Model-based design of a novel modulation technique.

Using FPGA based motor control board for traction inverter.

By

Andrea Piccioni

******

Supervisor(s):

Prof. P. Guglielmi, Supervisor

Prof. E. Armando, Co-Supervisor

Prof. R. Ruffo. , PhD Student support

Politecnico di Torino

2017

I would like to dedicate this thesis to my loving parents

Acknowledgements

I would first like to thank my thesis advisor Paolo Guglielmi of the Polytechnicuniversityof Turin. The door to Prof. Guglielmi office was always open whenever I ran into a trouble spot or had a question about my research or writing. He consistently allowed this paper to be my own work, but steered me in the right the direction whenever he thought I needed it. I would also like to acknowledge Eric Armando of the PolytechnicUniversityof Turin as my co-advisor to support me in FPGA field. I would also like to thank the expert who were involved in the validation survey for this research project: Ing. Riccardo Ruffo. Without their passionate participation and input, the validation survey could not have been successfully conducted. Finally, I must express my very profound gratitude to my parents for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them.

Abstract

This paper analyzes the new scenario of embedded technologies from the point of view of the control of a three-phase electric motor. Given the emergence of electric mobility, either hybrid (HEV) or full electric (BEV), it will become the most profitable but equally demanding market, given that the vehicle system is a complex system full of increasingly stringent regulations. In fact, one of the main themes in the automotive field is that of safety, of the single component and of the overall vehicle system, called functional safety IEC-61508, and ISO-26262. So the base of a control design engineer's pyramid becomes security, that is design embedded software for high-integrity systems that meet industry standards with tools that provide documentation, test cases, and procedures that let qualify each subsystem. Therefore, the fundamental requirement is reliability, but above all to be able to guarantee it at any workplace that is eligible in the specifications. To ensure that there is a need for an accurate stage of designing software, but above all an equally hard working debugging job, during all stages of process development. It seems more than desirable, therefore, given the needs, that the control of a car should come from a model, composed of several subsystem models, specifically validated. All this will be implemented through dedicated tools, thus avoiding unpleasant inconveniences and slowdowns in process product and vehicle. By entering into the motor control specific, one of the most critical tasks the currents-control, given their dynamics a real time task must be assured. In addition to this task, the inverter will have to control other slower variables, detect any faults, communicate with the vehicle control unit, etc. In short, a hard work for the embedded engineer who will have to force hack to an ASIC, as the general purpose microcontrollers would be in trouble, especially for interrupt handling for task execution. Designing and validating an ASIC is a costly and long process that can be avoided with the advent of the FPGA control board, real-time reprogrammable, high performance (hundreds

of MHz) hardware circuits, to be entrusted with real-time tasks, leaving the static tasks to common processors.

Contents

List of Figures

Chapter 1

Background

1.1 Introduction

Choosing which silicon based technology to use for their design is getting harder for embedded expert, due to the technology increase in electronic devices used for control. Moreover, when new hybrid silicon is becoming available that blurs the lines between these device types. The designer, based on their experience, knowledge in base on performance (general-purpose, high dynamics, connectivity etc) and time to market (prototyping, final sample, etc) in the control application, which devices are best suited. A matter of fact as with most decisions, an analysis of the particular final scope must be done to discover which device or combination of devices may be satisfactory. The silicon candidates include a general-purpose microprocessor, DSP, and FPGA, or the combination of such. If the application must access a wide range of I/O, reprogram logic or run closed-loop control at rates exceeding 1 MHz, an FPGA is probably the right fit. A micro-controller is the more appropriate choice if designers want general-purpose functionality and programming simplicity. If ultimate optimization and processing speed are needed and the engineer is familiar with underlying chip architecture and DSP programming, a DSP is the best device to use.

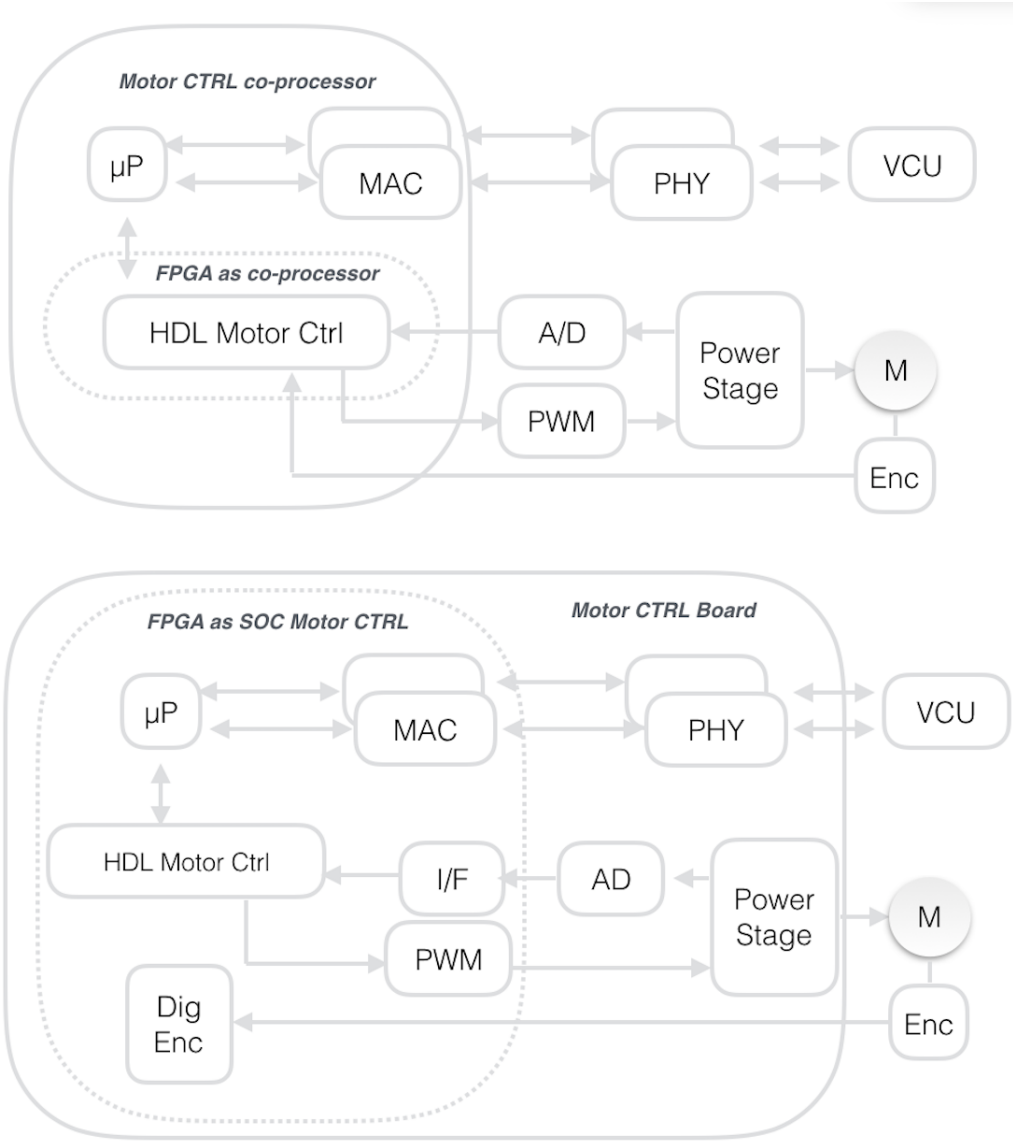

Fig. 1.1 Motor control board: FPGA Co-processor, FPGA SoC.

Furthermore embedded developers traditionally have had limited options for accelerating performance near the end of a design cycle, as quick and dirty solutions including buying a faster processor or last minute hand made tuning of assembly subroutines. While both options can be effective, the tradeoffs they bring are too large to ignore, comparing with the advantage of new hybrid solutions with FPGA. For example, an overloaded processor scenario can be avoided using FPGAs as a coprocessor (the difference between System on Chip (SoC) FPGA and Coprocessor FPGA is shown in figure 1), providing simple glue logic. This technique allows

off-loads tasks, such as communications, motor control, I/O modules, the primary microcontroller or digital signal processor device in applications. Much final application exceeds bandwidth, and if they do not, they will definitely do so soon especially with the increase of regulations, required to run drive motor tasks and I/O. As a result, you may be required to add an extra device, often an expensive solution if your actual layout board could not accommodate this extra device.

1.2 Direct comparison

Strictlyconsider FPGA and DSP, it is obvious that DSP wins the battle for motor control and in a general embedded system design perspective. The reason is FPGAs are blank chips while DSP chips having microprocessor and peripherals meaning that instantly you can begin to develop and debug your software on a DSP; while you cannot on an FPGA. The FPGAs way needs a design of HW layer first, then proceed to SW development and finally debug. Hence FPGAs have one level of start up complexity higher than DSP and while this can become on one side an advantage increased flexibility for new features, it is on the other side a drawback because the same solution is going to take more time to develop. Not even talking about the fact that most motor control department, is currently embedded expert with DSP/MCU hence have not necessarily the skills for HW development, thus could not appreciate the infinite degree of freedom given by FPGAs solutions. FPGA and IP, that is completely different, with an FPGA and IP approach, the HW development phase is reduced to its minimum which is to integrate IP components together (processor IP, motor control IP, communications IP, HMI IP, etc.). While this process can be a nightmare if not done correctly, it takes only a few minutes if done with the correct tools (e.g. using SOPC Builder in the case of Altera FPGAs or Xilinx's Platform Studio in case of Spartan). Because of their low-cost programmable hardware, integration of embedded processors, and ability to incorporate motor-control IP, FPGA-based systems offer new features for many applications that cannot be found in any MCU- or DSP- based system. Controlling motors with high-performance motor controller significantly improves the overall system efficiency and dynamics, also reduce volume (in E-motor this is strictly linked with costs and weight of active materials) and increase durability in the mechanical point of view. To reach high-

performance control the nowadays trend is a typically multi-chip solution (Table 1):

- For faster math focused operations, today a Digital Signal Processing is the best choice
- A micro-processor to run the slower as Temperature control loops and do all the other interfacing and I/O.
- ASIC whenever there is a need for personalization, communication, a fast task or some interface other than the usual standards.

Table 1	**Silicon technology**		
Skill	***DSP***	***FPGA***	***FPGA and IP***
Out of box	Great	Low	Great
Processor	Fixed	No	Configurable
HW phi	Fixed	No	Configurable
Cost maintenance	High	Very high	Low
Integration	Tedius	Easy	Easy
Expert required	Yes	Yes	No

Choosing microprocessors instead of DSPs, for control applications, traditionally offer better on-chip peripherals, such as standard PWM and other integrated I/O for multiple, asynchronous tasks. MCU is less specialized and is more flexible, which usually makes them easier to understand and program, instead of low level, also called Hardware languages Verilog and VHDL programming. Finally, if the engineer is really lazy, a single GUI-friendly silicon access tool is easy to find by the type of FPGA supplier. The rough difference between an FPGA and an MCU is that the first is fuzzier. Basically, what an FPGA is at the hardware level? A lot of small SRAM cells, all connected to a dense matrix of multiplexers. Basically, an FPGA is a whole pile of discrete logic that can beelectronicallyre-wired simply by reprogramming the multiplexers and SRAM cells; easy.

