

# Norvento HIL emulator

## User Manual

Author: Jorge Rodríguez de Frutos

Revision: Antonio Lázaro Blanco

Leganés, 03/12/2018



## General Index:

<b>Figure index:</b>	<b>5</b>
<b>Table index:</b>	<b>7</b>
<b>Introduction</b>	<b>8</b>
1.1 Objectives	8
<b>2 Scope</b>	<b>8</b>
<b>3 Mathematical Models</b>	<b>9</b>
3.1 Blocks equations	9
3.1.1 <i>Grid</i>	9
3.1.2 <i>LCL filter</i>	10
3.1.3 <i>Circuit Breakers</i>	11
3.1.4 <i>Grid Inverter and generator inverter</i>	12
3.1.5 <i>DC Bus and Chopper</i>	14
3.1.6 <i>PMSM</i>	15
3.2 PSIM blocks	17
3.2.1 <i>Grid</i>	18
3.2.2 <i>LCL filter</i>	18
3.2.3 <i>Circuit Breakers</i>	19
3.2.4 <i>Grid Inverter</i>	19
3.2.5 <i>DC Bus and Chopper</i>	20
3.2.6 <i>Generator Inverter</i>	20
3.2.7 <i>PMSM</i>	21
3.3 PSIM template for simulation of the complete B2B system blocks	24
<b>4 Vivado HLS Models</b>	<b>28</b>
4.1 General concepts	28
4.1.1 <i>Data type, Clocking and time step</i>	28
4.1.2 <i>Data flow and interfacing ports</i>	29
4.2 Description of the main blocks	31
4.2.1 <i>Grid</i>	31
4.2.2 <i>LCL filter</i>	31
4.2.3 <i>Circuit Breakers</i>	32
4.2.4 <i>Grid and PMSM Inverter</i>	32
4.2.5 <i>DC Bus and Chopper</i>	32

4.2.6	<i>PMSM</i> .....	33
4.3	Source Code Release .....	33
<b>5</b>	<b>Vivado Project</b> .....	<b>34</b>
5.1	IP integrator overview .....	34
5.2	IC drivers IPs and conversion IPs .....	38
5.3	User Console communications IPs.....	38
5.4	Start IP.....	39
5.5	PWM signals.....	40
<b>6</b>	<b>SDK and User Console</b> .....	<b>41</b>
6.1	General concepts .....	41
6.1.1	<i>Parameter list and parameter format</i> .....	43
6.1.2	<i>Communication description</i> .....	43
6.2	SDK communication protocol .....	44
6.3	Qt communications protocol .....	45
6.4	Source Code Release.....	46
<b>7</b>	<b>RTS Carrier Board</b> .....	<b>48</b>
7.1	General overview and description of the main blocks .....	48
7.2	<i>NTC sensors emulation</i> .....	51
7.3	<i>Grounding</i> .....	52
7.4	<i>Lay-out and signal integrity</i> .....	53
7.5	PSC Carrier Boar Project release .....	53
7.6	<i>Current amplifiers and DAC chain</i> .....	53
<b>8</b>	<b>Annex A: rebuild Vivado project:</b> .....	<b>56</b>
8.1	Introduction .....	56
8.2	Path assignments .....	56
8.3	Vivado HLS .....	56
8.4	Vivado .....	57
8.5	SDK .....	57
8.6	Notes.....	59
<b>9</b>	<b>Annex B: Current driver in depth analysis:</b> .....	<b>60</b>
9.1	General concepts .....	60
9.2	G_input (s) calculation .....	61
9.3	G_LMH6321(s) calculation .....	62

9.4	G_AD8130(s) calculation.....	64
9.5	Closed loop analysis.....	65
9.6	LTSpice comparison .....	67

## Figure index:

Figure 3. 1: Grid equivalent circuit .....	10
Figure 3. 2: LCL filter circuit.....	10
Figure 3. 3: LCL filter circuit.....	12
Figure 3. 4: PMSM power interface .....	12
Figure 3. 5: Three phase inverter equivalent circuit .....	13
Figure 3. 6: DC bus equivalent circuit.....	14
Figure 3. 7: Lack of accuracy in DC bus voltage.....	15
Figure 3. 8: PMSM whole circuit .....	16
Figure 3. 9: LCL filter model with ideal grid .....	18
Figure 3. 10: LCL filter C code.....	18
Figure 3. 11: Grid circuit breaker C code.....	19
Figure 3. 12: PMSM circuit breaker C code .....	19
Figure 3. 13: Grid inverter C code .....	20
Figure 3. 14: DC bus and chopper C code .....	20
Figure 3. 15: Generator three phase inverter C code .....	21
Figure 3. 16: Sine and cosine calculator C code .....	22
Figure 3. 17: abc to dq transformation C code .....	22
Figure 3. 18: dq to abc transformation C code .....	22
Figure 3. 19: Integrator C code.....	23
Figure 3. 20: Derivation C code .....	23
Figure 3. 21: PMSM_2 function C code .....	23
Figure 3. 22: PMSM top function C code .....	24
Figure 3. 23: PMSM function hierarchy .....	24
Figure 3. 24: Psim whole system.....	26
Figure 3. 25: simulation of DC bus preload from PMSM side in Psim.....	27
Figure 4. 1: HLS IPs dataflow and interfacing.....	30
Figure 4. 2: Grid HLS code .....	31
Figure 4. 3: Variable grid amplitude.....	31
Figure 4. 4: LCL HLS C code.....	32
Figure 4. 5: LCL inductor current filtering action .....	32
Figure 4. 6: Circuit breaker HLS C code .....	32
Figure 4. 7: DC bus HLS C code.....	33
Figure 5. 1: AXI IP .....	34
Figure 5. 2: HLS IP.....	35
Figure 5. 3: VHDL IP .....	35
Figure 5. 4: Vivado IP integrator view .....	36
Figure 5. 5: Bidirectional AXI IP .....	39
Figure 6. 1: User console tabs .....	42
Figure 6. 2: User console .....	43

Figure 6. 3: user console terminal window .....	44
Figure 6. 4: SDK communication protocol.....	44
Figure 6. 5: QT default parameter values .....	45
Figure 6. 6: QT engineering prefixes. ....	45
Figure 6. 7: Qt serial communication .....	46
Figure 6. 8: Qt communicating protocol .....	47
Figure 7. 1: PWM signal driver .....	48
Figure 7. 2: MicroZED connectors and signal naming.....	49
Figure 7. 3: Fan speed circuit .....	49
Figure 7. 4: Digital inputs and multiplexing.....	50
Figure 7. 5: Digital outputs circuit .....	50
Figure 7. 6: PSC carrier board main blocks.....	51
Figure 7. 7: Digital potentiometer schematics .....	52
Figure 7. 8: PSC carrier card ground plane.....	53
Figure 7. 9: Current amplifier .....	54
Figure 8. 1: HLS IP folder .....	56
Figure 8. 2: Run a script from Vivado.....	57
Figure 8. 3: Create an SDK empty project .....	57
Figure 8. 4: Create a SDK FSBL.....	58
Figure 8. 5: Create the boot.ini file in SDK .....	58
Figure 8. 6: MicroZED jumper configuration to boot from uSD card .....	59
Figure 9. 1: Current driver circuit .....	60
Figure 9. 2 Current driver circuit main blocks .....	60
Figure 9. 3: Input voltage divider and filter.....	61
Figure 9. 4: Bode plot of G_input (s). Module.....	61
Figure 9. 5: Bode plot of G_input (s). Phase.....	62
Figure 9. 6: Current buffer circuit.....	62
Figure 9. 7: Bode plot of G_LMH6321 (s). Module.....	63
Figure 9. 8: Bode plot of G_LMH6321 (s). Phase.....	63
Figure 9. 9: Differential amplifier and feedback signal .....	64
Figure 9. 10: Bode plot of G_L AD8130 (s). Module.....	65
Figure 9. 11: Bode plot of G_L AD8130 (s). Phase.....	65
Figure 9. 12: Current driver block design .....	66
Figure 9. 13: Bode plot of G_closed_loop (s). Module.....	66
Figure 9. 14: Bode plot of G_closed_loop (s). Phase.....	67
Figure 9. 15: LTSpice time domain schematic.....	68
Figure 9. 16: LTSpice time domain simulation .....	68
Figure 9. 17: LTSpice frequency domain schematic.....	69
Figure 9. 18: LTSpice frequency domain closed loop simulation .....	69

## Table index:

Table 4. 1: HIL HLS models list.....	28
Table 4. 2: Clock and time step per HLS IP .....	29
Table 5. 1: Full list of custom IPs used in Vivado.....	38
Table 7. 1: Available voltages .....	48

## Introduction

### 1.1 Objectives

A Hardware in the loop (HIL) emulator system has been designed in order to emulate the analog response of a wind power plant. So the HIL system will emulate and provide every signal required by Norvento Control Card.

In order to fulfil these objectives, it has been necessary to develop the following models:

1. Three phase Grid.
2. Three phase inverter bridge.
3. Three phase LCL filter.
4. Dc bus dynamic.
5. Permanent Magnet Synchronous Machine (PMSM).

As mentioned in the specification (document D0044-TET-0006 – Especificación de componentes electrónicos: Tarjeta simuladora), absolute accuracy in the plant behaviour is not required and the following details are out of scope:

1. Transitory response.
2. Harmonics.
3. Grid codes

## 2 Scope

This system has been design to allow:

1. The validation of the SW versions before using them in the real system.
2. Validate Norvento Control Card hardware.
3. Allow the improvement of all programs related with the control of the system.

All the system has been developed for Avnet MicroZED card with Xilinx 7X030 SoC.



### 3 Mathematical Models

To create a HIL system, several models has been done to emulate the behaviour of real analog components. In this chapter, those models and its algorithms has been covered.

#### 3.1 Blocks equations

All the mathematical expressions of the models are based on the trapezoidal integration rule where the following assumption can be made: [3.1] and [3.2].

$$\frac{dx}{dt} = x'(t) \quad [3.1]$$

$$x(t) = x(t - \Delta t) + x'(t) * \Delta t \quad [3.2]$$

##### 3.1.1 Grid

The grid model is divided into two different parts:

1. The ideal grid model:

The ideal behaviour of the grid is achieved by sweeping a 500 points vector which contains a unitary full sine-wave period with three different indexes, one per phase. Each phase follows this equation: [3.3], [3.4] and [3.5].

$$V_{grid_a} = (ampl_a * full\_sine[index_a]) \quad [3.3]$$

$$V_{grid_b} = (ampl_b * full\_sine[index_a + 333]) \quad [3.4]$$

$$V_{grid_c} = (ampl_c * full\_sine[index_a + 167]) \quad [3.5]$$

To get the correct amplitude, the unitary vector is multiplied by several constants (ampl\_a, ampl\_b and ampl\_c).

2. The grid rL impedance

To generate the grid impedance effect, its impedance (rL) has been added to the LCL filter, specifically to grid side of the LCL filter.

To obtained the voltage of the whole grid (the ideal one with the rL effect), the following math has been included for phase 'a': [3.6].

$$V_{WholeGrid_a} = V_{grid_a} + IL_{2a} * r_{grid} + \frac{\Delta IL_{2a}}{dt} \quad [3.6]$$

Note that phases 'b' and 'c' expressions are completely equivalent.

Figure 3. 1 shows the grid equivalent circuit with the impedance effect included.

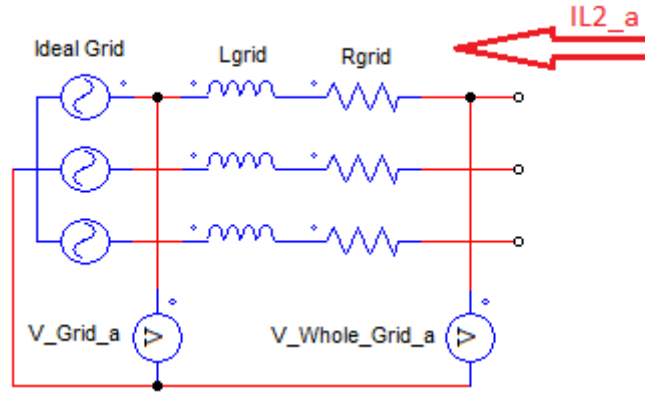


FIGURE 3. 1: GRID EQUIVALENT CIRCUIT

### 3.1.2 LCL filter

Three phase LCL filter is composed by six inductors with its parasitic resistor with three capacitors with its damping resistor. This filter follows this schematics of Figure 3. 2.

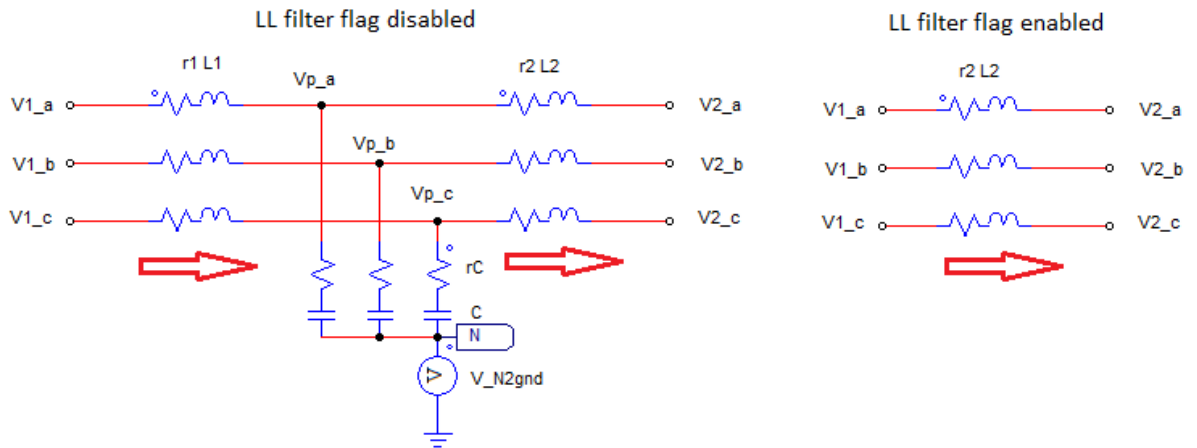


FIGURE 3. 2: LCL FILTER CIRCUIT

As the state-variables are current through the inductors and voltage across capacitors, the following expression has been made: equations [3.7] to [3.13].

$$\Delta IL_1 \begin{bmatrix} a \\ b \\ c \end{bmatrix} [k] = \frac{\Delta t}{L1} * \left( V1 \begin{bmatrix} a \\ b \\ c \end{bmatrix} [k-1] - V_p \begin{bmatrix} a \\ b \\ c \end{bmatrix} [k-1] - IL_1 \begin{bmatrix} a \\ b \\ c \end{bmatrix} [k-1] * r1 \right) \quad [3.7]$$

$$IL_1 \begin{bmatrix} a \\ b \\ c \end{bmatrix} [k] = IL_1 \begin{bmatrix} a \\ b \\ c \end{bmatrix} [k-1] + \Delta IL_1 \begin{bmatrix} a \\ b \\ c \end{bmatrix} [k] \quad [3.8]$$

$$\Delta IL_2 \begin{bmatrix} a \\ b \\ c \end{bmatrix} [k] = \frac{\Delta t}{L2} * \left( V_p \begin{bmatrix} a \\ b \\ c \end{bmatrix} [k-1] - V_2 \begin{bmatrix} a \\ b \\ c \end{bmatrix} [k-1] - V_{N2gnd} \begin{bmatrix} a \\ b \\ c \end{bmatrix} [k-1] - IL_2 \begin{bmatrix} a \\ b \\ c \end{bmatrix} [k-1] * r2 \right) \quad [3.9]$$

$$IL_2 \begin{bmatrix} a \\ b \\ c \end{bmatrix} [k] = IL_2 \begin{bmatrix} a \\ b \\ c \end{bmatrix} [k-1] + \Delta IL_{12} \begin{bmatrix} a \\ b \\ c \end{bmatrix} [k] \quad [3.10]$$

$$\Delta V_c \begin{bmatrix} a \\ b \\ c \end{bmatrix} [k] = \frac{\Delta t}{C} * \left( IL_1 \begin{bmatrix} a \\ b \\ c \end{bmatrix} [k-1] - IL_1 \begin{bmatrix} a \\ b \\ c \end{bmatrix} [k-1] \right) \quad [3.11]$$

$$V_c \begin{bmatrix} a \\ b \\ c \end{bmatrix} [k] = V_c \begin{bmatrix} a \\ b \\ c \end{bmatrix} [k-1] + \Delta V_c \begin{bmatrix} a \\ b \\ c \end{bmatrix} [k] \quad [3.12]$$

$$V_p \begin{bmatrix} a \\ b \\ c \end{bmatrix} [k] = \left( IL_1 \begin{bmatrix} a \\ b \\ c \end{bmatrix} [k-1] - IL_1 \begin{bmatrix} a \\ b \\ c \end{bmatrix} [k-1] \right) * rC + V_c \begin{bmatrix} a \\ b \\ c \end{bmatrix} [k-1] + V_{N2gnd} \begin{bmatrix} a \\ b \\ c \end{bmatrix} [k-1] \quad [3.13]$$

Notes:

1. All voltages are ground referred.
2. The positive current sense is flowing into the grid.
3. New values are made by the actual increment applied to the [k-1] currents or voltages.
4. To include the grid impedance effect, the following values has been retyped as: [3.14] and [3.15].

$$L2 = L2 + L_{grid} \quad [3.14]$$

$$r2 = r2 + r_{grid} \quad [3.15]$$

5. If LL filter flag is enabled, L1 and the capacitors branch are removed and rL2 includes rL1 effects plus rL\_grid effects: equations [3.16] to [3.19].

$$\Delta IL_2 \begin{bmatrix} a \\ b \\ c \end{bmatrix} [k] = \frac{\Delta t}{L2} * \left( V_1 \begin{bmatrix} a \\ b \\ c \end{bmatrix} [k-1] - V_2 \begin{bmatrix} a \\ b \\ c \end{bmatrix} [k-1] - V_{N2gnd} \begin{bmatrix} a \\ b \\ c \end{bmatrix} [k-1] - IL_2 \begin{bmatrix} a \\ b \\ c \end{bmatrix} [k-1] * r2 \right) \quad [3.16]$$

$$IL_2 \begin{bmatrix} a \\ b \\ c \end{bmatrix} [k] = IL_2 \begin{bmatrix} a \\ b \\ c \end{bmatrix} [k-1] + \Delta IL_{12} \begin{bmatrix} a \\ b \\ c \end{bmatrix} [k] \quad [3.17]$$

$$L2 = L2 + L1 + L_{grid} \quad [3.18]$$

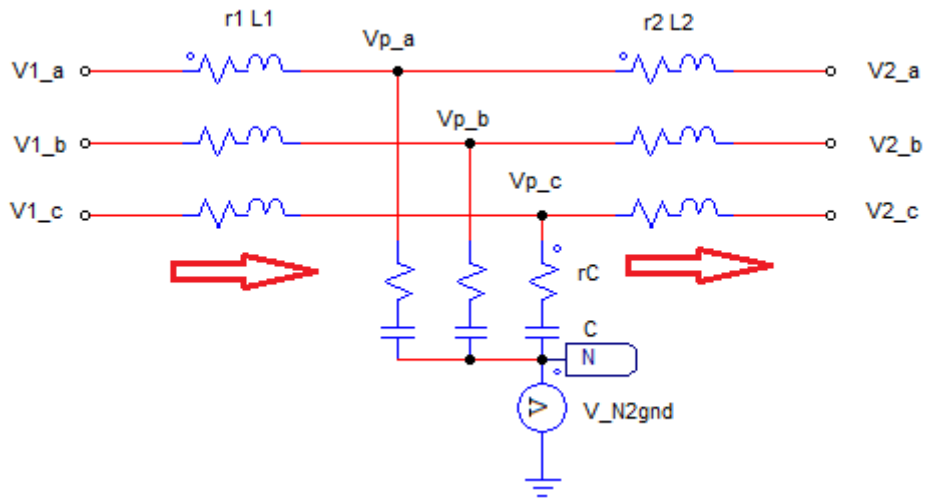
$$r2 = r2 + r1 + r_{grid} \quad [3.19]$$

### 3.1.3 Circuit Breakers

Circuit breakers due to convergence problems have been not modelled as switches, they have been modelled as 10000 ohm resistors.

No extra resistors have been included because circuit breakers affect to the parasitic resistor of other components. For example, the grid circuit breaker follows equation [3.20] logic.

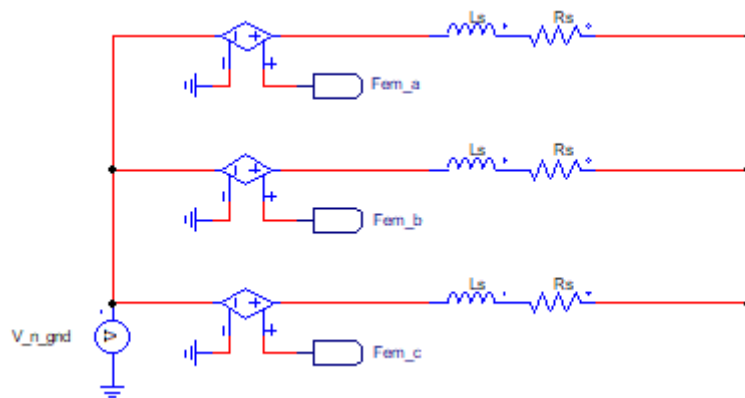
$$r2 = r2 + r1(if \text{ LL}_{filter}) + r_{grid} + 10k\Omega(if [k1 * CB1]) + R_{preload}(if \text{ k3}) \quad [3.20]$$



**FIGURE 3. 3: LCL FILTER CIRCUIT**

As it is denoted in Figure 3. 3, an extra resistor has not been included, the value of  $r_2$  has been changed instead.

The PMSM side circuit breakers has been done in the very same way. They alter the PMSM equivalent resistors:  $R_s$ . See Figure 3. 4.



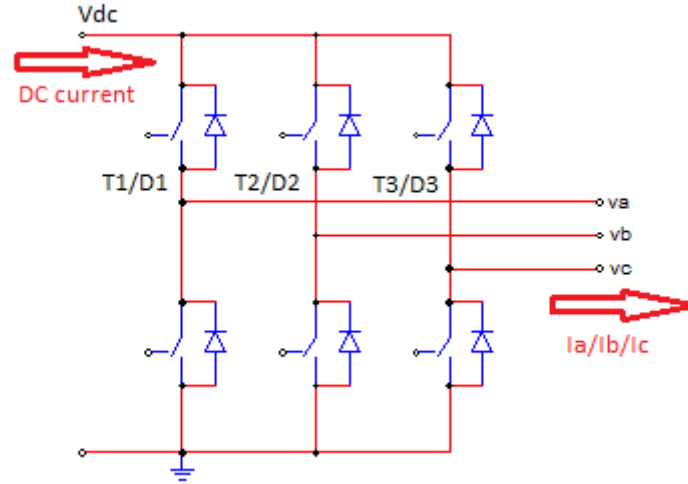
**FIGURE 3. 4: PMSM POWER INTERFACE**

Where:

$$R_s = R_s + 10k\Omega(\text{if } k2) \quad [3.21]$$

### 3.1.4 Grid Inverter and generator inverter

Three phase MOSFETs inverter bridges have been modelled as a collection of ideal switches with its antiparallel diodes being its equivalent circuit the one showed in Figure 3. 5.



