PM-Machine Field Oriented Control
SGL Implementation

Introduction

The FOC control algorithm has been implemented in LabVIEW by SGL numeric representation, and the resulting program is made up of three different layers, which are executed on dedicated hardware. The main part runs on the FPGA and it is related to the main control loops, generating PWM signals and managing hardware devices connected to the board. Furthermore, a second program is executed by the Real-Time target in order to perform the less demanding tasks and, if needed, to make the Graphical User Interface (GUI) available. A third layer running on the Host PC is also possible and it can be used to execute the non-deterministic tasks such as handling the GUI in order to lighten the Real-Time processor's workload.
The implemented algorithm block diagram is shown in Figure 1, where the blocks executed by different targets are represented using different colors.

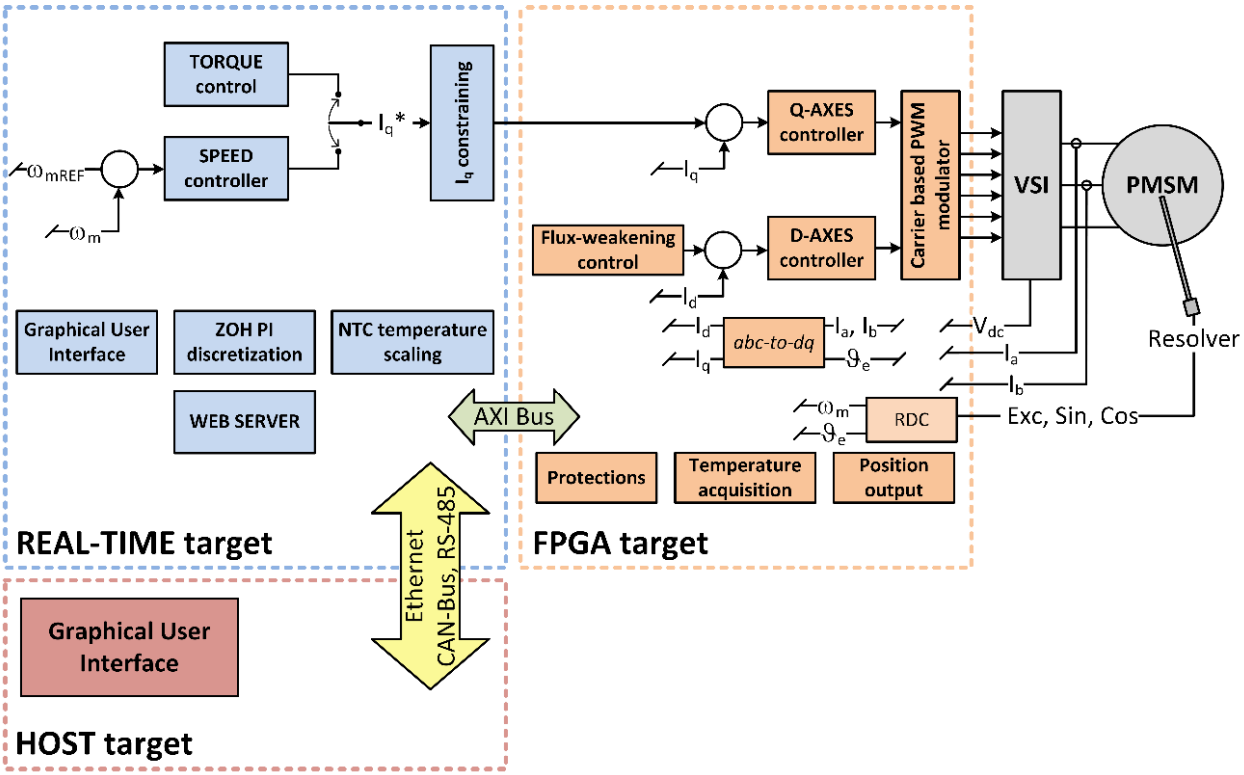


Figure 1. Algorithm block diagram highlighting the different tasks executed by the RT-Target (blue), the ones performed by the FPGA-Target (orange), and by the Host-Target (red).

Real-Time Target

The program executed by the Real-Time (RT) Target is used to load and to communicate with the FPGA top-level VI, to realize the speed controller and to handle the GUI.
It exhibits a simple structure than the FPGA VI and it is made up of a single high-speed loop that contains a Stacked Sequence Structure. The program starts loading the FPGA configuration VI and initializing the variable named "*PWMen*" to FALSE, in order to ensure that PWM output pulses are inhibited until the user decides to enable the control action. After that, the high-speed loop starts and it is executed every 2000 µs (this value can be chosen by the user modifying the variable named "*Period*") until the user presses the "*STOP*" button or an error occurs. The Stacked Sequence Structure is made up of four frames, extensively described in the following sections. Moreover, for test purposes, waveform charts for current components and rotor speed are made available.

Frame 0 – Reading from FPGA

In this frame, shown in Figure 2, the required quantities are read from the FPGA and they are reported on the GUI. In order to make the information more intuitive, the value of "*Sample Time (ticks@40MHz)*" is converted to µs by multiplying it by 0,025 and showed under the name of "*Exec Time [µs]*". Moreover, the measure of the rotor speed is filtered using a first order, low-pass Butterworth Filter with a cutoff frequency of 10 Hz and it is displayed on the Front Panel through the indicator named "*SpeedF [rpm]*".
The last part of the frame is responsible of providing the two temperature measures related to the machine's phases "B" and "C". For this purpose, the two raw values (T_{ADC}) are extracted from the array called "*NTC1,2*" and are multiplied by 0,01093. This constant results from the following expression.

$$\frac{R(T)}{R_{100}} = T_{ADC}\frac{V_{ADC\_Max}}{2^{n\_bit} I_{SET} R_{100}} = 0,01093 \cdot T_{ADC}$$

After that, the two temperature values are obtained as follows.

$$T[°C] = \frac{T_{100} B_{100/125}}{B_{100/125} + T_{100} \ln\left(\frac{R(T)}{R_{100}}\right)} - 273,15$$

Frame 1 – Protections

This frame is responsible to handle the errors that are detected by the FPGA. Every error is displayed on the GUI and it can be reset by pushing the *Reset Errors* button. Moreover, maximum values for I_s, V_{dc} and ω_m are set and they are sent to the FPGA.

Frame 2 – Speed controller and I_Q reference value calculation

In this frame, showed in Figure 4, a PI-type speed controller is implemented. The desired rotor speed is compared with the measured one and the controller provides the reference value for the i_q current component as its output, in order to minimize the speed error. If PWM output is disabled or Torque Control mode is chosen, the PI is disabled and reset.
The speed controller is active only if the "*speed control mode*" is enabled by pushing the corresponding button on the GUI. If the "*torque control mode*" is selected, i_q reference value is given directly by the user in the range between 0 % and 100 % of I_s^{MAX}. In both cases, though, i_q value is constrained in the range between $-i_q^{MAX}$ and $+i_q^{MAX}$, where i_q^{MAX} is obtained with the following expression.

$$i_q^{Max} = \sqrt{I_s^{Max2} - i_d^{\,2}}$$

This action is needed because the i_d current component can have a non-zero value, and this occurs in case of flux weakening operating condition.
The "*IqSat*" Boolean indicator warns the user if I_Q value has been coerced.

Frame 3 – Writing to FPGA

This frame, shown in Figure 5, is responsible for sending the required data to the FPGA program. After reading the controls on the GUI, the PI controllers parameters for I_Q, I_D and Flux Weakening control are converted to switch to the discrete domain according to the Zero Order Hold (ZOH) technique. For this purpose, the following expressions have been used.

$$K(z) = K_P$$

$$SampleTime[s] = SampleTime[ticks@40MHz] \cdot 25 \cdot 10^{-9}$$

$$K(z-1) = K_I SampleTime[s] - K_P$$

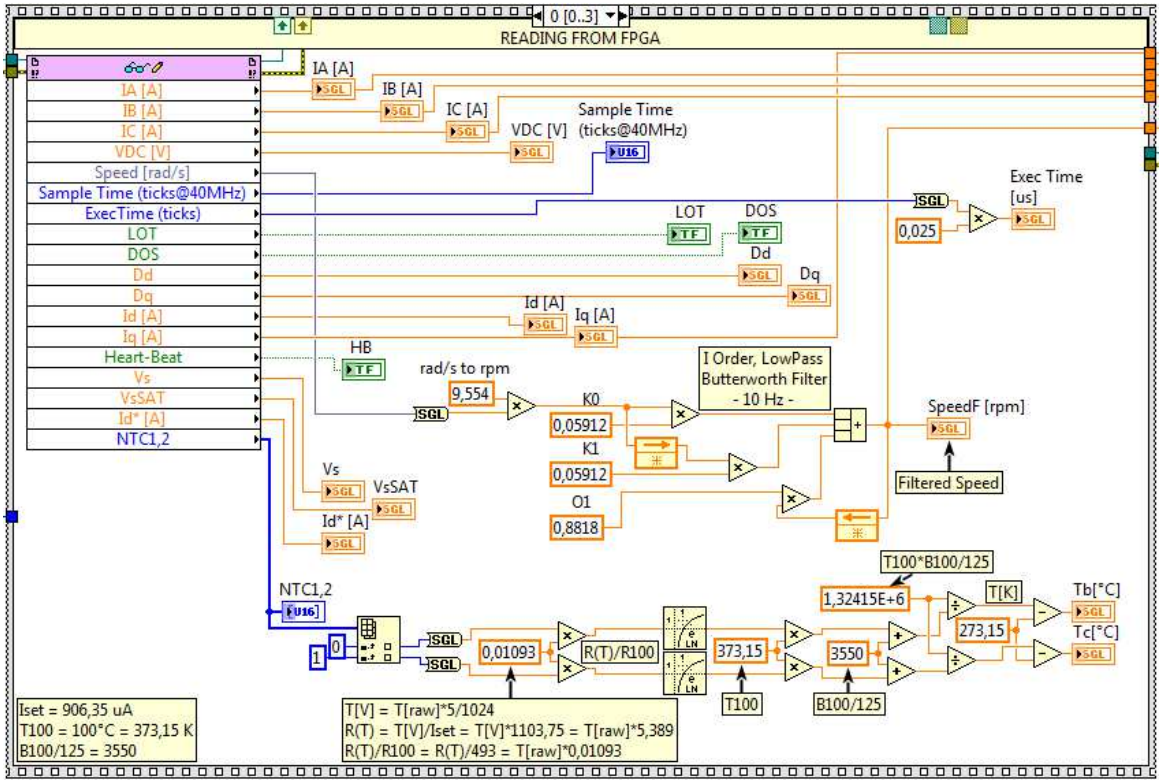


Figure 2. RT Target Frame 0 – Reading from FPGA.