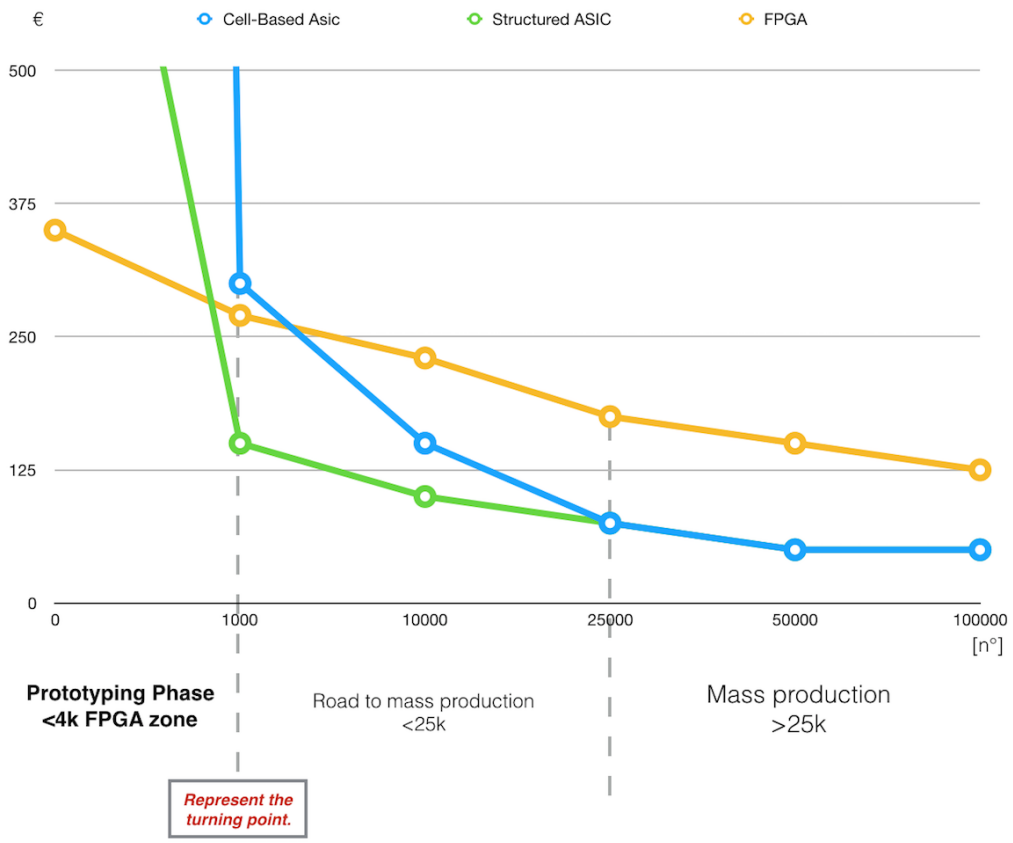

Fig. 1.2 Key role of FPGA into a product process.

A Field Programmable Gate Array can be seen as the prototyping stage of ASICs, which are very expensive to manufacture (sort of manufacturing stencil) and once it's made there is no going back. Given their fixed cost of development and the uniqueness of the hardware process, they are not very convenient, at less than large quantities of the life cycle products (see figure 2). But let's not forget that if you miss out something on validation, you have to bring the products back and replace the components physically. Apart from the fact that if in the course of product life, regulations become more stringent, you have to upgrade hardware or shorter time on the market product have to be accepted. Also, FPGAs if you power it off, you lose not only the current state but also your configuration. Now exist boards that add a FLASH and/or a MCU to load the configuration at startup so this tends to be a less important argument. Both ASICs and FPGAs can be configured with Hardware Description Languages, and sometimes FPGAs are used for the end product. But generally, ASICs kick in when the design is fixed. FPGAs as said execute multiple

operations in parallel, unlike MCU and DSP which are sequential machines and so execute one instruction at a time (MHz FPGA vs KHz microprocessor). The parallel nature of FPGA operation allows for higher speed computations, therefore, improving control performance. As a consequence it has the power to implement functions like Fast Fourier Transform (FFT), that allows the frequency domain analysis, so that see unwanted harmonics as for example the elimination of 5th and 7th in a three phase load. Unwanted frequencies can cause vibrations and interfere with precise speed control or cause resonance and vibrations that reduce the longevity of a motor assembly. Because no OS run on the FPGA, the code is implemented in a way that ensures maximum performance and reliability, FPGAs can perform closed-loop control at extremely fast loop rates. However, FPGAs by themselves are not as useful as when they complement microprocessor or DSPs. Interfacing an FPGA to the outside world through ADC and DAC controllers and a micro-controller offers complete system designs with fast I/O, appropriate control and rapid signal processing. Due to integration with lower bandwidth systems often speed is limited by the sensors, actuators and I/O modules rather than by the FPGA's processing performance.

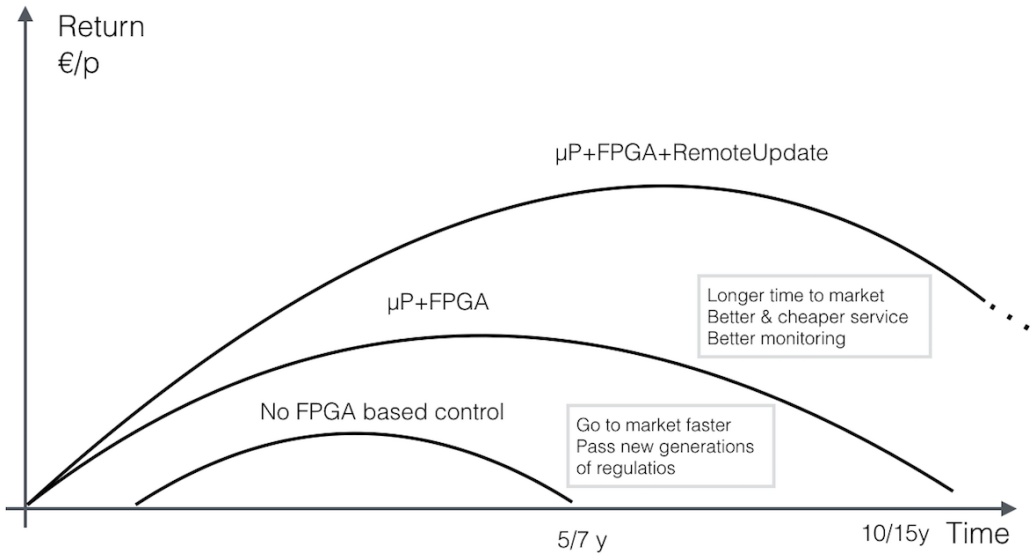

Fig. 1.3 Time in market, technology comparison.

Summarizing why to move on FPGA: reducing cost and time for prototyping to market, stay in market longer (reducing service cost), as shown in figure 3. FPGAs solutions sound like the godsend for Project Managers world: but on the other hand,

no enough Know How shared and consolidated. Summarizing the PRO point of FPGA based control in few point:

- Custom PWM technique.
- Multiple motors control.
- Custom structured power stage.
- Easy component integration.
- Parallel architectures.
- High bandwidth control loops.
- High reliability.

1.3 Fully customizable PWM

The crucial part of the current control strategy is the actuation of the referments voltages, using one of the many PWM techniques cited in the literature. A PWM technique controls the legs of an inverter, changing states in order to meet the time-average value of the voltage command. The true advantage of FPGAs is that it can make customizable what previously was fixed generic hardware in MCUs or DSP blocks. While optimizing DC bus voltage usage, PWM custom techniques can reduce losses in the motor and in the power converter. Furthermore, FPGA allows to cut useless (no dynamic scope) switch losses using variable frequency carrier, an explored solution given the power of computation: this technology converts the DC in instead of fixed-carrier frequency/variable-amplitude and frequency voltage in a variable-carrier frequency/variable-amplitude and frequency voltage. An application-specific PWM IP core in an FPGA, completely optimized in terms of energy efficiency and based on motor parameters, can replace the standard PWM block, forced in an MCU solution.

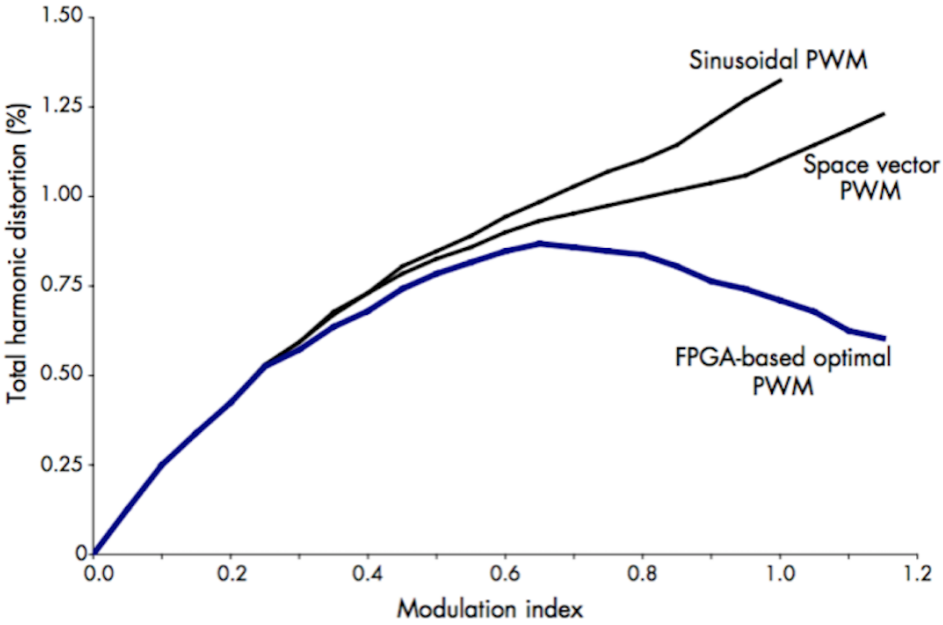

Fig. 1.4 Benefit of custom PWM technique.

Figure 4 shows a custom PWM in an FPGA can reduce the THD in over modulation by nearly 50 percent, compared to the standard PWM in MCUs or DSP blocks. In efficiency point of view, this reduces time-harmonic losses in the motor, from a mechanical point of view reduces audible noise and increases global motor reliability. The FPGA's hardware programmability enables easy implementation of dedicated high-performance logic circuits. Dedicating logic circuits for motor current and torque control, as opposed to software running on generic MCU or DSP blocks, allows a very high-frequency (more than 100 kHz) control-loop bandwidth. Such high-control bandwidth has a positive impact on current quality, previously cited, due to better regulation. From safety and durability point of view FPGAs allows the motor controller to extract critical information about the health of the motor during its operation, which then can be sent to the main appliance controller to notify the user about the risk of motor failure and to adjust motor-control parameters to reduce risk of failure and increase safety of operation.

1.4 State of art in Automotive control PEs.

Nowadays, FPGAs are available with one or more embedded processors, this class of FPGA is known as a SoC (System-on-Chip) FPGA. Using the embedded processor's cores in the SoC, to implement slower tasks as temperature or voltage bus monitoring, while using the FPGA fabric to accelerate components where a high update frequency will benefit the control loop, enables optimum usage of SoC resources and provides a best-of-both-worlds benefit to you, ductility and performance. A great example of a task of a motor control algorithm that can be implemented faster in FPGA is Field Oriented Control (FOC). You could use the FPGA portion of a SoC FPGA to accelerate your flux detection algorithm while using the embedded processor for lazy variables control loops.

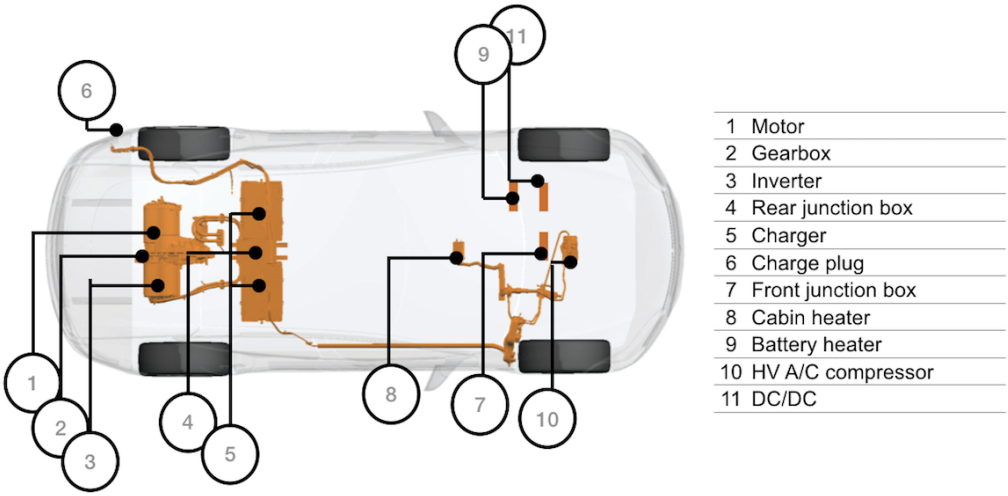

Fig. 1.5 Tesla Model S HV System

Tesla model S:

This part of the thesis will be a tribute to Tesla, because of its leadership in EV automotive field, just to share what for many is magic, but that with various tear down is becoming less and less magical, rather sometimes inexplicable as for the IM motor, despite having a lossless cooled copper rotor. Since focusing on controlling not on choosing that electric motor, an asynchronous means that you know exactly the d-q flux component, mandatory. As previously suggested, the use of SoC FPGA allows to improve the flow estimation, since the defense has been used for years

(mainly for image processing), rightly Elon Musk has thought well to follow the suggestions of U.S. Army and Co.