**FIGURE 3. 5: THREE PHASE INVERTER EQUIVALENT CIRCUIT**

This model is in charge of the following actions:

1. Stablish the intermediate voltage:  $v_a$ ,  $v_b$  and  $v_c$ .
2. Check the current sense per phase to check if any antiparallel diode must turn on.
3. Calculate  $V_{n2gnd}$
4. Calculate the DC current

To achieve this functionality, it has been necessary to define equations [3.22] to [3.32]:

$$D1_{onCondition} = 1 \text{ when } (T1 = 0 \text{ and } I_a < 0), \text{ else } 0 \quad [3.22]$$

$$D2_{onCondition} = 1 \text{ when } (T2 = 0 \text{ and } I_b < 0), \text{ else } 0 \quad [3.23]$$

$$D3_{onCondition} = 1 \text{ when } (T3 = 0 \text{ and } I_c < 0), \text{ else } 0 \quad [3.24]$$

$$T1_{eq} = T1 + D1_{onCondition} \quad [3.25]$$

$$T2_{eq} = T2 + D2_{onCondition} \quad [3.26]$$

$$T3_{eq} = T3 + D3_{onCondition} \quad [3.27]$$

$$V_a = Vdc[k-1] * T1_{eq} \quad [3.28]$$

$$V_b = Vdc[k-1] * T2_{eq} \quad [3.29]$$

$$V_c = Vdc[k-1] * T3_{eq} \quad [3.30]$$

$$V_{n2gnd} = (V_a[k-1] + V_b[k-1] + V_c[k-1]) * \frac{1}{3} \quad [3.31]$$

$$I_{dc} = I_a * T1_{eq} + I_b * T2_{eq} + I_c * T3_{eq} \quad [3.32]$$

It can be checked that it is necessary for a conducting diode that its antiparallel MOSFET is turned off and that the current sense is the one which can polarize it.

Parasitic elements like  $R_{dsON}$  or diode voltage drop has not been considered in this models.

### 3.1.5 DC Bus and Chopper

This model is in charge of calculating the DC bus voltage from the current balance between the injected one (PMSM side) and the extracted one (grid side).

It also includes the chopper resistor which can be seen as a PMSM injected current leak that generates a power loss.

Due to lack of numerical accuracy, the natural discharge of the bus produced by the always connected resistor has been calculated with the exponential discharge equation of a capacitor.

The equivalent circuit is the one shown in Figure 3. 6.

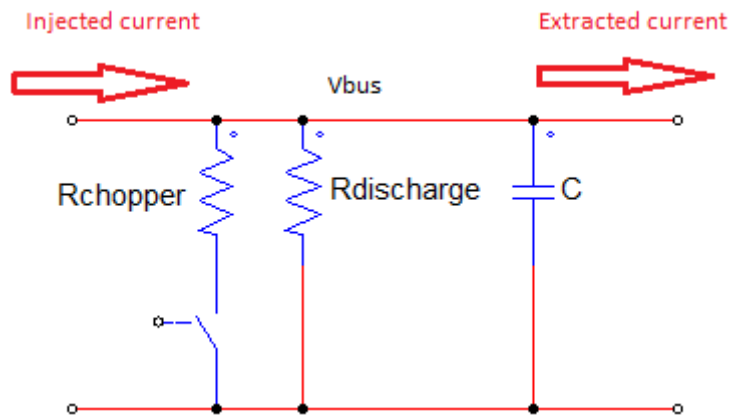


FIGURE 3. 6: DC BUS EQUIVALENT CIRCUIT

Whose mathematical equations are [3.33] and [3.34].

$$V_c[k] += \frac{\Delta t}{C} * (I_{injected}[k-1] - I_{extracted}[k-1] - Vbus[k-1] * \frac{1}{R_{equivalent}}) \quad [3.33]$$

$$V_{BUS.DC}[k] = \frac{\Delta t}{C} * \left( I_{injected}[k-1] - I_{extracted}[k-1] - Vbus[k-1] * \frac{1}{R_{equivalent}} \right) * R_c + V_c[k] \quad [3.34]$$

Where  $R_{equivalent}$  is the equivalent resistor of  $R_{chopper}$  and  $R_{discharge}$ .

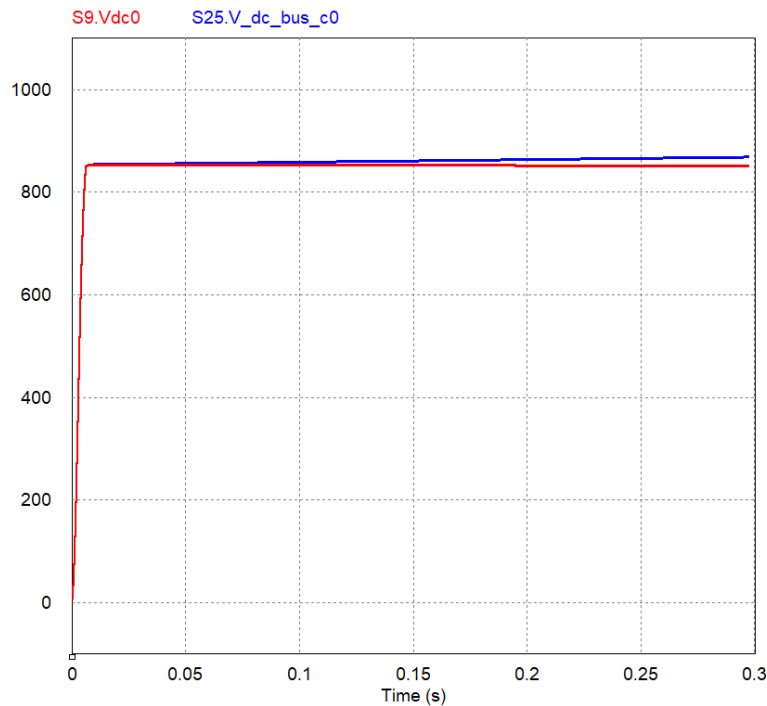
For the exponential discharge, this expression has been developed. Equations [3.35] and [3.36].

$$V_{BUS.DC}[k] = V_{BUS.DC}[k-1] * \exp(-0.00000297 * t) \quad [3.35]$$

$$0.00000297 = \frac{1}{R_{discharge} * C} = RC \text{ discharge time constant} \quad [3.36]$$

Where 't' is generated by means of a counter. The equivalent time step of the exponential discharge has been increased in order to save FPGA resources. Its equivalent time step is 0.005s; that is the speed with which 't' parameter is updated.

Note: the current balance has accuracy issues and they generate a small accumulative error in the DC bus voltage, this error is always increasing the voltage. This effect can be seen in Figure 3. 7 where it is shown a simplified DC bus preload from the grid:



**FIGURE 3. 7: LACK OF ACCURACY IN DC BUS VOLTAGE**

This error is very small but it is accumulative, and after several seconds, the DC bus voltage deviation is clearly notable. The solution to overcome this issue consist on reducing the value of Rdischarge resistor; this resistor will eliminate the extra voltage due to the lack of accuracy.

### 3.1.6 PMSM

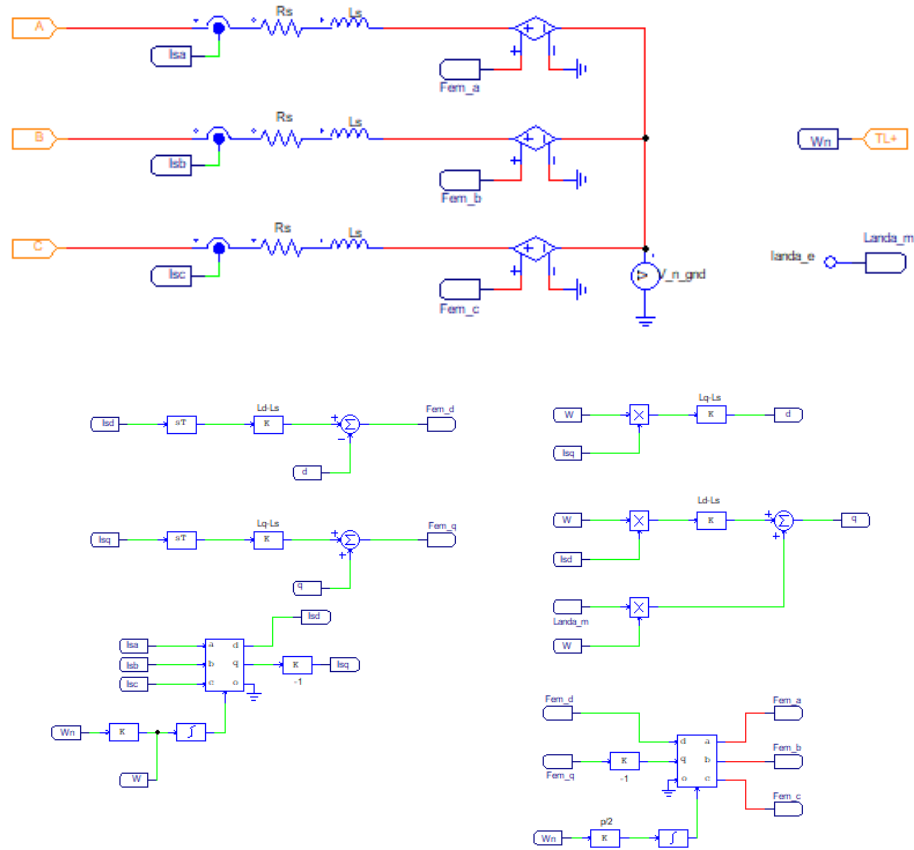
The electrical machine considered in this project is a Permanent Magnet Synchronous Machine (PMSM), this machine complies with the following:

1. It allows changing the electrical parameters such as Ls and Rs where the letter 's' comes from stator.
2. It allows changing the magnetic parameter of the machine like the pole number and the flux linkage value (parameter lambda\_e)

In order to make the tests easier, the mechanical speed of the machine shaft can be imposed. So details like the shaft power losses due to viscosity and the shaft inertia have not been considered.

The machine has been modelled in dq as it allows a higher level of abstraction so, dq to abc transformation and vice-versa has become necessary as well as a PLL for detecting the machine phase and feeding it into the transformation expressions.

The PMSM Psim block diagram is provided by Figure 3. 8 in order to make the assimilation of the equations easier:



**FIGURE 3. 8: PMSM WHOLE CIRCUIT**

The equivalent mathematical expressions are equations [3.37] to [3.48].

$$W = W_n * PolePairs \quad [3.37]$$

$$Theta = \int W dt \quad [3.38]$$

$$W = W_n * PolePairs \quad [3.39]$$

$$I_{sd} = \frac{2}{3} * (\cos(theta) * I_{sa} + \cos(theta - 120deg) * I_{sb} + \cos(theta + 120deg) * I_{sc}) \quad [3.40]$$

$$I_{sq} = \frac{-2}{3} * (\sin(theta) * I_{sa} + \sin(theta - 120deg) * I_{sb} + \sin(theta + 120deg) * I_{sc}) \quad [3.41]$$



$$d = W * I_{sq} * (Lq - Ls) \quad [3.42]$$

$$q = W * I_{sd} * (Ld - Ls) + \text{landa\_e} * W \quad [3.43]$$

$$Fem_d = \frac{dI_{sd}}{dt} * (Ld - Ls) - d \quad [3.44]$$

$$Fem_q = \frac{dI_{sq}}{dt} * (Lq - Ls) + q \quad [3.45]$$

$$Fem \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 1 \\ \cos(\theta - 120) & \sin(\theta - 120) & 1 \\ \cos(\theta + 120) & \sin(\theta + 120) & 1 \end{bmatrix} * \begin{bmatrix} Fem_d \\ Fem_q \\ 0 \end{bmatrix} \quad [3.46]$$

$$\Delta IL \begin{bmatrix} a \\ b \\ c \end{bmatrix} [k] = \frac{\Delta t}{Ls} * \left( Fem \begin{bmatrix} a \\ b \\ c \end{bmatrix} [k-1] + V_{n2gnd} \begin{bmatrix} a \\ b \\ c \end{bmatrix} [k-1] - V_s \begin{bmatrix} a \\ b \\ c \end{bmatrix} [k-1] - I_L \begin{bmatrix} a \\ b \\ c \end{bmatrix} [k-1] * R_s \right) \quad [3.47]$$

$$IL \begin{bmatrix} a \\ b \\ c \end{bmatrix} [k] = IL \begin{bmatrix} a \\ b \\ c \end{bmatrix} [k-1] + \Delta IL \begin{bmatrix} a \\ b \\ c \end{bmatrix} [k] \quad [3.48]$$

Where  $V_s \begin{bmatrix} a \\ b \\ c \end{bmatrix}$  represents the three phase inverter voltage referred to ground which is shared with the machine inductors.

### 3.2 PSIM blocks

In this chapter, it is explained how the previous expressions has been translated into C code.

The C code (not C++ code) generated in this step is a functional one. It means that they have not been thought for synthesis or implementation in a SOC system. Those codes have been done to check the algorithm functionality.

Among the following points, the main details of the C codes are reviewed.

Note: Psim only allows to define one time step per project, in this case, it has been fixed to 0.5us. The other required time step (1.5us) has been emulated my means of a counter which enables the evaluation of the code at the new “equivalent” time step.

The following modules run with a 0.5us time step:

1. Grid
2. LCL
3. Grid 3ph inverter
4. Dc bus

The following models are run with a 1.5us equivalent time step:

1. PMSM inverter
2. PMSM

### 3.2.1 Grid

The grid implemented in Psim consist on a pure sinusoidal voltage supply. No impedance effects have been considered in it. In this case, there is no Psim C code model.

### 3.2.2 LCL filter

LCL filter has been developed by using a Psim 'simplified C block', the schematic of the LCL filter and the grid can be observed in Figure 3. 9.

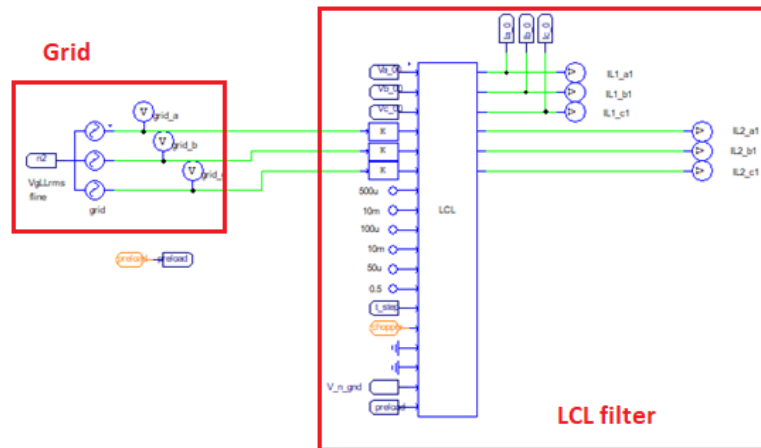


FIGURE 3. 9: LCL FILTER MODEL WITH IDEAL GRID

Figure 3. 10 shows the LCL filter code implemented in Psim:

```

1. //first inductor:
2. Delta_IL1_a= (t_step/L1)*(Va1 - vp_a - IL1_a*rL1);
3. Delta_IL1_b= (t_step/L1)*(Vb1 - vp_b - IL1_b*rL1);
4. Delta_IL1_c= (t_step/L1)*(Vc1 - vp_c - IL1_c*rL1);
5.
6. //second inductor:
7. Delta_IL2_a= (t_step/L2)*(vp_a - (Va2+V_N_gnd) - IL2_a*RLL2);
8. Delta_IL2_b= (t_step/L2)*(vp_b - (Vb2+V_N_gnd) - IL2_b*RLL2);
9. Delta_IL2_c= (t_step/L2)*(vp_c - (Vc2+V_N_gnd) - IL2_c*RLL2);
10.
11. //capacitor:
12. Delta_Vc_a= (t_step/C1)*(IL1_a - IL2_a) ;
13. Delta_Vc_b= (t_step/C1)*(IL1_b - IL2_b) ;
14. Delta_Vc_c= (t_step/C1)*(IL1_c - IL2_c) ;
15.
16. //voltage between inductors
17. vp_a = (IL1_a-IL2_a)*rC1 + Vc_a + V_N_gnd;
18. vp_b = (IL1_b-IL2_b)*rC1 + Vc_b + V_N_gnd;
19. vp_c = (IL1_c-IL2_c)*rC1 + Vc_c + V_N_gnd;
20.
21. IL1_a+=Delta_IL1_a;
22. IL1_b+=Delta_IL1_b;
23. IL1_c+=Delta_IL1_c;
24.
25. IL2_a+=Delta_IL2_a;
26. IL2_b+=Delta_IL2_b;
27. IL2_c+=Delta_IL2_c;
28.
29. Vc_a+=Delta_Vc_a;
30. Vc_b+=Delta_Vc_b;
31. Vc_c+=Delta_Vc_c;

```

FIGURE 3. 10: LCL FILTER C CODE

Where all the variables are static floats, so that they do not get reset in each run of the time step. No other detail has to be taken into account.

The coefficients in the left part of the LCL Psim block are the values of L1, r1, L2, r2, C and rC respectively.

### 3.2.3 Circuit Breakers

They only affect the parasitic resistor values of the adjacent components. In the LCL filter, this effect is achieved by means of code shown in Figure 3. 11.

```
1. if (preload==1)
2.     RLL2=11;
3. else if (K1==1)
4.     RLL2=10000;
5. else
6.     RLL2=rL2;
```

**FIGURE 3. 11: GRID CIRCUIT BREAKER C CODE**

Where “preload” bool variable emulates the precharge circuit breaker and K1 is the grid contactor. It can be seen how the RLL2 resistor value is adjusted depending on the situation.

The very same concept is applied to the PMSM K2 circuit breaker as it can be seen in Figure 3. 12.

```
1.
2. //contactor logic
3. if ( (SG1==0 && K2==0) || (SG1==0 && K2==1) || (SG1==1 && K2==0) )
4.     Rs=1000.0;
5. else
6.     Rs=Rs_param;
7.
```

**FIGURE 3. 12: PMSM CIRCUIT BREAKER C CODE**

### 3.2.4 Grid Inverter

Grid inverter includes the calculation of intermediate voltages,  $V_{N2gnd}$  voltage and DC bus current. Figure 3. 13 shows the code in charge of generating those outputs:

```

1. //conduction par T1
2. if (T1==0 && T4==0 && Ia<0)
3.     T1_eq=1;
4. else
5.     T1_eq=T1;
6. //conduction par T2
7. if (T2==0 && T5==0 && Ib<0)
8.     T2_eq=1;
9. else
10.    T2_eq=T2;
11. //conduction par T3
12. if (T3==0 && T6==0 && Ic<0)
13.     T3_eq=1;
14. else
15.     T3_eq=T3;
16.
17. if (T1_eq==1)
18.     a=Vdc;
19. else
20.     a=0;
21. if (T2_eq==1)
22.     b=Vdc;
23. else
24.     b=0;
25. if (T3_eq==1)
26.     c=Vdc;
27. else
28.     c=0;
29.
30. I_dc_bus=( Ia*T1_eq + Ib*T2_eq + Ic*T3_eq );
31. V_n_gnd=(a+b+c)*0.3333;

```

FIGURE 3. 13: GRID INVERTER C CODE

### 3.2.5 DC Bus and Chopper

In Psim, as the variable type is float, there is no remarkable accuracy error with the discharge resistor, so the exponential discharge behaviour has not been implemented in Psim code. However, this lack of accuracy does appear in the voltage stabilization of the bus.

To overcome this issue, the permanent resistor connected to the bus has been reduced so it will remove that “extra voltage”.

Figure 3. 14 shows the code which has been implemented in Psim:

```

1. //capacitor:
2. Delta_C= (t_step/C)*(I_in - I_out - V_bus_dc/R_eq);
3.
4. V_bus_dc = (I_in - I_out - V_bus_dc/R_eq)*rC + V_C ;
5.
6. V_C+=Delta_C;

```

FIGURE 3. 14: DC BUS AND CHOPPER C CODE

Where  $R_{eq}$  is the equivalent resistor of  $R_{chopper}$  and  $R_{discharge}$ .

### 3.2.6 Generator Inverter

Figure 3. 15 shows the code implemented in Psim:

```

1. //conduction par T1
2. if (T1==0 && T4==0 && Ia<0)
3.     T1_eq=1;
4. else
5.     T1_eq=T1;
6. //conduction par T2
7. if (T2==0 && T5==0 && Ib<0)
8.     T2_eq=1;
9. else
10.    T2_eq=T2;
11. //conduction par T3
12. if (T3==0 && T6==0 && Ic<0)
13.     T3_eq=1;
14. else
15.     T3_eq=T3;
16.
17. if (T1_eq==1)
18.     a=Vdc;
19. else
20.     a=0;
21. if (T2_eq==1)
22.     b=Vdc;
23. else
24.     b=0;
25. if (T3_eq==1)
26.     c=Vdc;
27. else
28.     c=0;
29.
30. I_out= - ( Ia*T1_eq + Ib*T2_eq + Ic*T3_eq );
31. V_n_gnd=(a+b+c)*0.3333;

```

**FIGURE 3. 15: GENERATOR THREE PHASE INVERTER C CODE**

The only difference with the grid inverter is the sign of I\_out variable. In this case, this variable is negative. Otherwise, both codes are the same.

### 3.2.7 PMSM

This model is such a complicated one and it contain some functionalities that are reused almost constantly like trigonometric mathematics. For this reason, this model has been divided into the following functions:

1. Sine and cosine calculator vector-based (in radians).

It can be seen how the sine and cosine functions are related by their mathematical symmetry. The variable “flag” is the one which determines if the operation will be a sine or a cosine.

The vector “sin\_funct” has not been included in this preview for simplicity.

