Thanks to the flexibility of the simulator board, it was possible to integrate more than one converter within the same firmware. This is the reason why the section "Converter" is displayed in the top left part of the GUI and also why some variables are not used for the three-phase full bridge (e.g.,  $V_p$ ,  $V_n$ ).

The **left column** is dedicated to fault injection, where it can be decided which fault to inject and at which instant of the sinusoid (from  $0^\circ$  to  $360^\circ$ ) with respect to  $V_{sourceA}$  the fault is injected. The last section of the left column is the gain, which is the factor that multiplies the variable to be changed. The faults that can be injected are the following:

1. **Vdc**: Overvoltage on the DC side.
2. **Idc**: Overcurrent on the DC side.
3. **Va+**: Overvoltage or undervoltage on the AC side. The voltage is multiplied by the gain only when it's positive.
4. **Va-**: Same as  $Va+$  but the gain is multiplied only when the voltage is negative. This can simulate a failure on AC network and the controller should act in order to prevent damaging components.

5.  $V_{b+}$ : Same as  $V_{a+}$  but on phase b.
6.  $V_{b-}$ : Same as  $V_{a-}$  but on phase b.
7.  $V_{c+}$ : Same as  $V_{a+}$  but on phase c.
8.  $V_{c-}$ : Same as  $V_{a-}$  but on phase c.
9.  $I_{a+}$ : Overcurrent on the AC side. It is divided into positive and negative parts, similar to  $V_{a+}$ .
10.  $I_{a-}$ : Same as  $I_{a+}$  but on the negative part of the currents.
11.  $I_{b+}$ : Same as  $I_{a+}$  but on phase b.
12.  $I_{b-}$ : Same as  $I_{a-}$  but on phase b.
13.  $I_{c+}$ : Same as  $I_{a+}$  but on phase c.
14.  $I_{c-}$ : Same as  $I_{a-}$  but on phase c.

## 5.4. Experimental tests

Experimental tests are first carried out based on Simulink simulations to validate the average model.

After that, a real-time simulation is conducted and compared with the desktop simulation. The objective is also to test the performance of the control.

Below is a table with parameters of the simulator (Table 5.1) with some of them derived from the board "STDES-BCBIDIR".

$R_{mos}$	45 $m\Omega$
$R_{inductor}$	36 $m\Omega$
$V_{PhaseToGroundRMS}$	220 $V$
Line Frequency	50 $Hz$
Sample time (Simulink)	1/(60 $kHz$ )
Sample time (real-time)	1/(65 $kHz$ )
Input inductor	255 $mH$
DC capacitor	500 $\mu F$
$R_{inrush}$	25 $Ohm$
Nominal power	11 $kW$

Table 5.1: Simulator parameters.

The resistance of the MOSFETs is taken directly from the datasheets, specifically [4], and the same applies to the inductor [1].

The sample time is different because, in Simulink, all sample times must be multiples, while in real-time, a frequency is chosen that is different from the PWM and its multiples to maintain asynchronous control and simulation.

From the controller side, parameters are listed in Tables 5.2, 5.3, 5.6, 5.4, and 5.5.

ADC_IAC_BIAS	2048
ADC_IAC_GAIN	58.404
ADC_IDC_BIAS	2048
ADC_IDC_GAIN	58.404
ADC_VAC_BIAS	2048
ADC_VAC_GAIN	5.4575
ADC_VDC_BIAS	0
ADC_VDC_GAIN	4.2667
AWD_HT	$(-IAC\_MAX*ADC\_IAC\_G+ADC\_IAC\_B)$
AWD_LT	$(-IAC\_MAX*ADC\_IAC\_G+ADC\_IAC\_B)$
AWD_HT	$(-IAC\_MAX*ADC\_IAC\_G+ADC\_IAC\_B)$

Table 5.2: ADC parameters - Three-phase full bridge.

PWM frequency	70 kHz
Deadtime	6 ns / 600 ns
Ictrl_ACFeedForward_enable	1
Ictrl_AWTG (anti-windup gain factor)	1379
Ictrl_Decoupling_enable	0
Ictrl_DSatNeg	-0.9
Ictrl_DSatPos	0.9
Ictrl_Ki	690
Ictrl_Kp	0.2275
Ictrl_Linductor	255 $\mu H$
Ictrl_LineFreq	50 Hz
Ictrl_QSatNeg	-0.1
Ictrl_QSatPos	0.1
Ictrl_Freq	30 kHz

Table 5.3: Current loop parameters - Three-phase full bridge.

The deadtime can be changed and adjusted depending on the real board construction and performance.

Simulations are carried out with two different deadtimes to evaluate the control performance with varying deadtimes, as these influence the behavior of AC side currents.

The cutoff frequency of the inner loop is set to 2 kHz, which is compatible with the PWM frequency.

SM_IDC_NO	0.5
SM_IDC_LOW	1
SM_IO_FAULTS_IDC_OC	15
SM_IO_FAULTS_V_BUS_MAX	880
SM_IO_FAULTS_VAC_PK_OV	$(250*\sqrt{2})$
SM_Main_BURST_Duty	0.1
SM_Main_BURST_ENABLE	1
SM_Main_BURST_V_MAX	820
SM_Main_IDLE_2_INIT_TO	$0.5*I_{ctrl\_Freq}$
SM_Main_INIT_2_START_TO	$0.5*I_{ctrl\_Freq}$
SM_Main_INRUSH_V_MAX	900
SM_Main_INRUSH_V_MIN	$220*\sqrt{2}*\sqrt{3}$
SM_Main_RUN_BURST_VREF_V	800
SM_VAC_PK_UV	$20*\sqrt{2}$
SM_VAC_RMS_UVLO	50

Table 5.4: State machine parameters - Three-phase full bridge.

Vctrl_AWTG (anti-windup gain factor)	16
Vctrl_Decoupling_enable	0
Vctrl_IdSatNeg	-0.85
Vctrl_IdSatPos	0.85
Vctrl_Ki	7.8817
Vctrl_Kp	0.2275
Vctrl_LineFreq	50 Hz
Vctrl_PSatNeg	-1000
Vctrl_PSatPos	1000
Vctrl_Freq	10 kHz
Vctrl_Ref	800 V

Table 5.5: Voltage loop parameters - Three-phase full bridge.

The cutoff frequency of the outer loop is set to 40 Hz.

This ensures a difference in the cutoff frequency that is more than 10 times lower than that of the current loop, which is the minimum requirement for system stability.

pllctrl_freq	50
pllctrl_Ki	20
pllctrl_Kp	5000

Table 5.6: PLL parameters - Three-phase full bridge.

The PLL is in the same interrupt of the voltage loop, so the execution frequency is 10 kHz.

The simulations aim to test different phases either to check the state machine behavior or the performance of the control:

- **Startup phase:** In this part, the objective is to test the state machine and check that the bus is correctly charged to the reference voltage.
- **Load Step:** Different load steps are given to check the response of AC currents and DC voltage.
- **PFC test:** Waveforms are checked at the nominal power.
- **Protection shut-down:** The simulator is stimulated with faults (directly through the GUI) and the correct response of the control is checked, ensuring that the fault is recognized correctly.

### 5.4.1. Desktop Simulation

The first desktop simulation is carried out with Simscape electrical components at a frequency of 8.4 MHz.

The result of the startup phase is shown in Fig. 5.26.

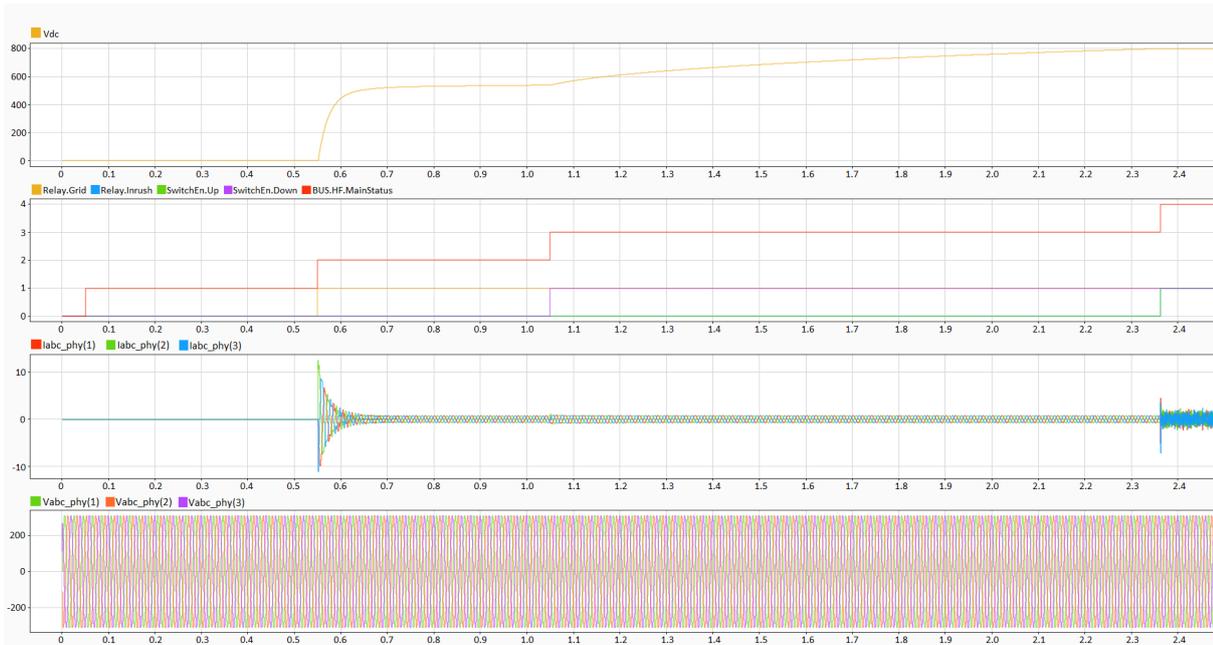


Figure 5.26: Simscape startup phase Three phase full bridge.

Starting from the top of Fig. 5.26, there is:

1. DC voltage
2. Status of the state machine, Relay grid, Relay inrush, enable of lower side switches, enable of high side switches
3. AC currents
4. AC grid voltages

At the beginning, the capacitor is discharged, so the DC voltage is 0 and the state machine status is 0 (*wait*).

After a certain time, and only if the PLL is synchronous with the voltage and the DC current is zero, the state machine moves to 1 (*Idle*).

After SM\_Main\_IDLE\_2\_INIT\_TO, the grid relay is closed (in the simulation, the grid relay is represented with a single signal that closes all relays, while in hardware simulation all three signals are considered) and the status goes to 2 (*init*).

There is an inrush current limited to 10 A thanks to  $R_{inrush}$ , and the voltage starts rising. The circuit works as a three-phase diode rectifier.

The voltage rises until it reaches the maximum  $V_{dc}$  allowed by the three-phase diode bridge.

When the voltage reaches  $SM\_Main\_INRUSH\_V\_MIN$ , the status of the state machine goes to 3 (*burst*).

In that phase, the lower switches of the converter are enabled, as seen in the second tab from the top in Fig. 5.26.

The voltage slowly rises, avoiding high currents on the AC side.

Once  $SM\_Main\_RUN\_BURST\_VREF\_V$  is reached, the state machine goes to the last status 4 (*PFC*), where the inrush relay is closed and all switches are on.

The control maintains the voltage at the reference  $V_{ctrl\_Ref}$ .

After completing the Simscape startup procedure, the same process is applied to the average model (Fig. 5.27).

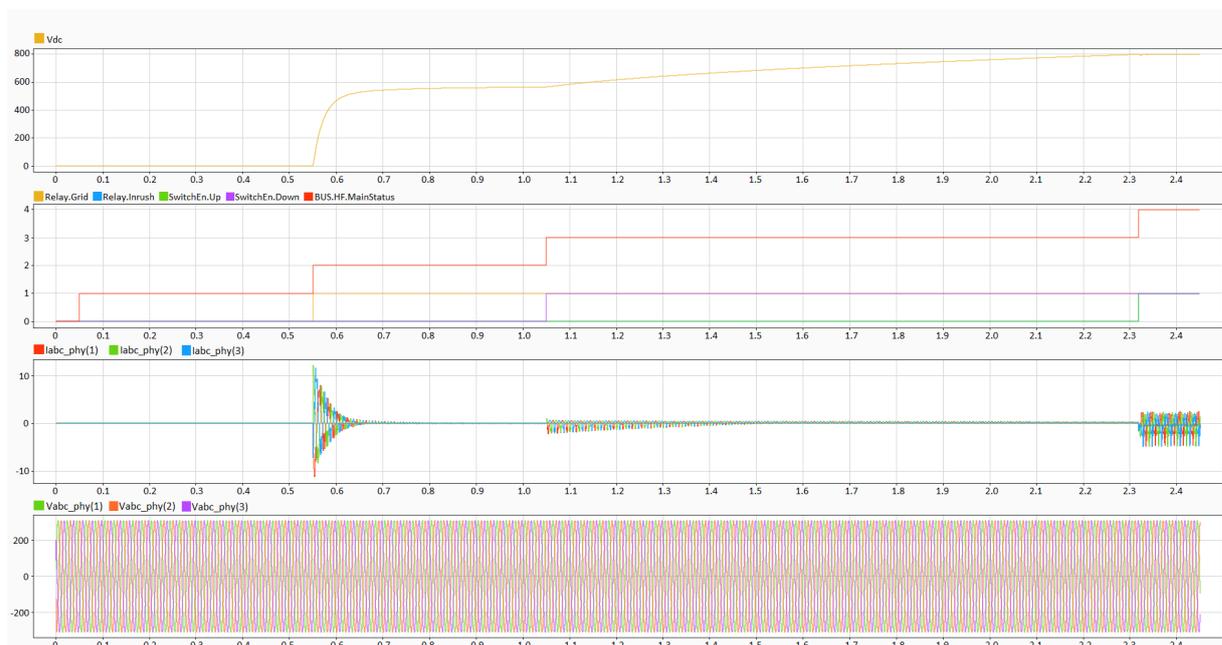
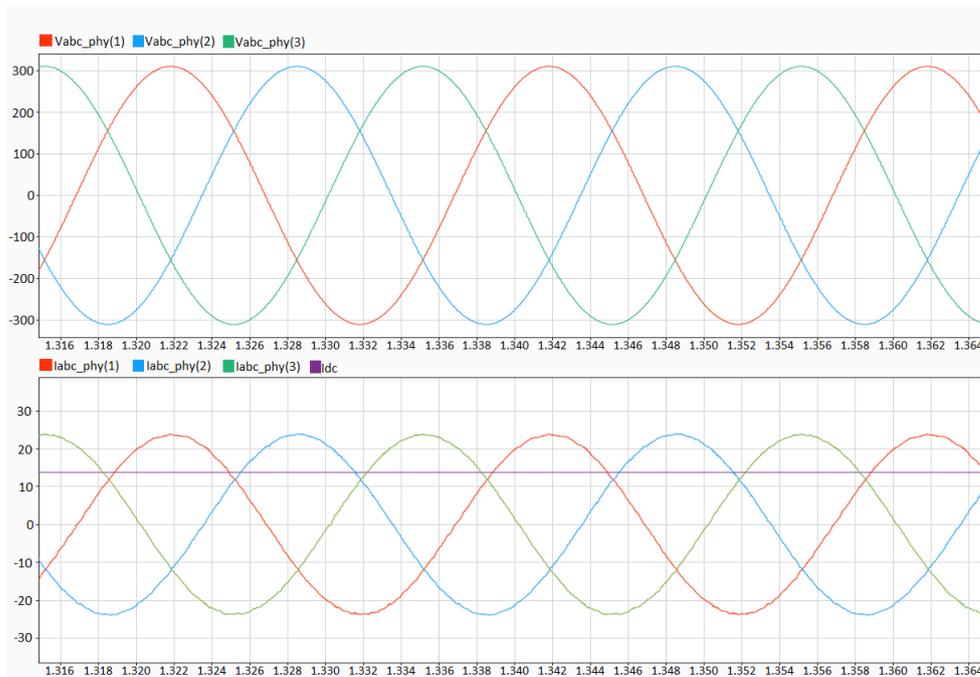