In figure 5 is explained the power train layout, an 85kWh battery pack that feeds the 270kW IM, controlled by a fully legs IGBT 2-L for the automotive purpose, summarizing all electrical architecture. Focusing only on electronic architecture, also called EE topology in the automotive field, connected by CAN protocol, its represent the nervous system of the vehicle. Mainly brain of the car is the Vehicle Control Unit, that sends, acquire and process all high-level variables, info, and data. Obviously, the battery and inverter have a crucial role in the drivetrain, so in control, them Tesla make a masterpiece of control.

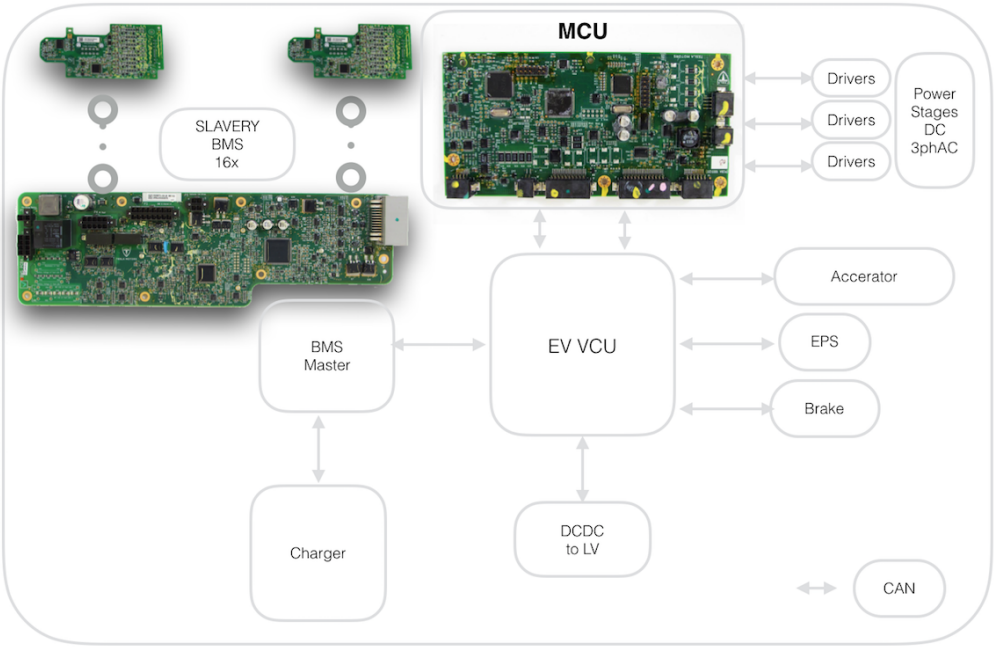

Fig. 1.6 Hierarchy of system control in Model S

As showed in figure 6 a FPGA based control board is reserved for the master BMS and MCU drive. This makes us understand where control is crucial:

- CTRL INV: FOC and Flux measurements and estimations.
- BMS Master: acquisition and processing of parameters for best estimation State of Charge.

By focusing on BMS (figure 7), the solution adopted by tesla is interesting:

- BMS Slaves: 6xCell Voltage Mis. and 6xCell Balancing, a 6 Series Cell Lithium-Ion Battery Monitor by Texas (76PL536A) and a Silicon labs general-purpose micro-C. Each Module BMS send bt SPI local data to Master.
- BMS Master: literally a SoC FPGA (micro-C TMS320F2809 + ProASIC 3 FPGA VQG100). The acquisition and processing of variables from the slave, in order to estimate the state of charge, is a very laborious task. Requiring much computing power as such will be delegated to the FPGA pipeline. State of health, pressure, failure, over-currents, over/under-voltages have to be monitored, too.

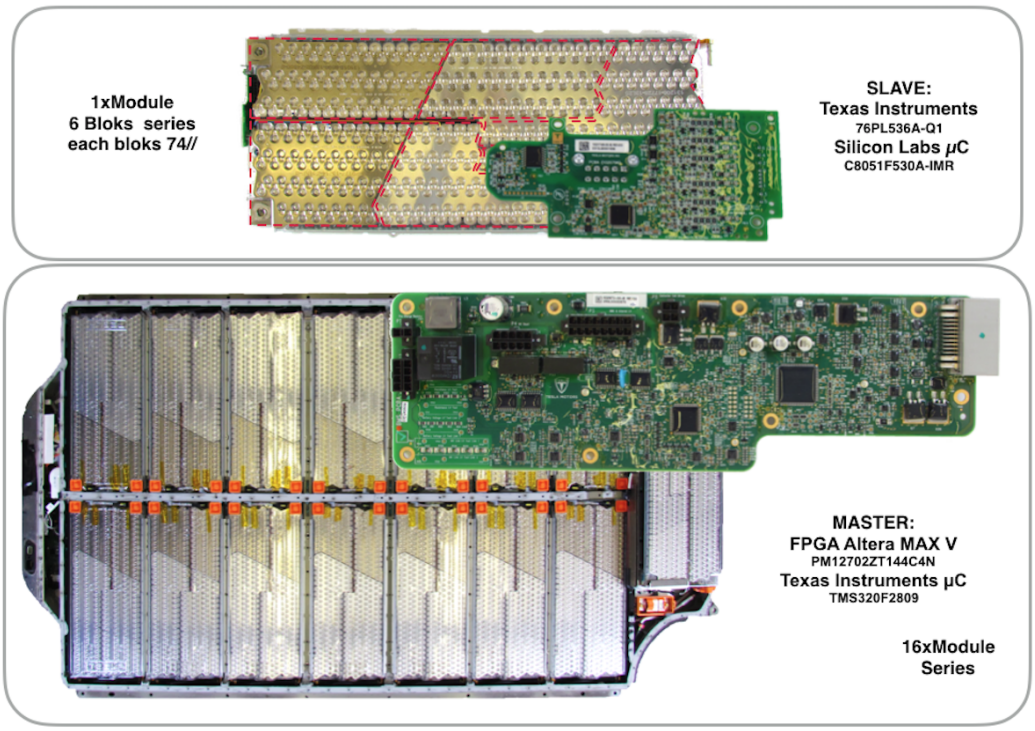

Fig. 1.7 Slaves modules and master pack.

Although Tesla's engineers have taken all the battery precautions, testing it the main understandable problem is hotspots mainly where connectors of HV for battery bus vehicle distribution. Analyzing the Power Stage control solution, it immediately leaps into the eye for its unusual cylindrical shape due to integration on the rear axle. This choice is a consequence of the fact that more

the inverter is close to the motor, less cable have to use (reducing not ideality cable), exploiting the advantage of the ripple reductions, so the stem also works from the EMI point of view. The structure is a classic two level 3 legs, but each leg has 32 IGBTs to be controlled (16xH and 16xL); this parallelism of currents should be carefully checked, controlling IGBT in parallel is definitely easier to control MOSFETs, but should not be underestimated. Everyone is wondering, why does this need to be parallelized? The answer is in Tesla requirements, 0 to 100kmh in 5sec. Boosting torque for few second, to reach 100kmh, a boosting current is needed: 700A more or less (1xIGBT=80A so a to expand SOA 16xIGBT=1300A is enough). The IGBT Infineon IKW75N60T, is a classical 600V 80A IGBT for automotive purpose, its main disadvantage is that during conduction a Von=0.6V is present in comparison with a FET solutions, so in high-torque high-current zone IGBTs solution have some advantage; this obviously leads to being disadvantaged in the low-torques low-current operating areas where the dissipation in ON mode in FET is smaller since there is no threshold component Von.

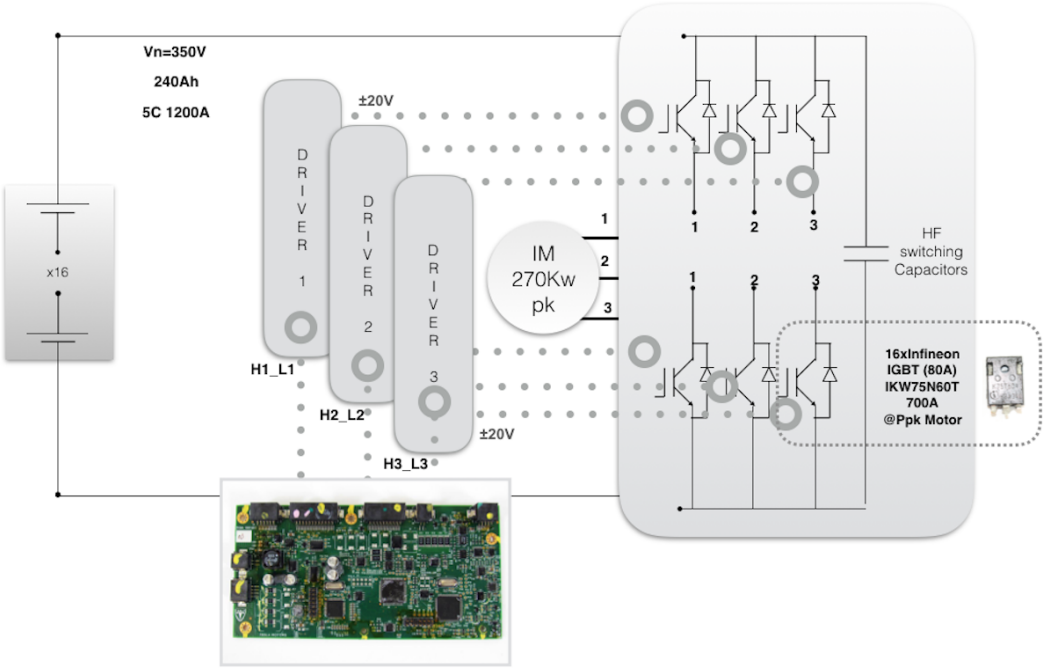

Fig. 1.8 Inverter summary structure.

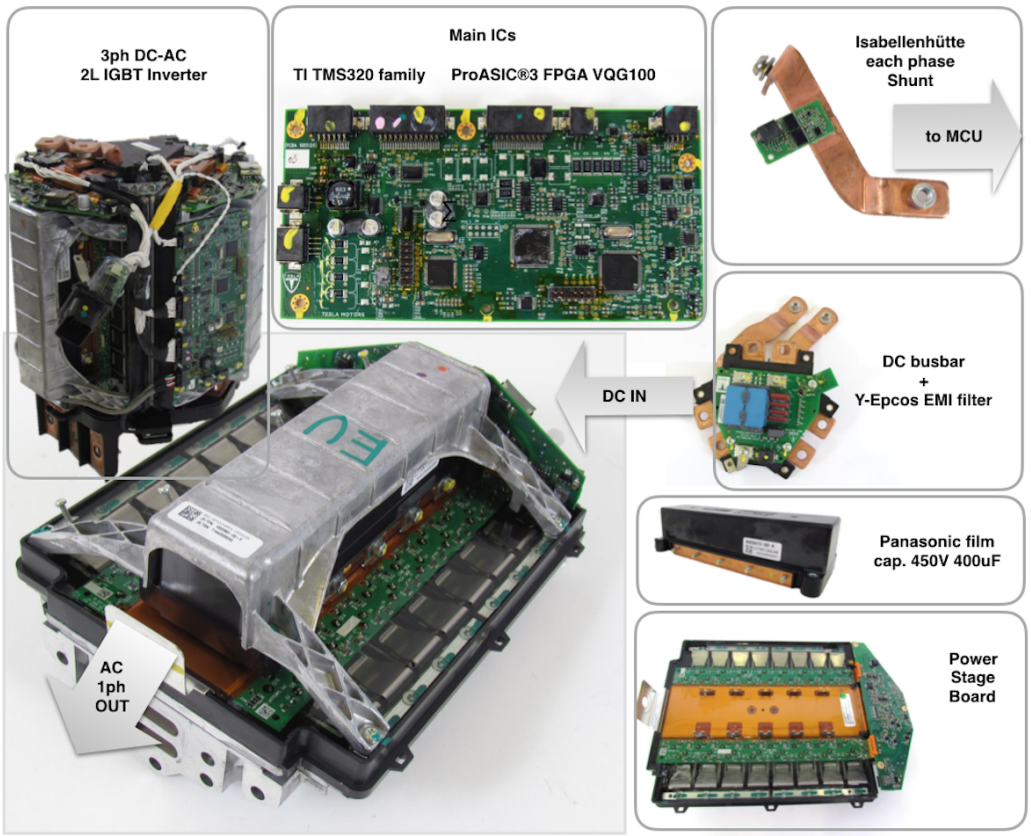

Fig. 1.9 Hardwere and PEs on Tesla drive for IM.