It has not been used the trigonometrical maths functions the C standard library math.h as it can not be efficiently included in the FPGA.

```

1. float calc_sin_cos (float theta_aux, bool flag){
2.
3.     const float sin_func[1024] = { ... };
4.
5.     const float _2pi      = 6.2831853;
6.     const float pi        = 3.1415926;
7.     const float pi_4      = 1.5707963;
8.     const float _3pi_4    = 4.7123889;
9.     const float sin_conv  = 651.898647;
10.    static float theta=0.01;
11.
12.    if (flag==1)
13.        theta=(float)1.57079f-theta_aux;
14.    else
15.        theta=theta_aux;
16.
17.    //restore theta format:
18.    if (theta>=_2pi)
19.        theta=theta-_2pi;
20.    if (theta<0)
21.        theta=theta+_2pi;
22.
23.    //restore theta format:
24.    if (theta>=_2pi)
25.        theta=theta-_2pi;
26.    if (theta<0)
27.        theta=theta+_2pi;
28.
29.    //0deg to 90deg
30.    if(0<=theta && theta<pi_4)
31.        return sin_func[(int)(theta * sin_conv)];
32.    //90deg to 180deg
33.    else if(pi_4<=theta && theta<pi)
34.        return sin_func[(int)((pi-theta) * sin_conv)];
35.    //180deg to 270deg
36.    else if(pi<=theta && theta<_3pi_4)
37.        return -sin_func[(int)((theta-pi) * sin_conv)];
38.    //270deg to 360deg
39.    else if(_3pi_4<=theta && theta<_2pi)
40.        return -sin_func[(int)((_2pi-theta) * sin_conv)];
41.
42.    else return 651.0f;
43. }

```

**FIGURE 3. 16: SINE AND COSINE CALCULATOR C CODE**

## 2. Transformation abc to dq.

It includes the abc to dq transformation including the '-' sign in q magnitude.

```

1. *d = dos_terc*( calc_sin_cos(theta,1)*a + calc_sin_cos(theta-_2pi_3,1)*b + calc_sin_cos(theta+_2pi_3,1)*c);
2. *q = -dos_terc*( calc_sin_cos(theta,0)*a + calc_sin_cos(theta-_2pi_3,0)*b + calc_sin_cos(theta+_2pi_3,0)*c);
3. *o = dos_terc*( un_med*a + un_med*b + un_med*c);
4.

```

**FIGURE 3. 17: ABC TO DQ TRANSFORMATION C CODE**

## 3. Transformation dq to abc.

```

1. *a = ( calc_sin_cos(theta,1)*d + calc_sin_cos(theta,0)*q + o);
2. *b = ( calc_sin_cos(theta-_2pi_3,1)*d + calc_sin_cos(theta-_2pi_3,0)*q + o);
3. *c = ( calc_sin_cos(theta+_2pi_3,1)*d + calc_sin_cos(theta+_2pi_3,0)*q + o);
4.

```

**FIGURE 3. 18: DQ TO ABC TRANSFORMATION C CODE**

## 4. Integrator.

```

1. out_act=out_ant+(un_med*dT)*(in_act+in_ant);
2.
3. //reset the integral action
4. if (out_act>= PI_2)
5.     out_act=0;
6.
7. in_ant=in_act;
8. out_ant=out_act;

```

**FIGURE 3. 19: INTEGRATOR C CODE**

It includes a discretization by means of the bilinear expression of an integrator according to expression [3.49].

$$Integrator(s) = \frac{1}{s} \quad || \text{ with } s = \frac{2}{T} * \frac{z-1}{z+1} \quad [3.49]$$

5. Two derivations.

Both derivations are exactly the same, so only one of them is shown:

```

1. aux=(in_act - in_ant)*dT_1;
2. in_ant = in_act;

```

**FIGURE 3. 20: DERIVATION C CODE**

It includes a discretization by means of the bilinear expression of a derivation according to expression [3.50].

$$derivation(s) = s \quad || \text{ with } s = \frac{2}{T} * \frac{z-1}{z+1} \quad [3.50]$$

6. A function called “pmsm\_2”

This function does all the maths except the current thought the stator inductors.

```

1. W = Wn_o*un_med*P;
2. theta_aux = integrate(W);
3.
4. abc_2_dqo(Is_a, Is_b, Is_c, theta_aux, &Is_d, &Is_q, &Is_o);
5.
6. d = W*Is_q*(Lq-Ls);
7. q = W*Is_d*(Ld-Ls) + landa_m*W;
8.
9. Is_d_dif = derivate_1(Is_d);
10. Is_q_dif = derivate_2(Is_q);
11.
12. Fem_d = Is_d_dif*(Ld-Ls)-d;
13. Fem_q = Is_q_dif*(Lq-Ls)+q;
14.
15. dqo_2_abc(Fem_d, -Fem_q, 0.0, theta_aux, &Fem_a_aux, &Fem_b_aux, &Fem_c_aux);
16.

```

**FIGURE 3. 21: PMSM\_2 FUNCTION C CODE**

7. A function called “pmsm”

This function calculates the current thought the stator inductors.

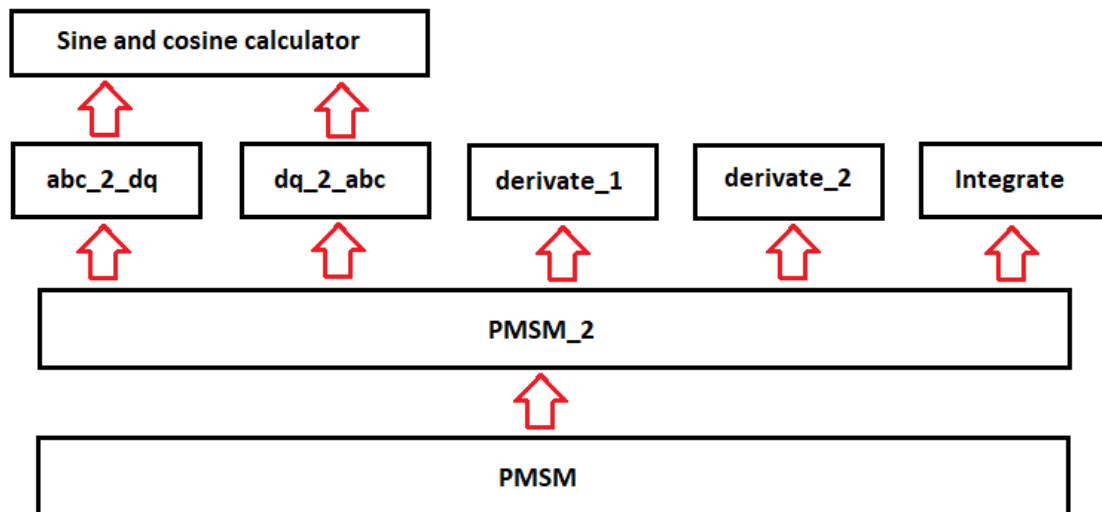
```

1. A_iIs_a = dT_Ls*(Fem_a+VN_gnd-Vs_a-iIs_a*Rs);
2. A_iIs_b = dT_Ls*(Fem_b+VN_gnd-Vs_b-iIs_b*Rs);
3. A_iIs_c = dT_Ls*(Fem_c+VN_gnd-Vs_c-iIs_c*Rs);
4.
5. iIs_a += A_iIs_a;
6. iIs_b += A_iIs_b;
7. iIs_c += A_iIs_c;
8.
9. pmsm_2(-iIs_a, -iIs_b, -iIs_c, Wn_o, Ld, Lq, &Fem_a, &Fem_b, &Fem_c, &Tem_aux);

```

**FIGURE 3. 22: PMSM TOP FUNCTION C CODE**

In this image it can be seen the dependences between all the functions implemented in this model and its hierarchy being PMSM the top function.



**FIGURE 3. 23: PMSM FUNCTION HIERARCHY**

### 3.3 PSIM template for simulation of the complete B2B system blocks

To check the functionality of all the models reviewed in the previous chapter it has been necessary to create a whole system in Psim to check how all the different models behave by themselves and how do they affects each other.

Those models included in Psim schematic capture tool are:

1. Grid
2. LCL filter
3. Grid inverter
4. DC bus and chopper
5. PMSM inverter
6. PMSM (the electrical machine model)

In addition to this models, it has been necessary to create certain stimulus which provides the possibility to test those models in real conditions. Those extra models are:

1. Grid control  
It includes a simplified dq control with an internal current loop and an external voltage loops which controls the Dc bus voltage.



The output of this IP are the switching signals for the grid three phase bridge.

2. PMSM control

It includes a Field Oriented Control (FOC) which measures the stator current of the PMSM and modified the current flow into the DC bus in order to maintain a shaft mechanical reference speed.

As the machine shaft mechanical speed is forced by user in last version, this block has become useless.

3. PMSM modulator

In Figure 3. 24 it is shown the whole Psim project created from the models.

Figure 3. 25 shows the result of a simulation in which the bus has been charged from the PMSM (it is like a machine side preload). To increase the current level, a low value resistor has been added to the DC bus.

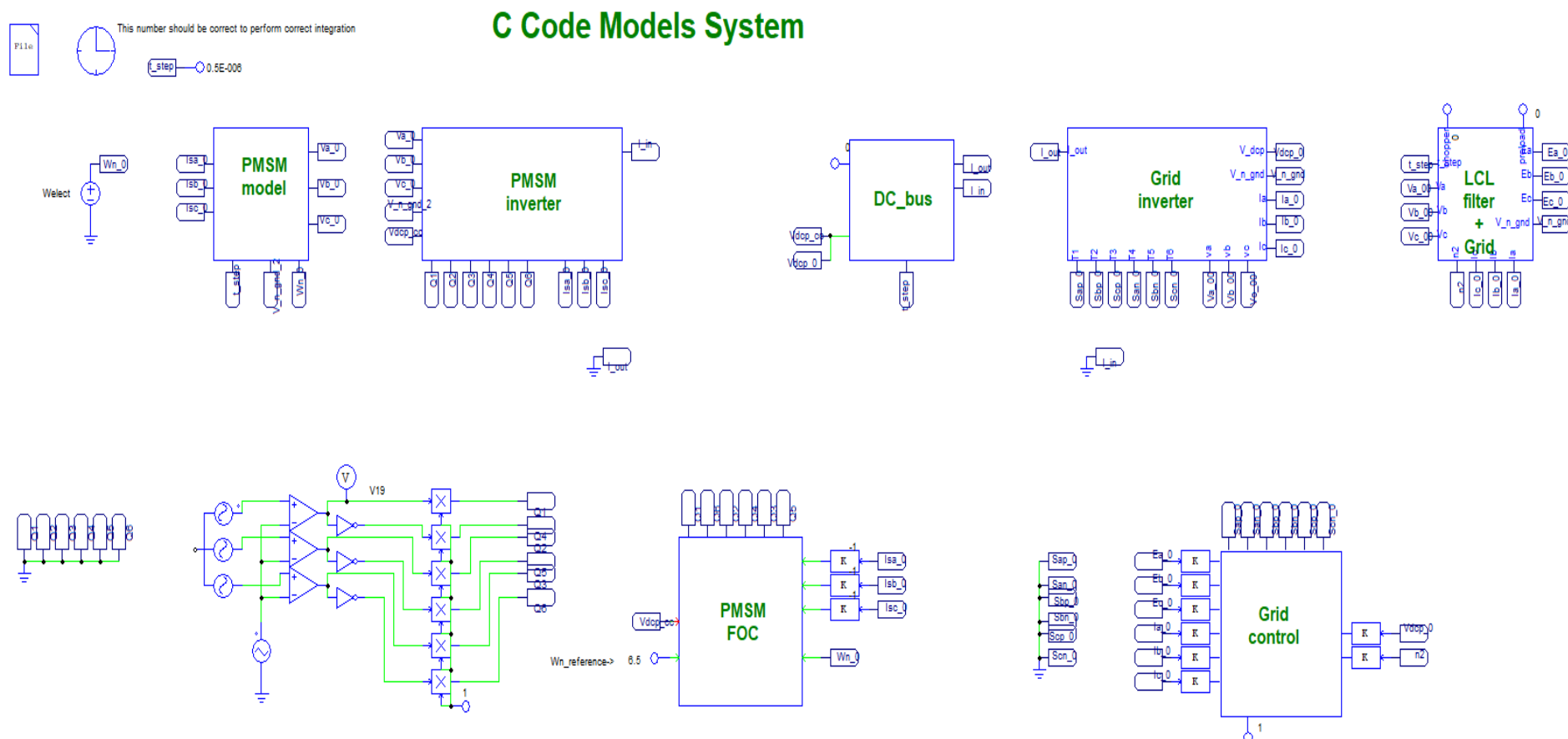


FIGURE 3. 24: PSIM WHOLE SYSTEM

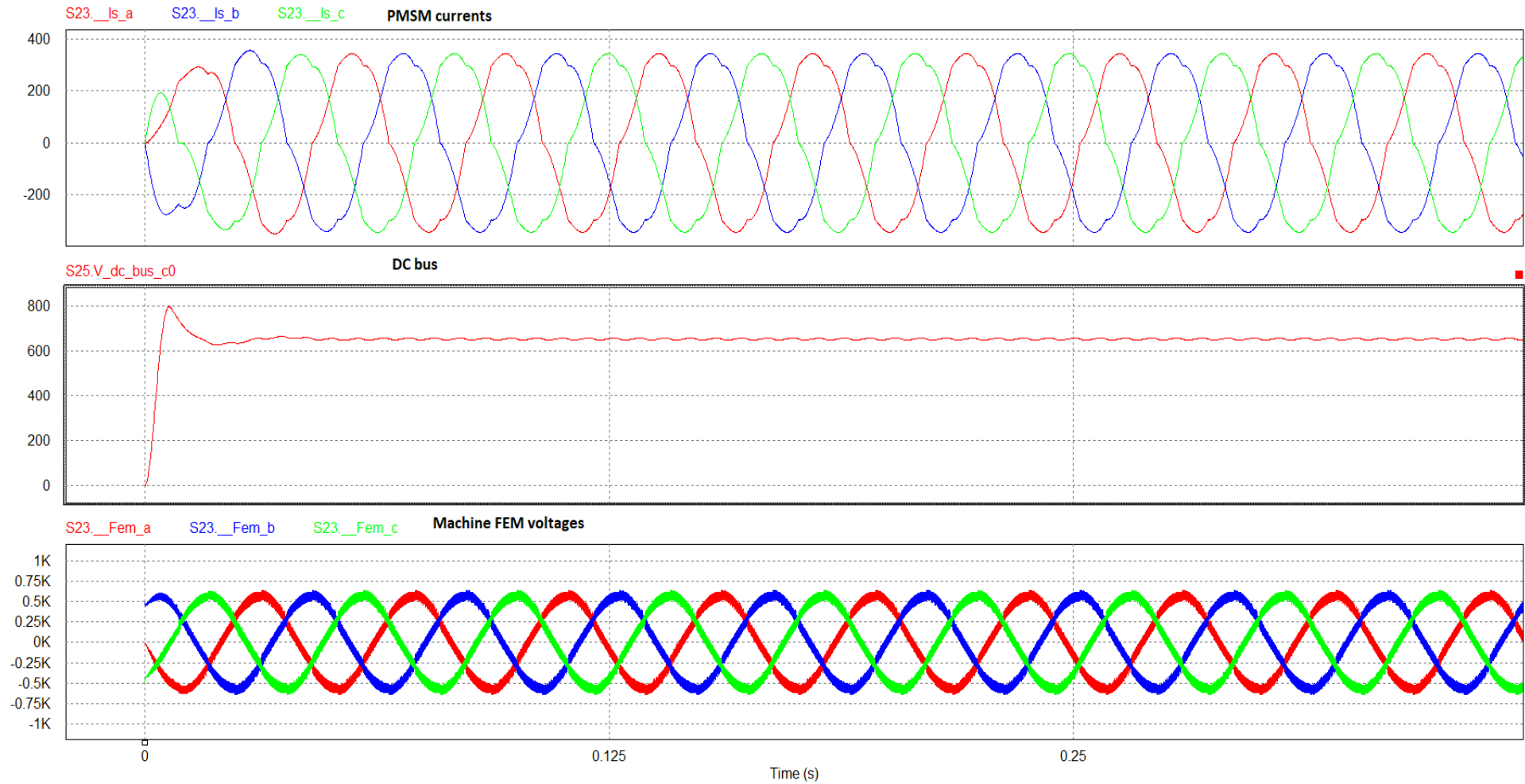


FIGURE 3. 25: SIMULATION OF DC BUS PRELOAD FROM PMSM SIDE IN Psim

## 4 Vivado HLS Models

### 4.1 General concepts

All the HIL models and HIL directly related models have been done in Vivado HLS as this provides an automatic C code translator which generates an equivalent VHDL code that can be used in Vivado as any other IP block.

In HLS have been done the IPs shown in Table 4. 1.

HLS IP name	Content of the IP
PMSM	Model
PMSM inverter	Model
DC bus	Model
Grid inverter	Model
LCL filter	Model
Grid	Model
Float converter	Data management
Gain offset truncation	Data management

**TABLE 4. 1: HIL HLS MODELS LIST**

All these IPs have been reviewed in the next sections.

#### 4.1.1 Data type, Clocking and time step

In order to obtain a tradeoff between employed resources and accuracy level, different data types have been chosen. Two main different types of data have been used:

1. Fixed point data  
It is used in the machine side as the mathematics included in those models have a higher complexity. To minimize the use of DSP48 resources in microZED board, variable width fixed point data have been employed. This applies to the following IPs: PMSM and PMSM\_inverter.
2. C standard data types  
For all the other IPs, C standard types like *bool*, *int* and *float* have been used.

Closely related with data type and use of available resources are the clock frequency and the time step used. In the next table it can be seen the clock domain and the *time step* frequency where time step is used as a synonym of the HLS *start* IP signal port.

HLS IP name	Clock	Time step	Data type
PMSM	50MHz	1.5us	Fixed
PMSM inverter	50MHz	1.5us	Fixed
DC bus	100MHz	0.5us	C standard
Grid inverter	100MHz	0.5us	C standard
LCL filter	100MHz	0.5us	C standard
Grid	100MHz	0.5us	C standard
Float converter	50MHz	1.5us	Fixed / C standard
Gain offset truncation	100MHz	0.5us	Fixed / C standard

**TABLE 4. 2: CLOCK AND TIME STEP PER HLS IP**

As it can be seen in Table 4. 2, two clock domains have been used in order to maximize the reutilization of resources, especially in the PMSM side. This made it necessary to register all the outputs of each IP block to ensure data will be stable when read.

Similarly, time step of the PMSM side is three times slower than the rest of HLS IPs this fact in combination with the fixed points and the slower clock have made it possible to fix everything inside the FPGA part of the microZED board.

Note, for fixed point support, it has been used the `ap_fixed.h` HLS C library; in the case of C standard types, it has been use the library `hls_math.h`.

#### 4.1.2 Data flow and interfacing ports

As different HLS IPs have been generated it has been necessary to stablish a proper standard for intercommunicating them.

For intercommunicating grid side blocks float typed variables have been used; in the case of PMSM side, 21 bits wide fixed point signals have been used being its format the following AP\_21\_13 (AP means “arbitrary precision”, 21 bits in total, 13 integer bits including the sign bit).

Figure 4. 1 shows one scheme which may prove useful to visualize data types, clocks and start (or time step) signals:

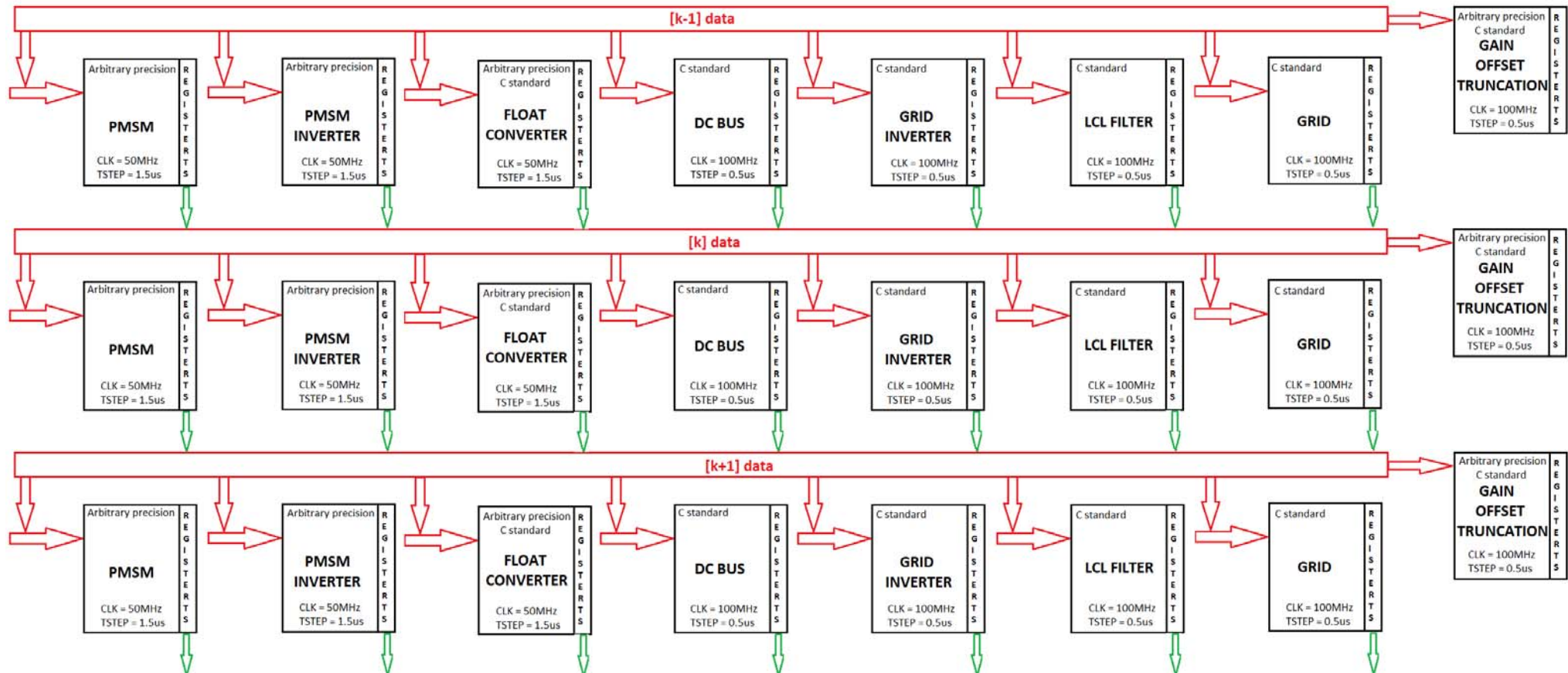


FIGURE 4. 1: HLS IPs DATAFLOW AND INTERFACING

## 4.2 Description of the main blocks

In this chapter all the HLS IP codes has been reviewed and its differences with the previous Psim codes has been outlined.

### 4.2.1 Grid

The grid has been implemented as a 500 points vector which is swept by three 120 degrees separated index.

Phase steps only affect at the beginning of the phase 'a' wave and they have been included in the way shown in Figure 4. 2.

```
1. if (flag_step==1 and index_a==0){
2.     index_a=index_a + phase_step;
3.     index_b=index_b + phase_step;
4.     index_c=index_c + phase_step;
5.     flag_step=0;
6. }
```

**FIGURE 4. 2: GRID HLS CODE**

Note: only the positive edge of the grid-step command is used. To introduce another step, this the command should be reset to zero and then enabled again.

To vary phase amplitudes, it has been used three different variables: ampl\_a, ampl\_b and ampl\_c as it can be seen in Figure 4. 3. Those values are the desired peak phase voltages of the grid typically  $230 * \sqrt{2}$ .

```
1. aux_a = (ampl_a*full_sine[index_a]);
2. aux_b = (ampl_b*full_sine[index_b]);
3. aux_c = (ampl_c*full_sine[index_c]);
```

**FIGURE 4. 3: VARIABLE GRID AMPLITUDE**

### 4.2.2 LCL filter

LCL filters HLS codes has suffer very few modifications compared with Psim LCL filter models.