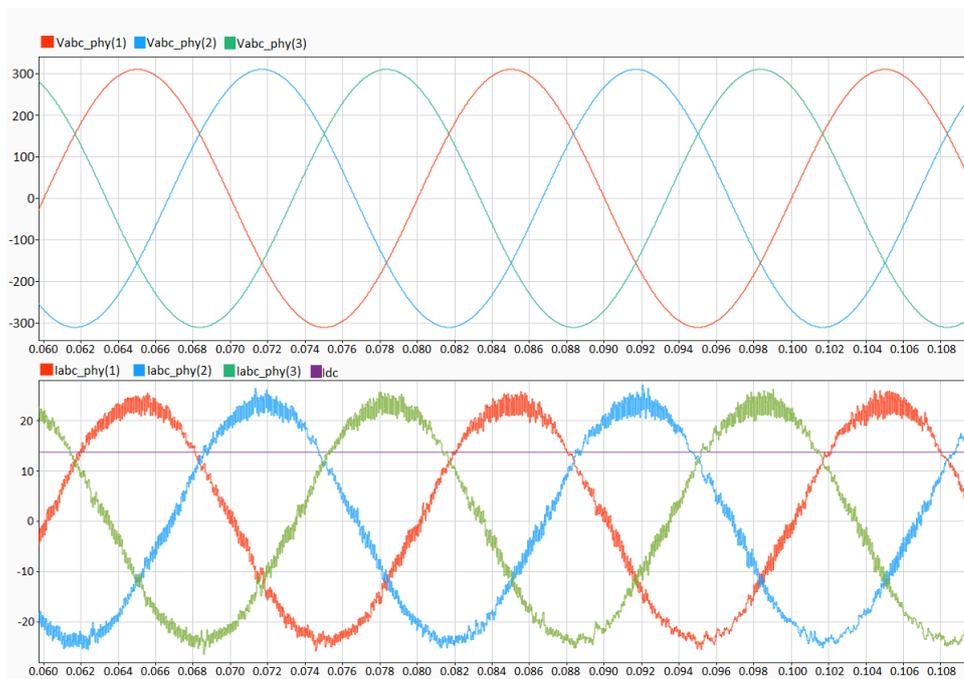
Figure 5.27: Average model startup phase Three phase full bridge.

The results shown in Fig. 5.27 and Fig. 5.26 are comparable in terms of duration and peak currents, with the advantage that the average simulation takes much less time: 0.2 sec of simulation need 3 minutes with Simscape model at 8.4 MHz and 2 second with the average model.

The next step is to test the PFC either in Simscape or in average. The converter is bidirectional and tests are carried out at 11 kW with DC current positive and negative. Fig. 5.28 and Fig. 5.29 shown results with a deadtime of 6 ns.

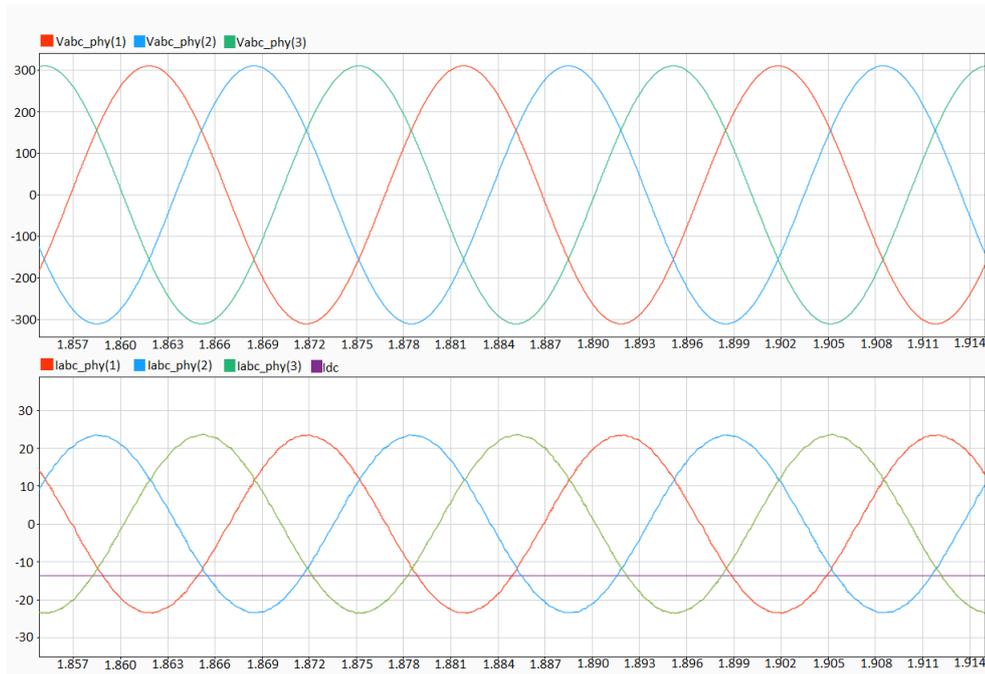


(a) Average model.

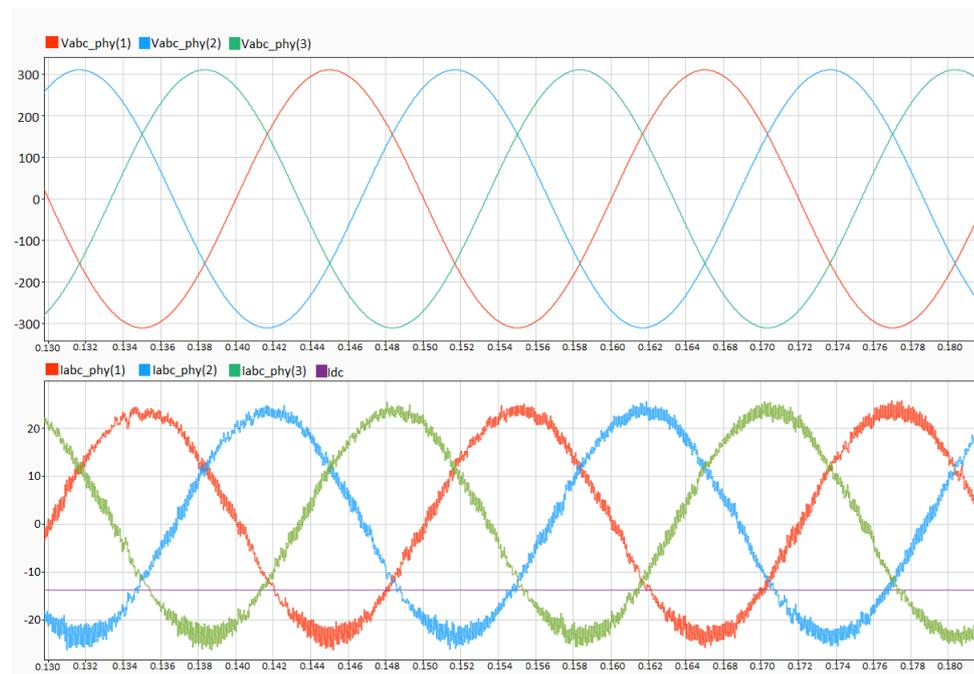


(b) Simscape model.

Figure 5.28: Simulation with deadtime = 6 ns and  $I_{dc} = 13.75$  A.



(a) Average model.



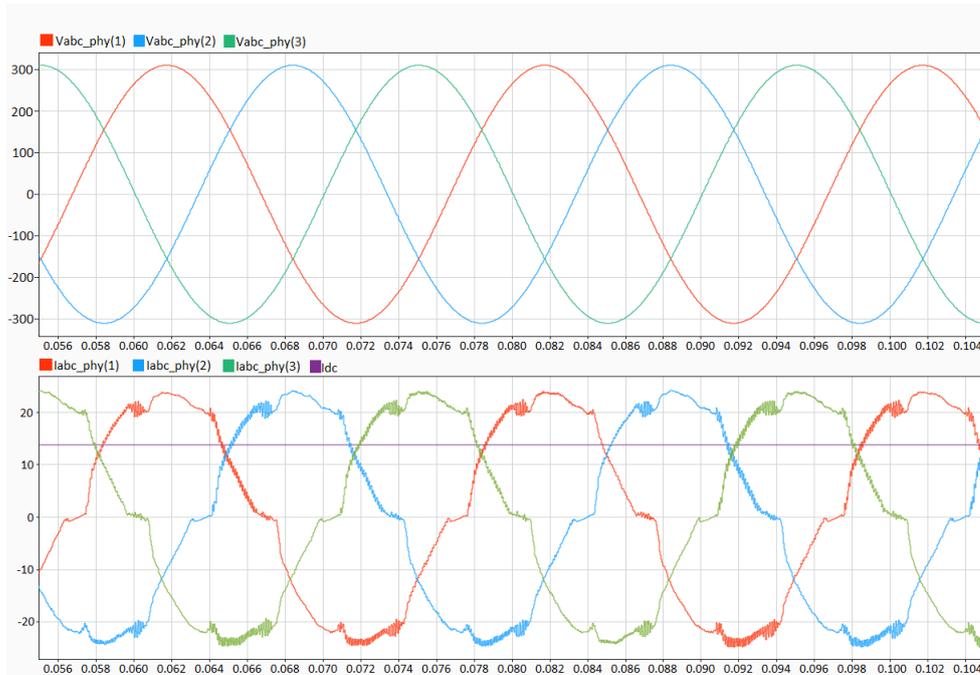
(b) Simscape model.

Figure 5.29: Simulation with deadtime = 6 ns and  $I_{dc} = -13.75$  A.

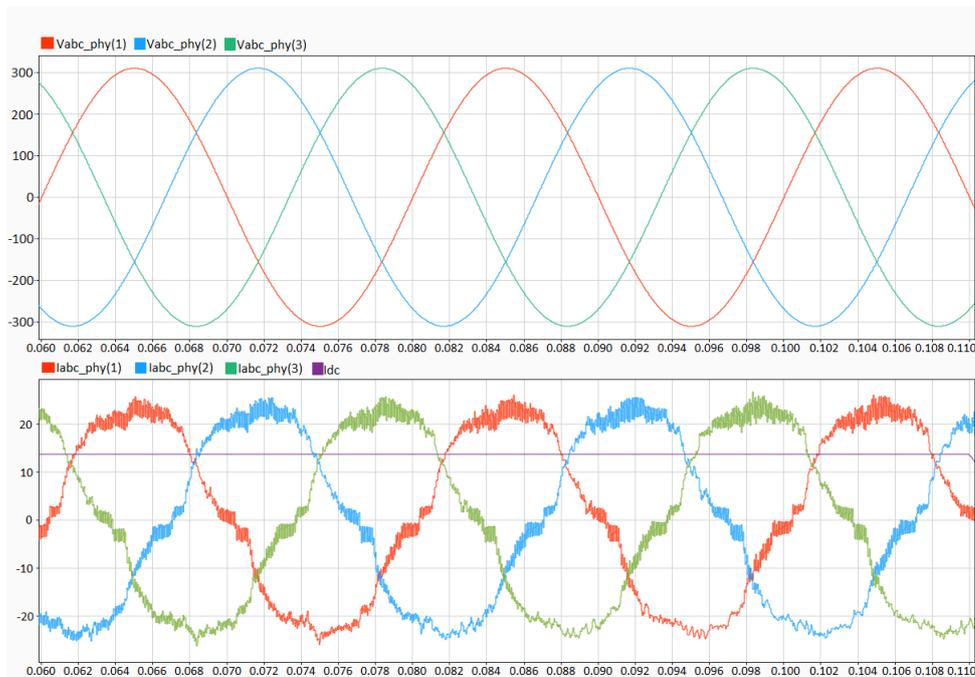
Fig. 5.28 and Fig. 5.29 demonstrates that the control is working as expected: the current is in phase with the voltage (or in counterphase when the DC current is negative) and the current absorption is sinusoidal. Furthermore, this figure validates the average model,

showing the same response as the Simscape model (except for the ripple on the current, which is not present in the average model). The deadtime is so small that the behavior of the current is not influenced.

The same simulation is carried out with a deadtime of 600 ns (Fig. 5.30 and Fig. 5.31).

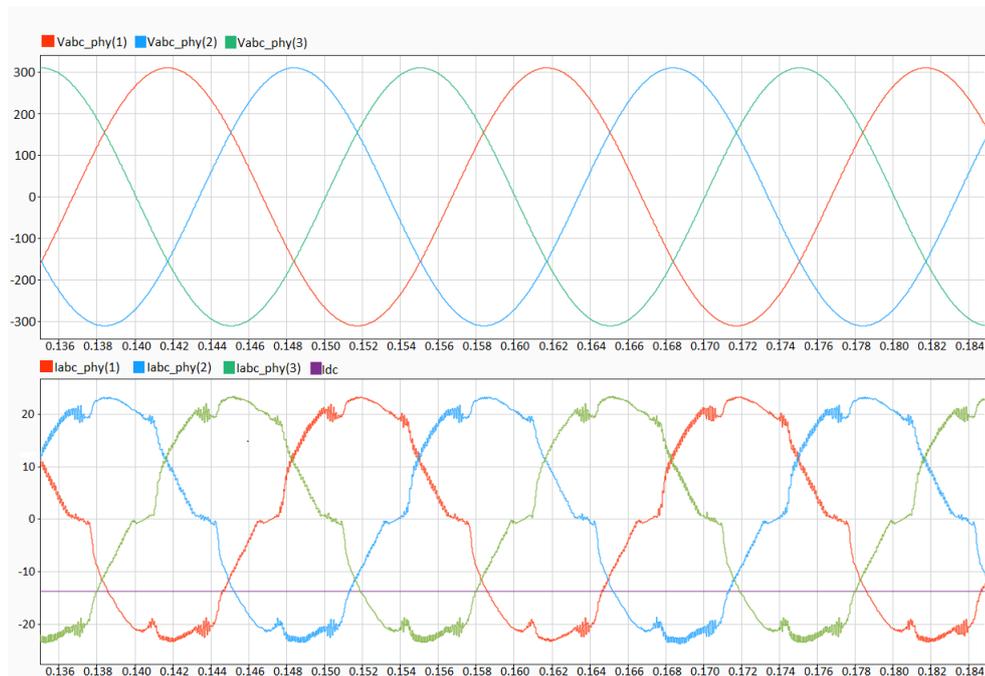


(a) Average model.