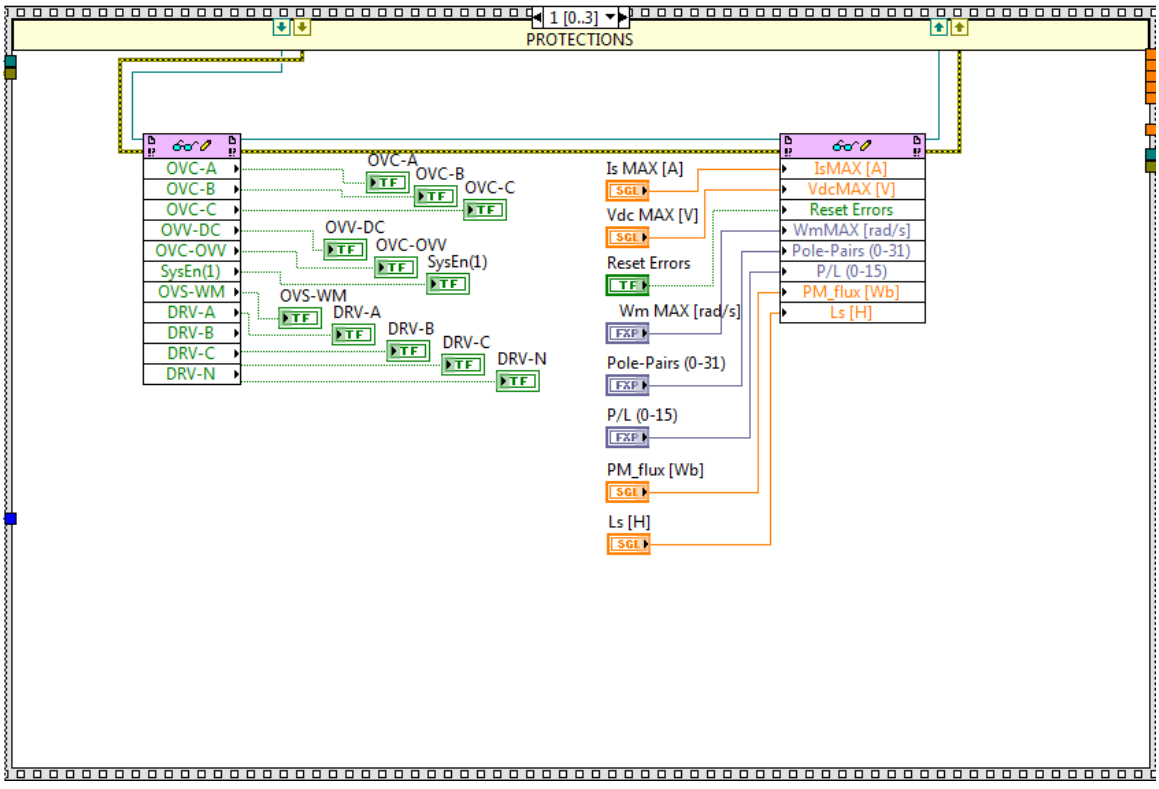


Figure 3. RT Target Frame 1 – Protections.

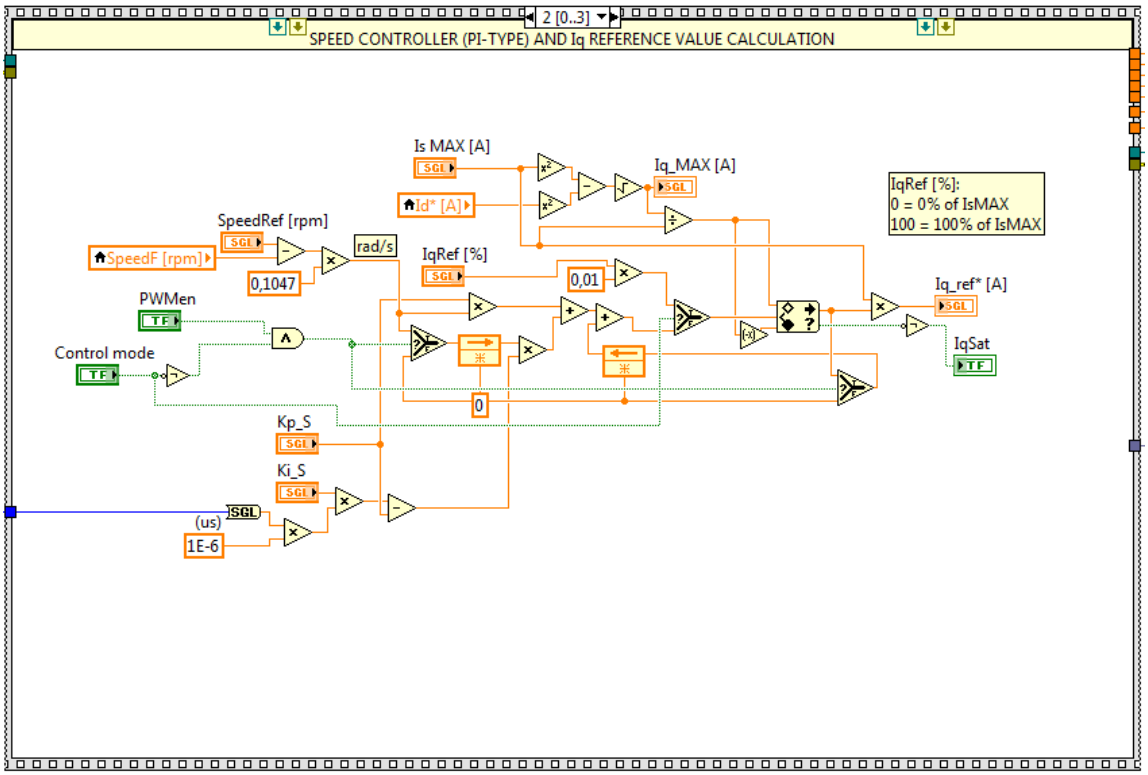


Figure 4. RT Target Frame 2 – Speed controller and I_Q reference value calculation.

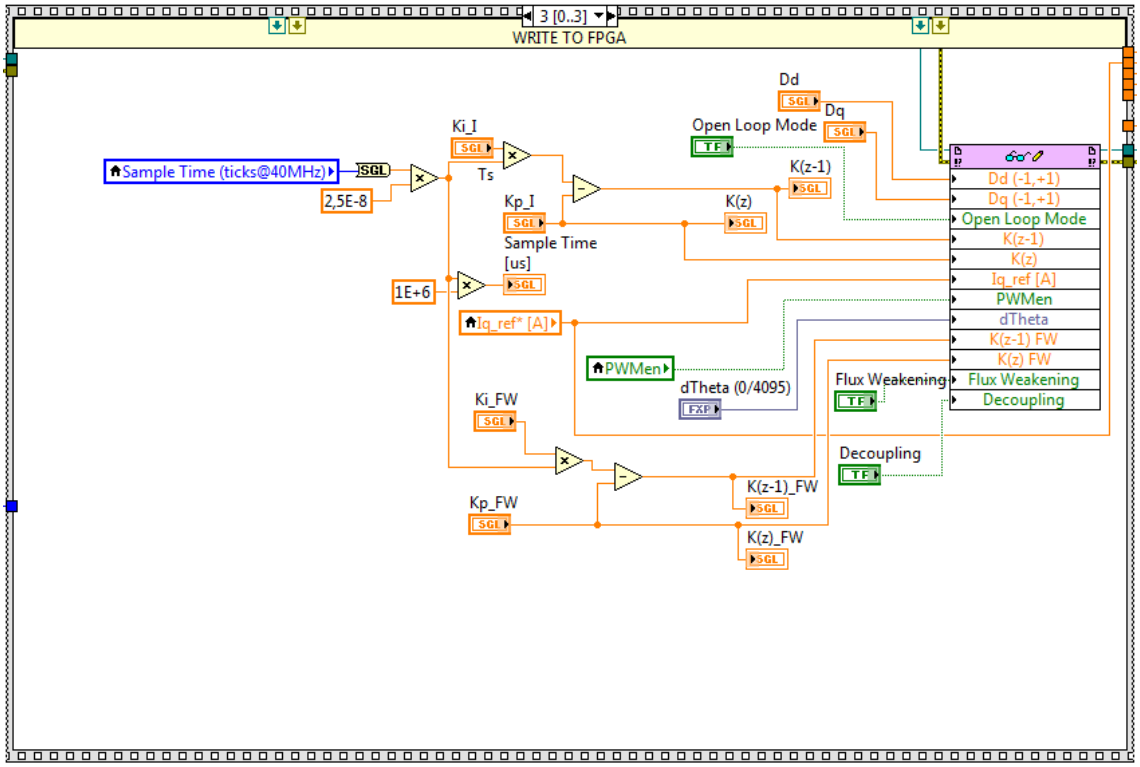


Figure 5. RT Target Frame 3 – Write to FPGA.

RT Target Front Panel

The realized GUI needed to manage the aforementioned program consists of a LabVIEW Front Panel, divided in three tabs.

The main tab, shown in Figure 6Figure, allows for the complete control of the electrical drive. The picture has been divided into sections for a simpler description.

In section *A*, the controls are mainly related to the electrical machine's parameters, such as maximum allowed current and voltage, pole-pair number, phase resistance and inductance.