The overall inverter structure and control is shown in figure 8 and figure 9, also drives are shown: they are the actuator of duty and also they are (opto)coupler between the power and the control part. As demonstrated by Tesla, a single FPGA can easily integrate part of your design into a single device, and allow to re-program your FPGA-based design at any time only with SW update. This approach maximizes your design's ability to change with evolving standards while minimizing the number of board designs required to support each protocol standard or each additional feature. FPGAs are ideal for parallel signal processing and therefore ideal for any system requiring a performance boost (HW acceleration). Parallel hardware in the FPGA means no performance cost to add more controllers and features; You can accelerate performance with embedded processors and IP blocks on an FPGA used as either solution as coprocessor or as SoC in your design FPGA devices, enable you to deliver products to market faster than with other technologies,

thus maximizing market share and extending the life cycle of your industrial designs: the success of Tesla.

1.5 pFARAD motor control board

pFarad team started in 2008 at polytechnic of Turin, in DENERG, them goals was design and prototype a fully electric kart with a high-performance drivetrain:

- 200kg
- 130km/h
- 3,8s from 0 up to 100km/h
- 2x 20kW IPM-Motors (REAR)

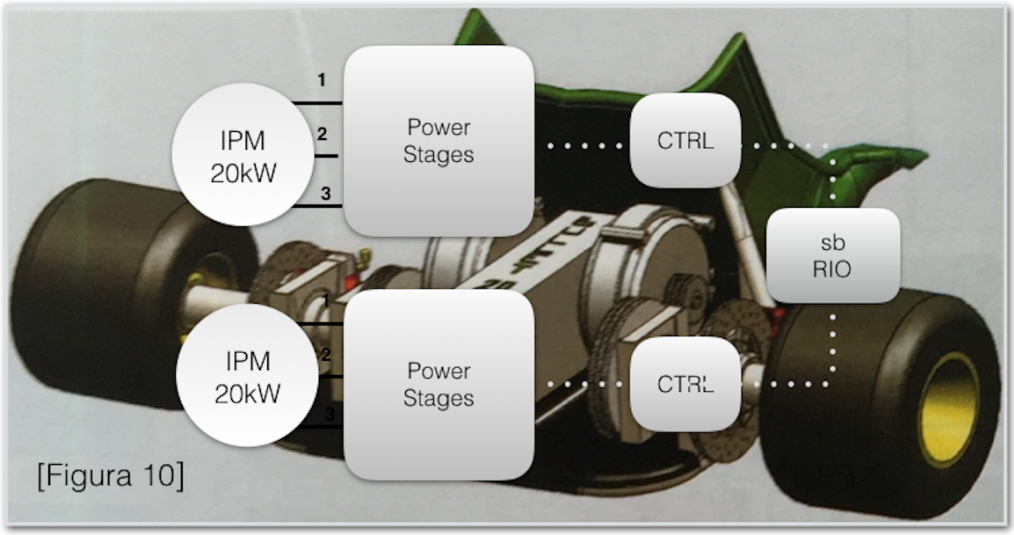

Fig. 1.10 pFarad powertrain summary.

The general drivetrain architecture is easy to understand, and it is show in figure 10. Focusing on control board as always, starting a preliminary analysis based on hardware, to clarify the main subject of this study. Schematics of the FPGA based board are in attached document (appendix 1). Instead a block diagram of the system on chip FPGA motor control board, show in figure 11 (logic and supply point of view):

- 3x LEM current sensors.
- AD7277 and OpAmp for analog signal as (temperature and voltage bus).
- 1x Xilinx XC3S200A VQ100.
- Flash PROM XCF02S
- A Joint Test Action Group
- 2xAMP mini CT 15 pin, one J1 for 2xSPI and one J2 for legs command (driver and power stage board).
- 1xLTC3560 switching 5V to 3.3V.
- 1xTPS62207 switching 3.3V to 1.2V for FPGA core.
- 1xTPS77633 linear 5V to 3.3 for LEMs etc..

There are all chips for a state of art FPGA based motor control board. The analysis will continue taking into account the single control board for elaborations and measurements.

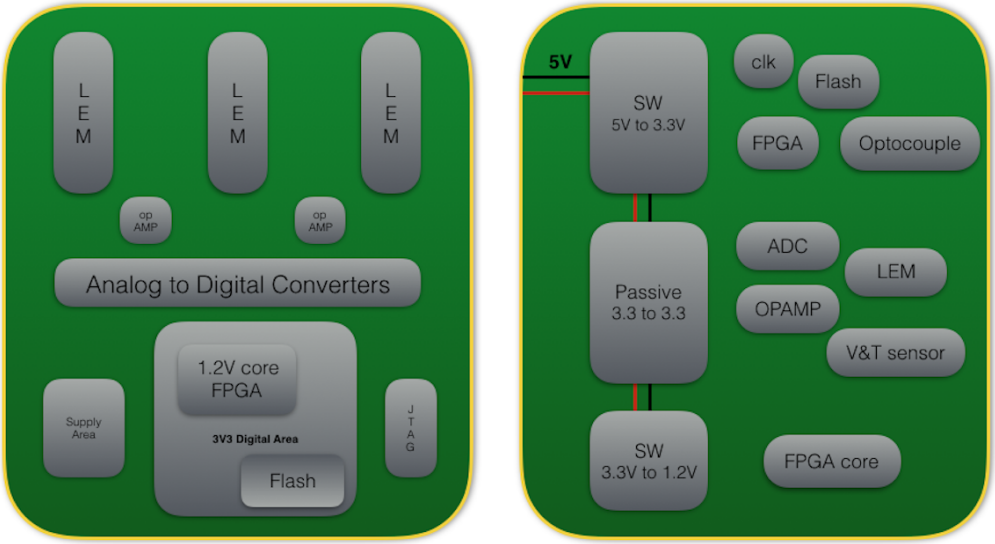

Fig. 1.11 pFarad control board HW summary.

Chapter 2

Product process design and validation

2.1 Process introduction

Since the aim is to provide innovation from the point of the modulation techniques, leveraging the FPGA parallelism capabilities; it will need to bring the concept explained in the appropriate paragraph from the virtual world (simulated) to the real world (HW implementation of control). This process is one of the classic development methods produced in the electronic and electrical industrial world.

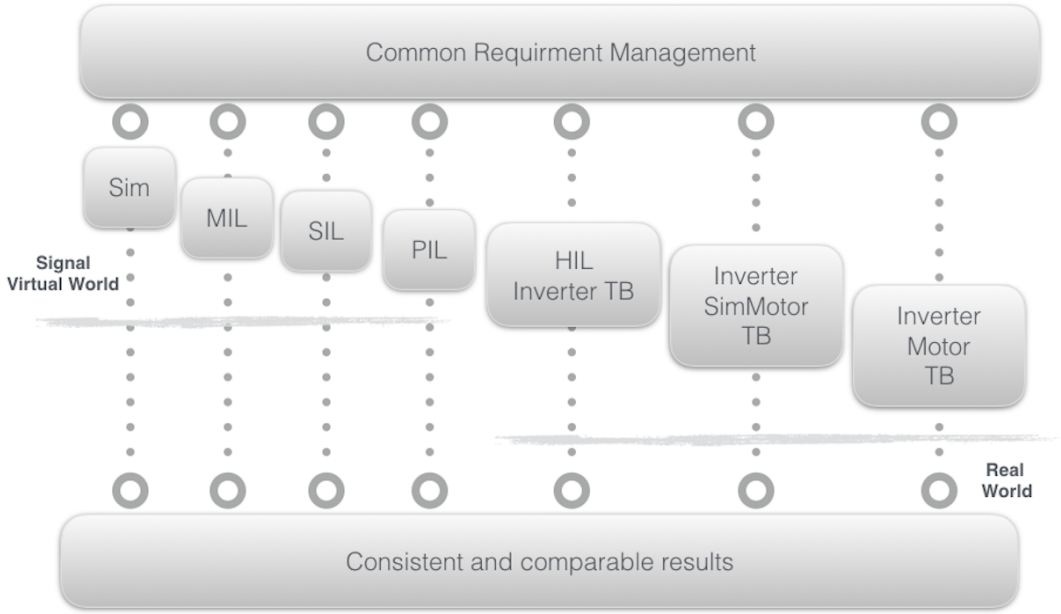

Fig. 2.1 State of art in product process development and validation.

The process from virtual to real, showed in figure 1, is the state of art of design and validation and are divided by this steps:

- Preliminary concept and draft simulation.
- Model in the loop (MIL): if you are designing the controller from scratch then you develop the model of the plant and controller (using Matlab, Simulik) and use model in the loop to verify or test whether it is implementable.
- Software in the loop (SIL): once your model is verified, the next stage is where you develop a software, in this case VHDL code, depending on the processor or FPGA, plan to use for final hardware implementation and run the simulations for the controller model (with the plant still a software model) with this code to verify it. If you experience any glitches you may have to go back to MIL and make necessary changes.
- Processor in the loop (PIL): once your controller is verified for its SIL implementation, you proceed to the next stage: processor in the loop, where you load the developed code onto the processor/FPGA and run the simulations on the modeled plant for verification. If there are glitches then go back to your code, SIL or MIL and rectify them.

- Hardware in the loop (HIL): Once your plant model has been verified using PIL, now you can replace the plant model with the original hardware: say lab model (if its a motor whose speed controller is being designed, then controller is in FPGA/processor which is now interfaced to the DC motor by connecting the inputs and outputs/states at the right points of sensors/transducers) and conduct the HIL test for verification.

Once you complete all the above tests, your controller is said to be ready for hardware implementation.

- Power stage test bench: use the legs command high and low to control the optocoupler and drivers for power stage integration.
- Power stage + E-Motor test bench: connect the 3 phase cable of power stage to a real (or simulated) motor.

Nowadays the transition from one step to the next in the production process is greatly facilitated by the wide choice of programs and tools that can accompany throughout the process, especially during virtual world part of the process. Often, these tools are easily interoperable, an example can be the strong link between the MathWorks world and Mentor graphics, bringing the process flowing, safe, and the right time to finish. The scope of figure 2 is to understand who are the partner, who will accompany the development of this product system, while the detailed purpose of each tool will be described in the next paragraphs dedicated to the appropriate step-process.

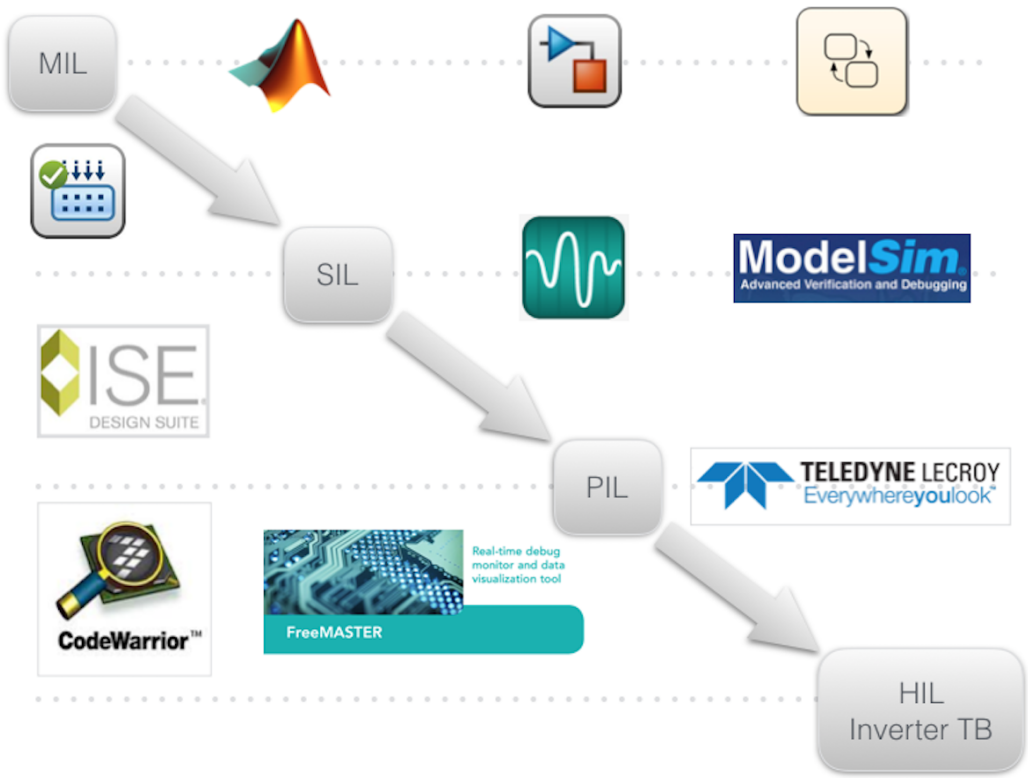

Fig. 2.2 Tools used for product process development and validation.