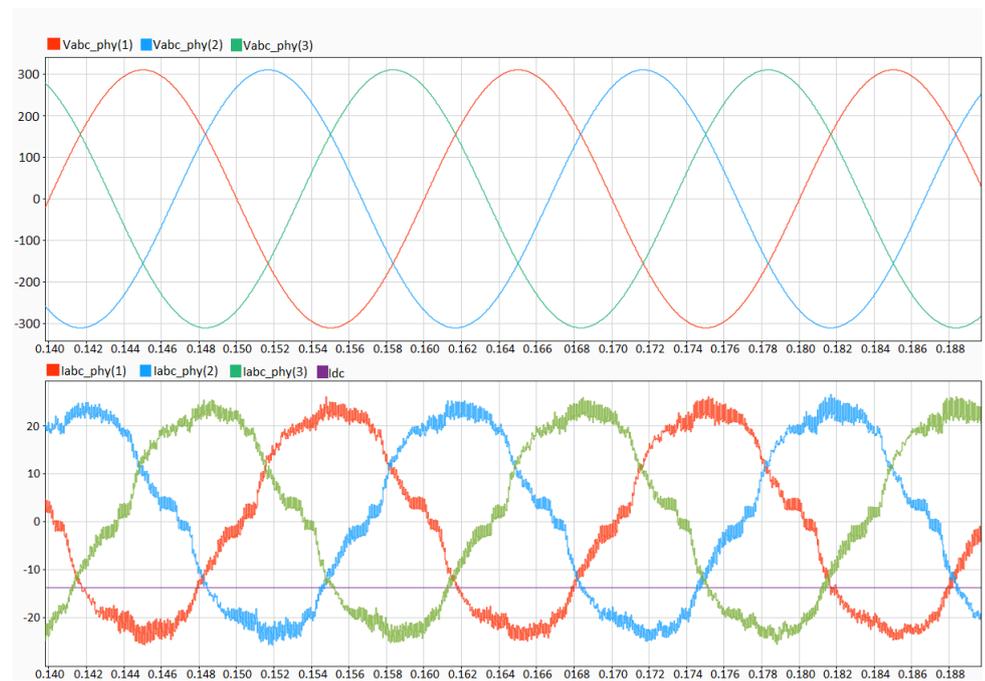


(b) Simscape model.

Figure 5.30: Simulation with deadtime = 600 ns and  $I_{dc} = 13.75$  A.



(a) Average model.



(b) Simscape model.

Figure 5.31: Simulation with deadtime = 600 ns and  $I_{dc} = -13.75$  A.

These figures validate the average model once again, as the results are comparable. The high value of deadtime significantly worsens the current waveforms, especially at the zero crossing. Despite this, the current is in phase with the voltage, and the closed-loop control

ensures the correct behavior of the converter.

### 5.4.2. Real time simulation

Once the desktop simulation reaches the desired results, a real-time simulation is carried out. Signals are shown with an oscilloscope, and the hardware used is already described in Chapter 4.

The startup phase is shown in Fig. 5.32.

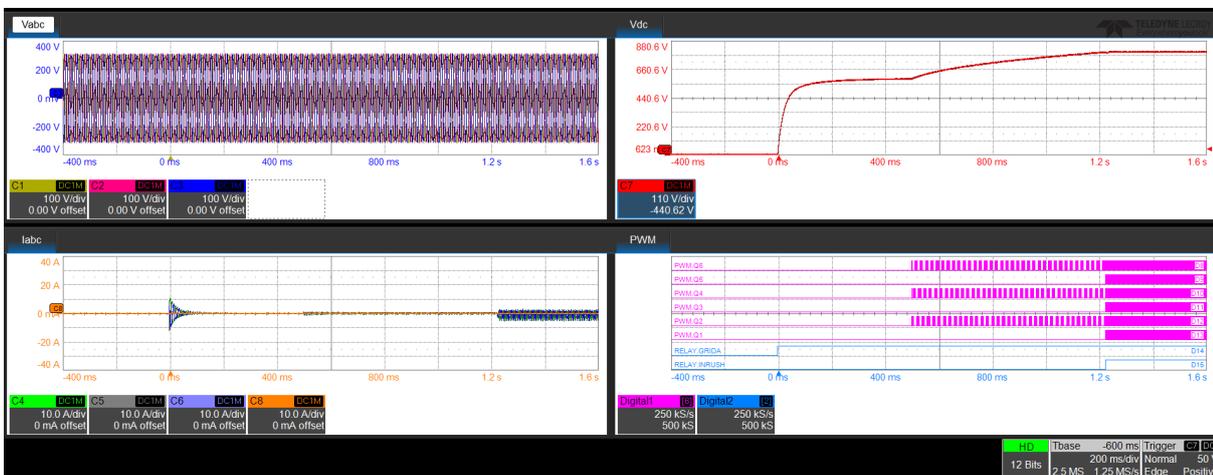
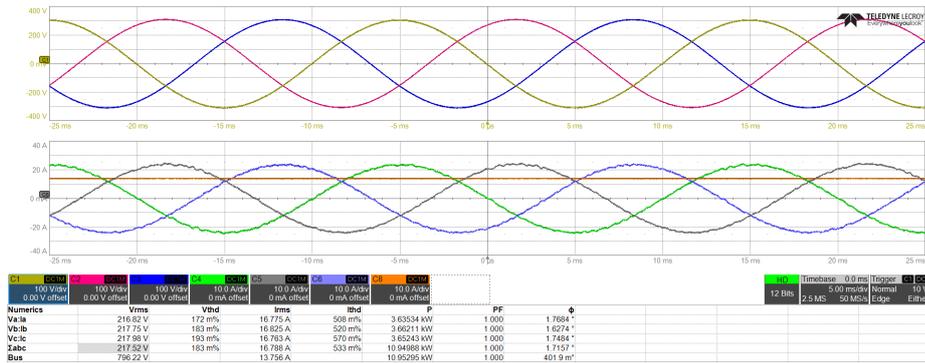


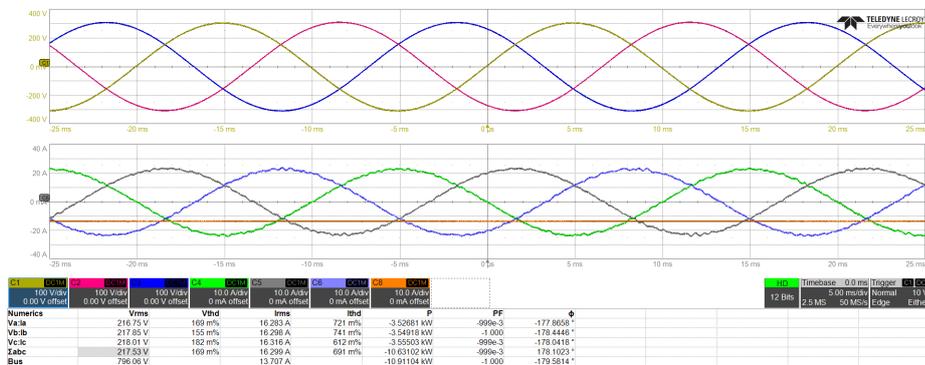
Figure 5.32: Startup of Three phase full bridge - Real Time.

The variables are shown in physical values, and the considerations are the same as those made for Fig. 5.26. In the PWM tab, the PWM signals are also shown. Thanks to that, we can appreciate the burst phase when only the lower switches (Q2, Q4, and Q6) are on, and all switches are turned off only when the DC bus voltage reaches 800 V. For simplicity, only RelayGrid A is shown, but the microcontroller also has RelayGrid B and C.

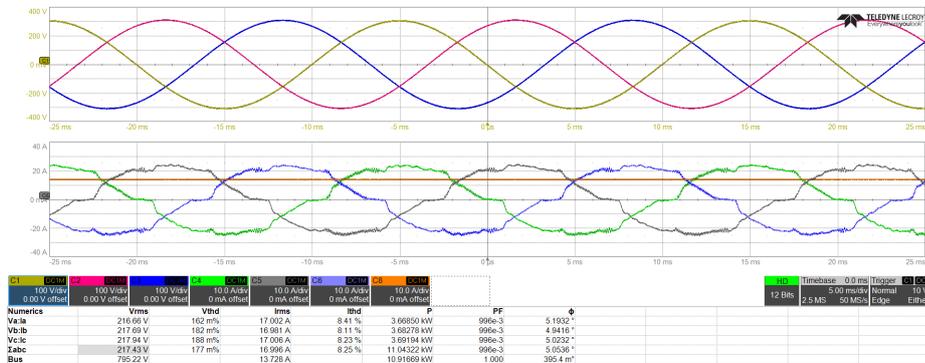
The next step is the full load test in Fig. 5.33.



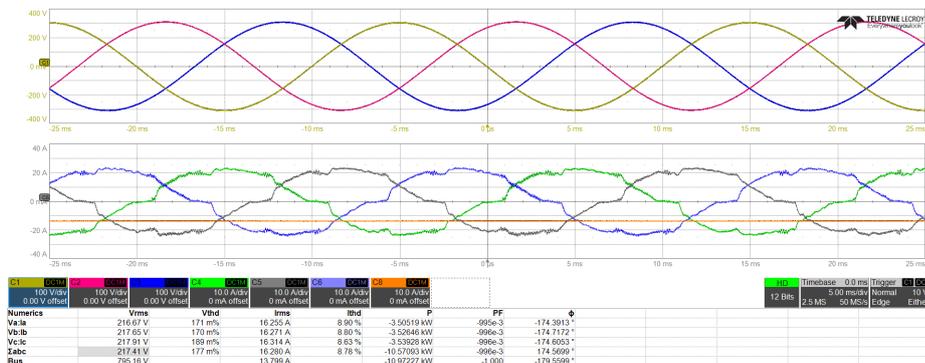
(a)  $I_{dc} = 13.75$  A and Deadtime = 6 ns.



(b)  $I_{dc} = -13.75$  A and Deadtime = 6 ns.



(c)  $I_{dc} = 13.75$  A and Deadtime = 600 ns.



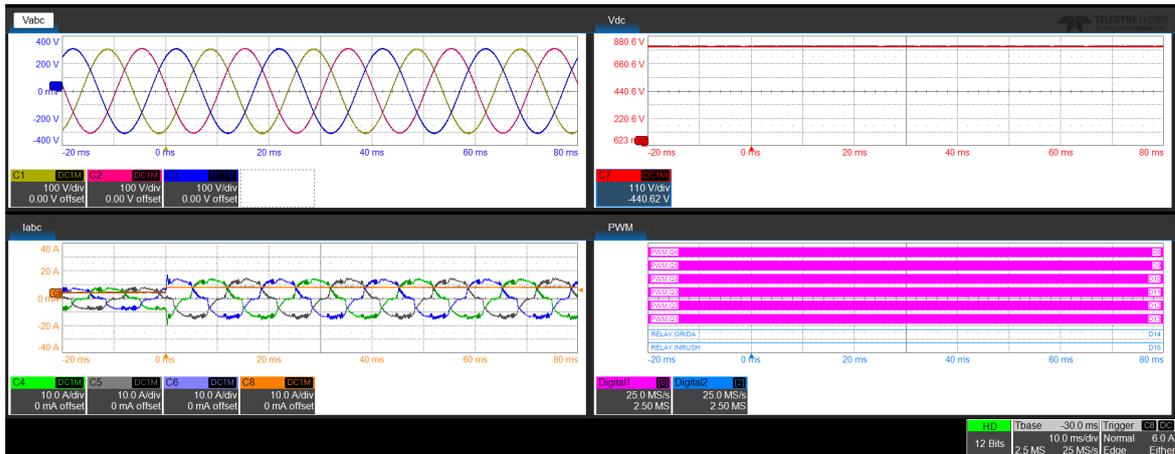
(d)  $I_{dc} = -13.75$  A and Deadtime = 600 ns.

Figure 5.33: Three phase full bridge real-time simulation at full load.

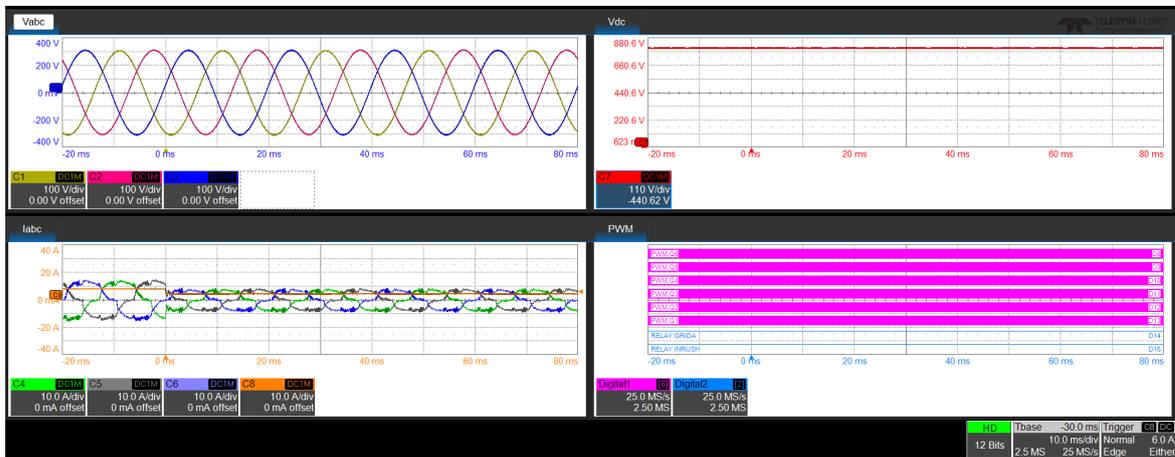
In the bottom part of each figure in Fig. 5.33, there is a 3-phase power analyzer tool that measures all currents and voltages on both the AC and DC sides. Fig. 5.33a and Fig. 6.12b show sinusoidal current absorption and a DC voltage at 800 V. Furthermore, the three-phase system is balanced, and the power absorbed from each phase is the same. The column related to the current harmonic distortion ( $I_{thd}$ ) is not realistic since we don't have the ripple on the current due to the limitations of the average model. Also, the power on the AC side is practically the same as on the DC side because the only losses we have are those simulated in the average model with  $R$ .

Fig. 5.33c and Fig. 5.33d show the results with a deadtime of 600 ns. As in the desktop simulation, the current shows a significant deterioration at the zero crossing. Nevertheless, the DC bus is at 800 V and the power is still 11 kW (or -11 kW in the case of DC-AC conversion).

In order to stress and test the performance of the control, a **load step** test is performed. In practice, a step in the DC current is applied, and the effect on voltages and currents due to the step is observed. Steps are expressed in % with respect to the nominal DC current ( $I_{dc} = 13.75$  A).

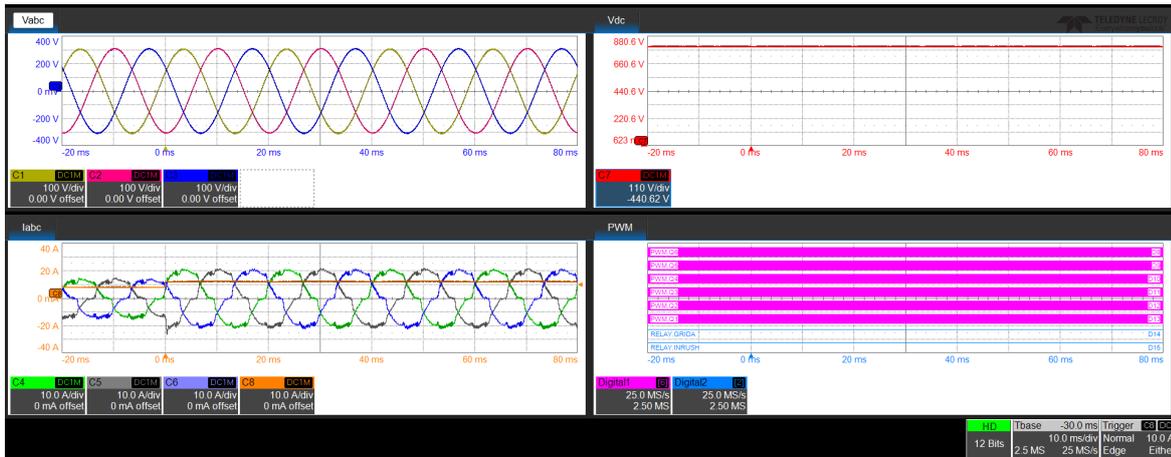


(a) 30% - 60%.