Section *B* is dedicated to error indicators, which are derived from over voltage, over current and over speed conditions or drivers' faults.

In section *C*, the measured values about currents and voltage and the resulting components as well as duty cycles used in the algorithm are shown. The measures about inverter temperatures are placed in section *D*. Section *E* contains the controls available to the user to enable and disable the algorithm or specific functions such as Flux-Weakening algorithm. Section *F* is dedicated to resolver errors and to set the angle required for alignment. In section *G*. the parameters related to the code executed by the Real Time target are placed. Section *H* is related to the PI controllers of the DQ-axis currents. Controllers gains can be set in the s-domain, then they are discretized and sent to the FPGA. Similarly, the parameters of the PI controller used in the Flux-Weakening algorithm can be set in section *I*. In section *L* the user can set the desired values of i_q, and consequently torque, or rotational speed. Section *M* shows if the FPGA is running (heart-beat), its execution time and sampling time.

Obviously, there is also a speed indicator. Moreover, a *Reset* button is present in order to reset the errors.

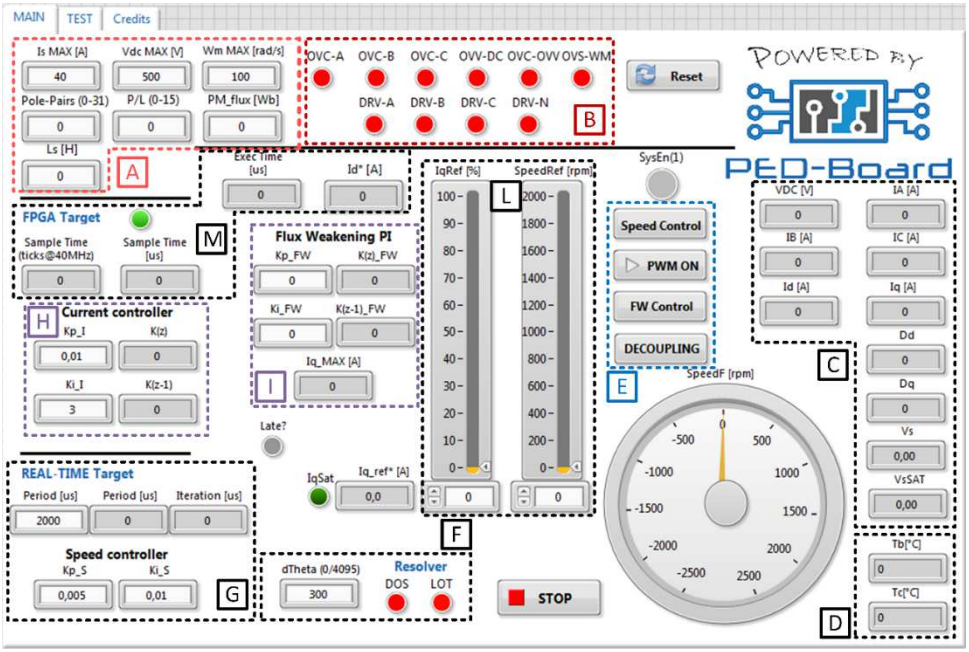


Figure 6. RT Target Front Panel – Main Tab.

The second tab, shown in Figure 7Figure, is realized only for test purposes. It contains two Waveform Charts where the user can visualize the measured values about mechanical speed and the machine current components. Moreover, there is a control named "*Open Loop Mode*" through which it is possible to disable the control algorithm and manually choose D_d and D_q values. It is important to note that these values will be checked in the program and they will be constrained if needed.
It is also present an indicator showing the acquired raw temperature values, called "*NTC1,2*".

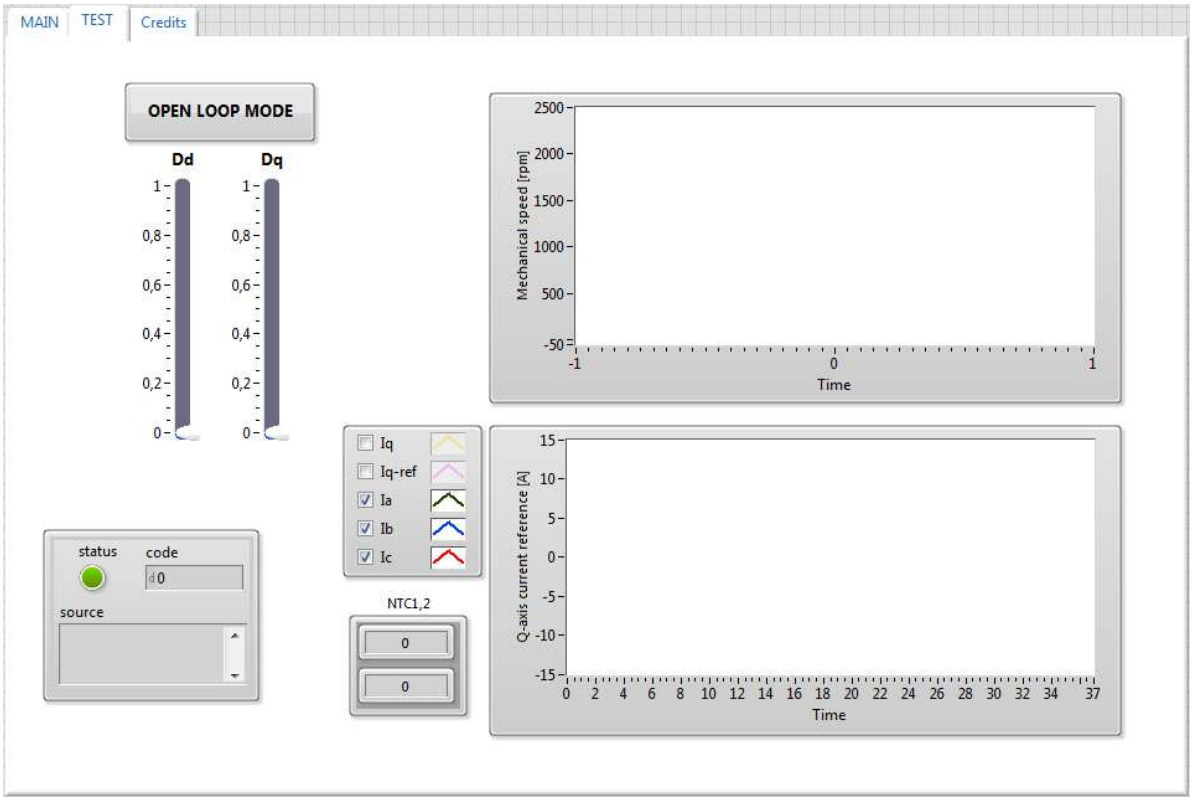


Figure 7. RT Target Front Panel – Test Tab.

FPGA Target

The program executed by the FPGA Target performs the most demanding tasks of the whole algorithm. In these applications, the code is usually realized using Fixed-Point math because of its high performances but, on the other side, it results hard to implement, to modify and to adapt to different systems. For this reason, the algorithm has been developed using both the Floating Point and the Fixed Point number representations, investigating how the flexibility improvement offered by Floating Point numbers translates into an increase of computational time and FPGA required space. For this purpose, the Floating Point Toolkit provided by National Instruments has been used.

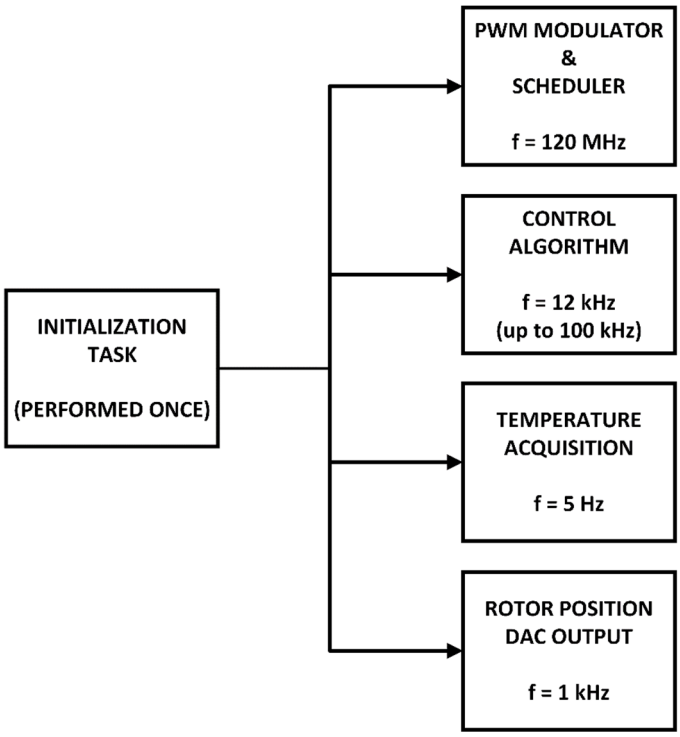


Figure 8. FPGA program layout's scheme indicating the loop's frequencies.