2.2 Preliminary concept and draft simulation.

First of all we should analyze the system to understand the variables and the imposed constraints. It is shown in figure 3 and consists of:

- A DC source, conveniently splited to produce medium point.
- A 3Ø-inverter which can drive both signs of the current, and its control.
- A balanced 3Ø-load.

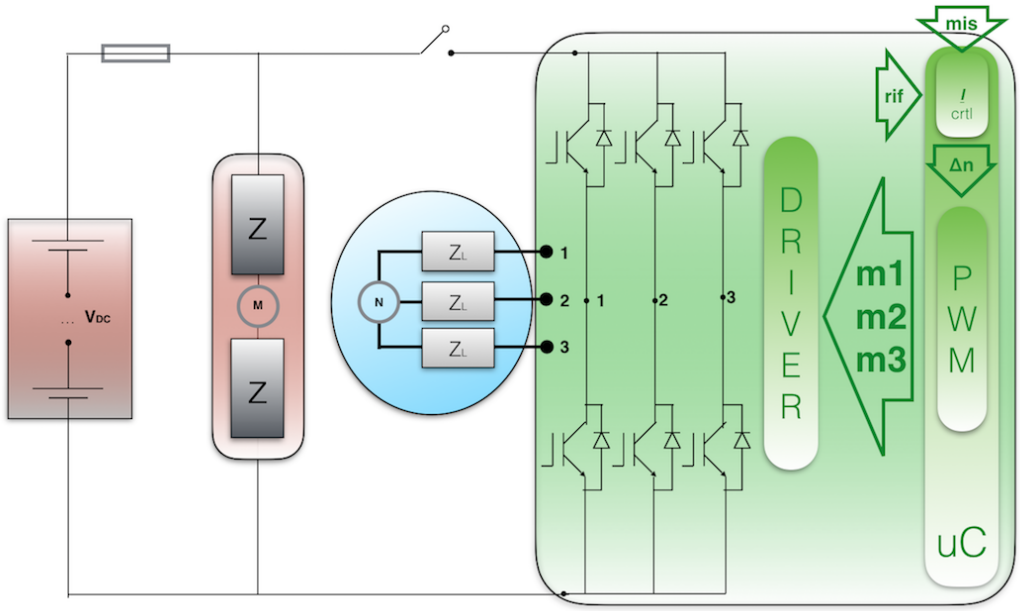

Fig. 2.3 3Ø-load supplied by DC source

Once the variables displayed write some relationships visible:

$$\left.\begin{aligned} \sum_{ph=1}^{3} I_{ph} = 0 \\ Z_{L1} = Z_{L2} = Z_{L3} = Z_L \\ \sum_{ph=1}^{3} V_{ph-n} = 0 \end{aligned}\right\} \qquad \text{3Ø-load constraint.} \tag{2.1}$$

$$V_{ph-m} = V_{ph-n} + V_{n-m} \tag{2.2}$$

using the degree of freedom as going from a linear dependence we lose, then could be:

$$\sum_{f=1}^{3} V_{fM} \neq 0 \tag{2.3}$$

To describe the inverter 3Øsystem certainly the phases to neutral voltage are a need, but are only two out of three necessary to determine its status. Because the sum of three phases current is equal to zero, so the sum of phases-neutral voltages is zero, too. Meanwhile, the phase-medium voltages are limited by

the amplitude of the DC physical system, as shown below:

$$-V_{DC} \leq \hat{V_{ph-ph}} \leq V_{DC} \tag{2.4}$$

So the limit mentioned above will move to the phase-neutral components with the implicit coefficient:

$$V_{ph-ph} = \sqrt{3} V_{ph-n} \tag{2.5}$$

$$-\left(\frac{V_{DC}}{\sqrt{3}}\right) \leq \hat{V_{ph-n}} \leq \left(\frac{V_{DC}}{\sqrt{3}}\right) \tag{2.6}$$

$$n_{ph} = \left(\frac{V_{ph-n}}{\frac{V_{DC}}{2}}\right) \tag{2.7}$$

$$\left(\frac{-2}{\sqrt{3}}\right) \leq n_{ph} \leq \left(\frac{2}{\sqrt{3}}\right) \tag{2.8}$$

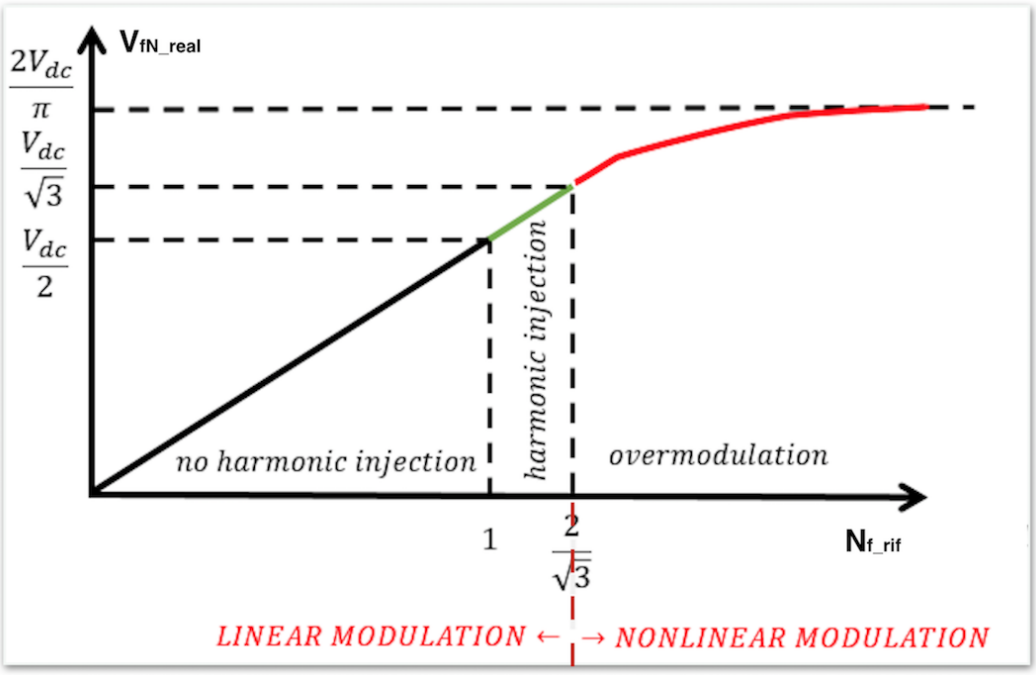

Fig. 2.4 Limit of modulation, with third harmonic injection

So extending the forcing up to 15% in the control, resulting in a linear behavior of the voltages at the load. This is clearly visible in figure 4 showing the linear limit, then with a completely logical reasoning continuing to ask more as

a reference, having however, not a more linear response due to the PWM saturation. So asking more going into a non-linear zone, due to the loss of reality in the modulator, called a quasi-six-step zone, up to the limit that is six-step mode. A further characteristic of the three-phase system visible is that: in every sixth of the period, there are always 2 phases in envelopes, respectively one positive and one negative, while the third phase is in transition. Accordingly another definition of three-phase variables:

$$\left.\begin{aligned} V_{max} &= max(V_{ph-n}) \\ V_{mid} &= mid(V_{ph-n}) \\ V_{min} &= min(V_{ph-n}) \end{aligned}\right\} \tag{2.9}$$

This feature of the three-phase system allows you to easily obtain third harmonic functions for zero-sequence injection. There has been so much talk in the scientific literature, on which is the right function to be added to pulse width modulation techniques. Concluding that for the use of DC voltage and harmonic spectrum the Balance Envelope Modulation results to be the best, since it modulate longer than the other techniques.

$$zero_{seq.} = \frac{V_{mid}}{2} \tag{2.10}$$

Since modular means switching, switching means to lose by switching and losses are heat: it may be useful in some conditions to use the degree of freedom offered to reduce losses. In this regard [17] , it analyzes a discontinuous modulation technique in order to reduce losses in an IGBTs VSI power inverter. In fact, from the equation of the zero sequence it can be seen that given the availability of the maximum, minimum and average value, there are only a few sums to be implemented in the previous BEM technique. This technique is called discontinues, so when:

$$zero_{seq.} = \begin{cases} -1 - V_{min} & \text{if } V_{mid} > 0 \\ 1 - V_{max} & \text{otherwise} \end{cases} \tag{2.11}$$

This kind of technique, as demonstrated by the results of the preliminary draft simulation shown in the figure, has a worst harmonic spectrum, but every sixth of a period there is a phase that does not lose in switching (or is ON or OFF). It

will not be ideal for network applications but greatly improves the appearance of leakage, so significant thermal and reliability benefits can be achieved.

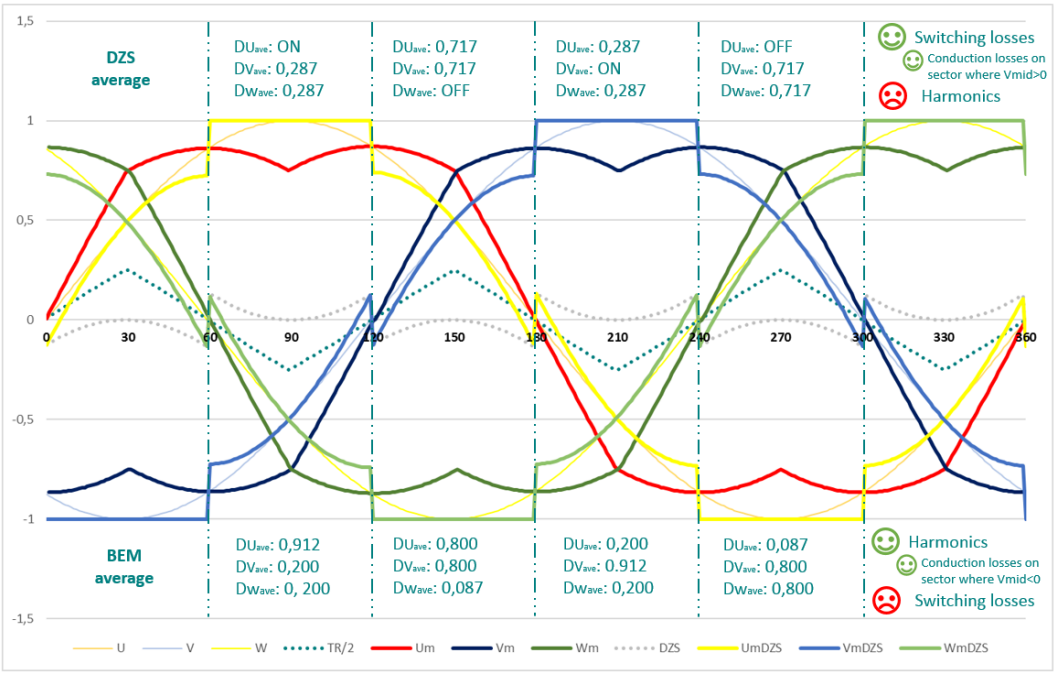

Fig. 2.5 Comparison of modulation technique, SINE (no zero seq. injection) vs BEM vs DZS, in a electrical period.