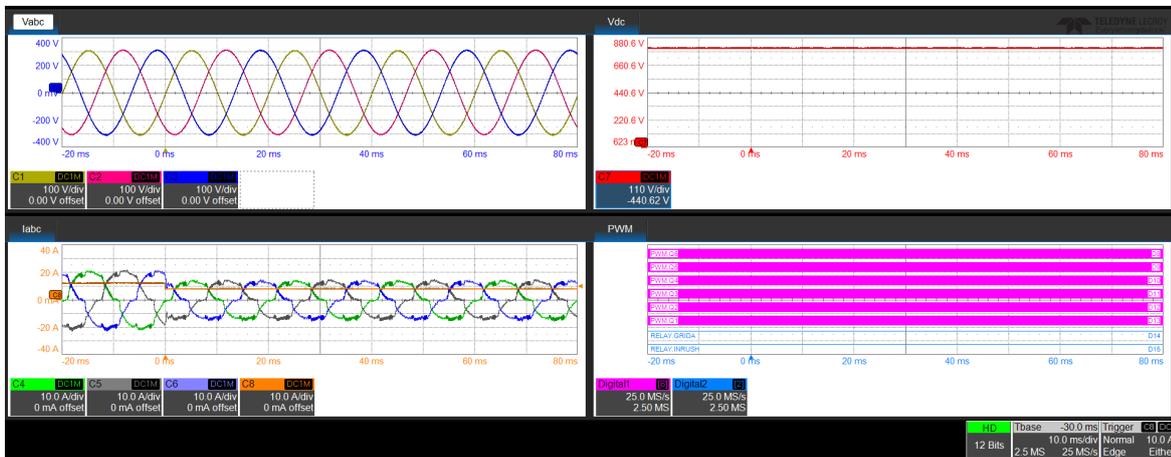


(b) 60% - 30%.

Figure 5.34: Three phase full bridge load step 30-60.

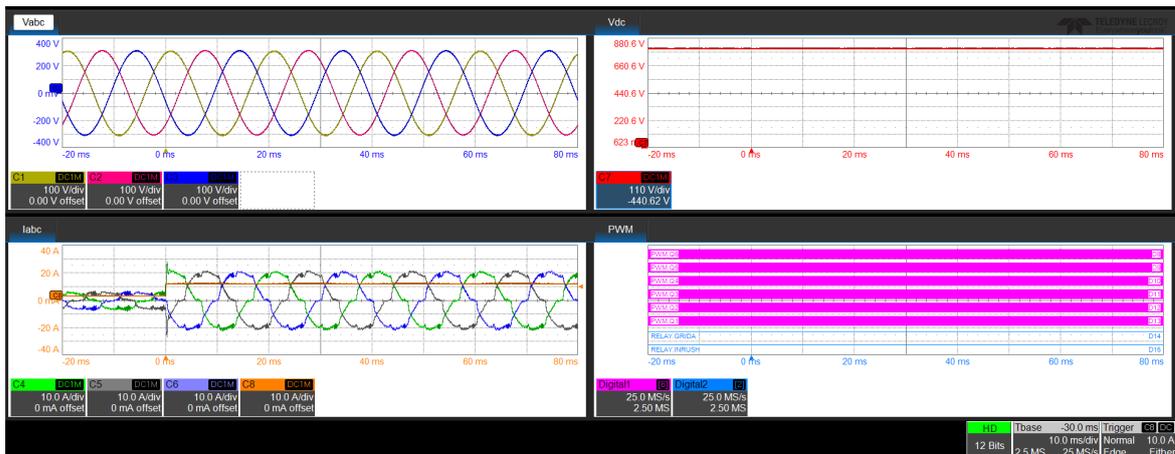


(a) 60% - 90%.

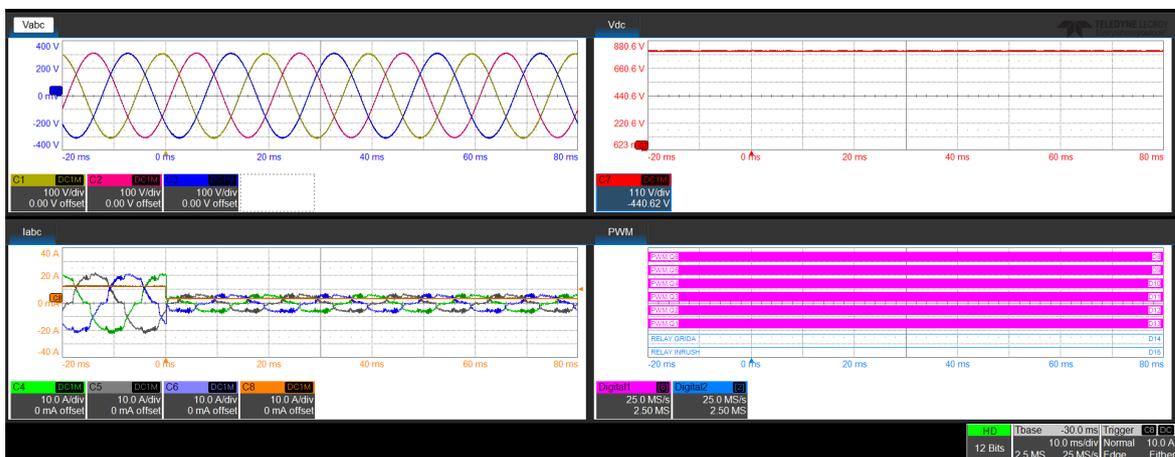


(b) 90% - 60%.

Figure 5.35: Three phase full bridge load step 60-90.



(a) 30% - 90%.



(b) 90% - 30%.

Figure 5.36: Three phase full bridge load step 90-30.

All load step results show that when the step occurs, the DC voltage does not change its value and remains at the reference voltage (800 V). Additionally, the AC currents immediately adjust their amplitude without changing their behavior.

Finally, the control is test against **faults** thanks to the GUI. The response can be seen with the oscilloscope.



Figure 5.37: Three phase full bridge - Vdc Overvoltage.

Fig. 5.37 shows a fault injection on the DC bus. The voltage rises, and the control recognizes the overvoltage, switching off all PWM signals and opening the grid relay. On the controller side, it is verified that the fault recognized is indeed an overvoltage on the DC side.



Figure 5.38: Three phase full bridge - Vac Overvoltage.

Fig. 5.38 shows an overvoltage on Va. In the Vabc tab, the yellow voltage shows a step, and at that moment, a fault is recognized, and everything is switched off as expected. Also in this case, it is verified on the controller side that the correct fault is detected.



(a) Ia overcurrent on positive values.



(b) Ia overcurrent on negative values.

Figure 5.39: Three phase full bridge - Iac overcurrent.

The last test is on AC currents. Specifically, Fig. 5.39 shows a fault injection in Ia. It is divided into positive and negative signs, which is particularly important for AC currents since the protection is managed by the analog watchdog that is set with low and high thresholds. In both cases, the control recognizes the fault. Tests on AC quantities should be repeated for every phase to ensure that everything is working correctly.

From the oscilloscope, the CPU occupation of each task is measured on both the simulation board and the control board. All these measurements are reported in Fig. 5.40.

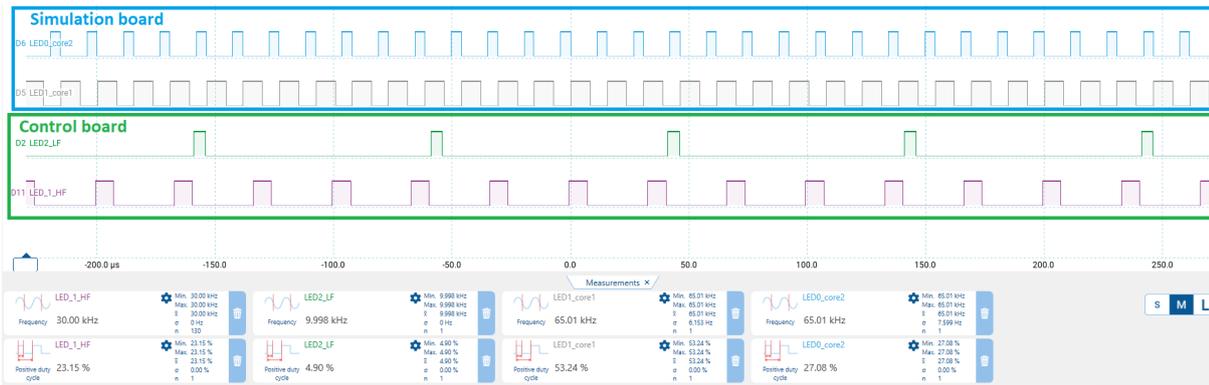


Figure 5.40: CPU usage.

Both the simulation and control boards have margin to increase the frequency of each task. In the case of the simulation, the frequency is near the switching frequency, so there are no advantages to going higher than that frequency. On the control side, it should be possible to increase the frequency of the inner loop or to move some parts of the low-frequency tasks to the higher frequency tasks.

# 6 | Application of Three Level T-Type Converter

This chapter will describe the application of a new converter with the same system (same hardware) so some parts are similar to the Three-phase full bridge and will not be repeated in this chapter.

## 6.1. Model of the system

The Three Level T-Type (TLTT from now on) is a three-level, three-phase bidirectional converter consisting of 12 MOSFETs. The detailed model is shown in Fig. 6.1.

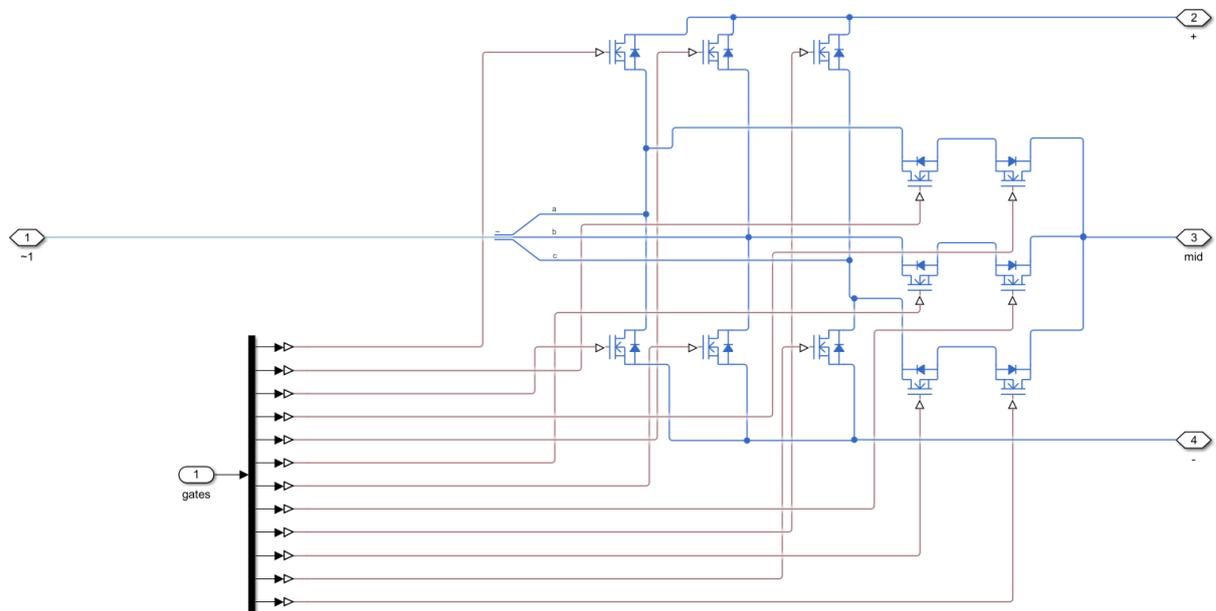


Figure 6.1: Three Level T-Type Simulink model.

The Simulink model setup is the same as the Three-phase full bridge (Fig. 5.2). The main difference is that in this case, the third input of the control is not  $V_{dc}$  but  $V_p$  and  $V_n$ , where  $V_p$  is the voltage between the mid output of the converter and the positive

terminal of the DC side, while  $V_n$  is the voltage between the midpoint and the negative terminal of the DC side.

## 6.2. Definition of the control

The control implemented for TLTT is the same as that already described for the three-phase full bridge. The main difference is the duty computation block (Fig. 3.10). This converter has 12 switches, where 6 of them are the negations of the others. In particular, from the modulation signals, 6 duty cycles are derived that go to the vertical switches, and the horizontal switches are consequently modulated. The detailed subsystem of "duty computation" of TLTT in in Fig. 6.2.

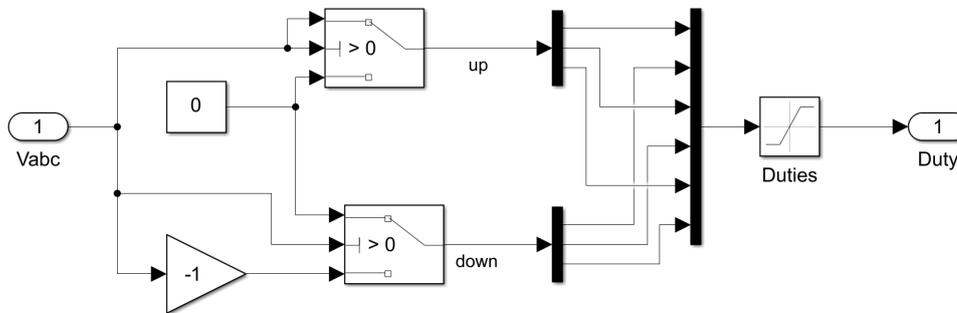


Figure 6.2: Three Level T-Type duty computation.

Of course, considering one vertical leg, the upper and lower switches cannot be on simultaneously. This was managed directly in hardware with the three-phase full bridge, while here it is done in software. The upper switch of each leg is modulated together with the left switch of the horizontal side, while the lower switch of the vertical leg is modulated with the right switch of the horizontal side.

### 6.2.1. ST Evaluation board with power mosfet

The evaluation board of the bidirectional converter is the "STDES-PFCBIDIR," which is a 15 kW, three-phase, three-level Active Front End (AFE) bidirectional converter for industrial and electric vehicle DC fast charging applications [5]. The board is shown in Fig. 6.3.