Initialization task

The VI starts performing once the initialization task, shown in Figure 9Figure, which is necessary for the proper functioning of the peripherals.
The developed algorithm uses the following devices, which are integrated on the PED-Board:

- ADC-1
- ADC-3
- DAC
- Resolver interface

The drivers required for the proper use of these peripherals are supplied with the board in the form of subVIs. Each driver is made up of two subVIs: one named INIT must be run only once for the initialization and the other has to be used each time it is necessary to read/write from/to the device.
Default values of Switching Frequency (F_{SW}) and Dead Time (DT) are set and they are stored in two local variables. The loop containing the triangular carrier generation and the modulator is executed at 120 MHz, and the counter is realized using clock ticks: ((1) and ((2) show the formula used for calculating the F_{SW} and DT in ticks.

$$F_{SW}[ticks] = \frac{Core\_Clock[kHz]}{2 \cdot F_{SW}[kHz]} \quad (1)$$

$$DT[ticks] = \frac{DT[\mu s] \cdot Core\_Clock[MHz]}{2} \quad (2)$$

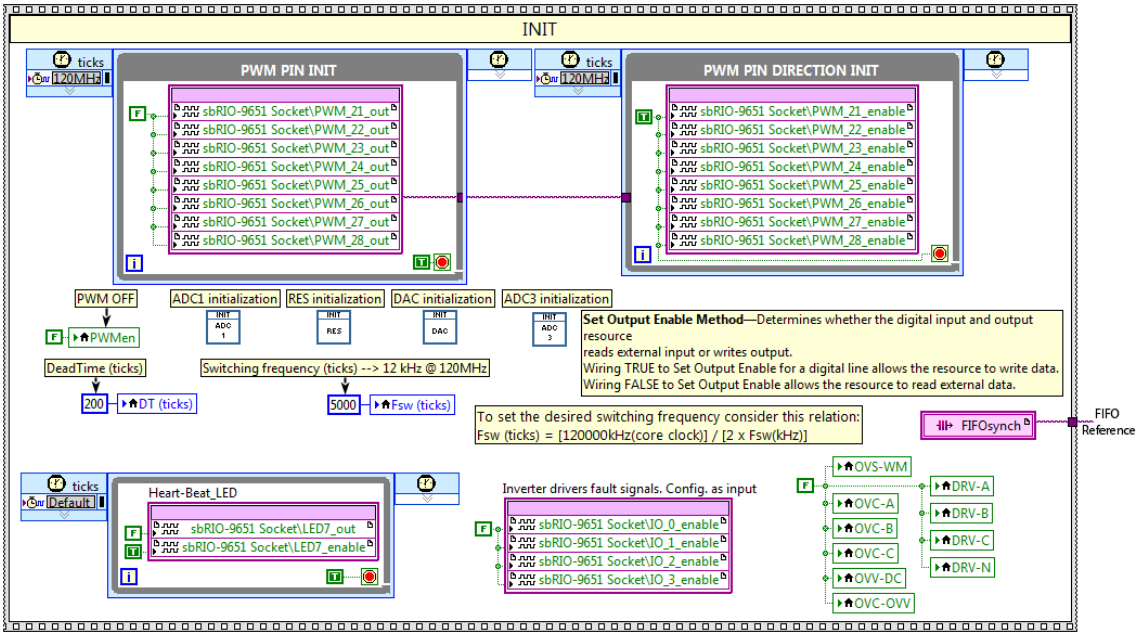


Figure 9. Initialization frame on the FPGA Target.

The resulting values are shown in Table 1.

Table 1. Default switching frequency and dead time values

PARAMETER	VALUE	TICKS @120 MHz
Switching frequency	12 kHz	5000
Dead Time	3.3 µs	200

Furthermore, for safety reasons, all PWM output pins are set LOW before being enabled. It is important to note that PMW output pins must be initialized and used under the same clock domain, which, in this specific case corresponds to 120 MHz. The pins used in this application are stated in Table 2.

Table 2. Board PINs connection

BOARD PIN	DIRECTION	FUNCTIONALITY
LED1	OUTPUT	Board LED – FPGA Heart-Beat
PWM_21	OUTPUT	Switch Top - Phase A
PWM_22	OUTPUT	Switch Bottom - Phase A
PWM_23	OUTPUT	Switch Top - Phase B
PWM_24	OUTPUT	Switch Bottom - Phase B
PWM_25	OUTPUT	Switch Top - Phase C
PWM_26	OUTPUT	Switch Bottom - Phase C
PWM_27	OUTPUT	Not connected – Possible Neutral
PWM_28	OUTPUT	Not connected – Possible Neutral
IO_0	INPUT	Inverter driver fault signal - Phase A
IO_1	INPUT	Inverter driver fault signal - Phase B
IO_2	INPUT	Inverter driver fault signal - Phase C
IO_3	INPUT	Inverter driver fault signal - Neutral

In order to synchronize the execution of the two main parallel loops a FIFO queue is created during the initialization task. This solution allows for running the loops at different speeds that the user can easily adjust by changing a constant in the program. The FIFO Reference wire is also used to force the INIT frame to be executed first.

PWM modulator and scheduler

FPGA Target main loop (PWM modulator and scheduler), shown in Figure 10, runs at 120 MHz (i.e. every 8.3 ns) and it is responsible for sending the control pulses to the inverter drivers and to schedule the control loops.

A triangular carrier wave is generated using a counter realized with unsigned integer 16-bit numbers and then compared to the modulating signals according to PWM technique. The carrier signal is an isosceles triangle waveform with a maximum value of 5000 (set by user according to the desired switching frequency), so the resulting period is 10000 ticks @ 120 MHz.

Output pulses can be stopped by setting the *SysEn* Boolean variable to FALSE. This happens when an error occurs or it can be done by the user from the GUI dedicated button.

Furthermore, for each carrier period, a synchronization signal enables a Case Structure responsible of the Control Loop undersampling, if needed. Indeed, by changing the constant highlighted in the picture it is possible to choose the desired Control Loop frequency with a maximum value equal to the carrier frequency. After that, an element is written in the queue using the *FIFO Write* block and it will be used as the starting signal for the Control Loop.

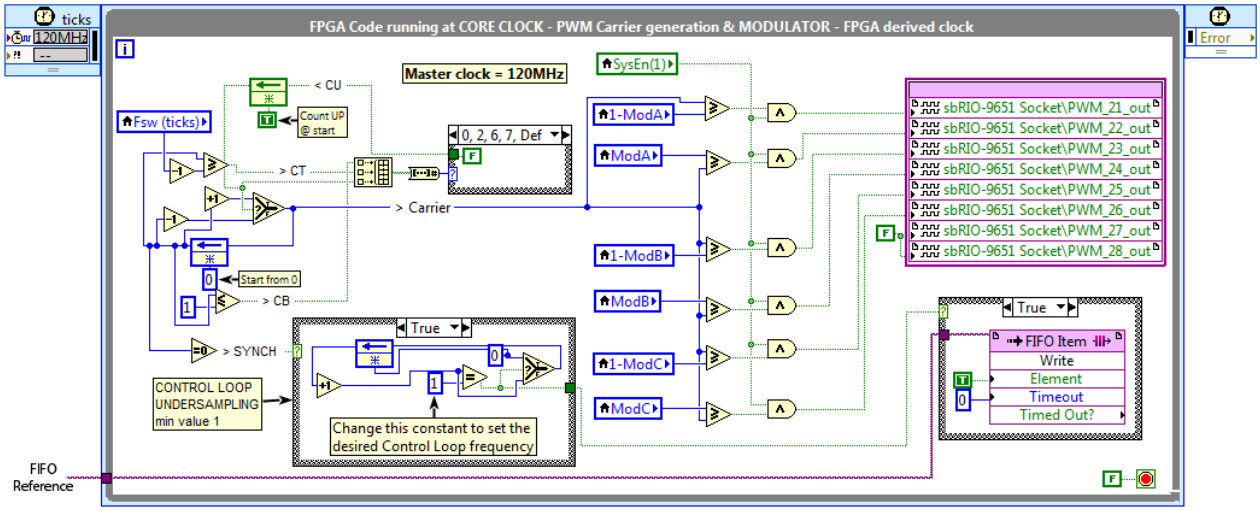


Figure 10. PWM modulator on the FPGA Target.

Control loops

This loop contains a Stacked Sequence Structure that is composed of six different frames, which run sequentially.

Frame 0 – Sync signal receiver (Synchronization between the PWM modulator and the control loop)

In this frame, a block named "*FIFO Read*" waits indefinitely for the presence of an element in the FIFO queue to be processed. After the arrival of an element, a *Tick Count* block is executed and it will be used to evaluate the execution time of the entire control loop.

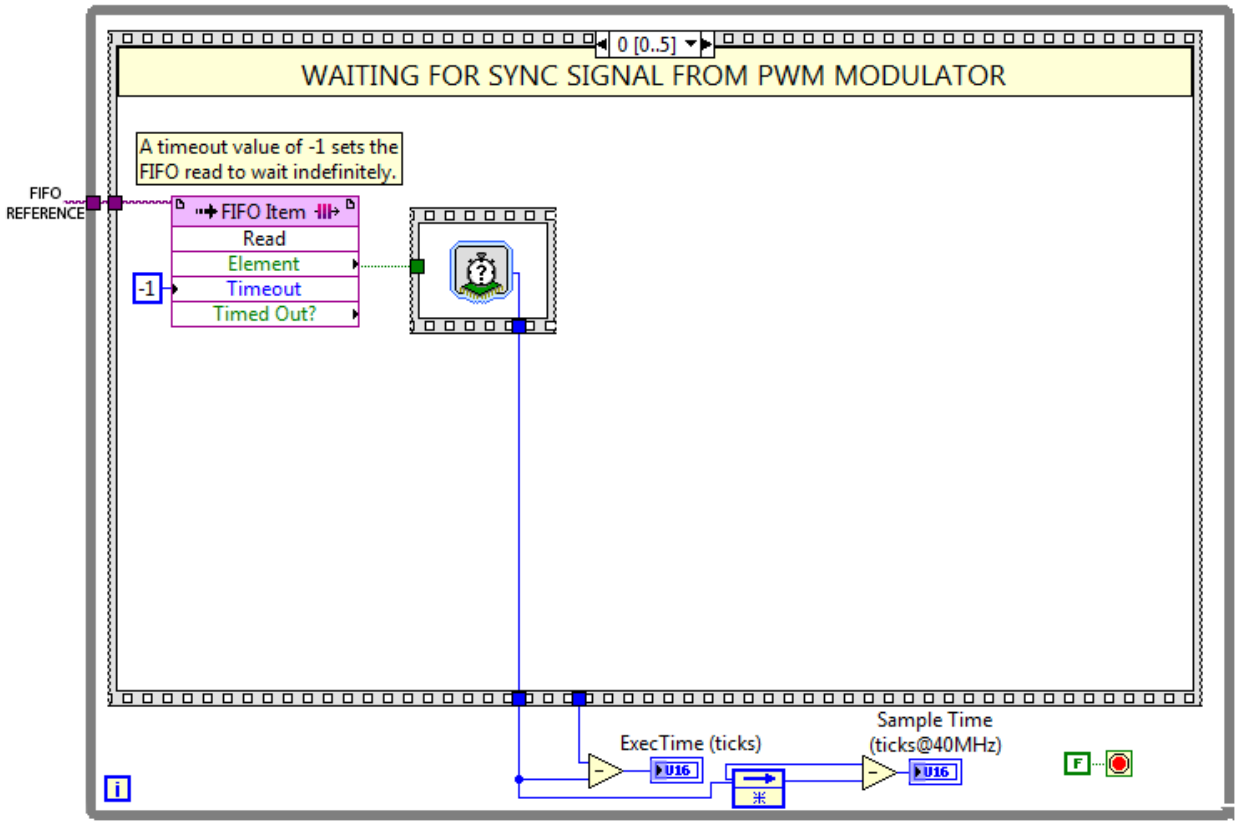


Figure 11. FPGA Target Frame 0 – Sync signal receiver.