Listed below are the differences:

1. Intermediate variables have been created to allow HLS to reuse FPGA functional blocks like DSP48 and reduce the use of resources. Figure 4. 4 shows the code which generates the current variation through rL1 inductor; as it can be seen, intermediate variables called "aux\_1\_x" has been used. The very same procedure has been followed with rL2 inductor.

```

1. aux_1_a = I_L1_a*rL1;
2. aux_1_b = I_L1_b*rL1;
3. aux_1_c = I_L1_c*rL1;
4.
5. //First LCL inductor:
6. A_I_L1_a = dT_L1 * (V1_a - Vp_a - aux_1_a);
7. A_I_L1_b = dT_L1 * (V1_b - Vp_b - aux_1_b);
8. A_I_L1_c = dT_L1 * (V1_c - Vp_c - aux_1_c);

```

**FIGURE 4. 4: LCL HLS C CODE**

Mathematical expressions related with the capacitor voltage has not required the use of intermediate variables.

2. If the inductor parasitic resistor becomes big enough in an instant (this means bigger than 12 ohms), for example, when opening a circuit breaker; a filtering action is done in the inductor currents (this is required because a huge variation in an inductor current generates a much bigger overvoltage which makes the HIL system saturate). This filtering action is produced by a moving average filter. See Figure 4. 5.

```

1. if (rL2_real>12){
2.     I_L2_a = (I_L2_a*cero_uno+I_L2_a_prev);
3.     I_L2_b = (I_L2_b*cero_uno+I_L2_b_prev);
4.     I_L2_c = (I_L2_c*cero_uno+I_L2_c_prev);
5.
6.     I_L2_a_prev=I_L2_a*cero_nueve;
7.     I_L2_b_prev=I_L2_b*cero_nueve;
8.     I_L2_c_prev=I_L2_c*cero_nueve;
9. }

```

**FIGURE 4. 5: LCL INDUCTOR CURRENT FILTERING ACTION**

### 4.2.3 Circuit Breakers

Circuit breaker HLS codes include no differences with Psim ones. For example, the effect of the grid Circuit Breaker on the LCL filter is shown in Figure 4. 6.

```

1. if (CB1_K1_K3==0 or CB1_K1_K3==1 or CB1_K1_K3==2 or CB1_K1_K3==3 or CB1_K1_K3==4)
2.     rL2_real=R_open_contactor;
3. else if (CB1_K1_K3==5)
4.     rL2_real=Preload;
5. else
6.     rL2_real=rL2;

```

**FIGURE 4. 6: CIRCUIT BREAKER HLS C CODE**

### 4.2.4 Grid and PMSM Inverter

HLS codes include no difference compared with Psim ones.

### 4.2.5 DC Bus and Chopper

HLS codes are, from a functional point of view, equal to the Psim ones; however, they include the use of intermediate variables to allow HLS to reuse FPGA resources, specially the DSP48 ones.



The use of intermediate variables in combination with the change in resistor value depending on the chopper status can be seen in Figure 4. 7.

```

1. if (ena_chop_res==0){
2.     V_R = V_bus_dc*(inv_R_eq);
3.     auxx=(I_in - I_out_ - V_R)*rC;
4.     aux=(I_in - I_out_ - V_R);
5. }
6. else{
7.     V_R2 = V_bus_dc*(inv_R_eq_2);
8.     auxx=(I_in - I_out_ - V_R2)*rC;
9.     aux=(I_in - I_out_ - V_R2);
10. }
11.
12. //capacitor:
13. V_bus_dc = (auxx + V_C);
14. V_C+=(t_step_C*aux);

```

**FIGURE 4. 7: DC BUS HLS C CODE**

The exponential curve associated to the natural discharge of the bus has not suffer any modification.

#### 4.2.6 PMSM

The code implemented in HLS contains very few differences compared with Psim one, being the following the most important:

1. All the input, output and intermediate variables are implemented in fixed point.
2. AP\_21\_13 signal types has been used for intercommunicating all the internal function variables corresponding to voltages and currents inside the PMSM model.
3. The resolution provided to each variable is approximately the smallest number which can be stored in the variable divided by 1000.
4. The precision of the PMSM whole model has been tested against a Psim PMSM machine. Differences smaller than a 5% has been seen.
5. Special care has been taken with the integrator and derivative internal variables as their resolution is better than the known smallest increment divided by 10000.

No other detail to highlight.

#### 4.3 Source Code Release

All the codes required to generate the HLS IPs are provided as well as different .tcl script which create the HLS project.

Document Instructions.pdf gives further detail about how to use them.

## 5 Vivado Project

Vivado is the software in which all hardware configuration of microZED SoC is done. All this configuration is condensed in a file called NV\_project\_wrapper.bit which is provided up to date.

### 5.1 IP integrator overview

This project has been developed and built in a modular manner, this modulus are called IPs. In this project, the following IP types has been used or developed:

1. IPs provided by Xilinx

This IPs include the processing system, reset, AXI interconnect, GPIOs, Xlconcat and Xlconstant. All of them are already available from Vivado IP browser. PSC does acquire any compromise for their proper functionality in any Vivado software version different from 2017.4.

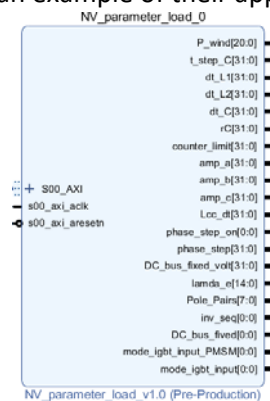
2. IPs created by Power Smart Control SL (PSC)

This IPs has been created by means of HLS codes or VHDL hand-written codes. All of them are fully re-generable and they are warrantied to work as they comply with C standard coding style and with VHDL-93 standard. This IPs can be divided in three categories:

a) Axi IPs

They have been created from a Vivado template and packaged to be capable of read and write in the AXI bus.

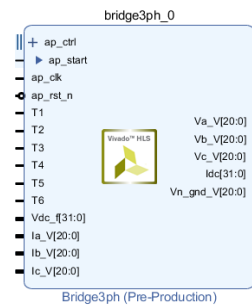
They are used for transferring data and parameters from ARM microcontroller to FPGA. Figure 5. 1 gives an example of their appearance.



**FIGURE 5. 1: AXI IP**

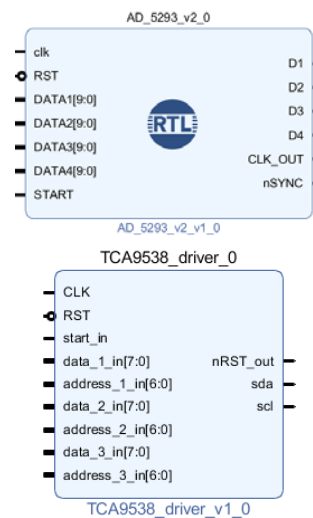
b) HLS IPs

They include an automate generated VHDL code from a C code by means of HLS high level synthesizer. Figure 5. 2 gives an example of their appearance.



**FIGURE 5. 2: HLS IP**

- c) VHDL IPs (packed in IPs or included as RTL-VHDL codes)  
They include a hand-typed VHDL code, the ones with the RTL symbol are unpacked by Vivado. Figure 5. 3 gives an example of their appearance.



**FIGURE 5. 3: VHDL IP**

Figure 5. 4 provides a view of the IP integrator which included all the IPs used in the project.

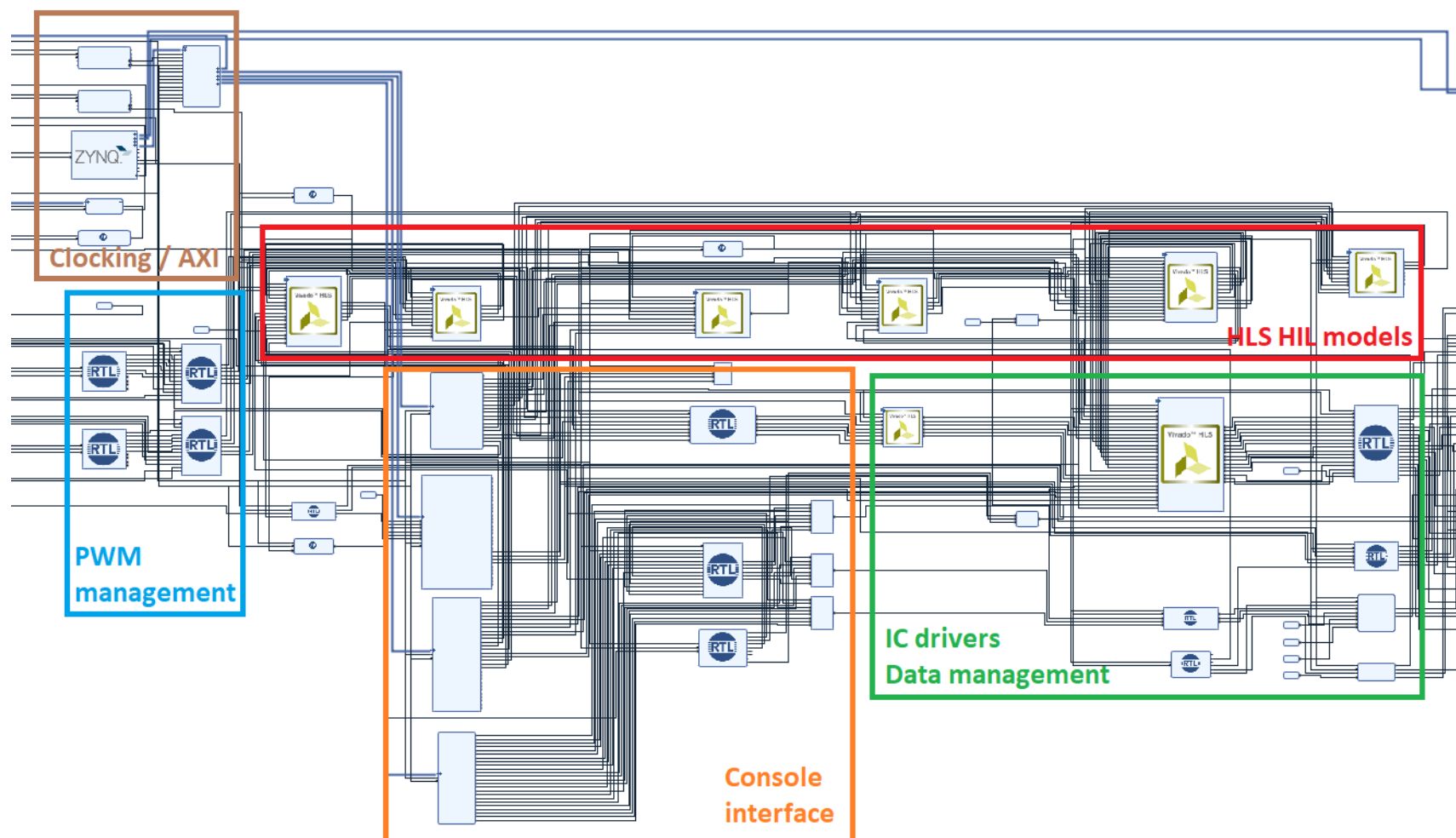


FIGURE 5. 4: VIVADO IP INTEGRATOR VIEW

Table 5. 1 shows all the PSC made IPs.

IP name	IP type	Function	CLK	Start	Data type
Serial interrupt	RTL	Generates 1 s interrupt for console communication	100 MHz	-	Standard VHDL
Gen_referencias	RTL	Generate open loop pulses for making tests	100 MHz	-	Standard VHDL
IBGT_input_selector	RTL	It select which pulses will be provided to the bridge: NV ones or Gen_referencias IP ones	100 MHz	-	Standard VHDL
Start_PMSM	RTL	It generates the PMSM side start signal (time step)	50 MHz	-	Standard VHDL
PMSM	HLS	Model of PMSM	50 MHz	1.5 us	Arbitrary precision
Bridge_3ph	HLS	Model of PMSM inverter	50 MHz	1.5 us	Arbitrary precision
NV_parameter_load	AXI	Communicate user console with FPGA	100 MHz	-	Standard VHDL
AD1_refComp	RTL	Input driver for DAC121s101	50 MHz	1 kHz	Standard VHDL
NOT	RTL	It toggles the input bits	100 MHz	-	Standard VHDL
NV_io_controller	AXI	Communicate user console with FPGA	100 MHz	-	Standard VHDL
Parameter_load_2	AXI	Communicate user console with FPGA	100 MHz	-	Standard VHDL
Digital_output	AXI	Communicate user console with FPGA	100 MHz	-	Standard VHDL
startGrid	RTL	It generates the grid side start signal (time step)	100 MHz	-	Standard VHDL
DC_bus_dynamic	HLS	Model of DC bus	100 MHz	0.5 us	Standard C
Truncate_bits	RTL	It truncate 21 bits signals to 12 bits (for DACs)	50 MHz	1.5 us	Standard VHDL
NV_multiplier	RTL	It is a multiplexer for PSC digital outputs	100 MHz	-	Standard VHDL
De_multiplexer	RTL	It is a demultiplexer for PSC digital inputs	100 MHz	-	Standard VHDL
Grid_bridge	HLS	Model of grid inverter	100 MHz	0.5 us	Standard C
Float_conversor	HLS	Convert arbitrary precision PMSM side signals into float typed signals	50 MHz	1.5 us	Mixed
LCL_filter	HLS	Model of LCL filter	100 MHz	0.5 us	Standard C
Gain_offset_truncation	HLS	It converts float signals to std_logic_vector of 12 bits	100 MHz	0.5 us	Standard C

Bit_order_adjust_tca9538	RTL	It rotates the input vectors so MSB become LSB	100 MHz	-	Standard VHDL
Start_peripheral	RTL	It gives start signals to IC drivers IPs	100 MHz	-	Standard VHDL
Grid	HLS	Model of grid	100 MHz	0.5 us	Standard C
Dac_121s101	RTL	Driver for DAC_121s101	50 MHz	Cont	Standard VHDL
AD_5293	RTL	Driver for AD_5293	100 MHz	100 kHz	Standard VHDL
TCA_9538	RTL	Driver for TCA_9538	100 MHz	3 kHz	Standard VHDL
TCA_9538_inputs	RTL	Driver for TCA_9538	100 MHz	4.2 kHz	Standard VHDL

**TABLE 5. 1: FULL LIST OF CUSTOM IPs USED IN VIVADO**

## 5.2 IC drivers IPs and conversion IPs

The carrier board made use of the following Integrated Circuits (ICs):

1. TCA9538. This is an 8-Bit I2C Low-Power I/O Expander.
2. DAC121s101. This is a 12-Bit Micro Power SPI Digital-to-Analog Converter.
3. AD5293. This is a single-channel, 10 bit, SPI, 1% Tolerance Digital Potentiometer

Each of them require a driver and all the drivers in the projects has been defined in the FPGA section of the SoC. Those drivers generate the required signals like clocks, /SYNC, data, etc.

As data inputs they require 8 bit wide signals or 12 bits wide signals. In order to convert the existing signal in the FPGA to the required data type, the following IPs has been developed:

1. Bit\_order\_adjust\_tca9538.  
It converts the MSB into the LSB and so on.
2. Gain\_offset\_truncation.  
It adjusts the gain of the signals to compensate the carrier board gains and adapt them to NV specifications. It also injects a 50% offset (2048 over 4096), convert float signals into std\_logic\_vectors of 12 bits and round them.  
Only one signal is converted each time step to reduce the use of resources, so, a whole conversion takes 8us (16 signals \* 0.5us time step=8us). As the DACs bandwidth is considerably smaller, this equivalent time step reduction has no visible effects.
3. Truncate\_bits.  
It adjusts the bit number of the signals.
4. Float\_conversor.

Their basic functionality has been covered in the list above and extended in the enumeration above.

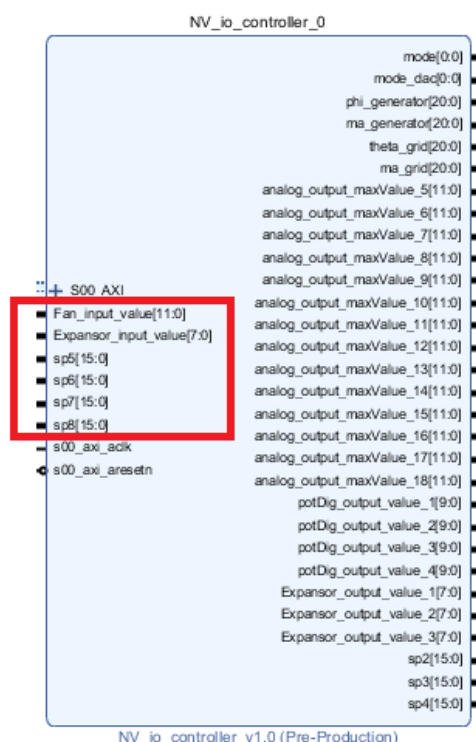
## 5.3 User Console communications IPs

They consist of a set of registers accessible from the microcontroller and the FPGA.

The microcontroller is in charge of establishing the communication with the computer and writing these registers with the information collected as well as reading some registers and sending its value to the user console.

The communication path used is AMBA bus, concretely, the AXI bus. To allow the communications, it has been necessary to generate IPs compatible with AXI bus with the ability to read from it or to write in it. This IPs are the ones which configures the registers.

Note, all AXI IPs generated have a slave role, it means that, theoretically, they could not write in the bus. However, with a modification of the driver provided by Vivado it has been able to write in it also. This effect can be observed in NV\_io\_controller IP, Figure 5.5.



**FIGURE 5. 5: BIDIRECTIONAL AXI IP**

Note that 4 signals (from sp5 to sp8) has been left spare in case they become necessary.

## 5.4 Start IP

To create a real time step, all HIL related blocks have a start signal which behave as a time step signal. When the signal change from '0' to '1', the IP is run.

Three start IP block have become necessary:

1. Start\_peripheral. This IP is in charge of launching the ICs serial protocol.
2. StartGrid. This IP generates the time step signal of the PMSM side HLS IPs.
3. Start\_PMSM. This IP generates the time step signal of the grid side HLS IPs.

Note: as it has been explained before, the time step for the grid side is 0.5us; and, for the PMSM side, is 1.5us.

## 5.5 PWM signals

PWM signals are sampled without any FPGA driver, consequently:

1. They are read at the main frequency of the FPGA: 100MHz.
2. They are used at the time step edge, so at 0.5us for grid side and 1.5us for PMSM side.

Additionally, for PSC internal test, two open loop modulator IPs has been created and two multiplexers allow the user the possibility to decide which are the real pulses: the ones read by the carrier or the ones generated internally by the modulators. This choice is not accessible from the user console but it is from microcontroller code.



## 6 SDK and User Console

### 6.1 General concepts

SDK is an Eclipse based software provided by Xilinx to program the microcontrollers included in the SoC of the microZED card.

In this project only ARM0 microcontroller has been used and it is in charge of maintaining the communication with the user console.

For this communication a serial protocol RS-232 has been used with the following details:

1. Baud Rate: 115200
2. Data bits: 8
3. Parity: None
4. Stop bits: 1
5. Flow control: None

For establishing the communication with the microZED board, it is required to have installed the Silicon\_Labs\_CP210x\_USB\_to\_UART\_bridge driver.

The user console allows the parametrization of the following parameters:

1. PMSM
  - a)  $L_d$ : d component stator inductance in H.
  - b)  $L_q$ : q component stator inductance in H.
  - c)  $R_s$ : stator phase resistor in ohm.
  - d)  $f_e$ : electric frequency imposed to the machine in Hz(electric).
  - e)  $\lambda_e$ : flux linkage of the machine in V/Rad/s.
  - f) Pole Pairs: Machine pole pairs.
2. DC bus
  - a)  $C_{bus\ dc}$ : Dc bus capacitor value in F.
  - b) Dc bus fixed (bool): when '1' it fixed the bus voltage.
  - c) V bus voltage: value to use when the voltage is fixed by user.
3. LCL filter
  - a) L filter/ LCL filter (bool): it changes from LL filter to LCL filter.
  - b)  $L_1$ : DC bus side LCL inductor value in H.
  - c)  $L_2$ : grid side LCL inductor value in H.
  - d) C: capacitor value in F.
  - e)  $r_{L1}$ : DC bus side LCL inductor parasitic resistor in ohm.
  - f)  $r_{L2}$ : grid side LCL inductor parasitic resistor in ohm.
  - g)  $r_{Precharge}$ : precharge resistor value in ohm.
  - h)  $R_c$ : capacitor parasitic resistor in ohm.
4. Grid
  - a) Phase step: angle to jump when enabled the phase step option in degrees.
  - b) Phase step on (bool): in '0' to '1' transition generates the phase step.
  - c) Freq grid: electrical frequency of the grid in Hz.
  - d)  $V_{a, b, c}$ : RMS grid phase voltage in V.
  - e) Inv seq (bool): when enabled, inject grid inverse sequence.

- f) Lcc: Grid short-circuit inductance in H.
- g) Rcc: Grid short-circuit resistance in ohm.
- 5. Digital outputs
  - a) Contactor returns (bool)
 

Signals: CB1, R grid K1, R gen K2, R rep K3, R heatK4 and R vent K5

This signal is generated automatically by FPGA, when “enabled” is selected this signal is sent to NV Control Card; when “disabled” is selected, the signal sent to NV Control Card is always 0.

This options can prove useful for testing circuit breaker return signal errors and how are they managed.
  - b) Other outputs (bool)
 

Signals: R break vent, Rdif, R SG1, R emerg, R var, R UAC, R temp max, R temp min, R hydr, R break heat, R start stop, R reset and Spare 1 to spare 5.

When blue, a ‘1’ is sent to NV Control Card; when grey, a ‘0’ is sent.
- 6. NTCs and PTCs values
 

Signals: NTC 1 ISO, PTC ISO, NTC MED and PT10000 NISO in ohm.

The resistor values specified in ohm are the ones to which PSC card digital potentiometers are set.

A shows the following data

- 1. Digital inputs (bool)
 

Signals: K1, K2, K3, K4, K5, K6, CB1 and Led Status.

They show PSC digital inputs.
- 2. Fan speed: it gives a relative 0 to 100% measure of the fan speed. Where 100% is correlated a 10V DC signal.
- 3. PMSM Wn (mechanical speed): it provides the actual mechanical speed of the PMSM shaft.

Figure 6. 1 and Figure 6. 2 shows the user console whole appearance and all the tabs expanded.

All the tabs and fields of the console are fully customizable by the user; they allow:

- 1. Resize and remove sections.
- 2. Change tabs order.
- 3. Hide the circuit scheme.

The figure displays four panels of the PSC user console, each showing a different tab of the 'RTS Parameters' section. The tabs are PMSM, DC Bus, LCL Filter, and Grid. Each panel contains a list of parameters with input fields for their values.