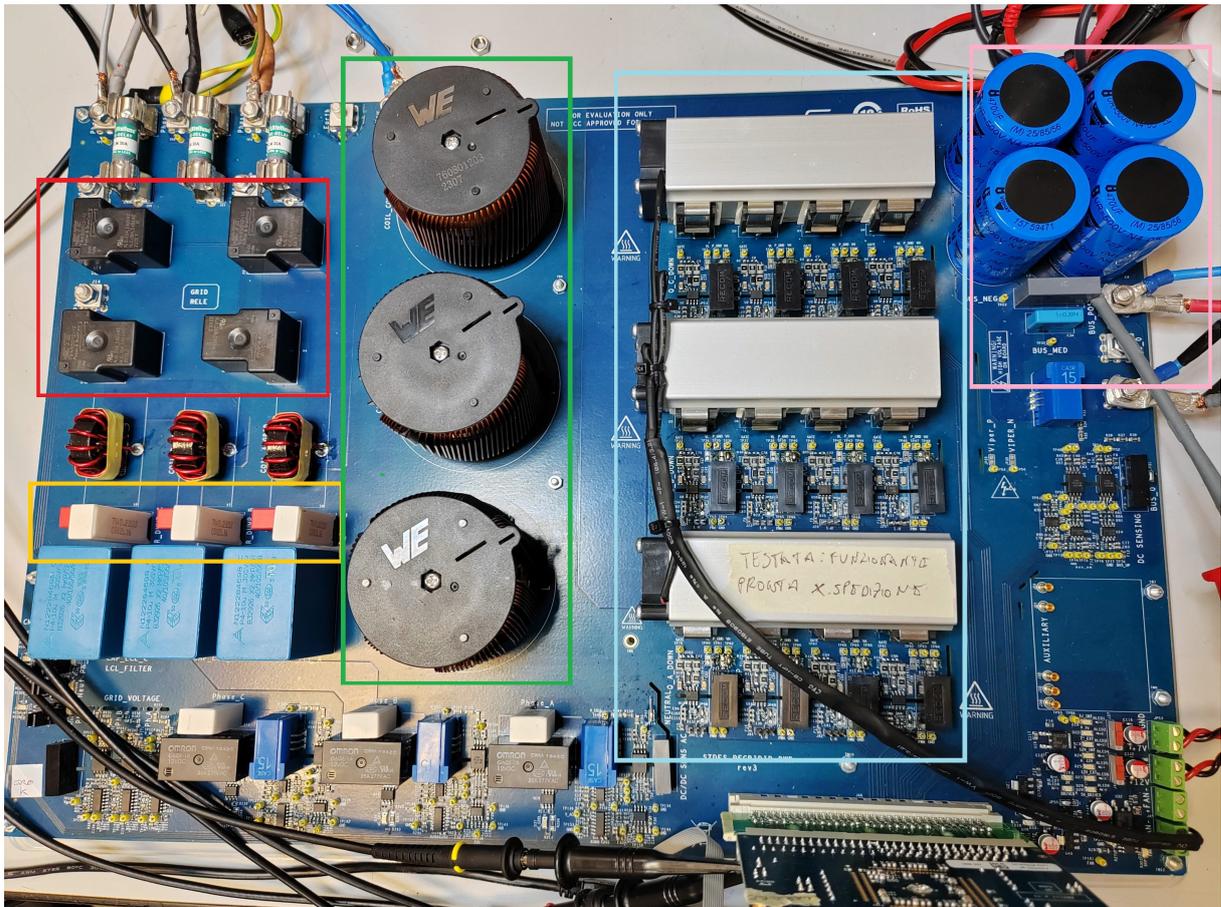


Figure 6.3: STDES-PFCBIDIR board.

From Fig. 6.3, it is possible to recognize all components described in Chapter 2:

- **Red area:** Relay grid. There are 4 relays because one is for the neutral.
- **Yellow area:** Inrush relay.
- **Green area:** Boost inductors.
- **Light blue area:** MOSFETs with gate drivers.
- **Pink area:** DC capacitors.

The connector of the board is the same as the STDES-BCBIDIR. The pin mapping is shown in Fig. 6.4. The pins used are highlighted in the yellow box.

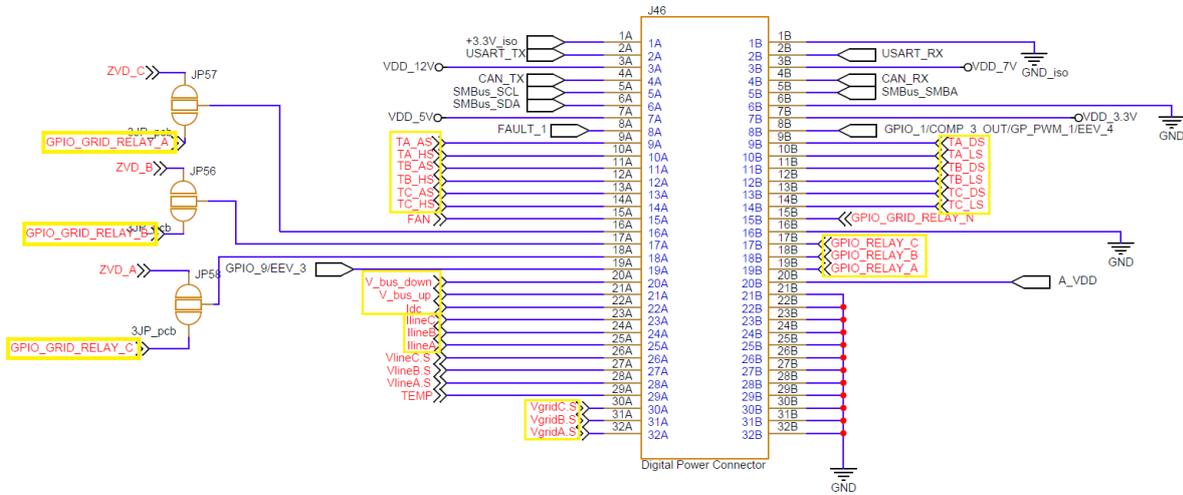


Figure 6.4: STDES-PFCBIDIR Connector.

The differences with the three-phase full bridge are that here there are 12 switches (the control board with the adapter is compatible in terms of PWM), and the inrush relay is commanded with three signals (GPIO\_RELAY\_A, GPIO\_RELAY\_B, GPIO\_RELAY\_C) instead of one. Additionally,  $V_p$  and  $V_n$  are sensed instead of  $V_{dc}$ , as already mentioned.

The IP's configuration, HF, and LF tasks are the same as in Chapter 5, so they will not be analyzed in detail here. Since TLTT has 12 switches, all timers of the HRTIM are necessary (from A to F) and channels of ADC are different.

### 6.2.2. Burst mode

During burst mode, only the horizontal switches are turned on with a fixed duty cycle. In this way, the vertical switches work as a three-phase diode bridge, and the horizontal switches allow for current absorption and charging of the DC bus. The implementation is in Fig. 6.5.

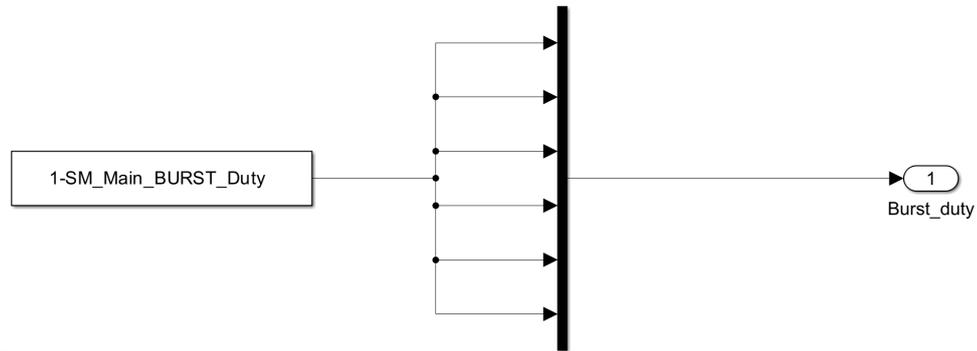


Figure 6.5: Burst mode TLTT.

The duty cycle is  $1 - duty_{burst}$  because the modulation that comes from the control goes to the vertical switches, while the horizontal ones are commanded with the negated signal, which is computed directly in hardware.

### 6.2.3. DC side Capacitor Balance

Given that the TLTT converter has a midpoint available, a load can also be connected to  $V_p$  or  $V_n$ , or there can be an imbalance that the control does not detect since the voltage reference is given in terms of the total voltage  $V_{dc}$ . A capacitor balance is implemented by adding an offset to the modulation waveforms  $V_{abc}$  before the duty computation block. The Simulink implementation of the capacitor balance is shown in Fig. 6.6.

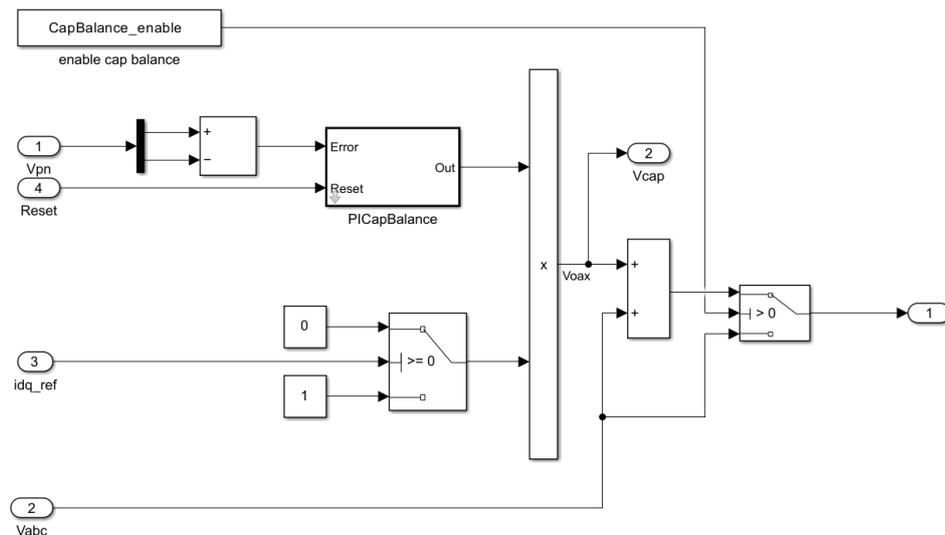


Figure 6.6: Capacitor balance.

It is implemented with a PI regulator that outputs an offset depending on the difference

between  $V_p$  and  $V_n$ , which should be zero. The capacitor balance acts only if the d-axis reference current is lower than zero.

### 6.3. Simulator of Three Level T-Type

The simulator board described in Chapter 4 is used for TLTT. The structure of the firmware is consistent with that of the three-phase full bridge, with the difference that the interrupt calls the differential equations of TLTT instead of the three-phase full bridge.

### 6.4. Simulation and real-time tests

Simulations are carried out starting from Simulink. Below is a table with parameters of the simulator (Table 6.1) derived from the board "STDES-PFCBIDIR" that can be taken directly from the ST website [6].

$R_{mos}$	75 $m\Omega$
$R_{inductor}$	50 $m\Omega$
$V_{PhaseToGroundRMS}$	220 $V$
Line Frequency	50 $Hz$
Sample time (Simulink)	1/(60 $kHz$ )
Sample time (real-time)	1/(65 $kHz$ )
Input inductor	381 $mH$
DC capacitors	1 $mF$
$R_{inrush}$	25 $Ohm$
Nominal power	15 $kW$

Table 6.1: Simulator parameters - TLTT.

From the controller side, parameters are listed in Tables 6.2, 6.3, 6.6, 6.4, and 6.5.

ADC_IAC_BIAS	2048
ADC_IAC_GAIN	42.67
ADC_IDC_BIAS	2048
ADC_IDC_GAIN	102.4
ADC_VAC_BIAS	2048
ADC_VAC_GAIN	5.505
ADC_VDC_BIAS	0
ADC_VDC_GAIN	8.6401
AWD_HT	$(-IAC\_MAX*ADC\_IAC\_G+ADC\_IAC\_B)$
AWD_LT	$(-IAC\_MAX*ADC\_IAC\_G+ADC\_IAC\_B)$
AWD_HT	$(-IAC\_MAX*ADC\_IAC\_G+ADC\_IAC\_B)$

Table 6.2: ADC parameters - TLTT.

The gain for DC voltage is called Vdc, but what is sensed is Vp and Vn.

PWM frequency	70 kHz
Deadtime	600 ns
Ictrl_ACFeedForward_enable	1
Ictrl_AWTG (anti-windup gain factor)	2034
Ictrl_Decoupling_enable	0
Ictrl_DSatNeg	-0.9
Ictrl_DSatPos	0.9
Ictrl_Ki	1017
Ictrl_Kp	0.3399
Ictrl_Linductor	400 $\mu H$
Ictrl_LineFreq	50 Hz
Ictrl_QSatNeg	-0.1
Ictrl_QSatPos	0.1
Ictrl_Freq	30 kHz

Table 6.3: Current loop parameters - TLTT.

The deadtime can be changed and adjusted depending on the real board construction and performance.

Simulations are carried out with two different deadtimes to evaluate the control performance with varying deadtimes, as these influence the behavior of AC side currents.

The cutoff frequency of the inner loop is set to 2 kHz, which is compatible with the PWM frequency.

SM_IDC_NO	0.5
SM_IDC_LOW	1
SM_IO_FAULTS_IDC_OC	19.3
SM_IO_FAULTS_V_BUS_MAX	880
SM_IO_FAULTS_V_CAP_MAX	450
SM_IO_FAULTS_VAC_PK_OV	$(250*\sqrt{2})$
SM_Main_BURST_Duty	0.1
SM_Main_BURST_ENABLE	1
SM_Main_BURST_V_MAX	820
SM_Main_IDLE_2_INIT_TO	$0.5*I_{ctrl\_Freq}$
SM_Main_INIT_2_START_TO	$0.5*I_{ctrl\_Freq}$
SM_Main_INRUSH_V_MAX	900
SM_Main_INRUSH_V_MIN	$220*\sqrt{2}*\sqrt{3}$
SM_Main_RUN_BURST_VREF_V	800
SM_VAC_PK_UV	$20*\sqrt{2}$
SM_VAC_RMS_UVLO	50

Table 6.4: State machine parameters - TLTT.

Vctrl_AWTG (anti-windup gain factor)	32
Vctrl_IdSatNeg	-0.7
Vctrl_IdSatPos	0.7
Vctrl_Ki	4
Vctrl_Kp	0.1
Vctrl_PSatNeg	-500
Vctrl_PSatPos	500
Vctrl_Freq	10 kHz
Vctrl_Ref	800 V

Table 6.5: Voltage loop parameters - TLTT.

The cutoff frequency of the outer loop is set to 40 Hz.

pllctrl_freq	50
pllctrl_Ki	20
pllctrl_Kp	5000

Table 6.6: PLL parameters - TLTT.

The PLL is in the same interrupt of the voltage loop, so the execution frequency is 10 kHz.

### 6.4.1. Desktop simulation

The desktop simulation is carried out first with Simscape at a frequency of 8.4 MHz. The startup phase is shown in Fig. 6.7.