Frame 1 - Front Panel interaction

This frame contains the controls placed on the Front Panel. Pressing the "*Reset Errors*" button sets each error to the FALSE state.
FPGA's Front Panel is shown in Figure 13. It will not be used as the GUI, which is in fact handled by the Real Time Target, but it is only needed to contain the indicators and controls of the whole algorithm.

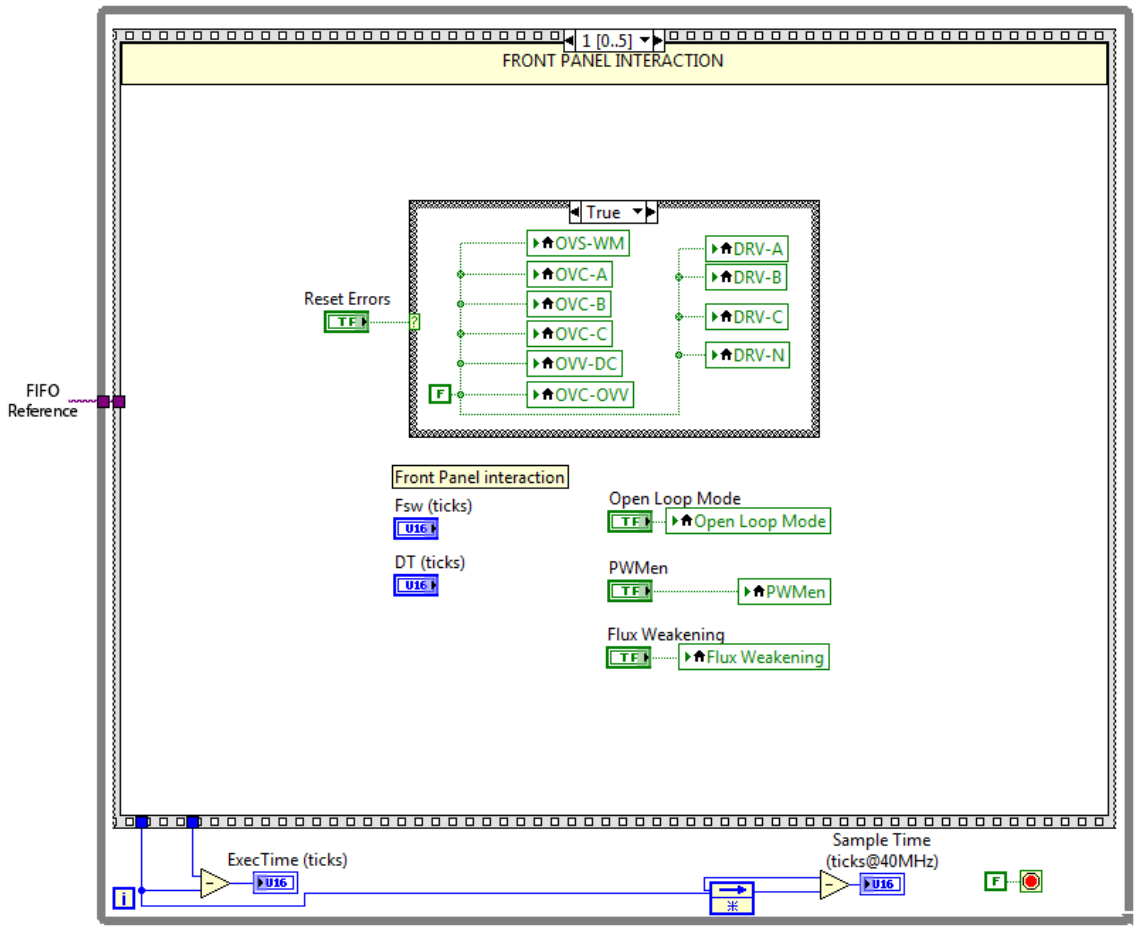


Figure 12. FPGA Target Frame 1 – Front Panel Interaction.

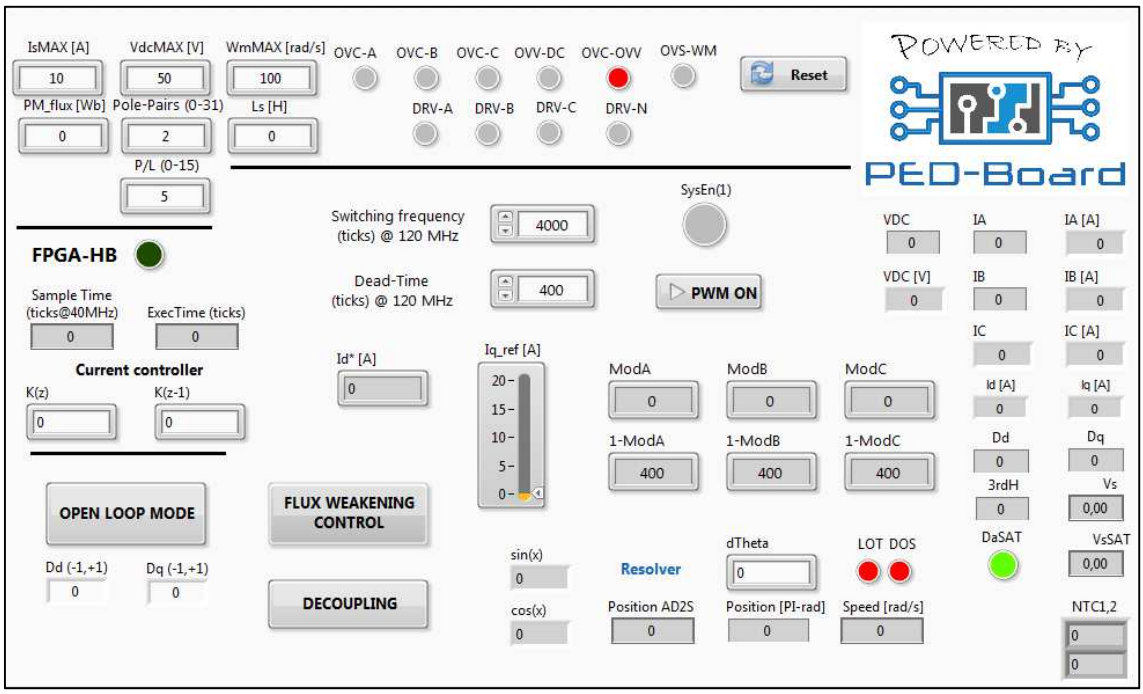


Figure 13. Front Panel on the FPGA Target.

Frame 2 - Measures

This frame executes the acquisition tasks for currents, DC-link voltage, rotor speed and position measures.
Raw data about current and voltage are converted in the proper units of measure by multiplying them by the corresponding LSBs. The resulting measurements are then compared with the maximum allowed values in order to generate an error if an over-voltage or over-current state is detected. If an error occurs, a variable named "OVC-OVV" is set to TRUE and the output PWM pulses are automatically inhibited.

Table 3. Current and voltage LSBs and machine parameters

PARAMETER	VALUE
Current LSB [A/bit]	0,013161
Voltage LSB [V/bit]	0,380108
I_s^{MAX} [A]	90
V_{dc}^{MAX} [V]	60

At the same time, rotor position and speed are obtained from the resolver using the PED-Board dedicated driver named "*RES_read_FPGA.vi*". The position is then corrected considering the Machine/Resolver Pole-Pairs/Lobes ratio and constrained between 0 and 4095, having the resolver interface a 12-bit resolution. It is then summed to an angle adjustable by the user in order to align the resolver and to have the electrical position θ_{el} starting from the peak of the phase A-to-Neutral machine back-EMF (V_{AN}). For the correct resolver alignment, measuring a positive mechanical speed, the electrical back-EMF sequence should be A-B-C. This is required if the provided Clarke and Park transformation are used.
Furthermore, the Resolver driver provides two error signals coming from the resolver hardware interface: LOT and DOS. Their meaning is explained in Table 4.
The measured currents and the aligned rotor position are then used to switch from the *a-b-c* to the *d-q* reference frame. Using Clarke and Park transformations, I_D and I_Q current components are obtained.