- PMSM Tab:** Parameters include Ld (H), Lq (H), Rs (Ω), fe (Hz), Lambda\_e (V/Rad/s), Pole Pairs, Wn (rpm), and a note about the permanent magnet synchronous machine.
- DC Bus Tab:** Parameters include Cbus dc (F), Vbus voltage (V), and a checkbox for 'Bus dc fixed'.
- LCL Filter Tab:** Parameters include L filter (checkbox), L1 (H), L2 (H), C (F), rL1 (Ω), rL2 (Ω), rPrecharge (Ω), and rC (Ω).
- Grid Tab:** Parameters include Phase step (°), Freq Grid, Ampl a (Vrms), Ampl b (Vrms), Ampl c (Vrms), Inv seq (checkbox), Lcc (H), and Rcc (Ω).

**FIGURE 6. 1: USER CONSOLE TABS**

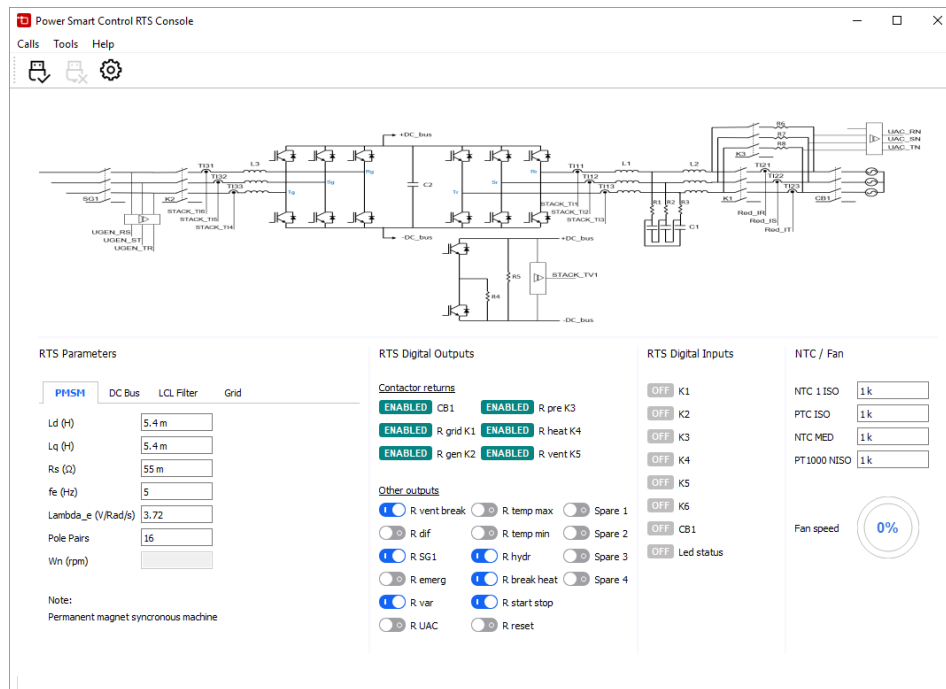


FIGURE 6. 2: USER CONSOLE

### 6.1.1 Parameter list and parameter format

All the parameters shared between user console and HIL system have a specific order and data type in the communication.

MicroZED SoC microprocessor hold the master role in the serial communication with the user console.

### 6.1.2 Communication description

The following tasks are done in a communicating cycle:

1. MicroZED sends:
  - a) The digital inputs values received from NV Control Board.
  - b) The 0 to 100% fan value calculated from NV Control Board fan signal.
  - c) The shaft speed value of the PMSM.
  - d) The last parameter command 'P pp' and a counter variable. This is the command for user console to start sending data. This variable increments with a unitary step every time the communication is relaunched (once per second).
2. User console sends (where the order is of parameters is important):
  - a) Other digital outputs: R break vent, Rdif, R SG1, R emerg, R var, R UAC, R temp max, R temp min, R hydr, R break heat, R start stop, R reset and Spare 1 to spare 5.
  - b) Contactor returns: CB1, R grid K1, R gen K2, R rep K3, R heatK4 and R vent K5.
  - c) NTCs and PTCs values NTC 1 ISO, PTC ISO, NTC MED and PT10000 NISO.
  - d) All the bool parameters: Dc bus fixed, Phase step on, Inv seq, L filter/ LCL filter.
  - e) All PMSM parameter: Ld, Lq, Rs, Fe, Lanbda\_e, Pole Pairs.
  - f) All dc bus parameters: Cbus dc, V bus voltage.
  - d) All LCL filter parameters: L1, L2, C, rL1, rL2, rPrecharge, rC.
  - e) All grid parameters: Phase step, Freq grid, Ampl a, Ampl b, Ampl c, Lcc, Rcc.

3. MicroZED sends (or can send) strings to user console terminal with command "NV". This allows the console to act as a simplified debugger.

This communication can be seen in the user console terminal window. Figure 6. 3.

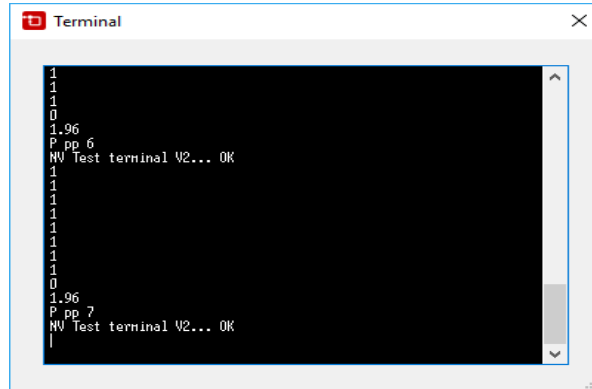


FIGURE 6. 3: USER CONSOLE TERMINAL WINDOW

Communication is established once per second as no extra speed has been necessary.

## 6.2 SDK communication protocol

The communication protocol is coded with the following details:

1. It uses a hardware (FPGA) produced 1 second interrupt.  
This interrupt is also used to make small mathematical evaluations or adaptations with the received data.
2. It uses an ARM A9 UART peripheral.
3. It sends data with printf command.
4. It receives data with scanf command.

```

1. //Digital inputs signals
2. printf("%c\n", charString[7]);
3. printf("%c\n", charString[6]);
4. printf("%c\n", charString[5]);
5. printf("%c\n", charString[4]);
6. printf("%c\n", charString[3]);
7. printf("%c\n", charString[2]);
8. printf("%c\n", charString[0]);
9. printf("%c\n", charString[1]);
10. //fan speed
11. printf("%i\n", Fan_speed);
12. //shaft speed
13. printf("%.2f\n", (float)(j_Wn/256.0f));
14. printf("P pp %i\n", counter);
15. scanf("%s", &my_chain);
16. sscanf(my_chain, "%d", &dig_out_1);
17. scanf("%d%c", &dig_out_3, &ch);
18. scanf("%d%c", &dig_out_5, &ch);
19. scanf("%d%c", &dig_out_6, &ch);
20. scanf("%d%c", &dig_out_10, &ch);
21. scanf("%d%c", &dig_out_11, &ch);
22. scanf("%d%c", &dig_out_12, &ch);
23. scanf("%d%c", &dig_out_13, &ch);
24. scanf("%d%c", &dig_out_14, &ch);
25. scanf("%d%c", &dig_out_15, &ch);
26. scanf("%d%c", &dig_out_17, &ch);
27. scanf("%d%c", &dig_out_18, &ch);
28. scanf("%d%c", &dig_out_19, &ch);
29. scanf("%d%c", &dig_out_20, &ch);
30. scanf("%d%c", &dig_out_21, &ch);
31. scanf("%d%c", &dig_out_22, &ch);
32. scanf("%d%c", &dig_out_7, &ch);
33. scanf("%d%c", &dig_out_8, &ch);
34. scanf("%d%c", &dig_out_4, &ch);
35. scanf("%d%c", &dig_out_9, &ch);
36. scanf("%d%c", &dig_out_16, &ch);
37. scanf("%d%c", &dig_out_2, &ch);
38. scanf("%f%c", &spotDig_output_value_1, &ch);
39. scanf("%f%c", &spotDig_output_value_2, &ch);
40. scanf("%f%c", &spotDig_output_value_3, &ch);
41. scanf("%f%c", &spotDig_output_value_4, &ch);
42. scanf("%d%c", &j_DC_bus_fixed, &ch);
43. scanf("%d%c", &j_phaseStep_on, &ch);
44. scanf("%d%c", &j_inv_seq_grid, &ch);
45. scanf("%d%c", &j_mode_L_LCL, &ch);
46. scanf("%f%c", &j_Ld, &ch);
47. scanf("%f%c", &j_Lq, &ch);
48. scanf("%f%c", &j_Rs, &ch);
49. scanf("%f%c", &j_fe, &ch);
50. scanf("%f%c", &j_lamda_e, &ch);
51. scanf("%f%c", &pole_pairs, &ch);
52. scanf("%f%c", &j_C_bus_dc, &ch);
53. scanf("%f%c", &j_V_bus_voltage, &ch);
54. scanf("%f%c", &j_L1, &ch);
55. scanf("%f%c", &j_L2, &ch);
56. scanf("%f%c", &j_C, &ch);
57. scanf("%f%c", &j_rL1, &ch);
58. scanf("%f%c", &j_rL2, &ch);
59. scanf("%f%c", &j_rPrecarga, &ch);
60. scanf("%f%c", &j_rC, &ch);
61. scanf("%f%c", &j_phaseStep, &ch);
62. scanf("%f%c", &j_freq_grid, &ch);
63. scanf("%f%c", &j_Ampl_a, &ch);
64. scanf("%f%c", &j_Ampl_b, &ch);
65. scanf("%f%c", &j_Ampl_c, &ch);
66. scanf("%f%c", &j_Lcc, &ch);
67. scanf("%f%c", &j_Rcc, &ch);
68. printf("NV Test terminal V2... OK \n");

```

FIGURE 6. 4: SDK COMMUNICATION PROTOCOL

This communication is implemented by means of the code collected in Figure 6. 4.

### 6.3 Qt communications protocol

User console is fully programmed in Qt software. It includes a communication protocol complementary with the one implemented in SDK. User console also included a terminal function which can prove useful as an easy and simplified debugging tool.

Most of buttons and controls have been created from scratch with the aim to simplify the interface and to make it as clear as possible. As a result, there are several files which defines those graphical controls.

In addition to that files, there are other files of importance which have been reviewed below.

1. Definition of user console default values have been done in the file “console.h”

```

1. // Console.h
2. static double initValuesRTS[] =
3. {
4.     5.4e-3,      // Ld          // 0
5.     5.4e-3,      // Lq          // 1
6.     55e-3,       // Rs          // 2
7.     5,           // fe          // 3
8.     3.72,        // Lambda_e    // 4
9.     16,          // Pole Pairs  // 5
10.    5.1e-3,       // Cbus dc     // 6
11.    800,          // Vbus voltage // 7
12.    500e-6,       // L1          // 8
13.    100e-6,       // L2          // 9
14.    33e-6,        // C           // 10 C_IDX
15.    0.01,         // rL1         // 11
16.    0.01,         // rL2         // 12
17.    11,           // rPrecharge  // 13
18.    0.5,          // rC          // 14 rC_IDX
19.    50,           // Phase step
20.    50,           // Freq Grid
21.    230,          // Ampl a
22.    230,          // Ampl b
23.    230,          // Ampl c
24.    100e-6,       // Lcc
25.    10e-3,        // Rcc
26. };
27.
28. static int initValuesDigOther[] =
29. {
30.     1,           // R vent break
31.     0,           // R dif
32.     1,           // R SG1
33.     0,           // R emerg
34.     1,           // R var
35.     0,           // R UAC
36.     0,           // R temp max
37.     0,           // R temp min
38.     1,           // R hydr
39.     1,           // R break heat
40.     1,           // R start stop
41.     0,           // R reset
42.     0,           // Spare 1
43.     0,           // Spare 2
44.     0,           // Spare 3
45.     0,           // Spare 4
46. };
47.
48. static double initValuesNTC[] =
49. {
50.     1000,        // NTC 1 ISO
51.     1000,        // PTC ISO
52.     1000,        // NTC MED
53.     1000,        // PT1000 NISO
54. };

```

FIGURE 6. 5: QT DEFAULT PARAMETER VALUES

2. Number format and the use of prefixes like ‘n’, ‘u’, ‘m’, ‘k’, ‘M’, etc. have been defined in file “ElectNunb.cpp”. It associates the engineering prefixes with mathematical constants as it can be seen in Figure 6. 6.

```

1. CString NumbToElect(double d, CString ft)
2. {
3.     CString str("0");
4.     if (fabs(d) < 1.0e-16)
5.         return str;
6.     if (fabs(d) < 1.0e-9)
7.     {
8.         str.Format(ft, d * 1.0e12);
9.         str += " p";
10.    }
11.    else if (fabs(d) < 1.0e-6)
12.    {
13.        str.Format(ft, d * 1.0e09);
14.        str += " n";
15.    }
16.    else if (fabs(d) < 1.0e-3)
17.    {
18.        str.Format(ft, d * 1.0e06);
19.        str += " u";
20.    }
21.    else if (fabs(d) < 1.0)
22.    {
23.        str.Format(ft, d * 1.0e03);
24.        str += " m";
25.    }
26.
27.    else if (fabs(d) < 1.0e+3)
28.    {
29.        str.Format(ft, d);
30.    }
31.    else if (fabs(d) < 1.0e+6)
32.    {
33.        str.Format(ft, d * 1.0e-3);
34.        str += " k";
35.    }
36.    else if (fabs(d) < 1.0e+9)
37.    {
38.        str.Format(ft, d * 1.0e-6);
39.        str += " M";
40.    }
41.    else
42.    {
43.        str.Format(ft, d * 1.0e-9);
44.        str += " G";
45.    }
46.    return str;

```

FIGURE 6. 6: QT ENGINEERING PREFIXES.

3. Data and serial communication have been held in “NVConsole.cpp” file. This functions are the ones that open, configure and close the serial port. See Figure 6. 7.

```

1. void MainWindow::openSerialPort()
2. {
3.     const SettingsDialog::Settings p = m_settings->settings();
4.     m_serial->setPortName(p.name);
5.     m_serial->setBaudRate(p.baudRate);
6.     m_serial->setDataBits(p.dataBits);
7.     m_serial->setParity(p.parity);
8.     m_serial->setStopBits(p.stopBits);
9.     m_serial->setFlowControl(p.flowControl);
10.    if (m_serial->open(QIODevice::ReadWrite)) {
11.        m_console->setEnabled(true);
12.        m_console->setLocalEchoEnabled(p.localEchoEnabled);
13.        m_ui->actionConnect->setEnabled(false);
14.        m_ui->actionDisconnect->setEnabled(true);
15.        m_ui->actionConfigure->setEnabled(false);
16.        showStatusMessage(tr("Connected to %1 : %2, %3, %4, %5, %6")
17.                           .arg(p.name).arg(p.stringBaudRate).arg(p.stringDataBits)
18.                           .arg(p.stringParity).arg(p.stringStopBits).arg(p.stringFlowControl));
19.    }
20.    else {
21.        QMessageBox::critical(this, tr("Error"), m_serial->errorString());
22.
23.        showStatusMessage(tr("Open error"));
24.    }
25. }
26.
27. void MainWindow::closeSerialPort()
28. {
29.     if (m_serial->isOpen())
30.         m_serial->close();
31.     m_console->setEnabled(false);
32.     m_ui->actionConnect->setEnabled(true);
33.     m_ui->actionDisconnect->setEnabled(false);
34.     m_ui->actionConfigure->setEnabled(true);
35.     showStatusMessage(tr("Disconnected"));

```

**FIGURE 6. 7: QT SERIAL COMMUNICATION**

4. The communication protocol implemented in user console is shown in Figure 6. 8.

## 6.4 Source Code Release

Code is provided in two different folders:

1. One folder with SDK code which includes all the microZED ARM code.
2. Other folder with the all the source codes required for the user console.

```

38. void MainWindow::writeData(const QByteArray &data)
39. {
40.     m_serial->write(data);
41. }
42.
43. void MainWindow::readData()
44. {
45.     const QByteArray arrayData = m_serial->readAll();
46.     m_console->putData(arrayData);
47.
48.     if (arrayData[0] == 'N')
49.         return;
50.
51.     QString stData(arrayData.data());
52.     QStringList stList = stData.split("\n");
53.
54.     if (stList.size() != NUM_RTS_DIGITAL + NUM_RTS_PAR_VALUE + 3 || stList[stList.size() - 2][0] != 'P')
55.     {
56.         QMessageBox::information(this, "Warning.", "Data received is not correct, please check the connection.", QMessageBox::Ok);
57.         return;
58.     }
59.
60.     QString data;
61.     int idx = 0;
62.
63.     for (ToggleSwitchIn * toggleSwitchIn : listInToggleSwitch)
64.     {
65.         data = stList[idx++];
66.         toggleSwitchIn->setStatus(data[0] != '0');
67.     }
68.
69.     data = stList[idx++];
70.     QString st(data.data());
71.     int fanspeed = st.toInt();
72.     fanSpeed->setProgress(fanspeed);
73.
74.     for (auto lineEdit : listInRTSLineEdits)
75.     {
76.         data = stList[idx++];
77.         QString st(data.data());
78.         lineEdit->setText(st);
79.     }
80.
81.     data = stList[idx++];
82.     if (data[0] == 'P')
83.     {
84.         //Sleep(10);
85.         SendData();
86.     }
87. }

```

FIGURE 6. 8: QT COMMUNICATING PROTOCOL

## 7 RTS Carrier Board

PSC carrier board provides analog signal adaptation circuits to transform microZED compatible signals with the ones provided or required by NV Control Card. These circuits have been reviewed in this chapter.

### 7.1 General overview and description of the main blocks

The carrier board main blocks are:

1. Power supply

All required voltages of the carrier are generated from the +-15V and +24V channels provided by NV Control Card directly through the connectors. In Table 7. 1 are collected all the carrier voltages:

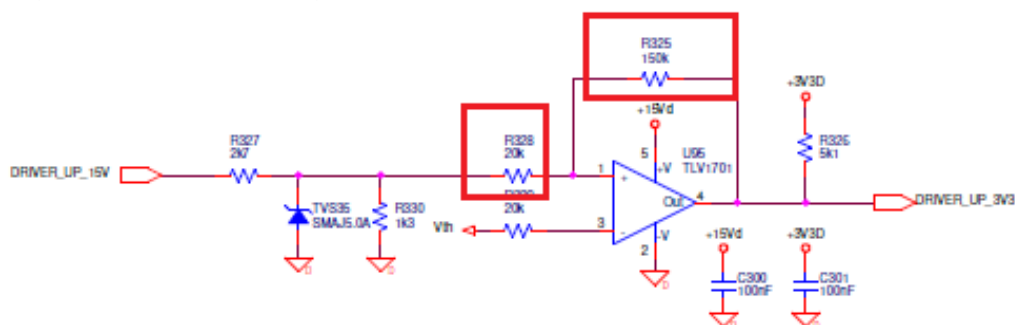
Voltage input source (NV Control Card)	Generated voltage	Regulated
+24V	+3.3V	Yes
+24V	+5V	Yes
+/-15V	+/-12V	Yes
+15V	+5V	Yes
+24V	+10V	Yes
+24V	+22V	Yes

**TABLE 7. 1: AVAILABLE VOLTAGES**

2. PWM signal driver

PWM signal conditioning consist on a voltage divider combined with an operational amplifier working as a comparator to reduce the voltage from 15V to 3.3V.

Note that the highlighted resistors introduce a hysteretic threshold response in the comparator to make it resistant to noise. The comparing '-' voltage is 0.75V ( $10\mu A * 75k\Omega = 0.750V$ ).



**FIGURE 7. 1: PWM SIGNAL DRIVER**

3. MicroZED connectors and signal assignments

Figure 7. 1 provides the relation between carrier signals and microZED board pins.



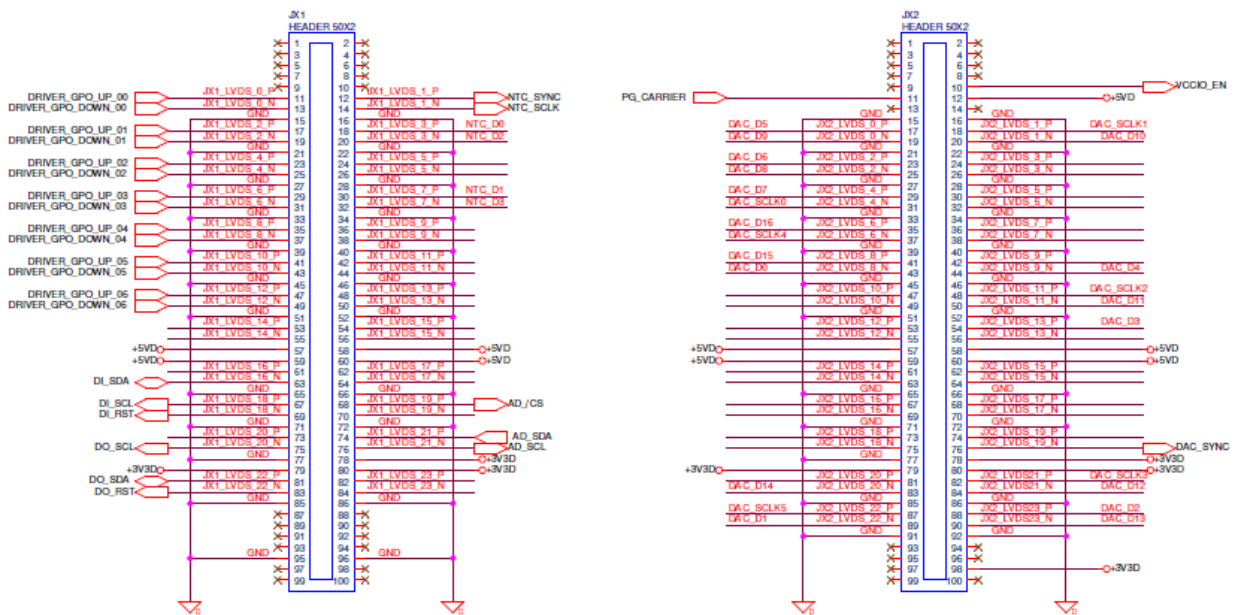


FIGURE 7. 2: MICROZED CONNECTORS AND SIGNAL NAMING

#### 4. Analog I/Os

Fan speed (a PWM signal) is the unique analog input signal. It is transformed from 10V<sub>peak</sub> to 3.3V<sub>peak</sub> ( $10V \cdot 68k\Omega / (68k\Omega + 137k\Omega) = 3.3V$ ) by means of a voltage divider, filtered with a time constant of 30ms ( $137k\Omega \cdot 220nF = 30.1ms$ ) and then injected into a 12bits ADC. See Figure 7. 3.