$$\left.\begin{aligned} n_{1\_rif} &= A \cdot \sin t \\ n_{2\_rif} &= A \cdot \sin\left(t - \frac{2 \cdot \pi}{3}\right) \\ n_{3\_rif} &= A \cdot \sin\left(t - \frac{4 \cdot \pi}{3}\right) \end{aligned}\right\} \tag{2.12}$$

$$m_{crif} = -\frac{Max(n_{f\_rif}) + Min(n_{f\_rif})}{2} \tag{2.13}$$

$$\left.\begin{aligned} m_{1\_rif} &= n_{1\_rif} + m_{c\_rif} \\ m_{2\_rif} &= n_{2\_rif} + m_{c\_rif} \\ m_{3\_rif} &= n_{3\_rif} + m_{c\_rif} \end{aligned}\right\} \tag{2.14}$$

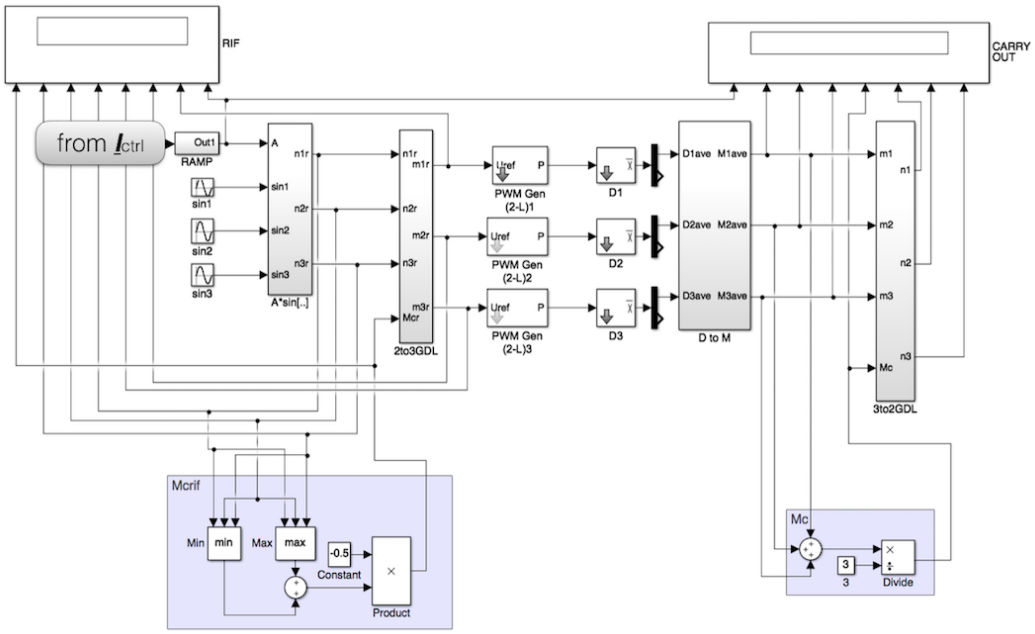

Fig. 2.6 From PI current control, to load 3Ø-index.

All command legs go into a 2-Level PWM generator, which transforms the continuous functions in time in binary functions: discontinuous at a frequency equal to the carrier one. Obviously, the leg is composed of two forced switching elements, then it will create 6 signals but 3 will be the negated of the other:

$$\left.\begin{aligned} D_{1\_rif} &= \frac{m_{1\_rif}}{2} + 0.5 \\ D_{2\_rif} &= \frac{m_{2\_rif}}{2} + 0.5 \\ D_{3\_rif} &= \frac{m_{3\_rif}}{2} + 0.5 \end{aligned}\right\} \tag{2.15}$$

Then the implementation takes place, that is, the saturation:

$$0 < D_{ph} < 1 \tag{2.16}$$

$$HP: f_{sw} \gg f_{bw} \tag{2.17}$$

Under this hypothesis by an average of discontinuous functions can be stated that:

$$\left.\begin{aligned}\tilde{D}_1 &= D_1\\ \tilde{D}_2 &= D_2\\ \tilde{D}_3 &= D_3\end{aligned}\right\} \tag{2.18}$$

Results of the Simulink model, visible in the scope of Figures 6 and 7, confirms the theoretical statements by means of an analysis of the equation system. While Figure 7 focuses on the voltages received by the load, showing the gain of 15%; Figure 8 instead explains where the limit is: the loss of modulation due to saturation.

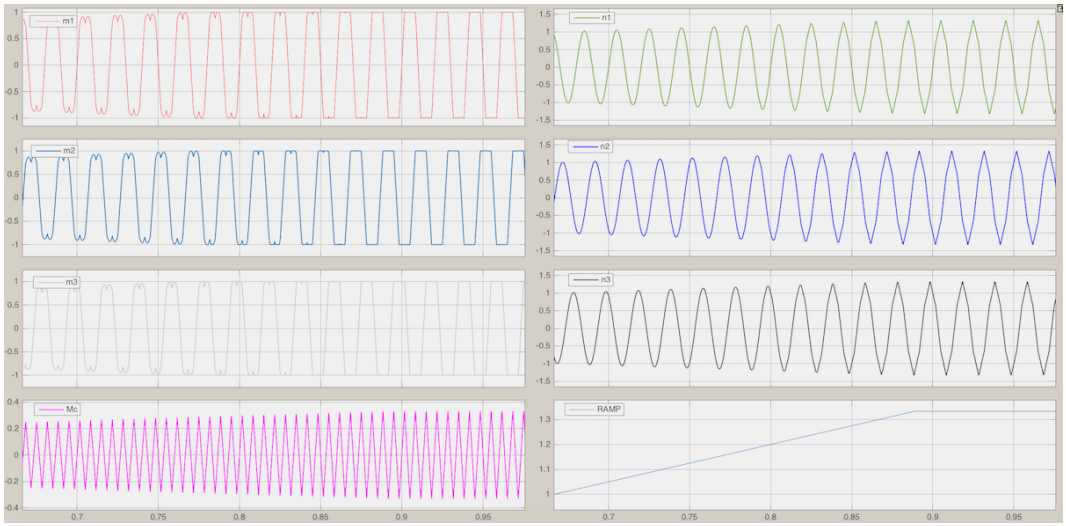

Fig. 2.7 3Ø-Load view, focus in shift from linear zone to 6 step zone.

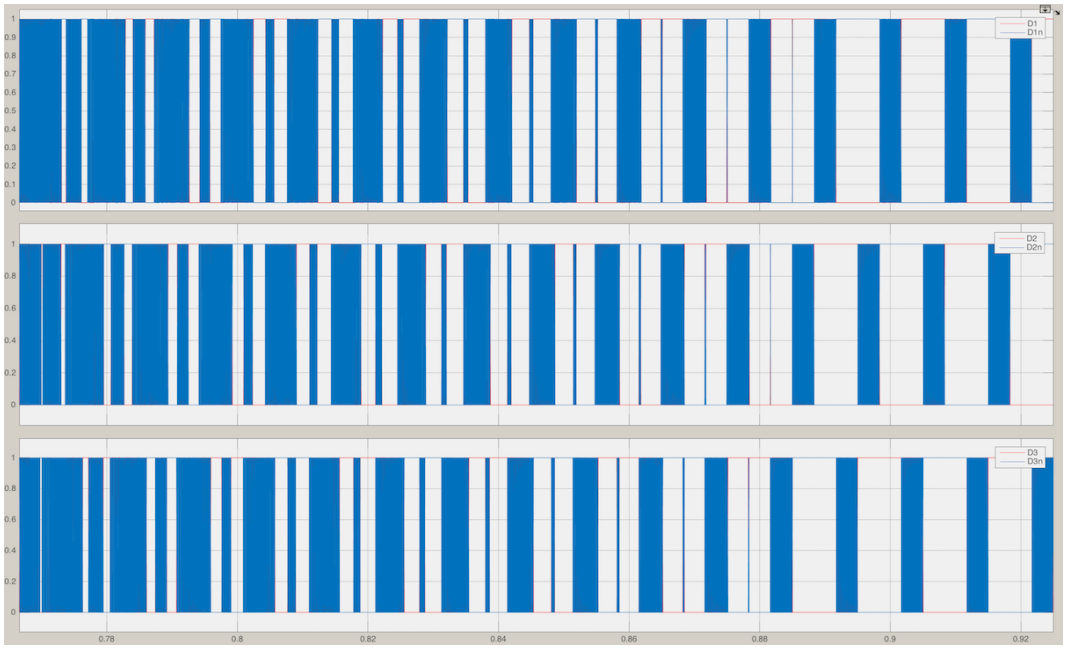

Fig. 2.8 Instantaneous duty, focus on shift from linear zone to quasi-6-step zone.

Summarizing the point, DZS allows a reduction of switching losses of about a third, due to each phase is ON or OFF one third of a period. Previously mentioned, even the harmonic spectrum seen from the load (phase-neutral point) has been simulated, thanks to the FTT block. As stated above, applying the BEM algorithm is confirmed the lower harmonic distortion in overmodulation. While the DZS tends to the worst situation (6-step operation) of harmonic spectrum, more suddenly at the same ramp.

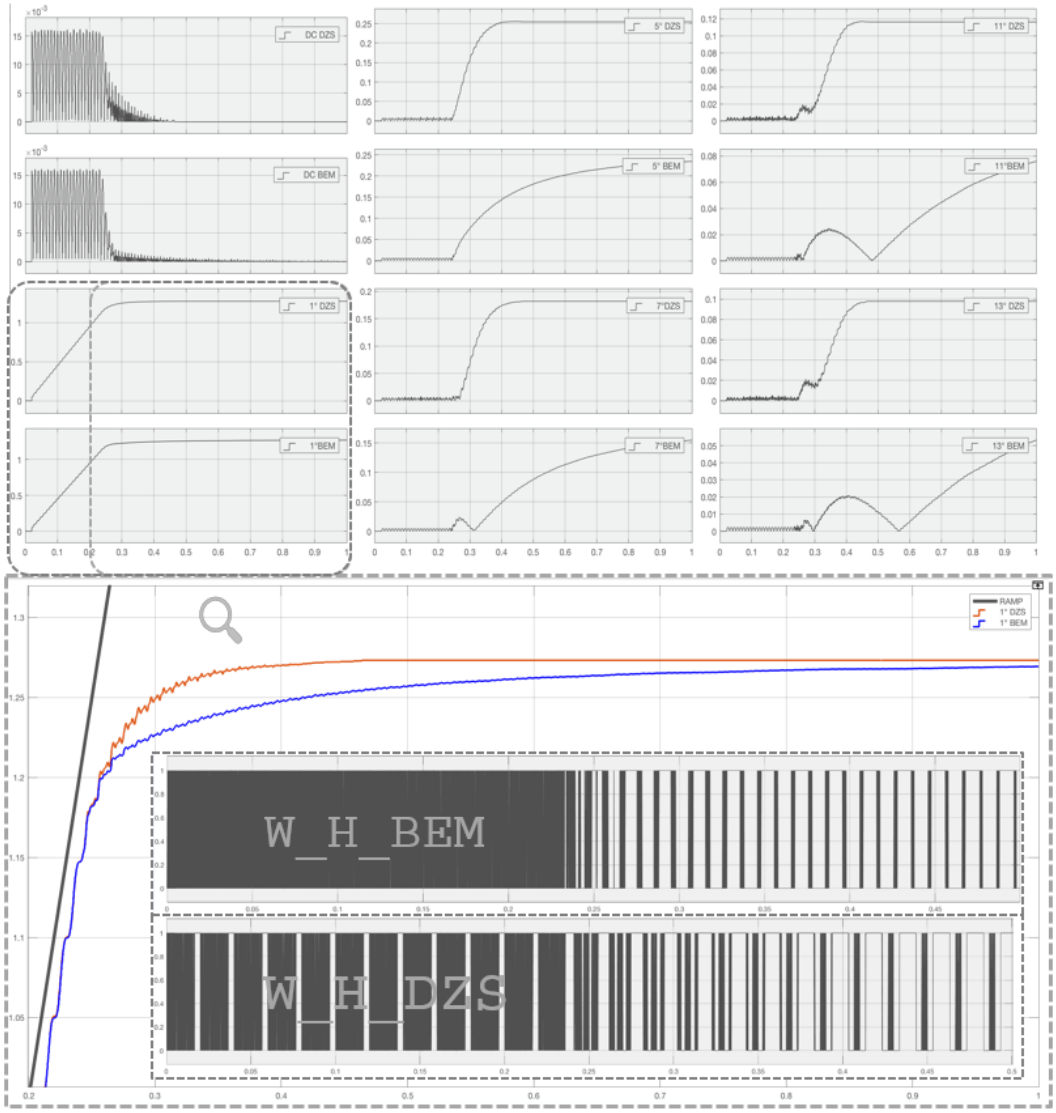

Fig. 2.9 Harmonic spectrum up to six-step operation. Comparison between BEM vs DZS