Table 4. Resolver fault states

Condition	DOS pin	LOT pin	Order of priority
Loss Of Signal (LOS)	0	0	1
Degradation Of Signal (DOS)	0	1	2
Loss Of Tracking (LOT)	1	0	3
No fault	1	1	

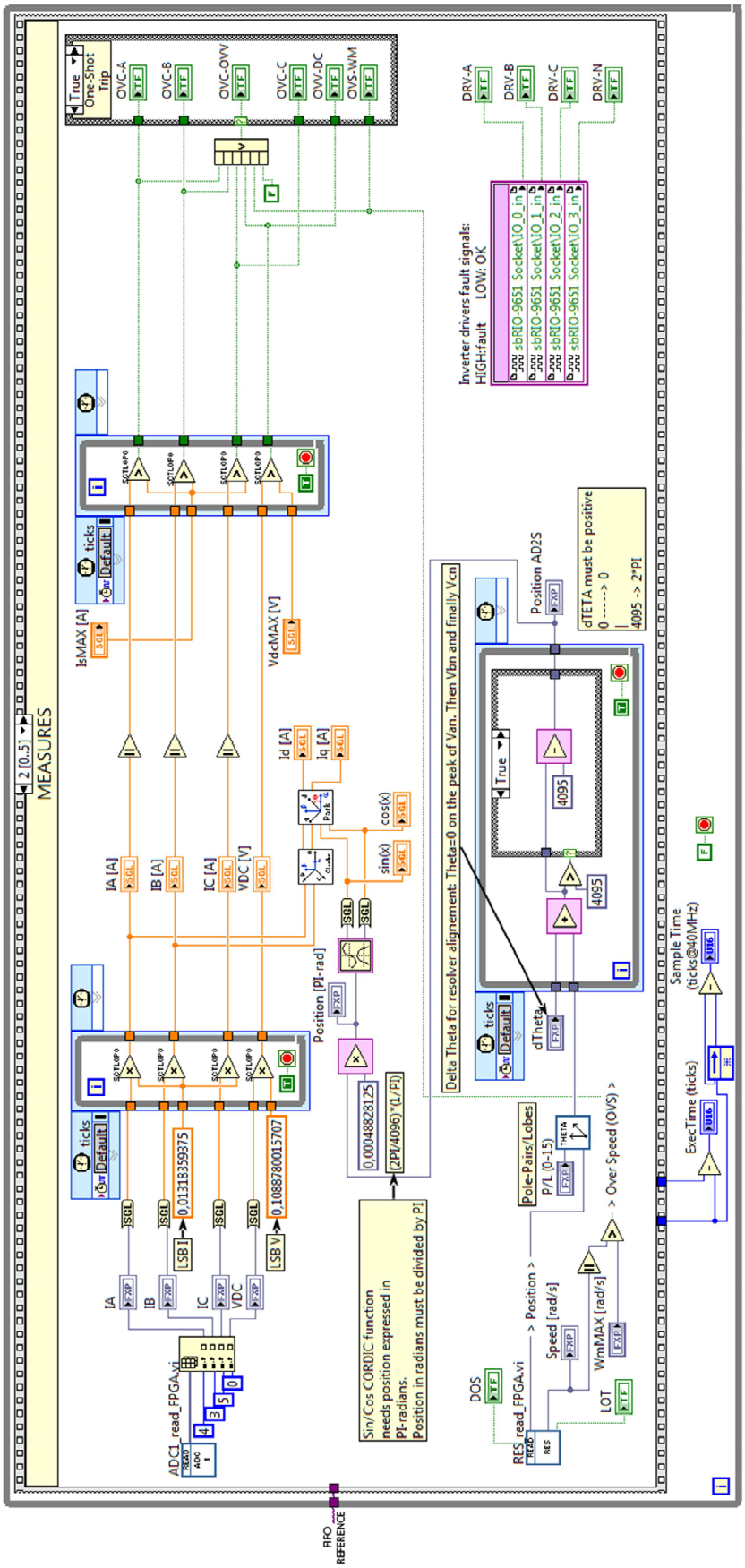

MEASURES
OVC-A
OVC-B
OVC-OVV
OVC-C
OVV-DC
OVS-VM
DRV-A
DRV-B
DRV-C
DRV-N
Inverter drivers fault signals:
HIGH:fault LOW: OK
IsMAX [A]
VdcMAX [V]
Id [A]
Iq [A]
IA [A]
IB [A]
IC [A]
VDC [V]
cos(x)
sin(x)
Position [PI-rad]
Position AD2S
dTETA must be positive
0 -----> 0
4095 -> 2*PI
Delta Theta for resolver alignement: Theta=0 on the peak of Van. Then Vbn and finally Vcn
dTheta
Sample Time (ticks@40MHz)
ExecTime (ticks)
0,00048828125
(2PI/4096)*(1/PI)
Sin/Cos CORDIC function needs position expressed in PI-radians. Position in radians must be divided by PI
0,01318359375
0,1088780015707
LSB I
LSB V
ADCI_read_FPGA.vi
RES_read_FPGA.vi
Pole-Pairs/Lobes
P/L (0-15)
> Position >
Speed [rad/s]
> Over Speed (OVS) >
WmMAX [rad/s]
DOS
LOT
RFO REFERENCE

Figure 14. FPGA Target Frame 2 – Measures. I_B is considered available from dedicated transducer. Alternatively, it can be derived from I_A and I_C.

Moreover, inverter drivers fault signals are acquired and showed on the GUI using four indicators named "*DRV-A*", "*DRV-B*", "*DRV-C*" and "*DRV-N*".

Frame 3 - High-speed current control loops

In this frame, D_d, D_q and the resulting voltage vector (V_s) needed by the FOC technique are evaluated. D_d and D_q correspond to the normalized values of the voltage components in the *dq* reference frame (v_d and v_q). They are not expressed in volt and they are in the range between -1,14 and +1,14 because of the third harmonic injection technique[1]. The normalized voltage vector V_s is obtained with the following expression (3).

$$V_s = \sqrt{D_d{}^2 + D_q{}^2} \tag{3}$$

The reference value of i_q can be set from the Real-Time target front panel, while the required i_d is provided by the Flux Weakening Control algorithm placed at the end of this frame. In order to chase the desired values, two PI controllers for i_d and i_q regulation are implemented in a SCTL that is executed with a single clock tick at 10 MHz instead of 40 MHz due of the SGL math. The controllers are discretized using the ZOH technique and the constants K(z) and K(z-1) are chosen according to [39] and [40].
PI output values are constrained in the range from -1,14 to +1,14 because of the third harmonic injection, and D_d^* and D_q^* are evaluated using the following expressions.

$$e_d = I_d^{Ref} - I_d \qquad e_q = I_q^{Ref} - I_q$$

$$D_d^*(z) = e_d(z)K(z) + e_d(z-1)K(z-1) + D_d^*(z-1) \quad D_d^*(z) = e_d(z)K(z) + e_d(z-1)K(z-1) + D_d^*(z-1)$$

After that, D_d^* and D_Q^* are added to the decoupling terms, expressed as follows.

$$V_{d\_Dec} = -\omega_{el} L_s I_q \qquad V_{q\_Dec} = \omega_{el} L_s I_d + \omega_{el} \lambda_{pm}$$

According to the whole algorithm, though, these terms must not be expressed as a pure voltage, but they have to be normalized. For this purpose, the following expressions are used.

$$D_{d\_Dec} = \frac{V_{d\_Dec}}{0{,}57 \cdot V_{dc}} \qquad D_{q\_Dec} = \frac{V_{q\_Dec}}{0{,}57 \cdot V_{dc}}$$

$$D_d = D_d^* + D_{d\_Dec} \qquad D_q = D_q^* + D_{q\_Dec}$$

The addition of the decoupling terms can be enabled or disabled through the GUI by pressing the button "*Decoupling*".

[1] The value of 1,14 has been used instead of 1,1547 to ensure a safety margin.

Using the new D_d and D_q values, the resulting voltage vector V_s is evaluated. If V_s is greater than V_s^{MAX}, it must be saturated to the maximum admissible value but it is important to preserve its phase. The following expressions are used to obtain the saturated values.

$$D_{d\_Sat} = \frac{V_s^{Max}}{V_s} D_d \qquad D_{q\_Sat} = \frac{V_s^{Max}}{V_s} D_q \qquad V_{s\_Sat} = \sqrt{D_{d\_Sat}^{\ 2} + D_{q\_Sat}^{\ 2}}$$

Furthermore, Open Loop Mode can be enabled by the provided front panel button. In this condition, D_d and D_q are set manually from the GUI in the range from -1 to +1.

The final part of the frame is related to the Flux Weakening (FW) operation. For this purpose, a PI controller is realized similarly to the others previously described. The regulator tries to minimize the difference between V_{s_sat} and V_s^{max} providing the I_D reference value as the output. This part can be enabled or disabled by a button on the GUI and the regulator is also reset if Open Loop Mode is chosen or if the "*SysEn*" variable is false. If the flux weakening control is disabled, i_d reference value is set to zero, otherwise, a negative value for i_d is evaluated by the FW PI controller.

The gains of the previously mentioned PI controllers can be set in the Real Time Target program considering the s-domain representation.

Frame 4 - DQ to ABC and modulating signals manipulation

This frame is responsible for generating the modulating signals needed to realize the PWM technique. For this purpose, inverse Clarke and inverse Park transformations are performed using the previously evaluated D_d, D_q, $sin(\vartheta_{el})$ and $cos(\vartheta_{el})$. The resulting output consists of three voltages in the ABC reference frame, normalized in the range between -1,15 and +1,15. These quantities are then multiplied by half of the switching frequency expressed in ticks. The resulting values, D_A, D_B and D_C, are processed by a subVI (Figure 15) in order to perform the triangular third harmonic injection, according to the following equations.

$$V_{3rd} = -\frac{\max(D_A, D_B, D_C) + \min(D_A, D_B, D_C)}{2}$$

$$D_{AO} = D_A + V_{3rd} \qquad D_{BO} = D_B + V_{3rd} \qquad D_{CO} = D_C + V_{3rd}$$

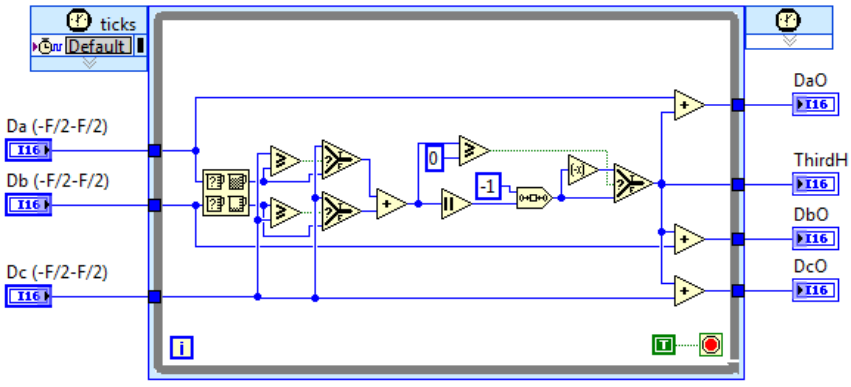


Figure 15. ThirdHinject.vi used to perform the triangular third harmonic injection.