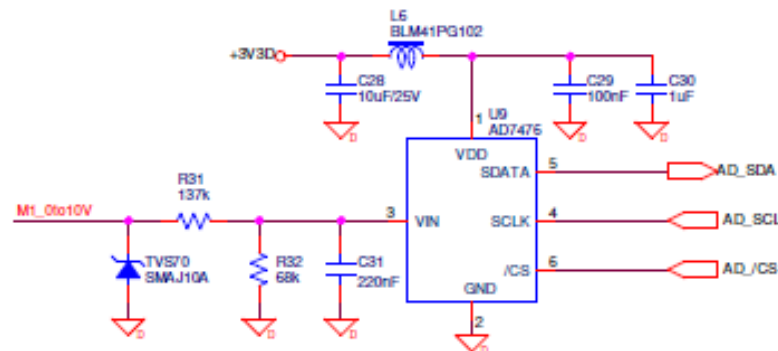


FIGURE 7. 3: FAN SPEED CIRCUIT

Analog outputs require a current output driver. This driver has been reviewed in another chapter due to its complexity.

#### 5. Digital I/Os

Digital inputs are filtered with a time constant of 720us ( $360\Omega \cdot 2 \cdot 1\mu F = 720us$ ), reduced to about 12V ( $24 \cdot 2k\Omega / (2k\Omega + 1k\Omega + 2 \cdot 360\Omega) = 12.9V$ ) and optocoupled, providing an output of 3.3V. The signals are then multiplexed with a TCA9538 IC.



### FIGURE 7. 5: DIGITAL OUTPUTS CIRCUIT

[www.powersmartcontrol.com](http://www.powersmartcontrol.com)

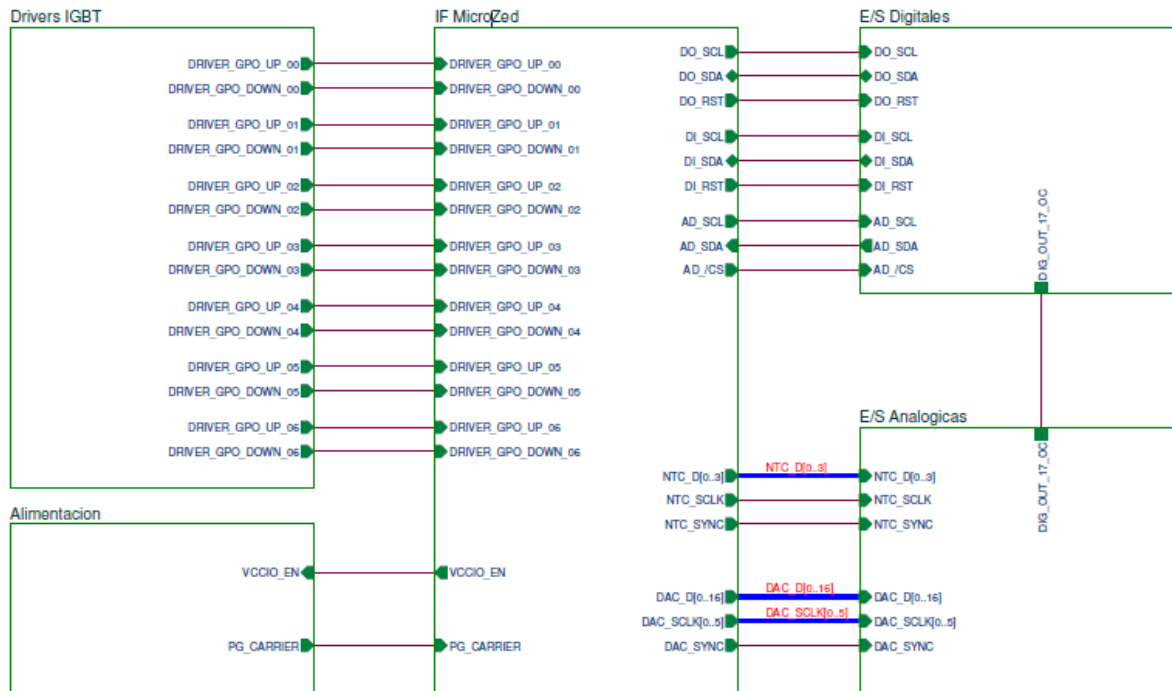


FIGURE 7. 6: PSC CARRIER BOARD MAIN BLOCKS

## 7.2 NTC sensors emulation

NTCs and PTCs have been emulated by means of four different digital potentiometers (AD5293 IC), those ICs require a SPI compatible serial communication whose driver is implemented in the FPGA part of the microZED board.

To simplify the routing of the system, /RST signal has been connected to Vlogic.

VHDL programed driver has the following details:

1. No reset command is sent to IC.
2. The first word sent is "0001100000000110" (0x1806) this word is necessary to:
  - a) Allow the wiper to update (removing turn on IC protection).
  - b) Set the standard mode. High precision mode does not follow the temporization shown in its datasheet so it has been discarded.
3. The rest of words are "000001" + "D9 to D0" where the first part is the writing command and D9 to D0 is the resistor data.

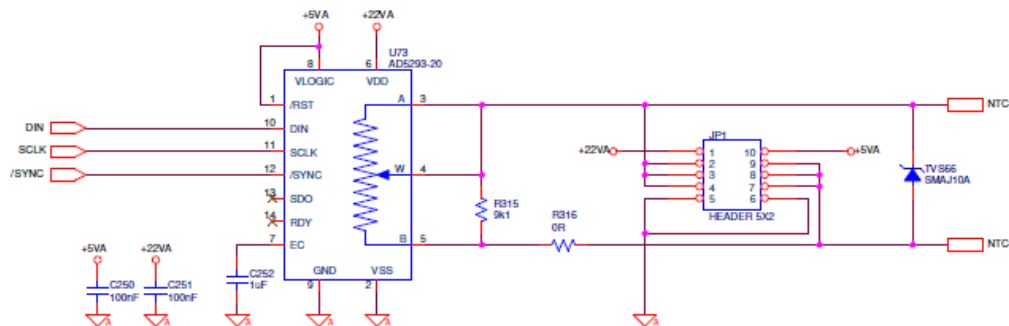
As there is a parallel resistor with the potentiometer it is necessary to calculate the required digital potentiometer resistor so the equivalent resistor is the one required from user console. The following math is required for the conversion:

$$D_{Digital\ Potentiometer} = \frac{(R_{eq} - 90) * 1024 * R_{aux}}{20k * R_{aux} - (R_{eq} - 90) * 20k} \quad [7.1]$$

Where:

1. Req is the resistor value collected from user console.
2. Raux is the value of the parallel resistor.
3. 20kohm is the whole value of the potentiometer.
4. 90ohm are subtracted as they are the wiper intrinsic resistor.




Figure 7. 7 shows the schematics of one digital potentiometer:



**FIGURE 7. 7: DIGITAL POTENTIOMETER SCHEMATICS**

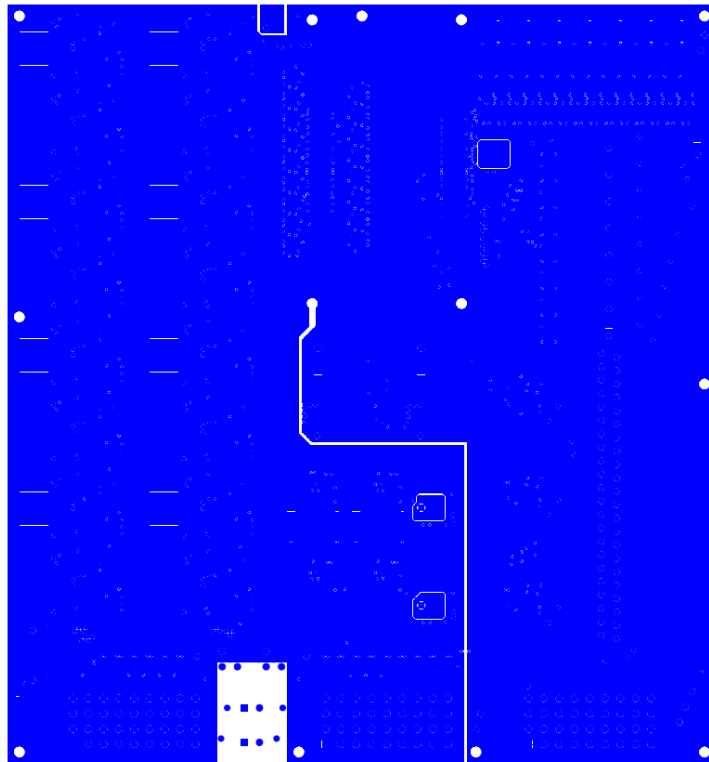
## 7.3 Grounding

In PSC carrier card three ground signals have been used:

1. Analog ground (  )  
It is connected to NV control Card, connector X2; 4, 8, 12, 16, 20, 24 and 28 pins.  
It is used in PSC carrier card for analog systems like: current drivers, digital potentiometers and DACs.
2. Digital ground (  )  
It is connected to NV control Card through the following connectors:  
A) PWM signal connectors: X19 to X22.  
B) Connector X12; 1 and 34 pins.  
C) Connector X3; 4, 8, 12, 16, 20, 24 and 28 pins.  
It is used in PSC carrier card for digital systems like: SoC, input and output expensor ICs, digital input optocouplers and PWM signals.
3. Power ground (  )  
It is connected to NV control Card, connector X3; 22 pin.  
It is used in PSC carrier card in one power supply that generates +12V.

Note: Analog ground and digital ground are connected internally in PSC carrier card. Power ground has been left isolated.

Note 2: The ground plane is partially divided in two areas: one for digital signals (digital ground) and another for analog signals (analog ground). It can be seen in the image below:



**FIGURE 7. 8: PSC CARRIER CARD GROUND PLANE**

## 7.4 Lay-out and signal integrity

There are no critical signals in PSC carrier board.

No special precautions have been taken while routing it.

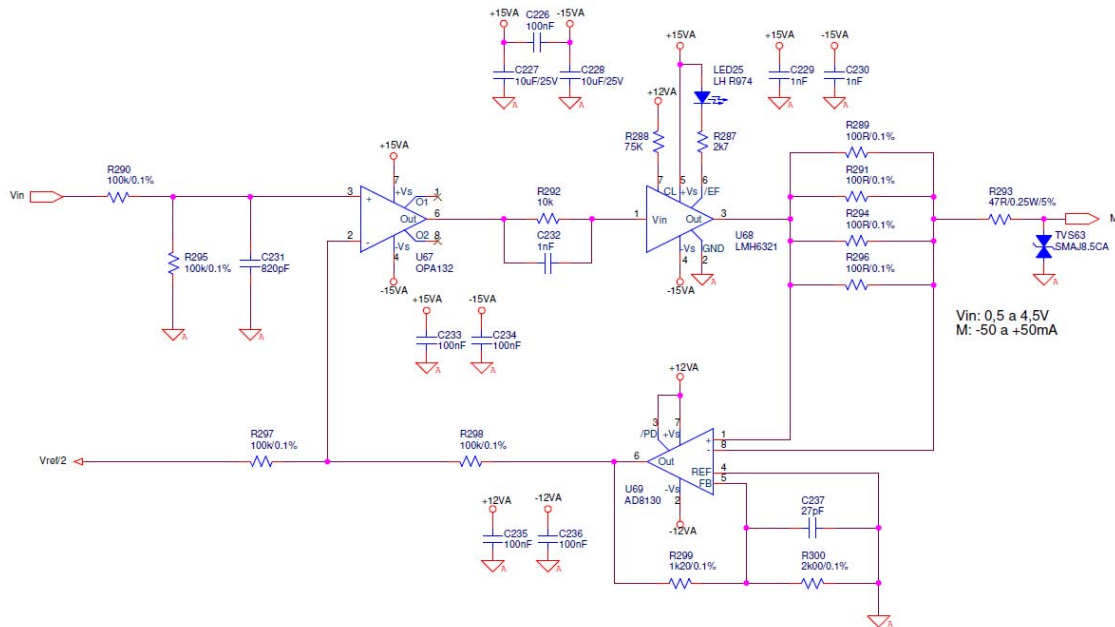
## 7.5 PSC Carrier Boar Project release

The following files are provided:

1. Whole set of schematics with annotations and version control.
2. Gerber files.
3. Datasheets of the ICs which include a serial communication protocol (DAC121s101, TCA9538 and AD5293).

## 7.6 Current amplifiers and DAC chain

This section has been described using Figure 7. 9 current amplifier as an example:



**FIGURE 7. 9: CURRENT AMPLIFIER**

This system includes:

1. An initial voltage divider with a small low pass filter. Its gain is collected in [7.2].

$$V_{out} = V_{in} * \frac{R_{295}}{R_{295} + R_{290}} = V_{in} * 0.5 \quad [7.2]$$

2. OPA132 is an operational amplifier with negative feedback so, according to virtual short-circuit theory, its positive input voltage must be equals to its negative input voltage.

$$V_{OPA132}^{+} = V_{OPA132}^{-} \quad [7.3]$$

3. The feedback of the noninverting OPA132 is done by a resistor which convert the output current into a voltage and provides that voltage to the AD8130 differential amplifier. As there are 4 100ohm paralleled resistors, the equivalent resistor has a value of 25ohm.

$$V_{AD8131} = I_{out} * R_{eq} = I_{out} * 25 \quad [7.4]$$

4. AD8130 has a gain provided by R299 and R300. See [7.5].

$$V_{-} = I_{out} * 25 * \left(1 + \frac{R_{299}}{R_{300}}\right) = I_{out} * 37.5 \quad [7.5]$$

5. As V+ has to be equals to V- in OPA132. See [7.6] and [7.7].

$$V_- = I_{out} * R_{eq} * \left(1 + \frac{R_{299}}{R_{300}}\right) = I_{out} * 37.5 = V_+ = \frac{R_{295}}{R_{295} + R_{290}} * V_{in} \quad [7.6]$$

$$I_{out} = V_{in} * \frac{R_{295}}{R_{295} + R_{290}} * \frac{1}{R_{eq}} * \left(1 + \frac{R_{299}}{R_{300}}\right)^{-1} \quad [7.7]$$

$$I_{out} = V_{in} * 13.33 * 10^{-3}$$

The voltage called Vref/2 in Figure 7. 9 is a voltage which removes the DACs 2.5V offset. This voltage is generated by means of another DAC to enable calibration of the current drivers.

## 8 Annex A: rebuild Vivado project:

### 8.1 Introduction

Vivado can create such a big and disgusting set of files for a project whose weight can make it clearly inoperative for tasks like version control or for sending it.

Included in this folder are all required files necessary to regenerate the whole Vivado project.

The procedure that must be followed relies in different scripts that will regenerate the whole project. The order in the execution of these scripts is critical because some of them are based in the previous ones.

The following software are involved: Vivado 2017.4 and Vivado HLS 2017.4; other versions of the software may require minor adjustments.

### 8.2 Path assignments

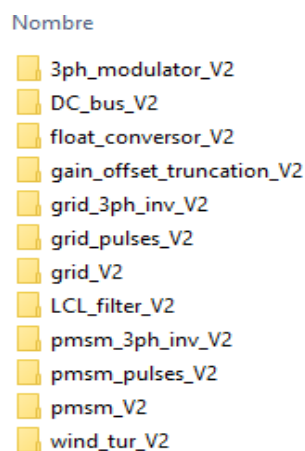
All paths are referred to the main scripts ones. No absolute paths are used during the project.

However, in order to avoid any issue, it is recommended to work in this path: C:\NV\_HIL

### 8.3 Vivado HLS

The first step of the whole process is to regenerate all the Vivado HLS IPs. Those IPs are the ones in which the HIL emulator equations are included.

Those IPs are shown in Figure 8. 1.



**FIGURE 8. 1: HLS IP FOLDER**

To generate those IPs, the following process must be done with all of them:

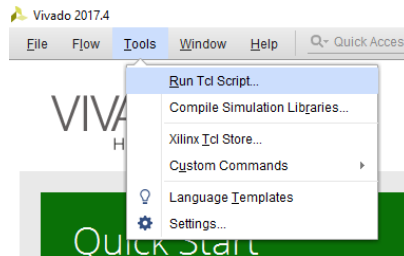
1. Open Vivado HLS command prompt
2. Type: `cd C:\NV_HIL\Vivado\HW_repo\HLS\3ph_modulator_V2`  
Where "C:\NV\_HIL\Vivado\HW\_repo\HLS\3ph\_modulator\_V2" is the path of run.tcl files
3. Type: `Vivado_hls -f run.tcl`
4. Wait for it until it finishes
5. In case it is wanted Vivado GUI to open, type: `Vivado_hls -p 3ph_modulator_V2.prj`  
Where 3ph\_modulator\_V2.prj is the folder where the HLS project has been created.
6. Repeat from 1 to 5 with all the HLS folders



## 8.4 Vivado

To fulfil this step all the Vivado HLS IPs must have been generated.

Open Vivado GUI and click in Tools/Run Tcl Script



**FIGURE 8. 2: RUN A SCRIPT FROM VIVADO**

Select the file run\_prj.tcl and click run.

run\_prj.tcl file internally only adjusts the initial path of Vivado and then called the file NV\_HIL\_prj.tcl which is the one that contains all hardware information of the project.

Wait until the script finishes - It can take up to one hour.

When finished, Vivado will export the bitstream file and will launch SDK.

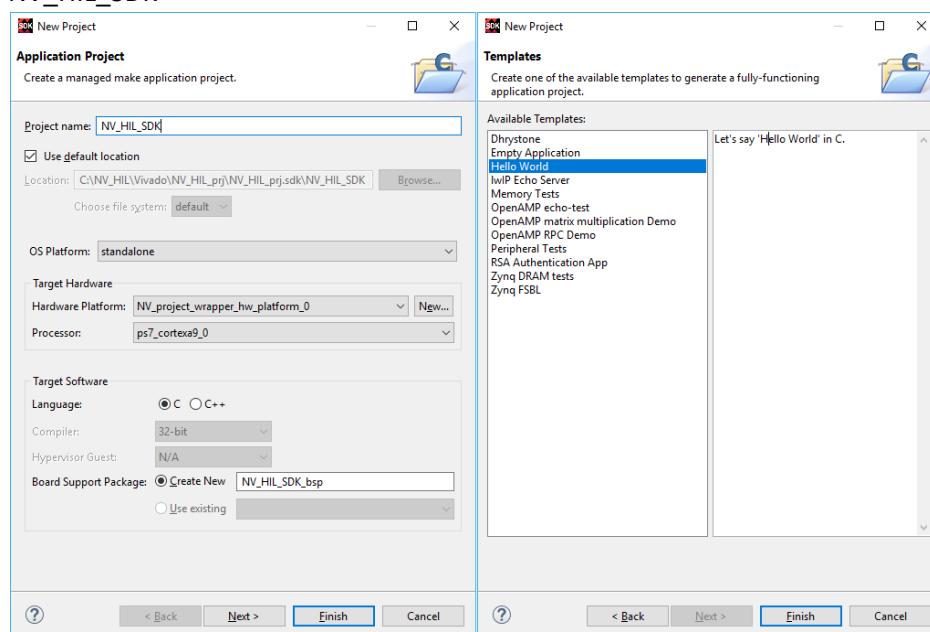
## 8.5 SDK

SDK will configure the ARM microprocessors.

Only one is used and it is in charge of managing the communication with the user desktop console by establishing a serial communication with 1 second of refresh rate.

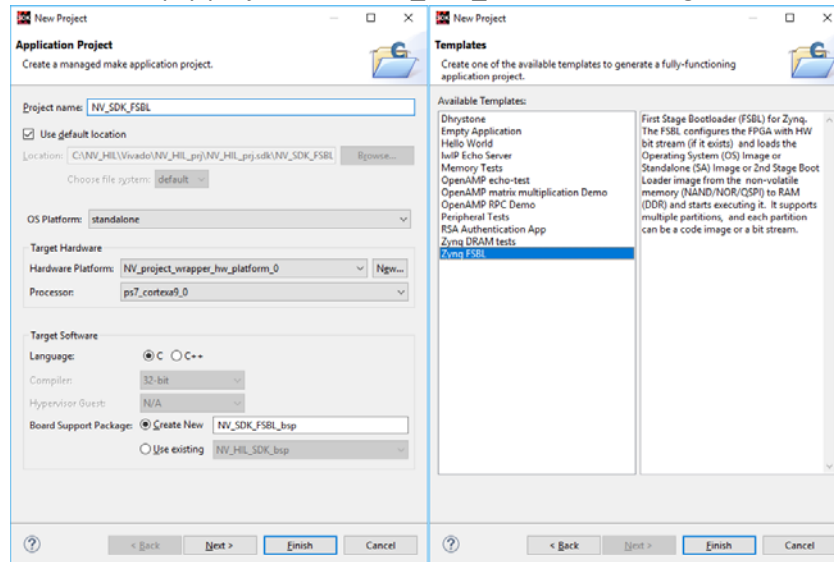
To configure SDK, the following steps must be accomplished:

1. Create a new empty project by clicking: File/New/Application Project with the name NV\_HIL\_SDK



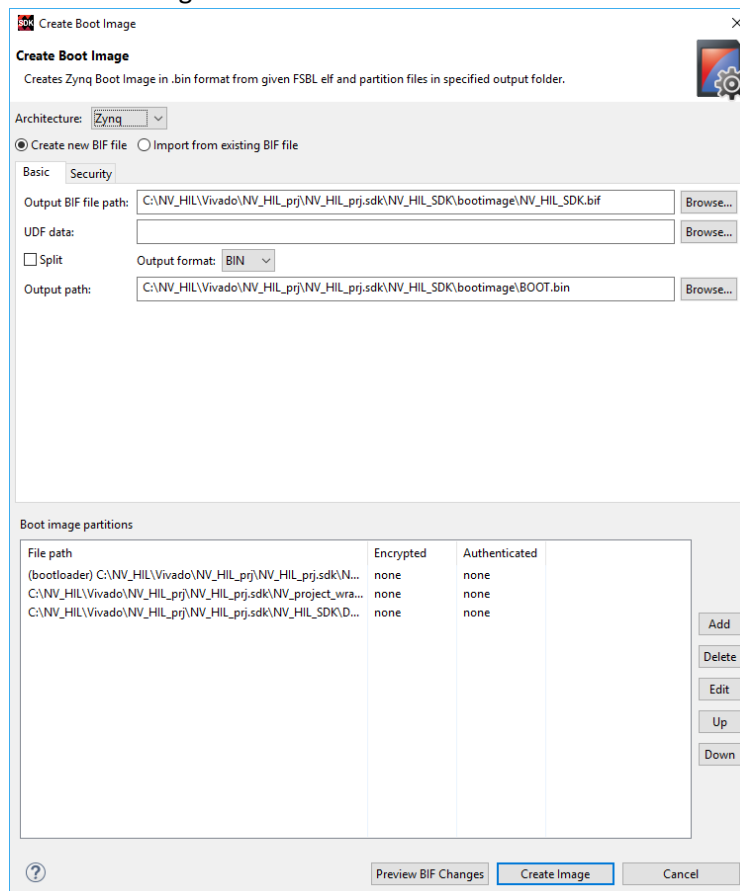
**FIGURE 8. 3: CREATE AN SDK EMPTY PROJECT**

2. Copy the contents of provided helloworld.cpp and paste it in SDK new project. Be sure to delete all previous code. This file can be found in SW\_repo folder.
3. Create a new empty project called NV\_SDK\_FSBL in order to generate the BOOT.ini file



**FIGURE 8. 4: CREATE A SDK FSBL**

4. Create the BOOT.ini file by selecting in SDK the folder NV\_HIL\_SDK and clicking in Xilinx/Create Boot Image menu.



**FIGURE 8. 5: CREATE THE BOOT.INI FILE IN SDK**

5. Go to this path in order to find the BOOT.ini file:  
C:\NV\_HIL\Vivado\NV\_HIL\_prj\NV\_HIL\_prj.sdk\NV\_HIL\_SDK\bootimage  
NOTE: the path may vary.
6. Insert an 8GB microSD card in the computer and format it as FAT32.
7. Paste BOOT.ini file in the microSD card without modifying its name.
8. Remember to configure correctly the microZED board jumpers to boot from microSD card.