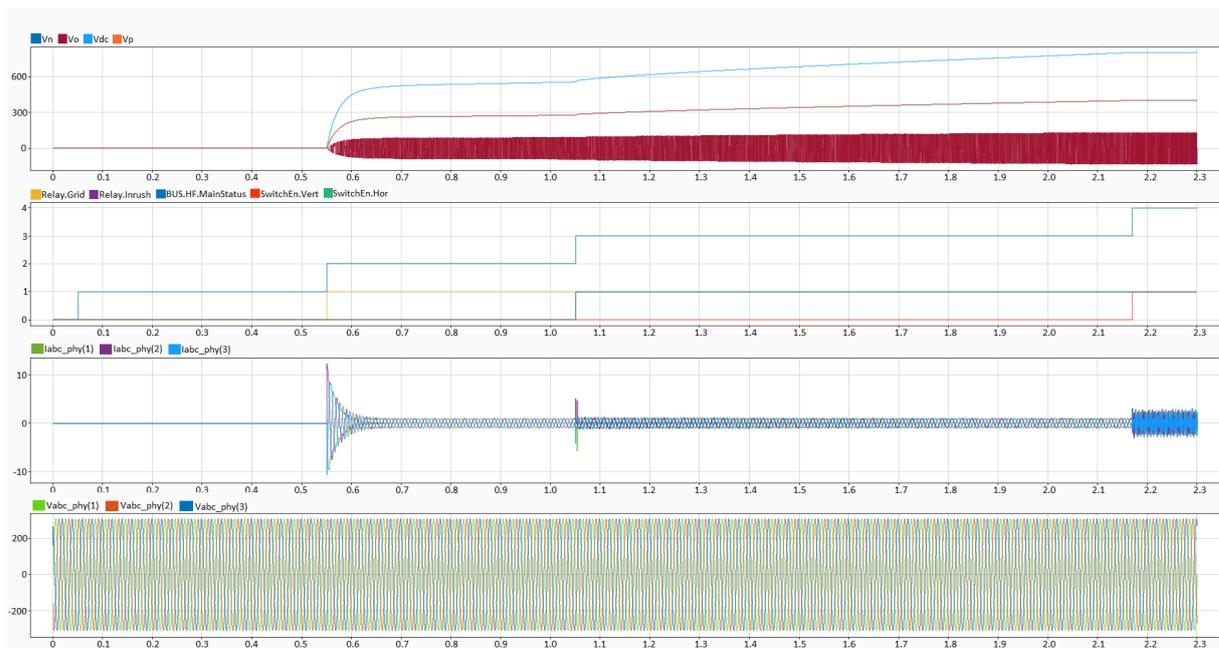


Figure 6.7: Simscape startup phase - TLTT.

The state machine is similar to the one used in the three-phase full bridge converter. Up to the burst phase, everything works in the same way. During the *burst* phase, however, the horizontal switches are turned on, and the DC voltage rises.  $V_p$  and  $V_n$  are balanced and rise with the same trend.

The average model simulation during startup is in Fig. 6.8.

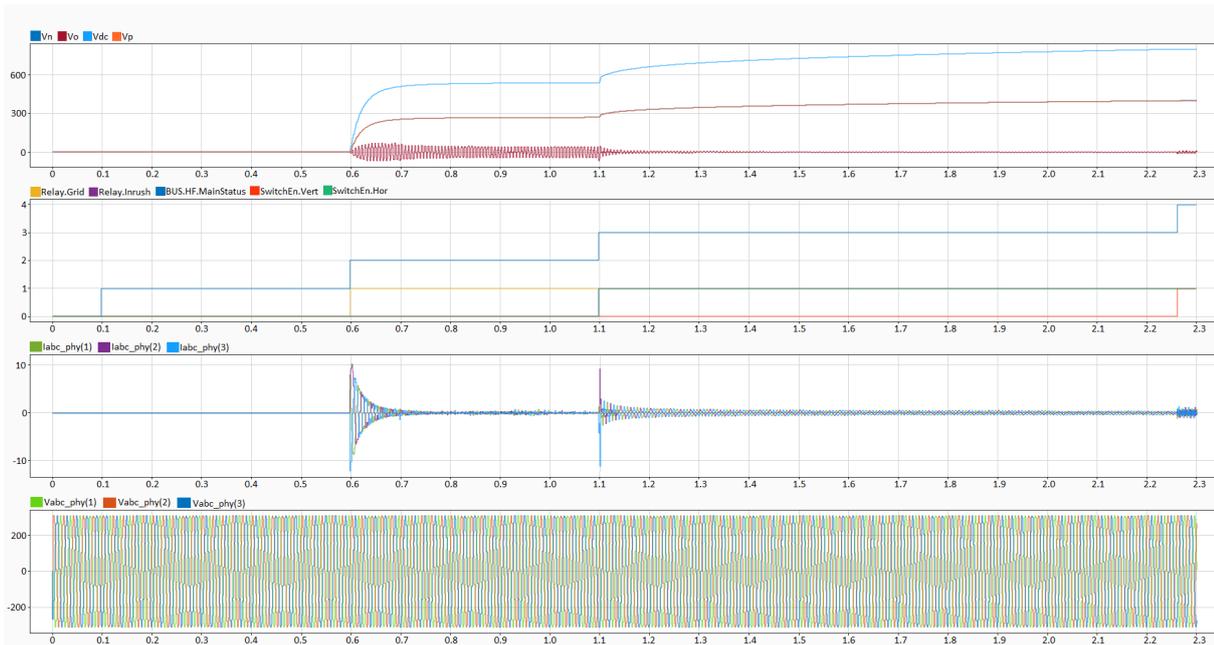
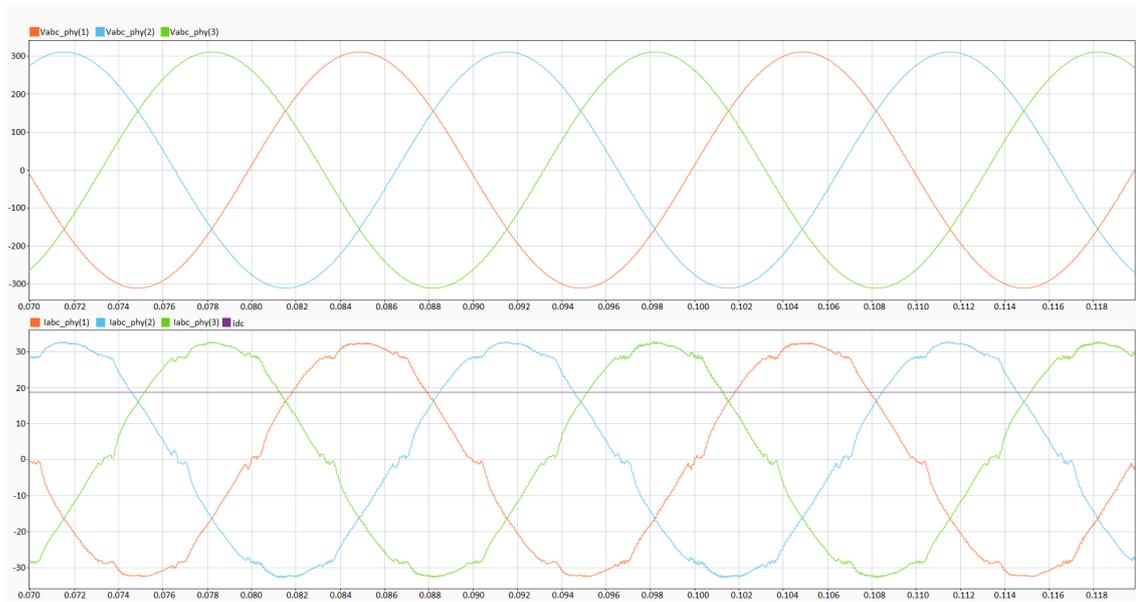


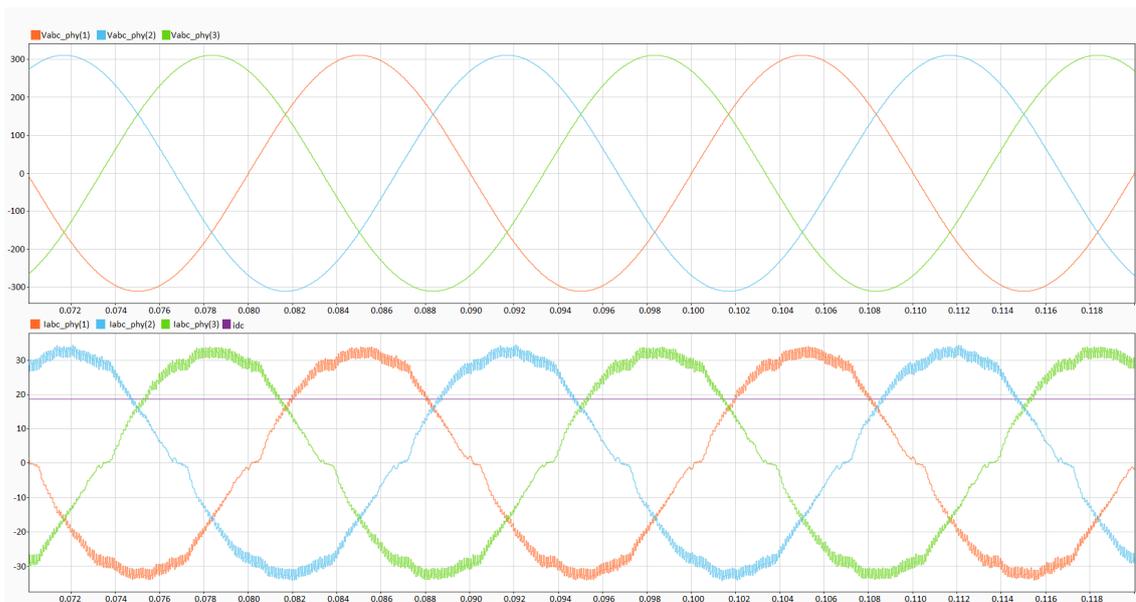
Figure 6.8: Average model startup phase - TLTT.

Fig. 6.7 and Fig. 6.8 shows same trends.

PFC tests are carried out at the nominal power of 15 kW (in both AC to DC conversion and DC to AC conversion) and are shown in Fig. 6.9 and Fig. 6.10. The deadtime value is set to 600 ns.



(a) Average model.



(b) Simscape model.

Figure 6.9: PFC test at full load -  $I_{dc} = 18.75$  A.

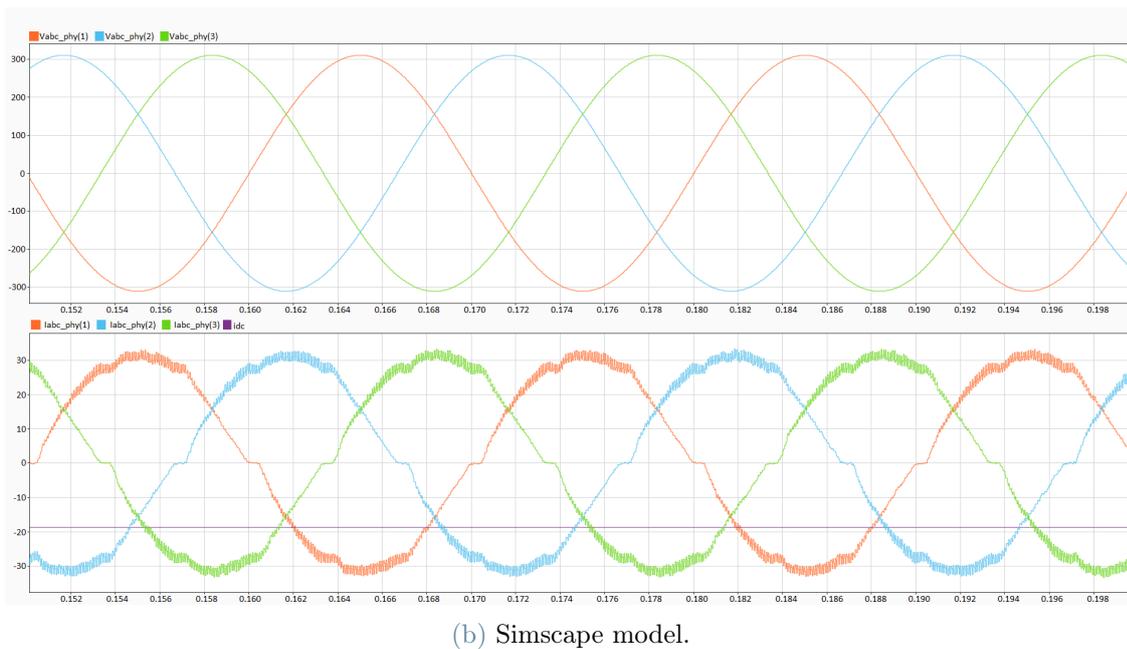
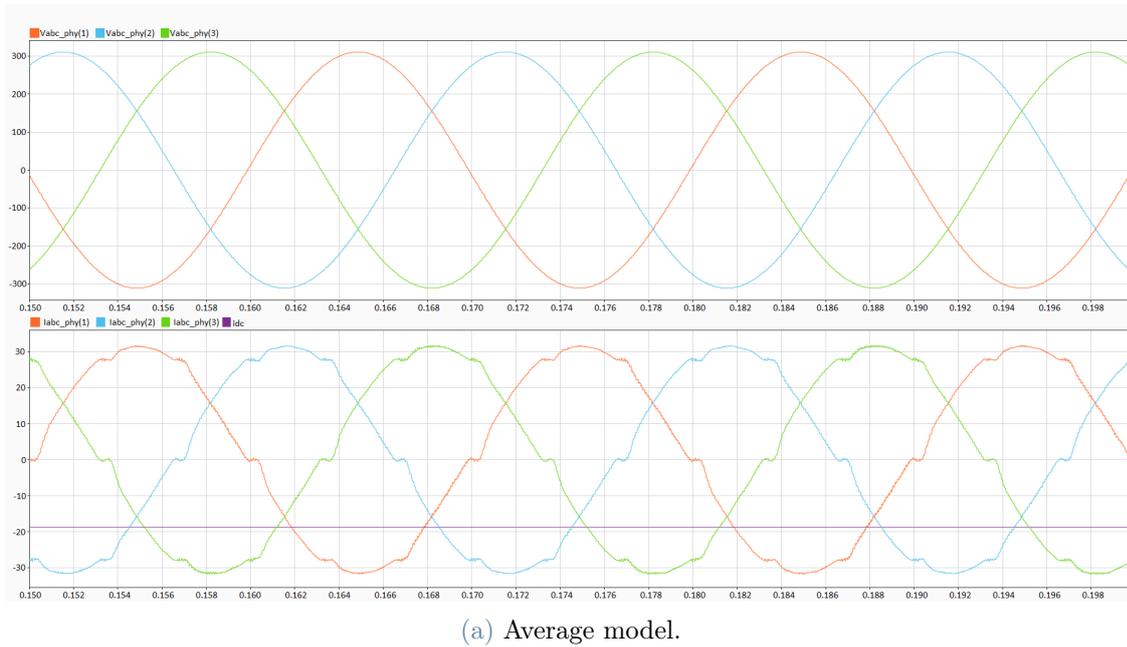


Figure 6.10: PFC test at full load -  $I_{dc} = -18.75$  A.

Except for the fact that the average model does not have ripple, the currents have the same sinusoidal trend in both models (average and Simscape). Due to the deadtime, the currents show a flattening at the zero crossing.

### 6.4.2. Real time simulation

Tests are performed in real time and results are below. The startup is in Fig. 6.11.