2.3 MIL: model in the loop.

Model in the loop (MIL) testing and simulation is a technique used to abstract the behavior of systems or sub-systems in a way that this model can be used to test, simulate and verify that mode, sounds like started from the bottom. Whether you want to design a control card from scratch, whether you want to implement new control tasks as in our case, starting from the model is always the best thing: first of all to verify the feasibility of our ideas, also

stability and ultimately the performance achieved. Model-based development and verification approaches are highly desirable in the development of safety-critical embedded systems because they help to identify functional and non-functional issues in the early development stage when verification complexity is relatively lower than that of the implemented systems. Verification and validation are the backbones of any robust model-based development process. It can be used to check the accuracy of the models and algorithms based on code generated by test hardware and software interactions. If the concept of why passing through the MIL has not yet been understood the figure 8 will take away any doubt, as it highlights all the potentialities of this Model process:

- Improve reliability due to design and instant simulation, "what if" analysis faster and easier.
- Testing design control, verification by scope to detect error and wrong loops earlier.
- Repeatability, easy to share and let's remember that the model is also accessible to those who do not have programming language properties.
- Self-generated code: Matlab coder (C/C++) or HDL coder (.VHDL/varilog).

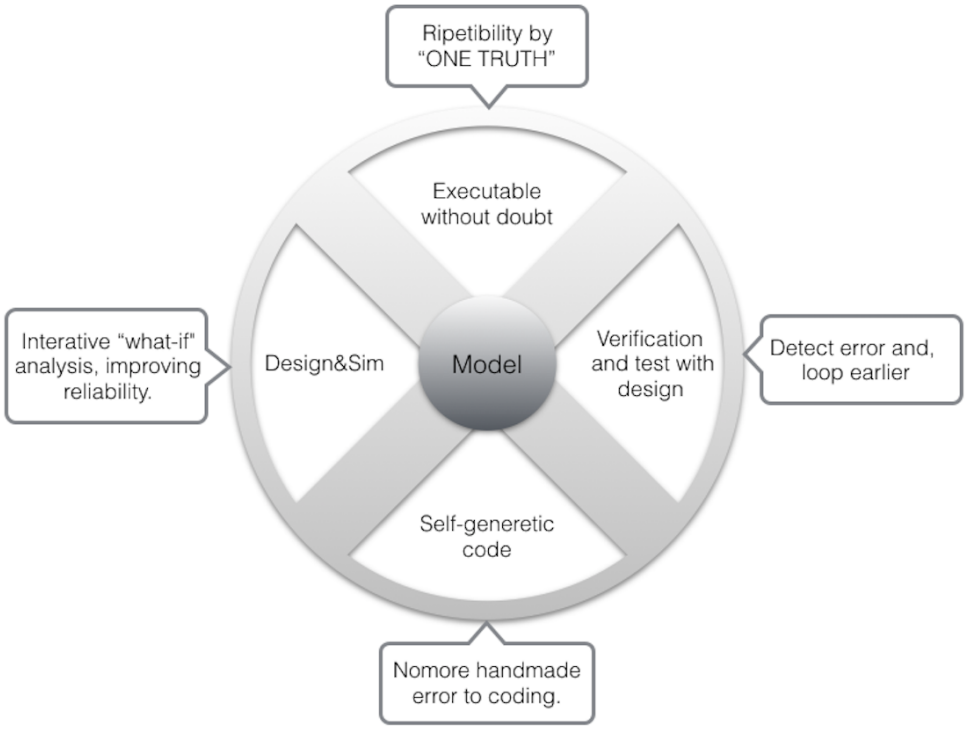

Fig. 2.10 Kay benefit of model in the loop.

The MathWorks word offers a wide range of possibilities for generating, testing and validating your own model. Obviously, when talking about models, it’s more logical to pull Simulink, but other packet environments are also well exploited, such as using .m for S-functions or flow charts (Stateflow MathWork’s app) to realize State machines. With this wide range of apps, it’s a flexible tool that lets the developer get to the ultimate model in really short time. In fact, the Mathworks ambient is so winner that the entire industrial world considers it mandatory as a requirement for embedded engineering for control purposes. Obviously, a complete model development of the "paradox" motor control board is already present as a background. The purpose of the elaborate remains, however, an implementation of a subset of modulation, not of the motor control, in order to improve the performance of the entire motor control set. As such, our interest focuses on the management of fixed axle voltages (d, q), which together with the angle will allow us to derive the 3 voltages, to be applied as modulating to the PWM. All is summed up in a simple diagram shown in figure 11; for a more in-depth discussion referring to

the dedicated chapter "Preliminary concept and draft simulation". The state of the modern art in modulation techniques is the injection of the third harmonic into the three-phase load, so as to achieve a linearity extension of + 15% of the fundamental harmonic.

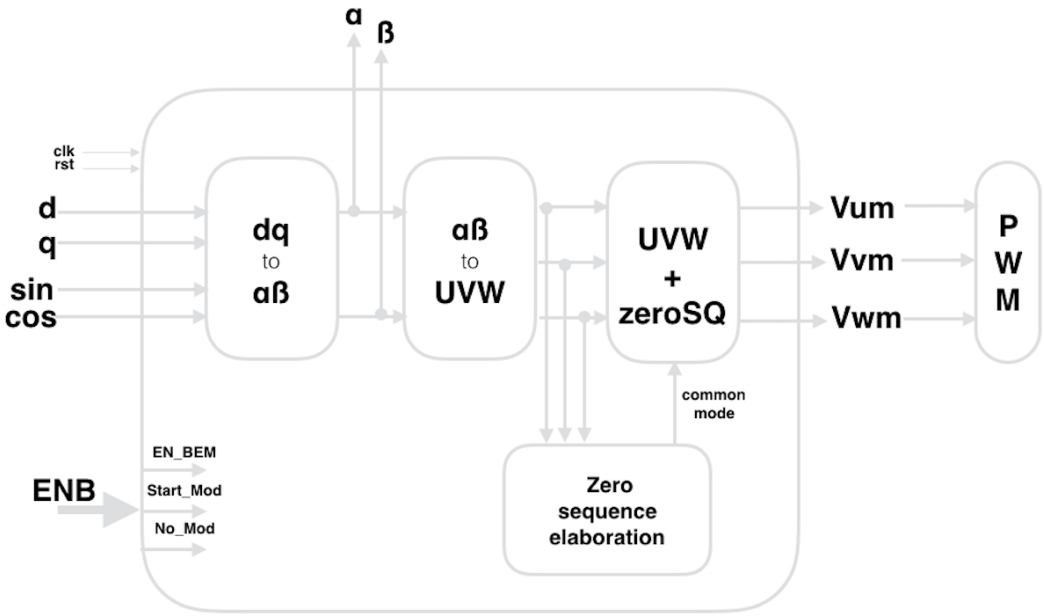

Fig. 2.11 Schematic of summary block design.

To bring the concept diagram of Figure 11, in a Simulink model some equations must be described to understand the process behind:

$$V_\alpha = V_d \cdot cos\theta - V_q \cdot sin\theta \tag{2.19}$$

$$V_\beta = V_q \cdot cos\theta + V_q \cdot sin\theta \tag{2.20}$$

$$V_u = V_\alpha \tag{2.21}$$

$$V_v = \frac{-V_\alpha + \sqrt{3}V_\beta}{2} \tag{2.22}$$

$$V_w = \frac{-V_\alpha - \sqrt{3}V_\beta}{2} \tag{2.23}$$

$$m_c = \frac{-EN - EP}{2} = \frac{TR}{2} \tag{2.24}$$

$$V_{um} = V_u + m_c \tag{2.25}$$

$$V_{vm} = V_v + m_c \tag{2.26}$$

$$V_{wm} = V_w + m_c \tag{2.27}$$

Having understood the logic, after listing the mathematical process through these equations, there is nothing to arm the Simulink libraries and to represent the model in .slx In addition to the above considerations, it should have in mind that the FPGA is a fixed point, which requires the correct scaling of the variables. We opted for a 12bit resolution, so every time:

- sfix12 x sfix12 = sfix24 so extract upper 12 bit
- sfix24 + sfix24 =sfix25 so extract upper 12 bit
- sfix12 + sfix12 = sfix13 so extract upper 12 bit

Having also specified the type of data to run the model, let's go for it. Let's see how there are 4 hierarchies in architecture, shown:

- Fig.21 control box.
- Fig.21 enable logics.
- Fig.22 from fixed axes to phase-neutral voltage.
- Fig.22 zero sequence injection to have phase-medium voltage.

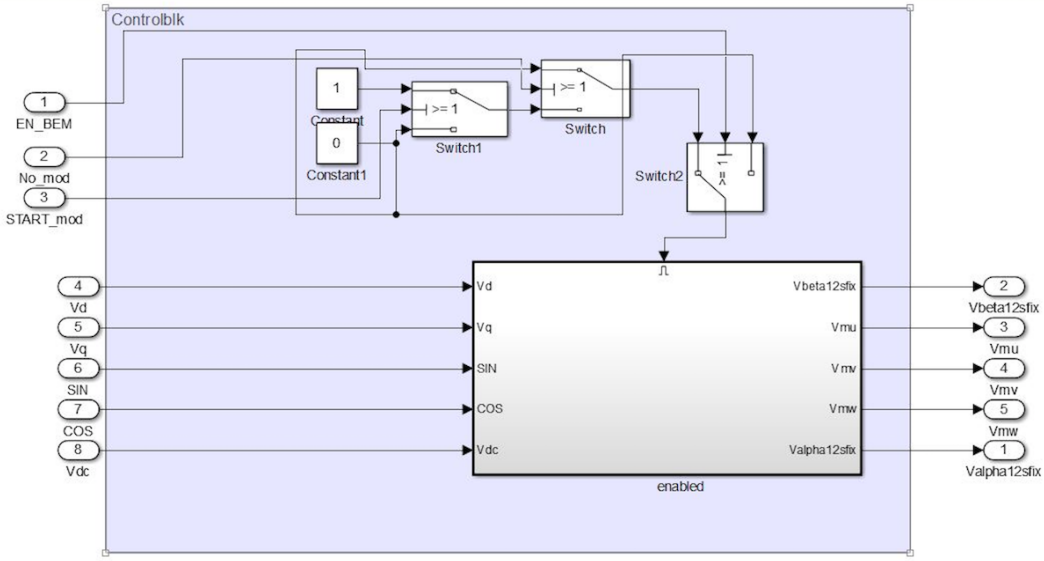

Fig. 2.12 Control and enalble subsystem.

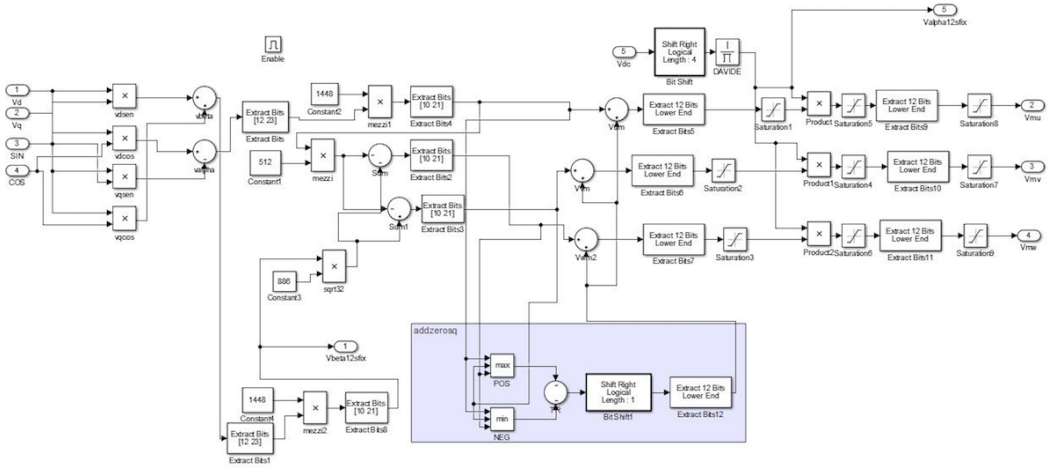

Fig. 2.13 Dq to ph-medium adding zero sequence block.