After that, using the "*DutyShift.vi*", shown in Figure, the three values are added to half of the switching frequency (expressed in ticks) in order to shift them in the range between 0 and F_{SW} (ticks), as required by the PWM modulator. The last subVI, named "*DutySat.vi*" (Figure 17), performs the dead time insertion and generates six different modulating signals (two for each phase), which will be sent to the PWM modulator. The three quantities are constrained in the range between DT and (f_{sw} - DT) and then they are added to DT. Furthermore, a Boolean indicator can be added to warn the user in case there is the saturation of the duty cycles.

At the same time, in order to provide a useful and intuitive way to check if the FPGA program is running, a loop for the "*Heart-Beat*" generation is executed, resulting in a blinking LED on the PED-Board as well as on the GUI. This "*Heart-Beat*" runs at a frequency lower than the control loop, thanks to an undersampling loop placed in a SCTL.

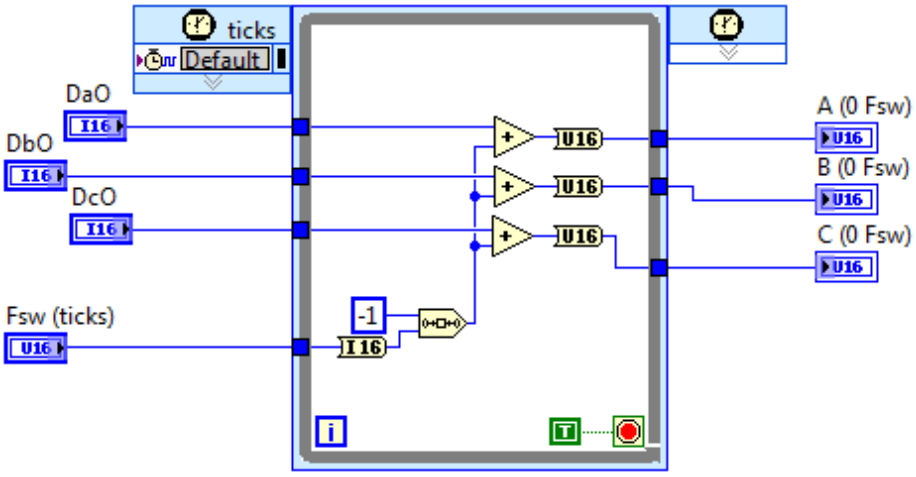


Figure 16. DutyShift.vi used to change the range of PWM duty cycles.

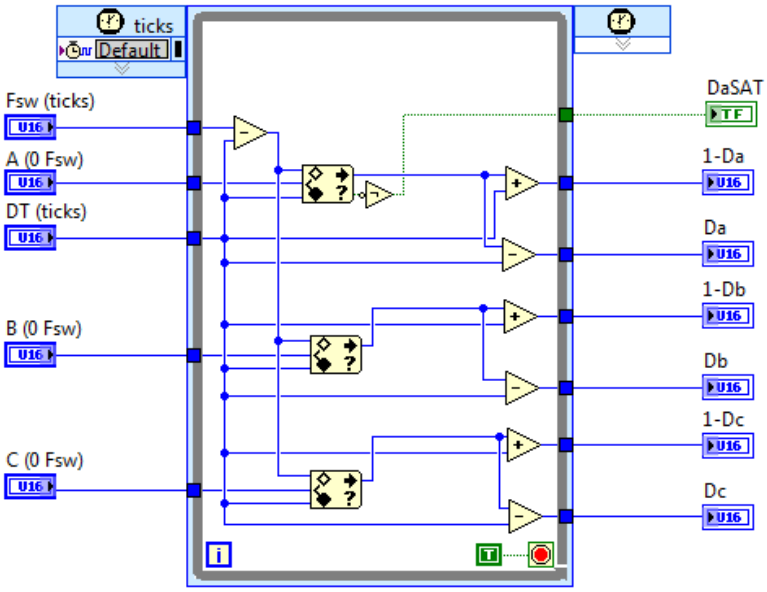


Figure 17. DutySat.vi used to insert the dead time to PWM duty cycles.

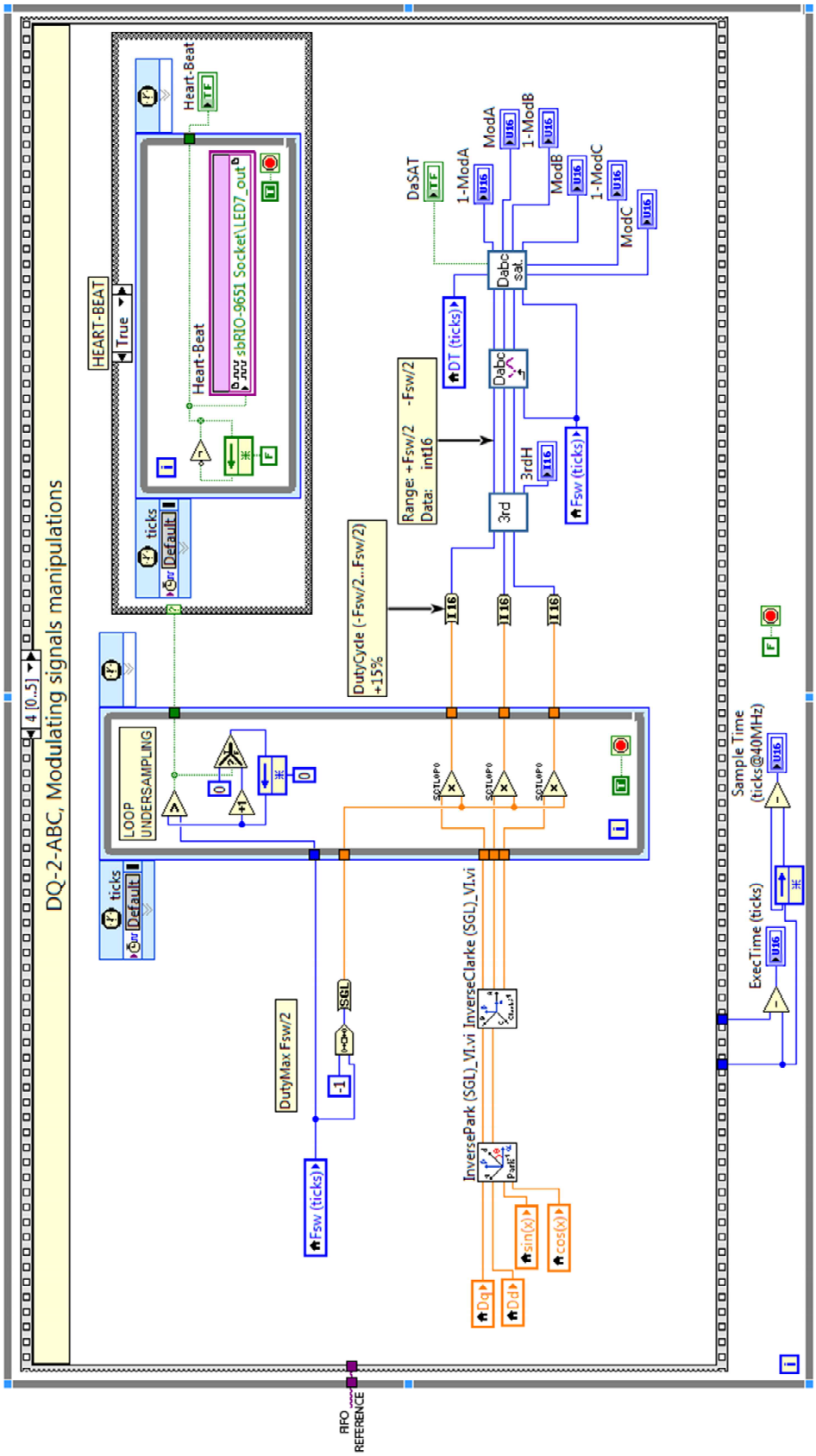


Figure 18. FPGA Target Frame 4 - DQ to ABC and modulating signals manipulation on the FPGA Target.

Frame 5 – Execution time measurement

This frame is the last one of the sequence. A "*Tick Count*" block is executed and it returns the value of a free running counter in µs. In order to evaluate the execution time of the entire control loop, the value obtained in "Frame 0" is subtracted from the one just got and the result is displayed on the Front Panel under the name of "*ExecTime (ticks)*". This is the minimum time needed by the code to be executed. The actual execution time of the control loop is set modifying the constant in the PWM modulator and it is saved in "*Sample Time (ticks@40MHz)*" variable.

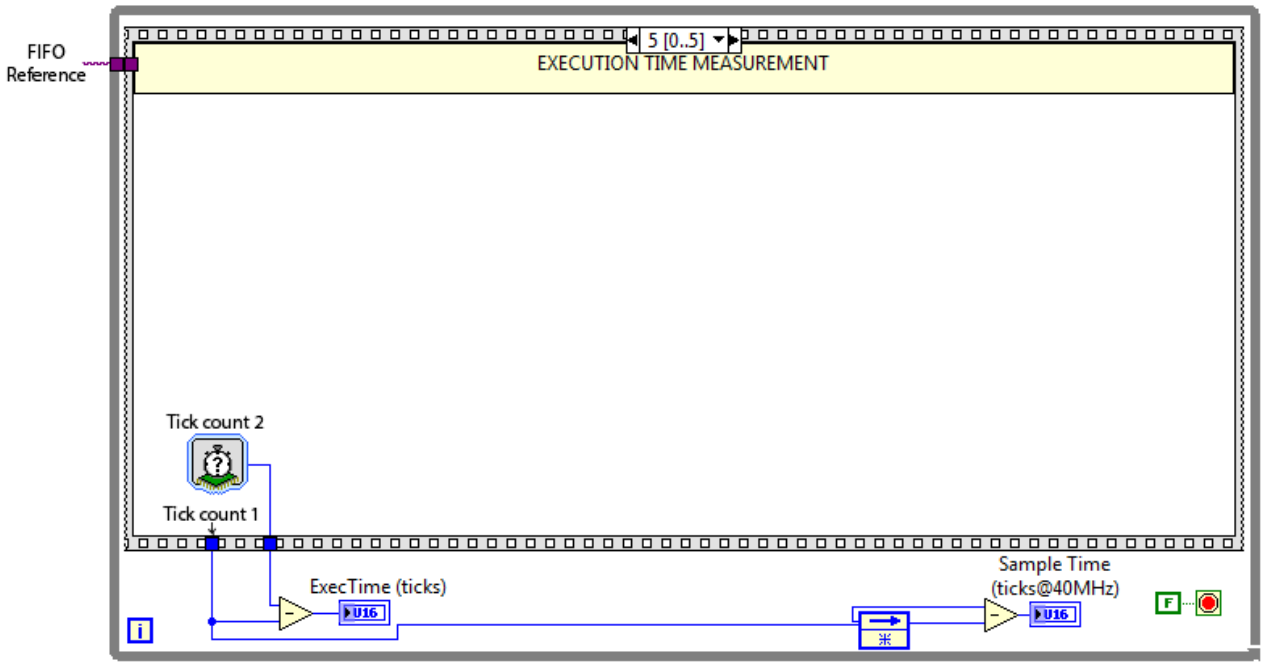


Figure 19. Frame 5 – Execution time measurement.