**FIGURE 8. 6: MICROZED JUMPER CONFIGURATION TO BOOT FROM USD CARD**

## 8.6 Notes

To avoid the user to do the whole regeneration of the project, the following files are provided as they will prove to be useful:

1. BOOT.bin
2. NV\_project\_wrapper.bit

Both of them are in programming\_files folder

## 9 Annex B: Current driver in depth analysis:

### 9.1 General concepts

In this annex it has been done a small signals analysis of the current driver circuit so it can be checked analog characteristics such as its gain and its bandwidth.

Figure 9. 1 shows one current driver circuit.

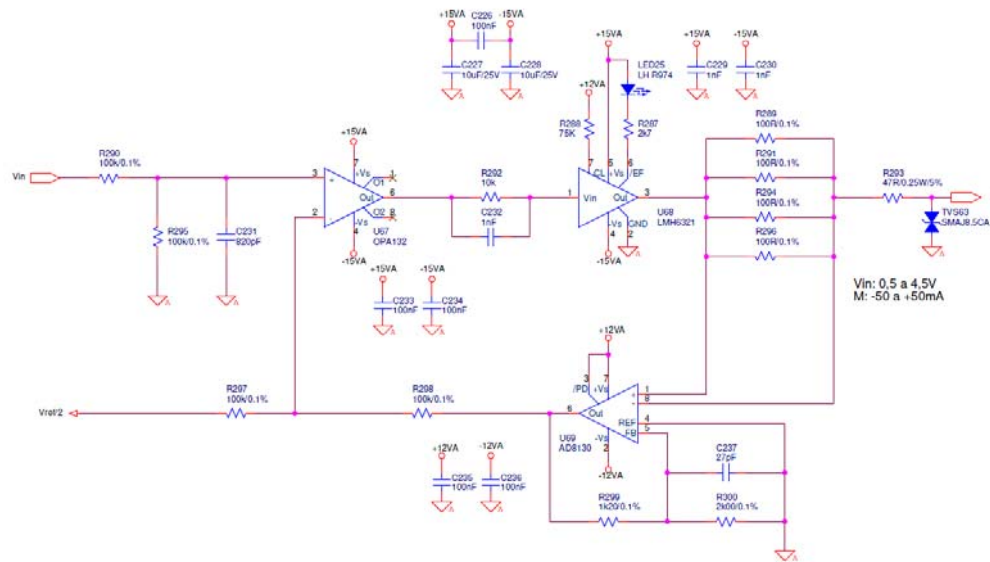


FIGURE 9. 1: CURRENT DRIVER CIRCUIT

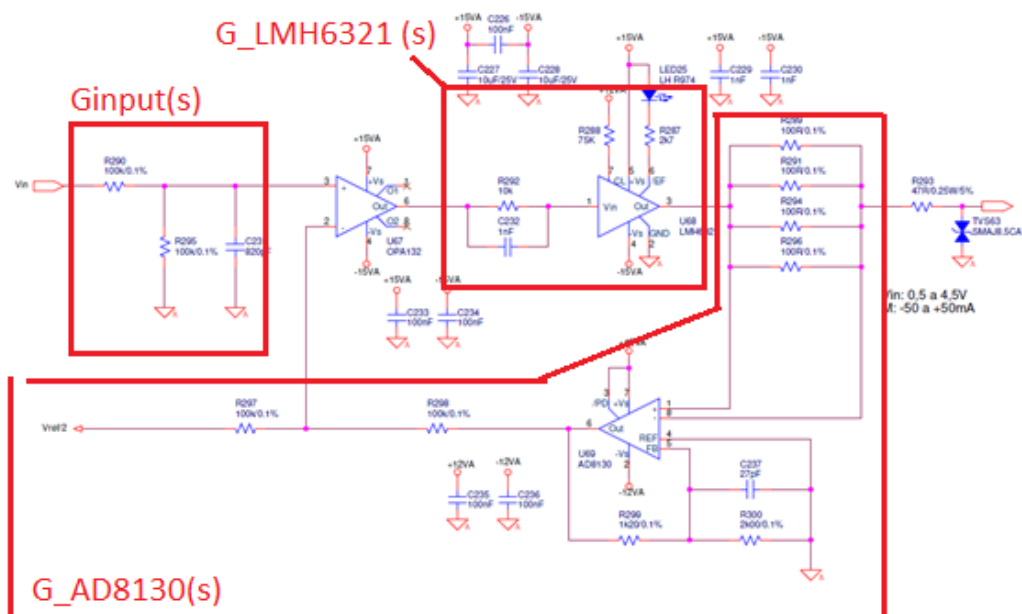


FIGURE 9. 2 CURRENT DRIVER CIRCUIT MAIN BLOCKS

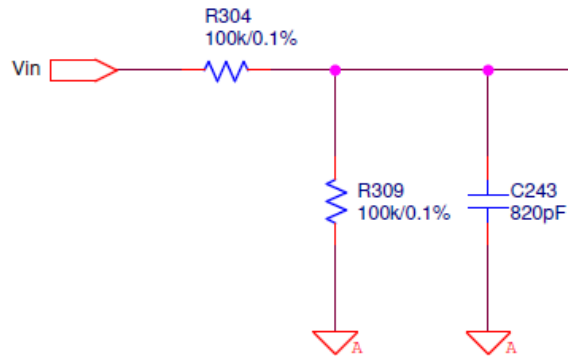
This complex circuit has been splitted on three main blocks as it can be seen in Figure 9. 2:

1.  $G_{input}(s)$ : input voltage divider and filtering
2.  $G_{LMH6321}(s)$ : It generates the output current from a voltage
3.  $G_{AD8130}(s)$ : It converts output current into voltage, applies a gain and feedback the signal.

Those blocks have been analysed deeper in the next sections.

## 9.2 $G_{input}(s)$ calculation

Figure 9. 3 provides the circuit of this block.

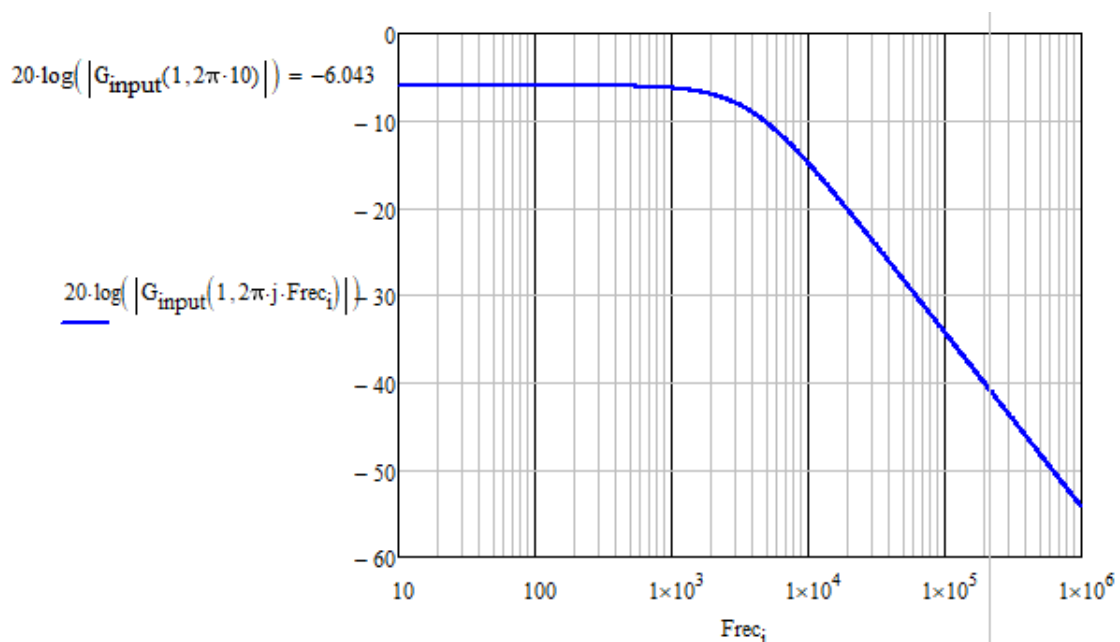


**FIGURE 9. 3: INPUT VOLTAGE DIVIDER AND FILTER**

Its transfer function is determined by the equivalent impedance. See [9.1].

$$G_{input}(s) = \frac{V_{out}}{V_{in}} = \frac{R_{309}}{1 + R_{309} * s * C_{243}} = \frac{500000}{41 * s + 1000000} \quad [9.1]$$

Figure 9. 4 and Figure 9. 5 provides a plot graph with the representation of  $G_{input}(s)$  transfer function. This function has a gain of -6dB @low freq and a BW of 4kHz @-45deg.



**FIGURE 9. 4: BODE PLOT OF  $G_{input}(s)$ . MODULE**

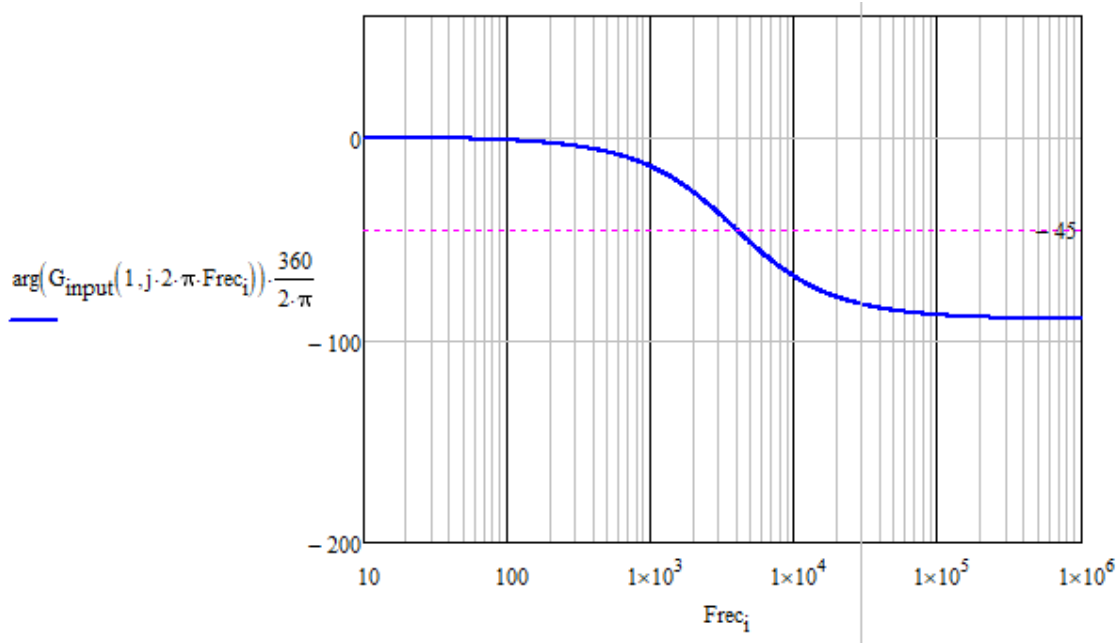


FIGURE 9. 5: BODE PLOT OF  $G\_INPUT(s)$ . PHASE

### 9.3 $G\_LMH6321(s)$ calculation

Figure 9. 6 provides the circuit of this block.

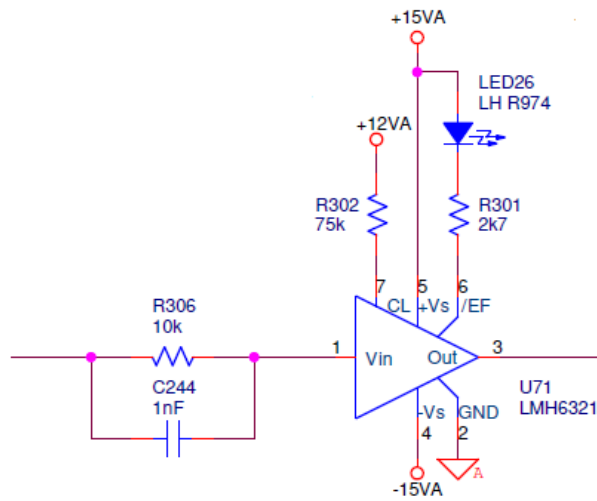


FIGURE 9. 6: CURRENT BUFFER CIRCUIT

The gain of this block and its BW are expressed by equations [9.2] to [9.5].

$$I_{EXT} = \frac{V_{prog}}{R_{ext}} = \frac{12V}{75k} = 160\mu A \quad [9.2]$$

$$V_{in} = 3V \Rightarrow I_{out} = 0.3A \Rightarrow Gain_{LMH6321} = \frac{I_{out}}{V_{in}} = \frac{0.3}{3} = 0.1 \frac{A}{V} \quad [9.3]$$

$$f_p = \frac{1}{2 \cdot \pi \cdot R1 \cdot C1} = 4.5MHz @ -20 \frac{dB}{dec} (datasheet) \quad [9.4]$$

$$G_{LMH6321}(s) = \frac{I_{out}}{V_{in}} = Gain * \frac{1}{1 + \frac{s}{2 * \pi * 4.5 * 10^6}} = \frac{2.857 * 10^6}{s + 2.857 * 10^7} \quad [9.5]$$

Figure 9. 7 and Figure 9. 8 provides a plot graph with the representation of  $G_{LMH6321}(s)$  transfer function. This function has a gain of -20dB @low freq and a BW of 6MHz @ -45deg.

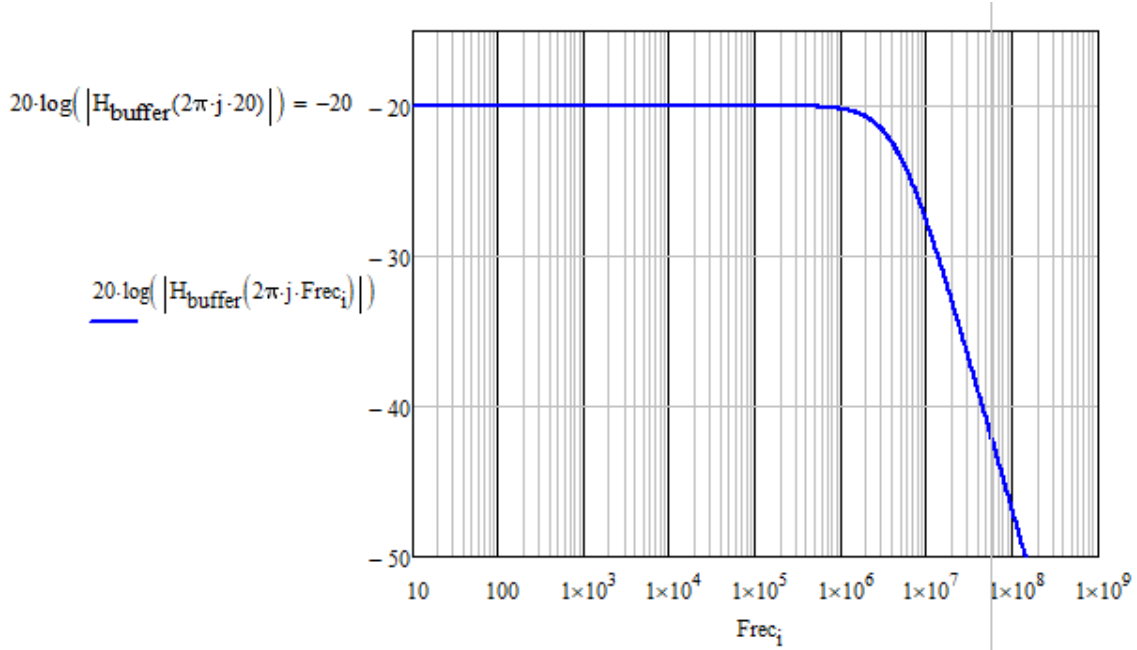


FIGURE 9. 7: BODE PLOT OF  $G_{LMH6321}(s)$ . MODULE

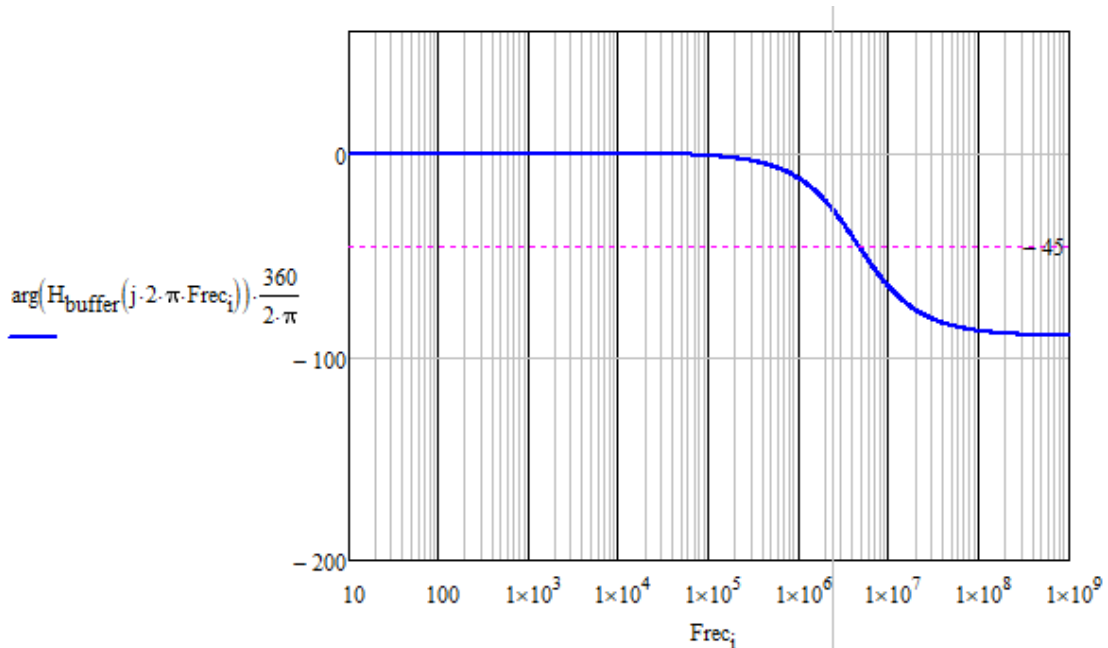
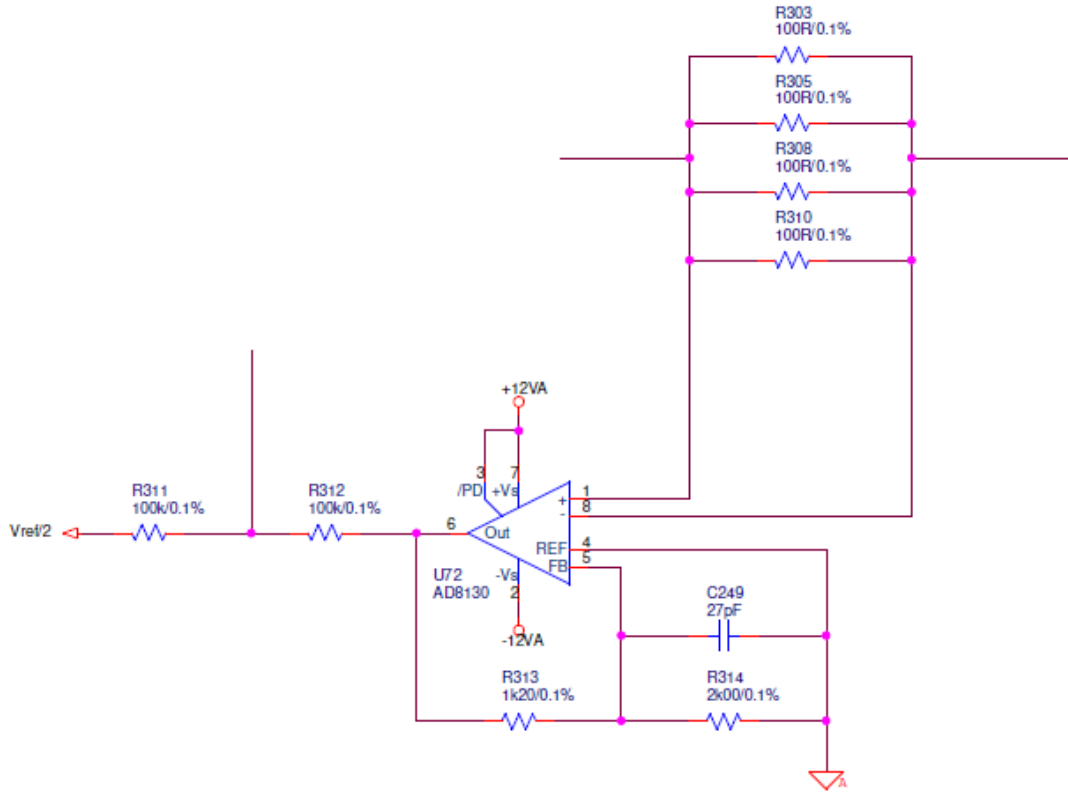


FIGURE 9. 8: BODE PLOT OF  $G_{LMH6321}(s)$ . PHASE

## 9.4 G\_AD8130(s) calculation

Figure 9. 9 provides the circuit of this block.



**FIGURE 9. 9: DIFFERENTIAL AMPLIFIER AND FEEDBACK SIGNAL**

This system contains two smaller blocks:

1. A resistor train which translates the output current to voltage
2. A differential amplifier which generates a gain.

The gain of this block and its BW are expressed by equations [9.6] to [9.5].

$$R_{eq} = R303 || R305 || R308 || R310 = \frac{100}{4} = 25 \text{ ohm} \quad [9.6]$$

$$V_{in} = I_{out} * R_{eq} \quad [9.7]$$

$$Gain = \frac{V_{out}}{V_{in}} = 1 + \frac{R313}{R314} = 1.6 \frac{V}{V} \quad [9.8]$$

$$G * BW = \text{constant} \Rightarrow G = 1 \text{ has } BW = 270\text{MHz} \text{ so } G = 1.6 \text{ has } BW = 168.75\text{MHz} \quad [9.9]$$

$$G_{AD8130}(s) = \frac{V_{out}}{I_{out}} = Gain * \frac{1}{1 + \frac{s}{2 * \pi * 168.75\text{MHz}}} = \frac{4 * 10^{31}}{9.43 * 10^{20} * s + 10^{30}} \quad [9.10]$$

Figure 9. 10 and Figure 9. 11 provides a plot graph with the representation of G\_ LMH6321 (s) transfer function. This function has a gain of 32dB @low freq.