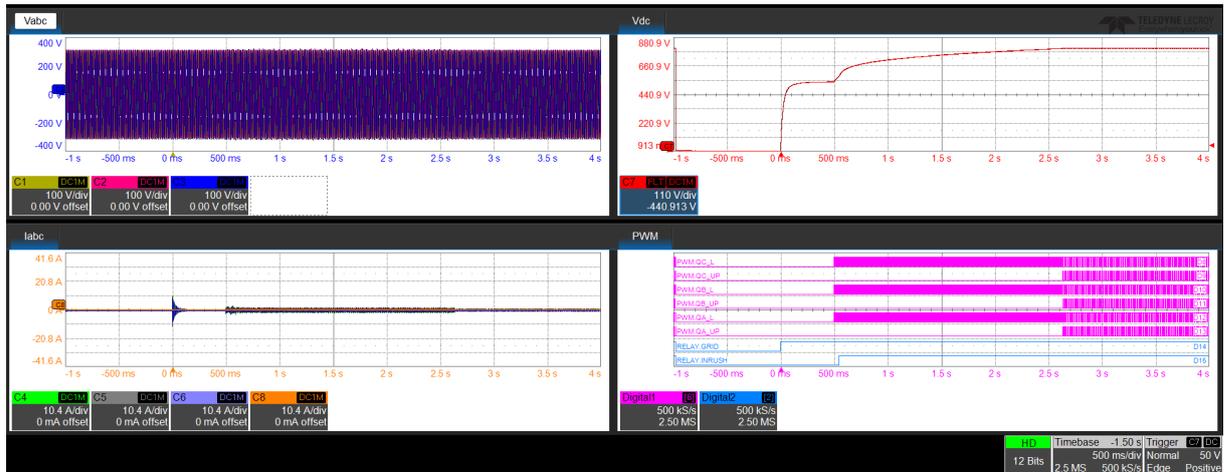
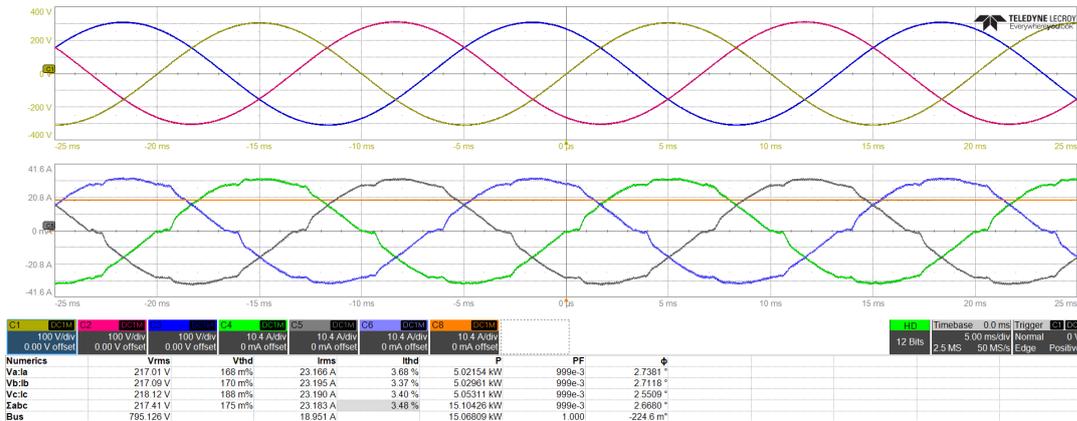


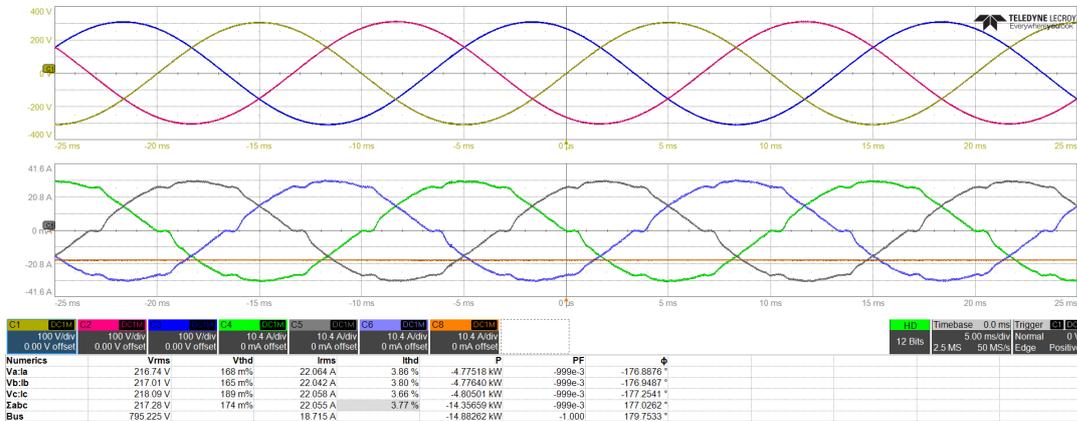
Figure 6.11: Startup of TLTT - Real Time.

Only 6 out of 12 PWMs are shown. The horizontal switches are turned on during burst mode (QA\_L, QB\_L, QC\_L), and all switches are turned on in PFC mode. The relay grid has a command for each phase (A, B, and C), but here only phase A is shown. The behavior is comparable to the desktop simulation.

The full load test is in Fig. 6.12.



(a)  $I_{dc} = 18.75$  A.



(b)  $I_{dc} = -18.75$  A.

Figure 6.12: TLTT real-time simulation at full load.

Despite having the same deadtime value, the currents of the three-phase full bridge exhibit a better sinusoidal shape. This is demonstrated by the current THD, which, although not realistic because it is an average model, provides an idea of the current distortion. The power is 15 kW (or -15 kw) and the voltage in phase with current (or in counterphase): PFC is working as expected.

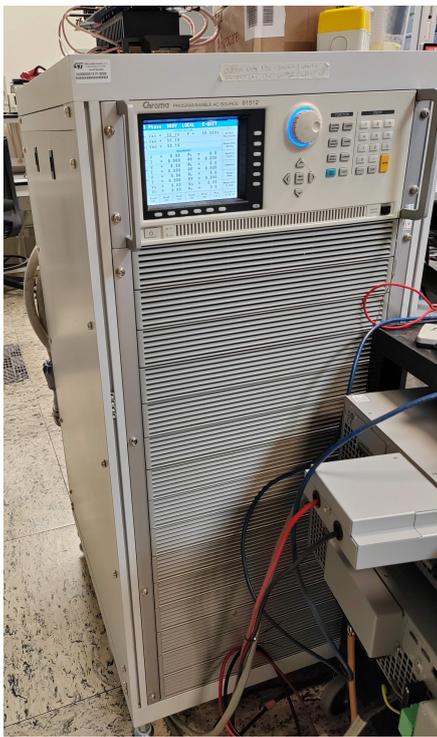
All fault are tested with the fault injection function of the GUI.

### 6.5. Lab tests with ST power board

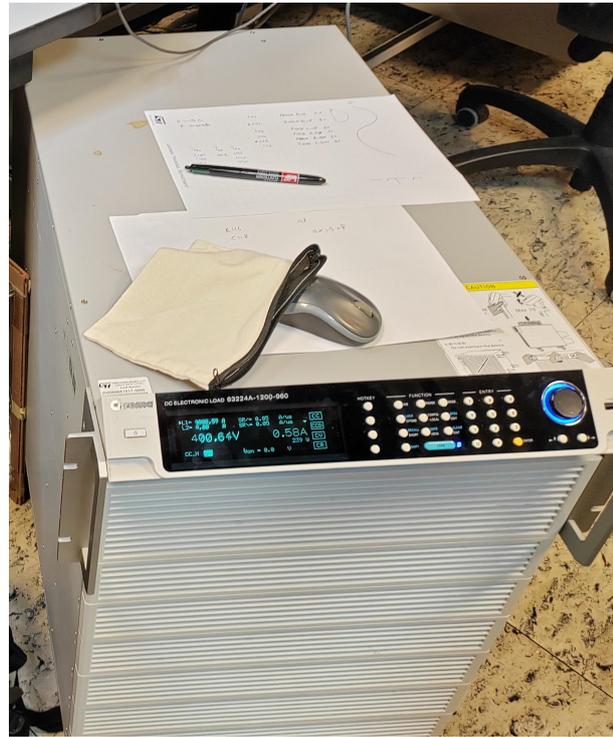
The HIL platform allows testing the control safely and without the need for a laboratory equipped with adequate instruments and power for a 15 kW converter. The control of the TLTT is tested and validate with HIL and it's ready to be tested on real board. Tests are carried out starting from half of nominal quantities that means 110 V phase to ground AC voltage and 400 V as a reference voltage on DC bus.

### 6.5.1. Setup of the lab

The board was already presented in Fig. 6.3. To complete the setup, a three phase AC generator (Fig. 6.13a) and a DC electronic load (Fig. 6.13b) is used.



(a) AC programmable source - 18 kVA.



(b) DC electronic load - 24 kW.

Figure 6.13: Laboratory instruments.

The AC programmable source is connected to the AC input of the STDES-PFCBIDIR, and the DC electronic load is connected to the DC bus of the STDES-PFCBIDIR.

The board has a low power supply section (Fig. 6.14) to supply the gate driver, relay, and control board. Using switches, it is possible to decide whether to power only the board (5 V DC supply) or also the relay and gate driver (12 V DC supply).



Figure 6.14: Low voltage supply section - STDES-PFCBIDIR.

### 6.5.2. PLL and PWM pattern

The voltages involved are high, and it is necessary to minimize risk by following a meticulous procedure and ensuring that everything is connected and set up correctly, both in terms of the connection between the board, the AC source, and the DC load, and in terms of the pin mapping of the control board.

For these reasons, the first procedure is to check the PLL in order to confirm the correct connection of the AC phases. The PLL synchronization is done by turning on the AC source at a phase-to-ground voltage of 50 V rms and turning on only the supply of the control board (5 V). Results from the oscilloscope are shown in Fig. 6.15.

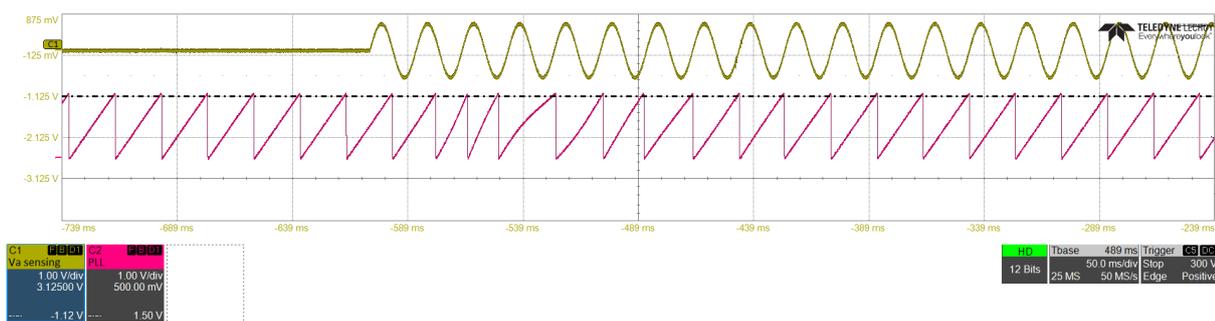


Figure 6.15: PLL synchronization.

In yellow, there is the sensing of  $V_a$ , while in pink, the angle of the PLL coming out from a DAC of the control board. When the AC source is turned on, the angle starts changing its waveform until, after 6-7 cycles of the AC voltage, the angle is aligned with the voltage  $V_a$  as expected.

Once the PLL is running correctly, the PWM pattern is checked as follows: with the AC voltage applied, everything is kept off and the control sends out PWM signals bypassing the state machine. The test points of each gate driver must match the expected pattern. After those checks also grid and drivers can be turned on

### 6.5.3. Three phase diode bridge and burst mode

Now the system is ready to start, and the relay and driver supplies are also turned on. The first check is to use the board as a three-phase diode bridge rectifier to ensure that the state machine is working correctly up to burst mode: all PWMs are disabled. In this way, the relay grid and relay inrush are closed, and the DC bus reaches the voltage of a three-phase diode bridge. After that, everything is switched off.

The next step is to test the burst mode in order to switch on horizontal relay: another step of the state machine is tested and the control board starts to control MOSFET's. Result is in Fig. 6.16.

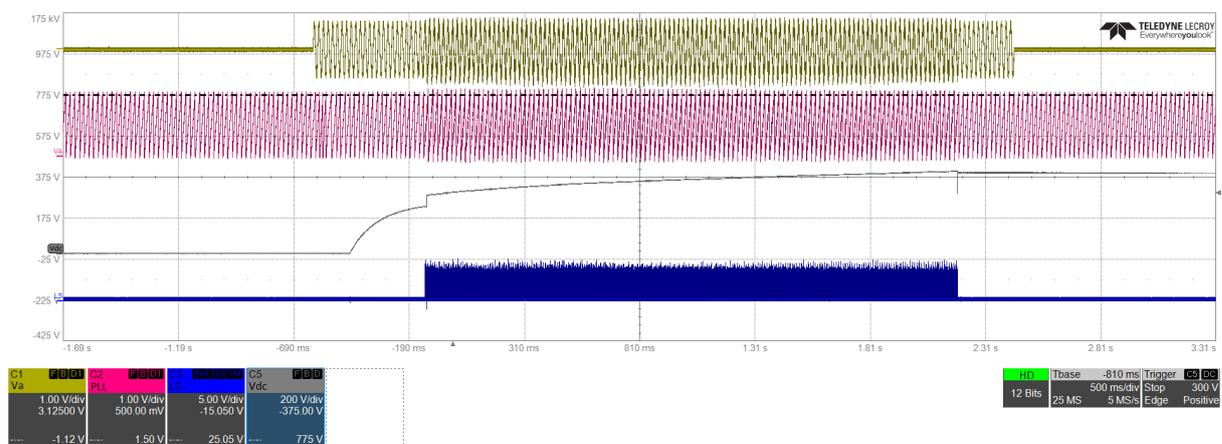
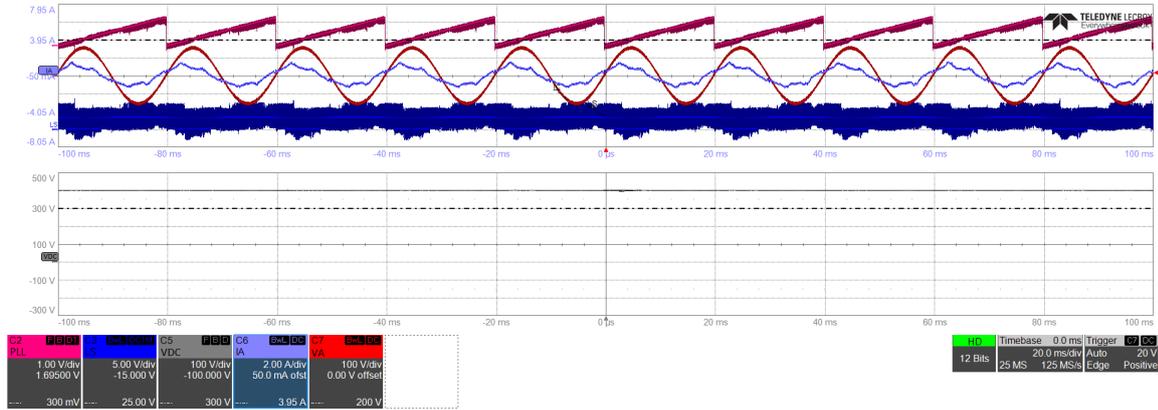


Figure 6.16: Burst test TLTT.

In **yellow** there is the sensing of  $V_a$ , in **purple** the PLL, in **grey** the DC voltage, and in **blue** the PWM of one horizontal switch. Once the grid relays are closed, the voltage starts rising until it reaches the three-phase diode bridge voltage. After that, burst mode starts, and the DC voltage rises up to the reference value (400 V). Then, the control switches off all PWM signals ( $t = 2.3$  sec) and all relays are opened. On the DC side, no load is connected, so the capacitors remain charged and the DC bus slowly rises. It is important to discharge the DC bus with an external load before restarting the system. Fig. 6.16 show the expected result.

### 6.5.4. PFC tests and PI tuning

All preliminary checks and tests have been successfully completed, and we are ready for the PFC test. A first PFC result is shown in Fig. 6.17.



(a) Oscilloscope waveforms.



(b) DC electronic load measurements.

Figure 6.17: PFC test - 240 W.