Temperature measures (Constant temperature control)

NTC_READ loop acquires temperature data from two NTC sensors that are connected to pins AIN17 and AIN18 of ADC3. These sensors are placed on the inverter switches of phases A and C. The loop runs every 100 ms in order to read the same sensor every 200 ms and it calls the "*ADC3_read_FPGA.vi*" subVI.

These measures are important to monitor the working conditions of the inverter and they can be used to turn off the system if a certain temperature has been reached, or to enable a sort of constant temperature mode of operation (not implemented). In fact, especially in demanding applications it is convenient to perform a constant-temperature control instead of a constant-torque one, regulating the torque to safeguard the system integrity. This is, in fact, a better solution because it allows for keeping the system operating, even with a reduced torque. These tasks can be easily done modifying the algorithm and using the measured temperature.

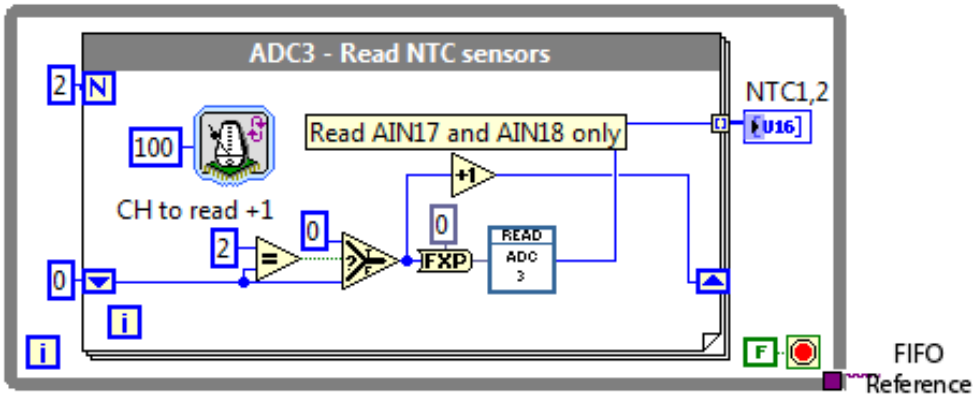


Figure 20. Temperature measures on the FPGA Target.

DAC output

DAC_OUTPUT loop writes the position acquired from the resolver on the first channel of the PED-Board DAC. This loop runs every 1 ms and it calls the "*DAC_write_FPGA.vi*" subVI. This task is performed so that the position can be easily displayed using an oscilloscope.

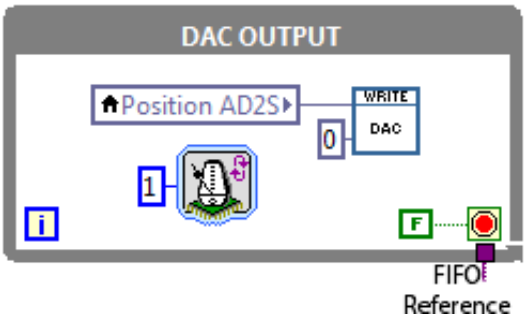


Figure 21. Position DAC Output on the FPGA Target.

Comparative results between FXP and SGL math implementations

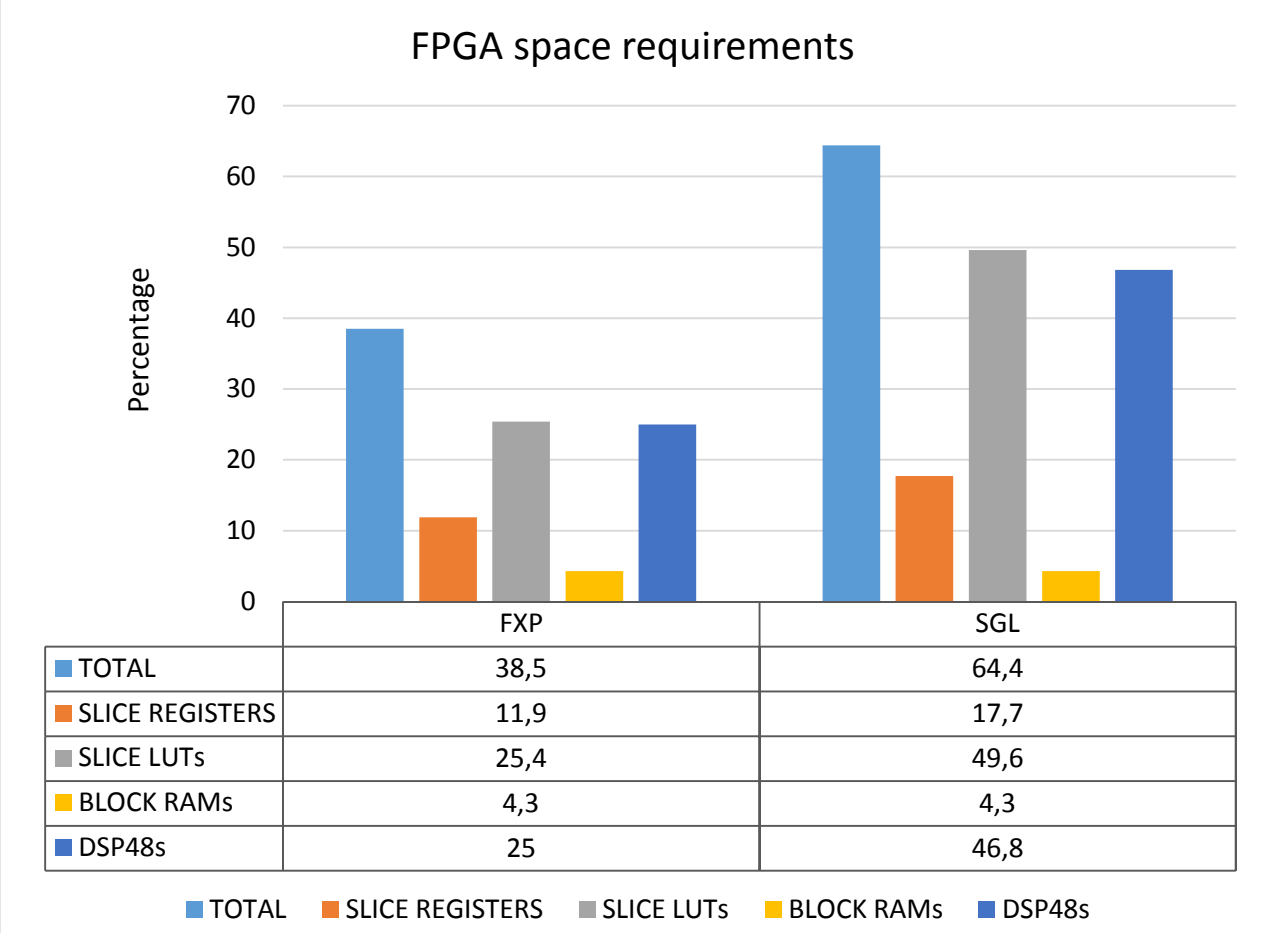
As previously mentioned, the FPGA algorithm has been implemented using both FXP and SGL number representations, in order to investigate the differences in terms of FPGA execution time and space requirements. For this purpose, the program has been realized at first using FXP arithmetic, and then gradually converted into SGL.

The parameters used to evaluate the FPGA space requirements are provided by the compiling tool. As can be seen in Figure 22, they represent the different resources physically present on the platform.

The starting revision, labelled as *"FXP"*, contains the complete FOC algorithm, including decoupling terms and Flux-Weakening control, realized using fixed-point math. The total space on the chosen FPGA (Xilinx Zynq-7020 included in sbRIO-9651) is shown in Figure 22. The execution time is around to 8 µs. This is the minimum time required to execute the whole code, allowing a theoretical sampling frequency of approximately 125 kHz.

The SGL program exhibits an execution time which is slightly increased with respect to the FXP implementation: 10,4 μs instead of 8 μs. It is important to note, though, that the flexibility improvement introduced by SGL leads to preferring SGL to FXP. Moreover, using SGL, although there is an increase of FPGA space occupancy (even if the FPGA is far from being considered full). Hence, there is space for many future implementations.

Another important consideration that is possible to make analyzing the results, is that realizing a "hybrid" algorithm with both the presence of FXP and SGL numbers is inefficient. This is due to the conversions between the two number representations, which results the most demanding tasks of the algorithm, especially FXP-to-SGL.



	FXP	SGL
TOTAL	38,5	64,4
SLICE REGISTERS	11,9	17,7
SLICE LUTs	25,4	49,6
BLOCK RAMs	4,3	4,3
DSP48s	25	46,8

Figure 22. FPGA resources utilization, FXP vs. SGL implementation. I_B current available from a dedicated sensor.