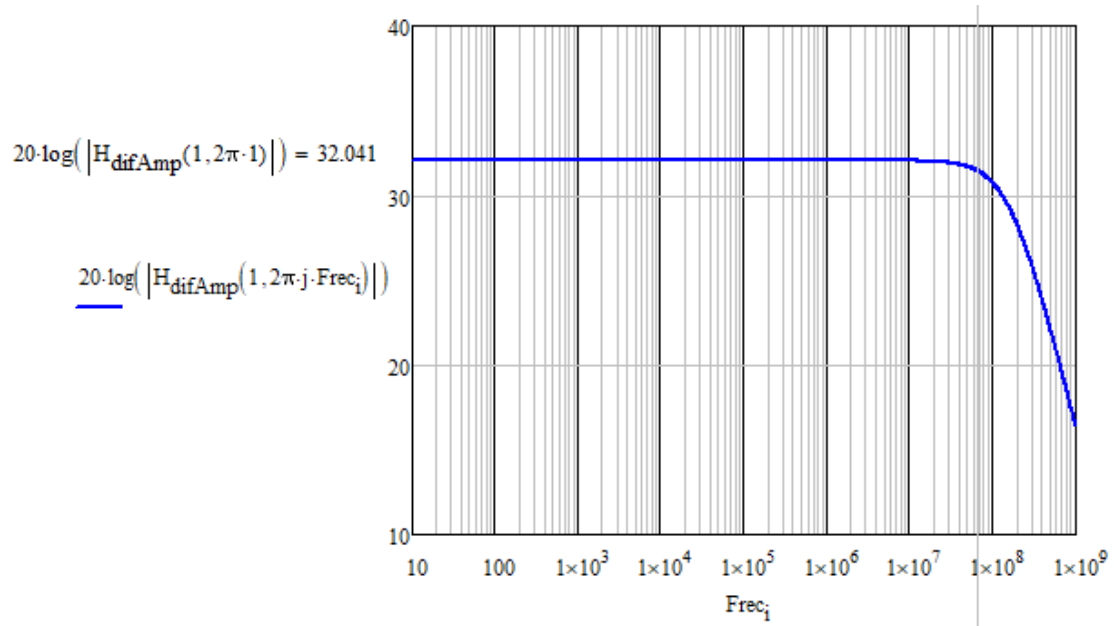


FIGURE 9. 10: BODE PLOT OF G<sub>L</sub> AD8130 (s). MODULE

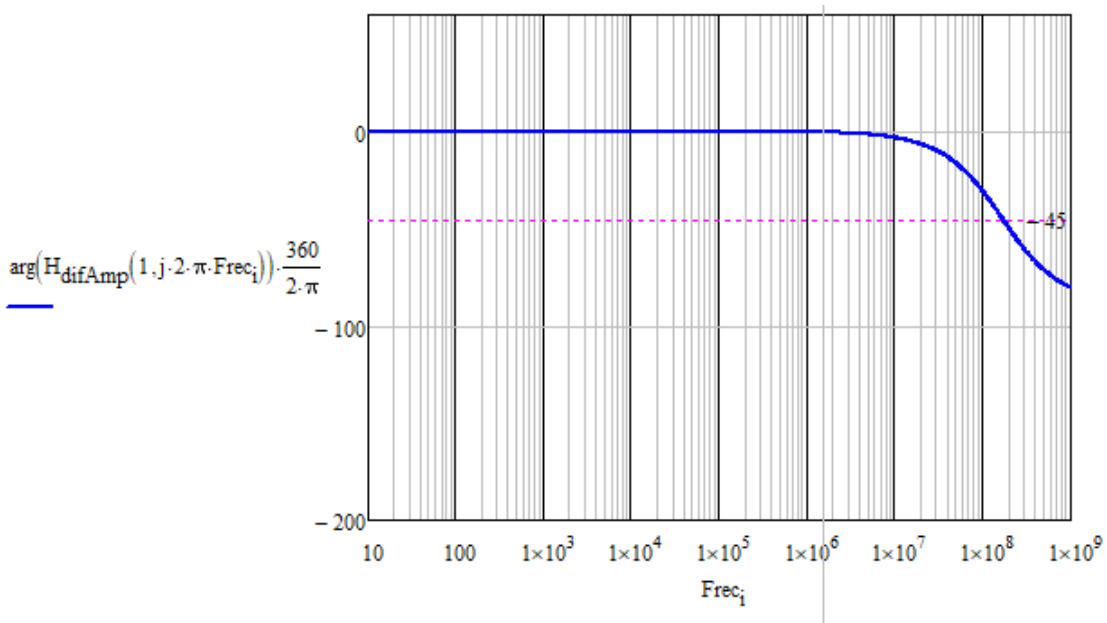


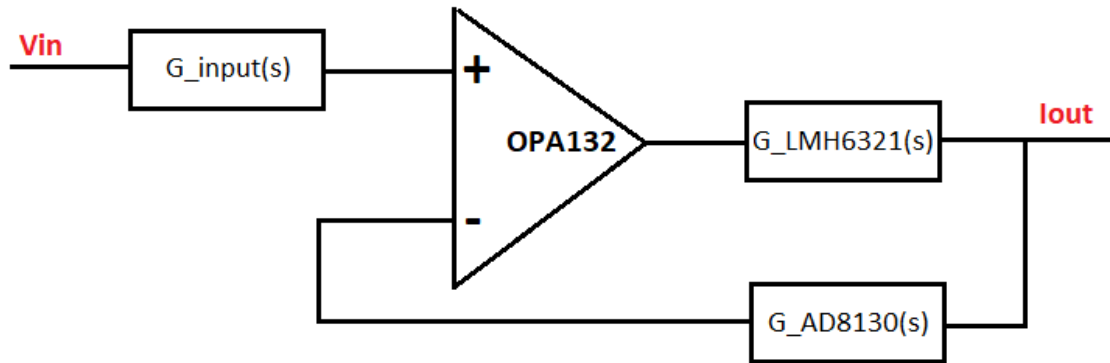
FIGURE 9. 11: BODE PLOT OF G<sub>L</sub> AD8130 (s). PHASE

## 9.5 Closed loop analysis

In previous sections, all the blocks have been defined and its transfer function achieved. In this section the close loop transfer function of the whole current driver has been calculated.

The closed loop transfer function is denoted by expression [9.11] and based on Figure 9. 12.

$$G_{ClosedLoop}(s) = \frac{I_{out}}{V_{in}} = \frac{Gain_{OPA132} * G_{LMH6321}(s)}{1 + Gain_{OPA132} * G_{LMH6321}(s) * G_{AD8130}(s)} * G_{input}(s) \quad [9.11]$$

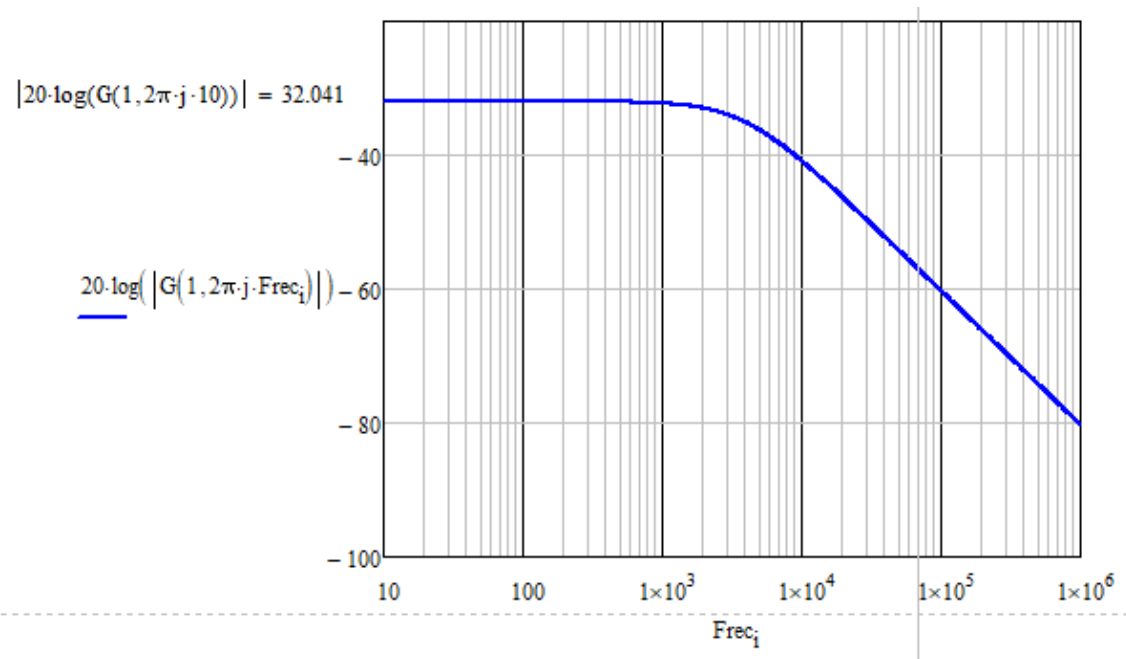


**FIGURE 9. 12: CURRENT DRIVER BLOCK DESIGN**

If [9.11] is substituted and operated, expression [9.12] is obtained.

$$G_{ClosedLoop}(s) = \frac{8.51 * 10^{51} * s + 9.03 * 10^{60}}{3.87 * 10^{33} * s^3 + 4.21 * 10^{42} * s^2 + 1.48 * 10^{58} * s + 3.61 * 10^{62}} \quad [9.12]$$

Figure 9. 13 and Figure 9. 14Figure 9. 8 provides a plot graph with the representation of  $G_{closed\_loop}(s)$  transfer function. This function has a gain of -32dB @low freq and a BW of 4kHz @-45deg.



**FIGURE 9. 13: BODE PLOT OF  $G_{CLOSED\_LOOP}(s)$ . MODULE**

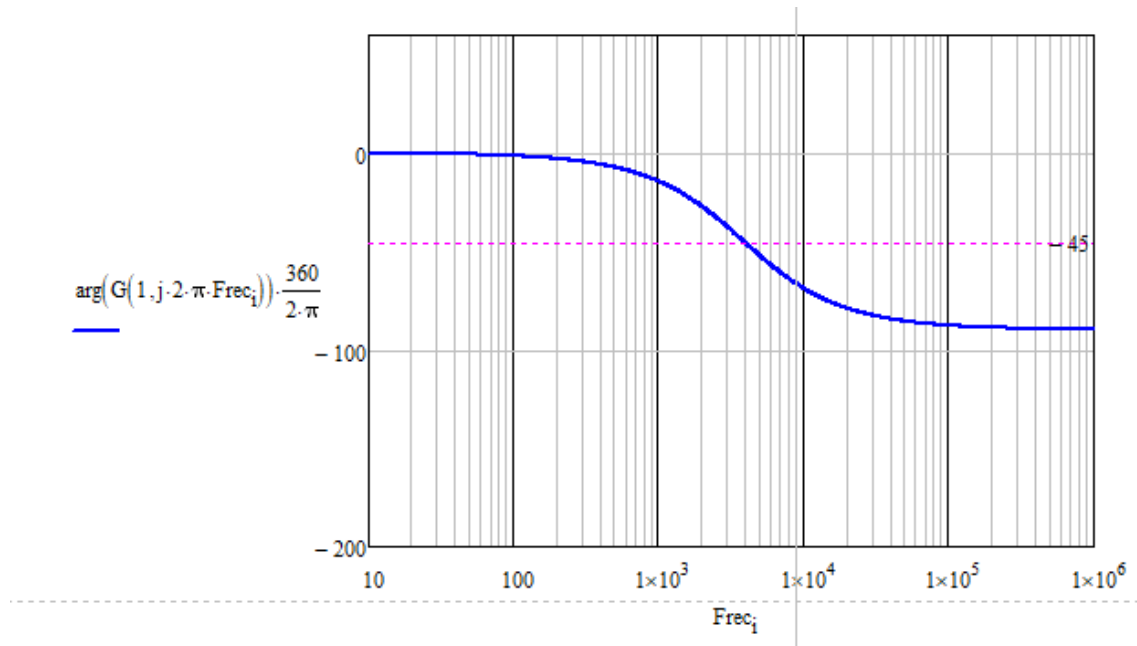


FIGURE 9. 14: BODE PLOT OF  $G_{\_CLOSED\_LOOP}(s)$ . PHASE

## 9.6 LTSpice comparison

In order to check the accuracy of all the expressions of previous sections, it have been made two simulations in LTSpice:

1. A time domain simulation. It is shown the input signal (voltage) and the output signal (current). As it can be seen in Figure 9. 15 and Figure 9. 16, a 4.5V input signal produces an output of 50mA. The output current has been measured with opposed polarity, it means that the current driver does not invert the signal.
2. A Frequency domain simulation in which the closed loop transfer function has been measured. See Figure 9. 17 and Figure 9. 18.

In order to carry out this comparison it has been necessary to create the models of the components in LTSpice. To do this, it has been used the spice models provided by IC manufacturers.

As it can be seen, Figure 9. 18 represent the same transfer function than the one represented by Figure 9. 13 and Figure 9. 14. Being it the current driver closed loop transfer function. See [9.13].

$$G_{ClosedLoop}(s) = \frac{8.51 * 10^{51} * s + 9.03 * 10^{60}}{3.87 * 10^{33} * s^3 + 4.21 * 10^{42} * s^2 + 1.48 * 10^{58} * s + 3.61 * 10^{62}} \quad [9.12]$$

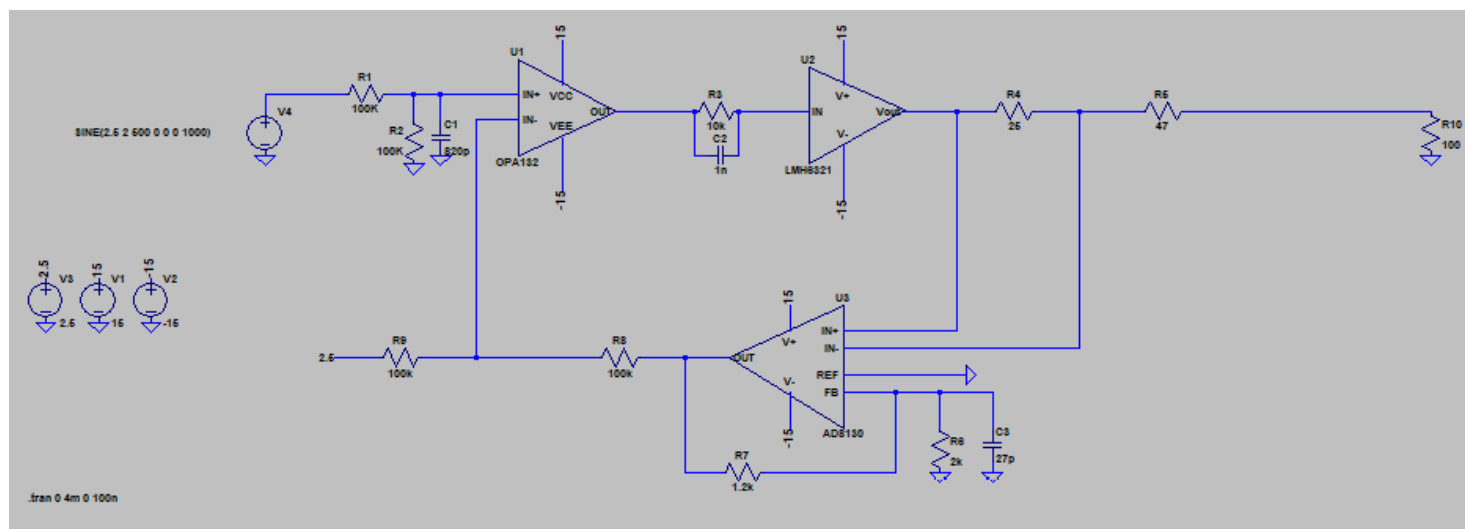


FIGURE 9. 15: LTSPICE TIME DOMAIN SCHEMATIC

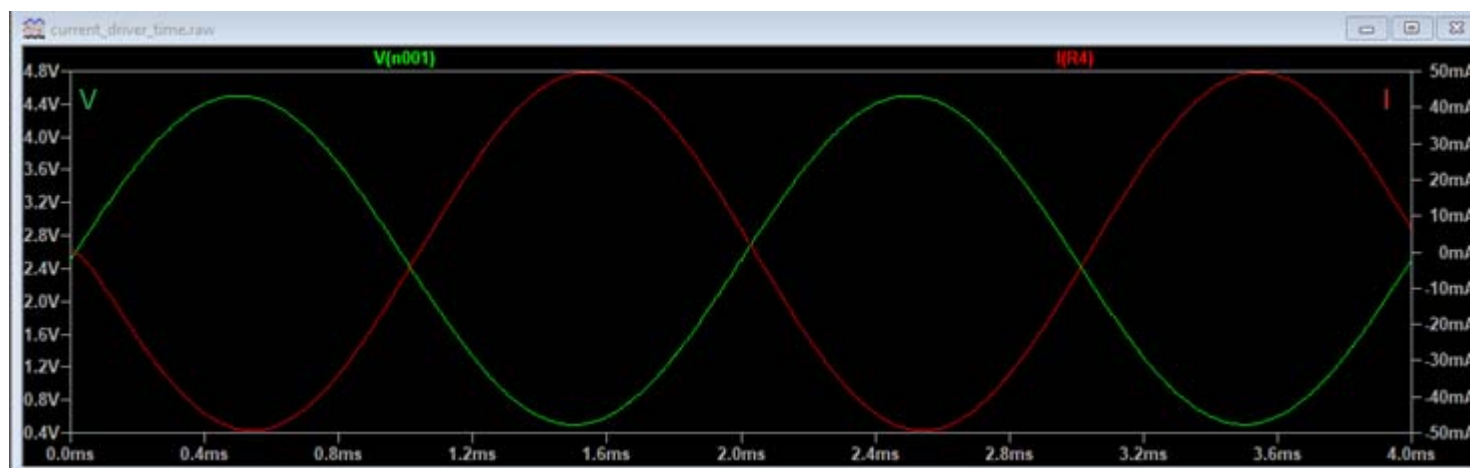


FIGURE 9. 16: LTSPICE TIME DOMAIN SIMULATION

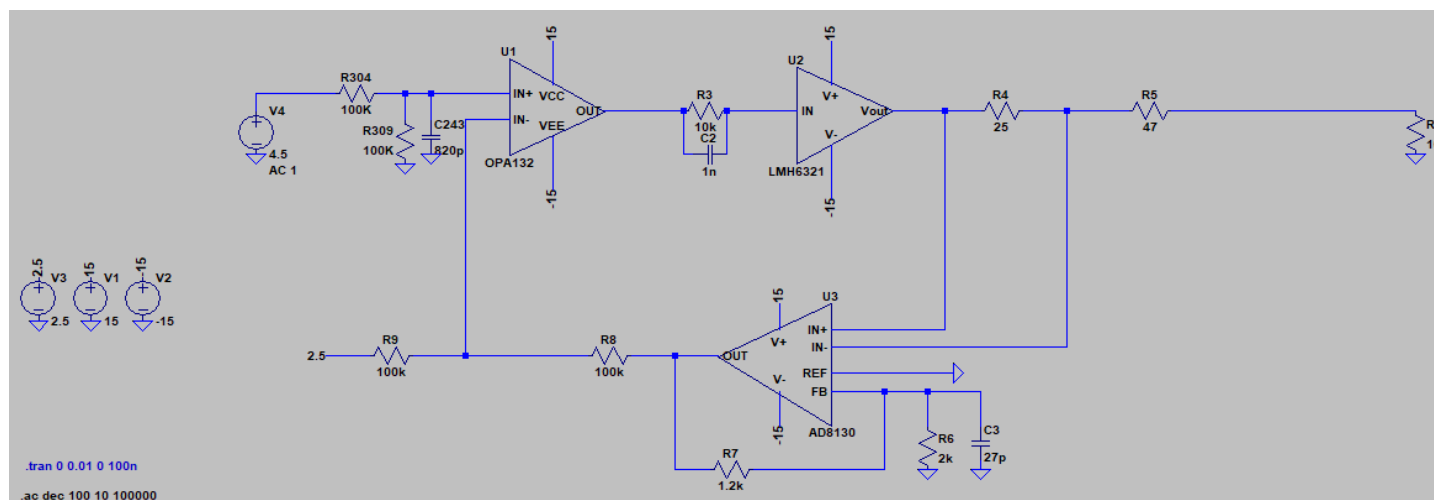


FIGURE 9. 17: LTSPICE FREQUENCY DOMAIN SCHEMATIC

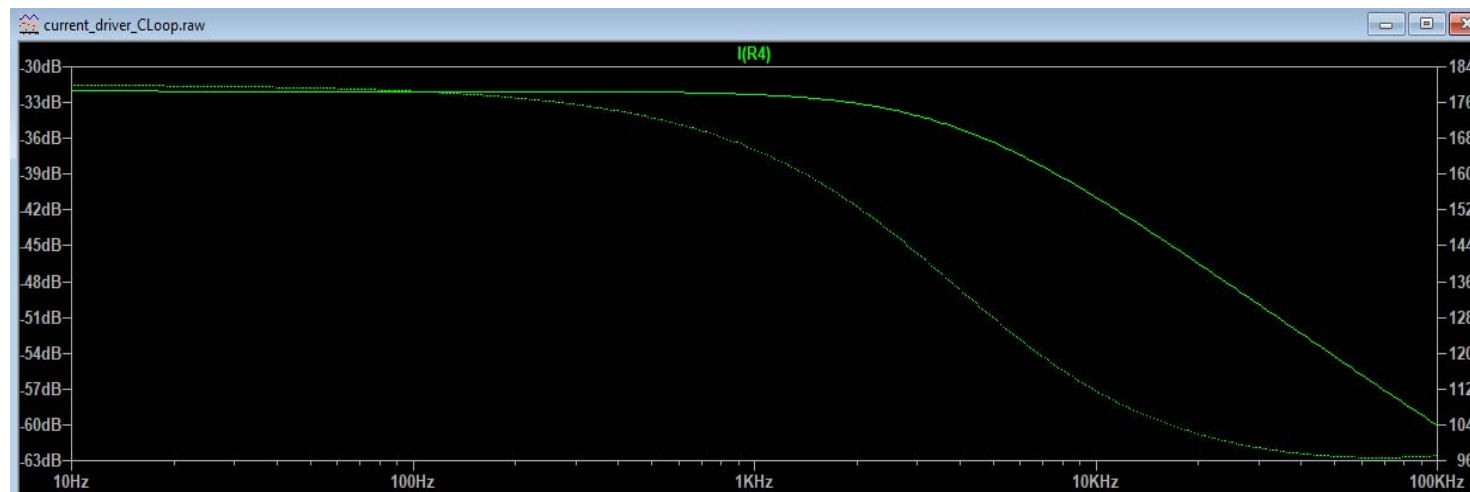


FIGURE 9. 18: LTSPICE FREQUENCY DOMAIN CLOSED LOOP SIMULATION