Fig. 6.17a shows in **red** the voltage  $V_a$  (in physical value and not sensed from the microcontroller) while in **light blue** the current  $I_a$  (also in physical value). In the lower part, in **grey**, the voltage of the DC bus is shown.

The DC voltage is maintained at 400 V, and we can also observe this from Fig. 6.17b, where the DC load shows the bus voltage and a current of 0.58 A. The fact that the DC

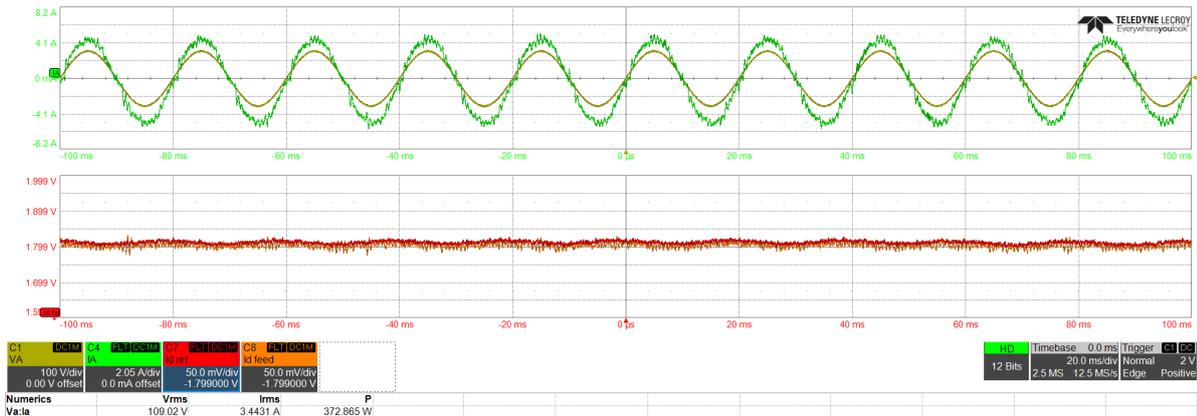
current is far from the nominal current and the test is at half of the nominal parameters means that the current in light blue is not sinusoidal. In fact, the PI tuning was done with the HIL test in a different working condition (nominal values).

The next objective is to test the control with the HIL setup but under the real board conditions (110 V on the AC side and 400 V on the DC side) in order to retune the PI controllers, particularly the current PI controller. Doing that, the new values for the PI controller are the following:

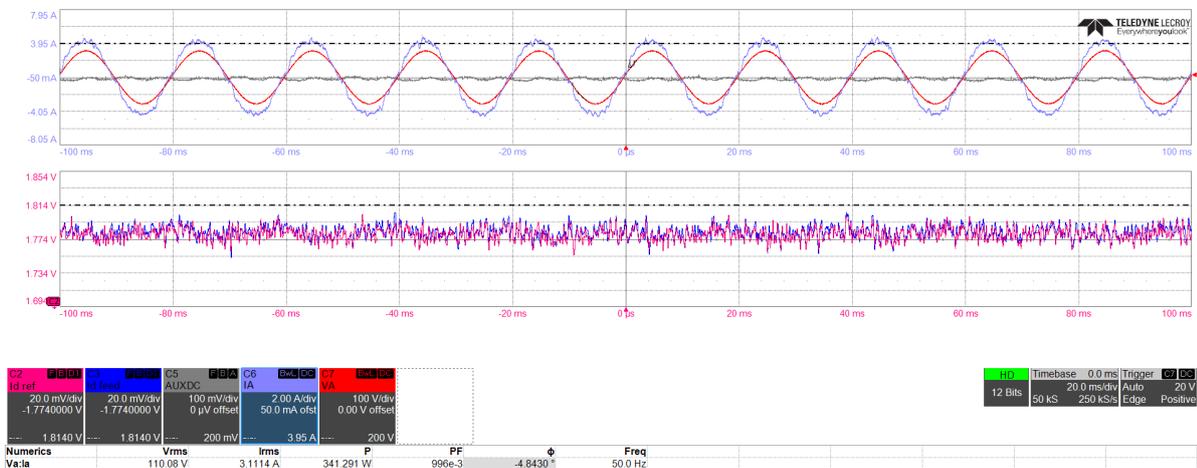
Ictrl_AWTG (anti-windup gain factor)	=Ictrl_Ki
Ictrl_Ki	5214
Ictrl_Kp	0.7088

Table 6.7: Current loop PI - 400 V.

A new test is carried out with those parameters and the result is shown in Fig. 6.18.



(a) Results with HIL.



(b) Results with STDES-PFCBIDIR.

Figure 6.18: 1 kW test - TLTT.

Fig. 6.18a and Fig. 6.18b have the same setup on the oscilloscope. On the high side of the oscilloscope, there is the voltage  $V_a$  and the current  $I_a$  in physical value (with the difference that in Fig. 6.18b variables are taken directly from the circuit while in Fig. 6.18a they are taken from test points of the control board and then rescaled through the oscilloscope). On the low side,  $i_{d\_ref}$  and  $i_{d\_feed}$  coming from DAC1 and DAC2 of the control board are shown. The power of phase A is shown in the bottom part of the oscilloscope. Assuming a balanced system, the power is about 1 kW. The current is sinusoidal,  $i_{d\_feed}$  follows  $i_{d\_ref}$ , and the results are comparable.

# 7 | Conclusions and future developments

This thesis has demonstrated the feasibility and effectiveness of a Hardware-in-the-Loop (HIL) platform developed for real-time digital power conversion simulation, with a specific focus on electric vehicle on-board chargers. Utilizing a model-based design approach, the HIL system successfully integrated control algorithms that address key challenges in power conversion, achieving both stability and high performance within the constraints of a microcontroller. The robust implementation of control and fault management systems highlights the potential of HIL environments as a reliable tool for real-time testing in power electronics applications.

The final experimental validation conducted with the real board was a pivotal step in verifying both the model-based development process and the HIL platform itself. This test demonstrated the platform's capability to replicate real-world operating conditions, effectively emulating the dynamics of actual power circuits. The experiment not only underscores the platform's readiness for complex testing scenarios but also confirms its value in supporting future innovations in digital power conversion.

Future developments could focus on enhancing the detail and accuracy of the switching component simulation, specifically the MOSFET. Improving the fidelity of the MOSFET model would allow the platform to capture switching transients and other nuances more accurately, thereby enabling more precise control algorithm testing and fault management strategies. This enhancement would extend the applicability of the HIL platform, making it a more powerful tool for advanced research and development in electric vehicle charging systems and broader power conversion technologies.



## Bibliography

- [1] W. Elektrtronik. We-torpfec toroidal pfc chokes, 2023. URL <https://www.we-online.com/en/components/products/WE-TORPFC>.
- [2] S. Y. P. Sarhadi. State of the art: hardware in the loop modeling and simulation with its applications in design, development and implementation of system and control software. *Dynamics and Control*, pages 470–479, 2024.
- [3] D. Pejovski. Dc microgrids operation and design. Course at Politecnico di Milano, 2023.
- [4] STMicroelectronics. 1200 v acepack dmt-32 power modules based on sic mosfet technology, 2023. URL [https://www.st.com/content/st\\_com/en/campaigns/stpower-sic-mosfet-based-1200v-acepack-dmt-32-automotive-power-modules.html](https://www.st.com/content/st_com/en/campaigns/stpower-sic-mosfet-based-1200v-acepack-dmt-32-automotive-power-modules.html).
- [5] STMicroelectronics. Stdes-pfcbidir, 2023. URL <https://www.st.com/en/evaluation-tools/stdes-pfcbidir.html#documentation>.
- [6] STMicroelectronics. Edesign suite for stdes-pfcbidir, 2023. URL <https://eds.st.com/vienna-bidir/>.
- [7] STMicroelectronics. *SR5E1 Reference Manual*, 2024. URL <https://www.st.com/en/automotive-microcontrollers/sr5e1e3.html#documentation>.
- [8] STMicroelectronics. Stdes-bcbidir, 2024. URL <https://www.st.com/en/evaluation-tools/stdes-bcbidir.html#overview>.
- [9] H. Wouters and W. Martinez. Bidirectional onboard chargers for electric vehicles: State-of-the-art and future trends. *IEEE Transactions on Power Electronics*, 39(1): 693–716, 2024. doi: 10.1109/TPEL.2023.3319996.



## List of Figures

1.1	Closed loop control system block diagram for an ideal HIL [2]. . . . .	1
1.2	Precision, time and cost trade-off in computer simulation, HIL simulation, and actual situation [2]. . . . .	2
1.3	EV main components [9]. . . . .	3
2.1	Equivalent circuit of boost inductor. . . . .	7
2.2	MOSFET schematic diagram. . . . .	8
2.3	Full System. . . . .	8
2.4	Simulink model. . . . .	9
2.5	Simulink average model. . . . .	11
2.6	Simulink average model - Interpreted function. . . . .	11
2.7	Gate signal. . . . .	13
2.8	Application timer configuration. . . . .	14
2.9	Check events. . . . .	15
2.10	Duty cycle computation. . . . .	16
2.11	Interrupt delay. . . . .	16
2.12	Core 2 application callbacks. . . . .	18
2.13	Second order low pass filter. . . . .	19
3.1	<i>Variant source</i> and <i>variant sink</i> blocks. . . . .	22
3.2	Controller subsystem. . . . .	22
3.3	Phase Locked Loop. . . . .	24
3.4	Equivalent single-phase circuit [3]. . . . .	24
3.5	Inner control loop. . . . .	25
3.6	CurrentControl subsystem. . . . .	26
3.7	ACFeedforward subsystem. . . . .	26
3.8	PI controller. . . . .	27
3.9	Voltage loop. . . . .	28
3.10	Duty computation. . . . .	29
3.11	IO FAULTS SM. . . . .	30

4.1	SR5E1 Control board. . . . .	34
4.2	Control board input filter. . . . .	35
4.3	Simulation board. . . . .	36
4.4	SR5E1 adapter. . . . .	40
4.5	Final control board. . . . .	41
4.6	Final hardware setup. . . . .	42
5.1	Three-phase full bridge Simulink model. . . . .	43
5.2	Full Simulink model of the three-phase full bridge. . . . .	44
5.3	Plant subsystem. . . . .	45
5.4	PFC connector of STDES-BCBIDIR. . . . .	46
5.5	ADC driver configuration. . . . .	47
5.6	ADC regular sequence configuration. . . . .	48
5.7	ADC trigger source [7]. . . . .	49
5.8	HRTIM configuration. . . . .	50
5.9	HRTIM timer unit. . . . .	50
5.10	HRTIM outputs configuration. . . . .	51
5.11	Output waveforms. . . . .	52
5.12	ADC trigger configuration. . . . .	53
5.13	TIMER configuration. . . . .	54
5.14	DAC configuration. . . . .	55
5.15	GPIO configuration. . . . .	55
5.16	Controller tasks. . . . .	56
5.17	Analog watchdog event configuration. . . . .	57
5.18	Counter and capture register format vs clock prescaling factor [7]. . . . .	58
5.19	Check limit of comparator. . . . .	59
5.20	Actuator output. . . . .	59
5.21	Burst mode for Three Phase Full Bridge. . . . .	60
5.22	.h file with defined variables. . . . .	61
5.23	Scheme of Three phase full bridge system. . . . .	62
5.24	CCM and DCM. . . . .	64
5.25	Simulator GUI. . . . .	67
5.26	Simscape startup phase Three phase full bridge. . . . .	72
5.27	Average model startup phase Three phase full bridge. . . . .	73
5.28	Simulation with deadtime = 6 ns and $I_{dc} = 13.75$ A. . . . .	74
5.29	Simulation with deadtime = 6 ns and $I_{dc} = -13.75$ A. . . . .	75
5.30	Simulation with deadtime = 600 ns and $I_{dc} = 13.75$ A. . . . .	76

5.31	Simulation with deadtime = 600 ns and $I_{dc} = -13.75$ A. . . . .	77
5.32	Startup of Three phase full bridge - Real Time. . . . .	78
5.33	Three phase full bridge real-time simulation at full load. . . . .	79
5.34	Three phase full bridge load step 30-60. . . . .	81
5.35	Three phase full bridge load step 60-90. . . . .	82
5.36	Three phase full bridge load step 90-30. . . . .	83
5.37	Three phase full bridge - Vdc Overvoltage. . . . .	84
5.38	Three phase full bridge - Vac Overvoltage. . . . .	84
5.39	Three phase full bridge - Iac overcurrent. . . . .	85
5.40	CPU usage. . . . .	86
6.1	Three Level T-Type Simulink model. . . . .	87
6.2	Three Level T-Type duty computation. . . . .	88
6.3	STDES-PFCBIDIR board. . . . .	89
6.4	STDES-PFCBIDIR Connector. . . . .	90
6.5	Burst mode TLTT. . . . .	91
6.6	Capacitor balance. . . . .	91
6.7	Simscape startup phase - TLTT. . . . .	95
6.8	Average model startup phase - TLTT. . . . .	96
6.9	PFC test at full load - $I_{dc} = 18.75$ A. . . . .	97
6.10	PFC test at full load - $I_{dc} = -18.75$ A. . . . .	98
6.11	Startup of TLTT - Real Time. . . . .	99
6.12	TLTT real-time simulation at full load. . . . .	100
6.13	Laboratory instruments. . . . .	101
6.14	Low voltage supply section - STDES-PFCBIDIR. . . . .	102
6.15	PLL synchronization. . . . .	102
6.16	Burst test TLTT. . . . .	103
6.17	PFC test - 240 W. . . . .	104
6.18	1 kW test - TLTT. . . . .	106



## List of Tables

4.1	64-pin connector. . . . .	39
4.2	New PWM configuration. . . . .	40
5.1	Simulator parameters. . . . .	68
5.2	ADC parameters - Three-phase full bridge. . . . .	69
5.3	Current loop parameters - Three-phase full bridge. . . . .	69
5.4	State machine parameters - Three-phase full bridge. . . . .	70
5.5	Voltage loop parameters - Three-phase full bridge. . . . .	71
5.6	PLL parameters - Three-phase full bridge. . . . .	71
6.1	Simulator parameters - TLTT. . . . .	92
6.2	ADC parameters - TLTT. . . . .	93
6.3	Current loop parameters - TLTT. . . . .	93
6.4	State machine parameters - TLTT. . . . .	94
6.5	Voltage loop parameters - TLTT. . . . .	94
6.6	PLL parameters - TLTT. . . . .	95
6.7	Current loop PI - 400 V. . . . .	105



## Acknowledgements

Special thanks to STMicroelectronics for providing me with the opportunity and the tools to carry out this thesis, offering a valuable and enriching work experience.

Ringrazio la mia famiglia per avermi dato la possibilità di intraprendere questo percorso.

Ringrazio il mio relatore, il Professor Piegari, che mi ha seguito, con disponibilità e gentilezza in ogni fase della realizzazione della tesi.

Ringrazio il mio Tutor e Correlatore Daniele Caltabiano per avermi dato questa immensa opportunità, per aver sempre creduto in me e soprattutto per avermi insegnato tanto. Porterò sempre con me il bagaglio di conoscenze che mi hai trasmesso.

Ringrazio i colleghi di ST con cui ho condiviso tanti momenti in questi mesi. Vi ringrazio per avermi accolto e messo a mio agio.

Ringrazio i miei amici, quelli di sempre e quelli universitari, per avermi alleggerito e distratto e per esserci stati in questo percorso.

Ringrazio Lilli per avermi supportato e ascoltato in ogni momento ed essere cresciuti insieme in questi anni.