Obviously, for graphics issues in the images shown in this model, there is no scope for displaying the magnitudes, but it is obvious that during development it should be used for immediate verification of the variables. Some of the fundamental variables, mainly those listed in the equations, can be viewed in the appendix (appendix 2, scopes of MIL). Checks on the model:

- Requirements
- Functional
- Equivalence
- Coverage
- Property Proving
- Virtual Platforms

Since the results shown in the appendix respect the model specifications, it is immediate that once the MIL is checked, you will go to the next step. The next step is to turn the model into real code, that is to switch from .m / .slx to .c / .VHDL files. If the software developer decides to implement in C, the Matlab Coder tool has been widely used for years: a lot of know-how is present and is now used in all industrial environments. The task becomes a bit more difficult if you have to implement a low-level language, as the presentation is almost nothing. Mass-production applications, using a hardware language, are only present in media acquisition and processing: so there is not much-deployed know-how about motors control.

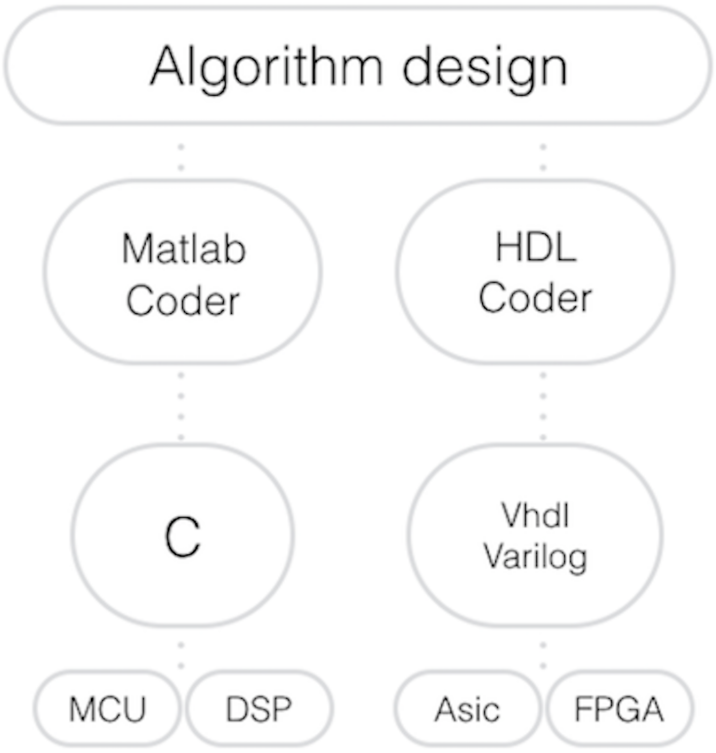

Fig. 2.14 From algorithm to code, C C++ or Vhdl Varilog.

2.4 MIL to SIL, HDL coder : Generating code.

"Writing VHDL is tedious, and the handwritten code still needs to be verified. With Simulink and Simulink HDL Coder, once we have simulated the model we can generate VHDL directly and prototype an FPGA. It saves a lot of time, and the generated code contains some optimizations we had not thought of." says Frantz Prianon, digital design manager at Semtech. So with the rapid advancement in FPGA design technologies, hardware, and software providers have been working towards producing more advanced and user-friendly tools for designing and testing FPGA programs. The emphasis lies on rapid prototyping of the FPGA design, even with minimal knowledge of Verilog and VHDL programming languages. To this end, MATLAB HDL Coder which can be used with MATLAB Simulink is tool enables the developers to use Simulink model environment, with drag and drop block sets, for designing their algorithms without having to write a single line of VHDL code, even for very large and complex designs. The HDL Workflow Advisor in HDL Coder automatically converts MATLAB code from floating-point to fixed-point and generates synthesizable VHDL and Verilog code. This capability lets you model your algorithm at a high level using abstract MATLAB constructs and System objects while providing options for generating HDL code that is optimized for hardware implementation. HDL Coder provides a library of ready-to-use logic elements, such as counters and timers, which are written in MATLAB. The HDL Workflow Advisor generates VHDL and Verilog code from Simulink and Stateflow, too. With Simulink, you can model your algorithm using a library of more than 200 blocks, including Stateflow charts. This library provides complex functions, such as the FFT, FIR filters, and so on for modeling signal processing and communications systems and generating HDL code. As shown in figure 15, the main features of HDL Coder are:

- Synthesizable VHDL and Verilog code.
- Code generation support for MATLAB functions, System objects, and Simulink blocks.
- Mealy and Moore finite-state machines and control logic implementations using Stateflow.
- Workflow advisor for programming Xilinx and Intel application boards.

- Resource sharing and retiming for area-speed tradeoffs.
- Legacy code integration

While designing with HDL Coder in Simulink, the first step is to filter the Simulink Library Browser, such that it only shows the model blocks that are compatible with the HDL Coder. To this end, by typing "HDL LIB" in MATLAB command prompt, one gets the Simulink Library Browser showing only the supported blocks. At present, HDL Coder supports over 200 Simulink blocks, using these blocks the developer can design the required communication and signal processing logic as a Simulink model file, by dragging and dropping various blocks into the design. Core Simulink Blocks: Basic and Array Arithmetic, Look-Up Tables, Signal Routing (Mux/Demux, Delays, Selectors), Logic and Bit Operations, Dual and single port RAMs, FIFOs, CORDICs, Busses. Signal Processing Blocks: NCOs, FFTs, Digital Filters (FIR, IIR, Multirate, Adaptive), Rate Changes (Up and Down Sample), Statistics (Min/Max) . Communications Blocks: Psuedo-random Sequence Generators, Modulators / Demodulators, Interleavers / Deinterleavers, Viterbi Decoders.

Furthermore, HDL coder automatically converts floating point numbers into fixed-point by Float-to-Fixed Workflow and support functions are written in MATLAB .m code so they can be integrated into the design. To check design results via simulations, different MATLAB tools could be used, including scopes, displays, etc. Blocks from Simulink Sources library can be used to generate test signals for the design (sounds like an adaptation of MIL for HDL Coder). Once the design results have been verified through simulations, the next step is to generate the HDL codes, which is done through Workflow Advisor, which comes with HDL coder. Through the Workflow Advisor, one can select different parameters for the design including:

- the target workflow frequency.
- the targeted platform.
- the targeted FPGA.
- check the global settings, algebraical loops, sample time and blocks compatibility.

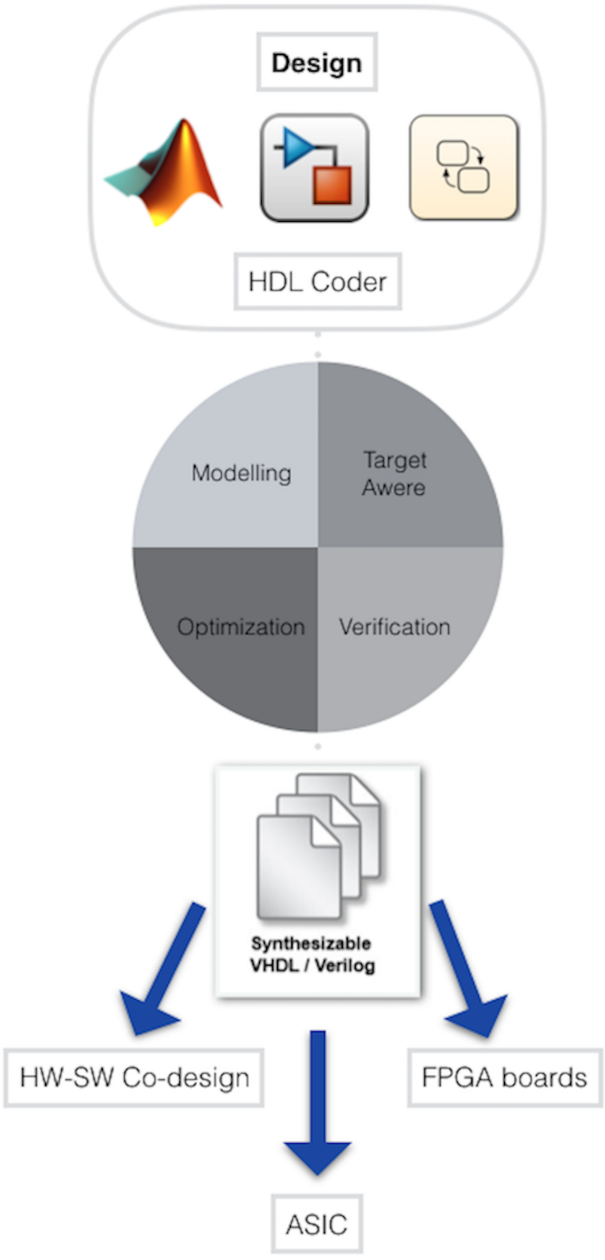

Fig. 2.15 HDL Corder keypoint.

- set basic options as: Vhdl or Varilog, the traceability of report as critical path, etc ...
- set advance options as: coding style, clocks and ports, optimizations etc ...
- optimize general parameter, pipeline and the sharing resources.
- finally generate code.

The code shown is for the zero sequence production (block model showed in figure 13): this allows to extend up to 15% the fundamental harmonic. The complete model, that transform (d,q) to (phase-medium), the generated code is shown in Appendix 3. Also, some report from HDL coder is show, that gives info to embedded engineer designer as:

- Clock Summary
- Code Interface Report
- Timing And Area Report
- High-level Resource Report
- Critical Path Estimation
- Optimization Report
- Distributed Pipelining
- Streaming and Sharing
- Delay Balancing
- Adaptive Pipelining
- Traceability Report

```
-- -------------------------------------------------------------
--
-- File Name: /Users/andreapiccioni/Desktop/vediquesti/20.5.17/hdlsrc/ppp
-- Created: 2017-08-01 12:45:38
--
-- Generated by MATLAB 9.2 and HDL Coder 3.10
--
-- -------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY bem IS
  PORT( Vu                                :   IN    signed(11 DOWNTO 0);
        Vv                                :   IN    signed(11 DOWNTO 0);
        Vw                                :   IN    signed(11 DOWNTO 0);
        Mc                                :   OUT   signed(11 DOWNTO 0)
        );
END bem;


ARCHITECTURE rtl OF bem IS

  -- Signals
  SIGNAL EP_stage1_1_val                  : signed(11 DOWNTO 0);  -- sfix
  SIGNAL EP_stage2_val                    : signed(11 DOWNTO 0);  -- sfix
  SIGNAL EN_stage1_1_val                  : signed(11 DOWNTO 0);  -- sfix
  SIGNAL EN_stage2_val                    : signed(11 DOWNTO 0);  -- sfix
  SIGNAL TR_out1                          : signed(13 DOWNTO 0);  -- sfix
  SIGNAL Bit_Shift_out1                   : signed(13 DOWNTO 0);  -- sfix
  SIGNAL Extract_Bits_out1                : signed(11 DOWNTO 0);  -- sfix
```

```
BEGIN
  ----- Tree max implementation -----
  --
  -- <S13>/EP

  EP_stage1_1_val <= Vu WHEN Vu >= Vv ELSE
      Vv;


  EP_stage2_val <= EP_stage1_1_val WHEN EP_stage1_1_val >= Vw ELSE
      Vw;

  ----- Tree min implementation -----
  --
  -- <S13>/EN

  EN_stage1_1_val <= Vu WHEN Vu <= Vv ELSE
      Vv;


  EN_stage2_val <= EN_stage1_1_val WHEN EN_stage1_1_val <= Vw ELSE
      Vw;

  -- <S13>/TR
  TR_out1 <= ( - (resize(EP_stage2_val, 14))) - resize(EN_stage2_val, 14);

  -- <S13>/Bit Shift
  Bit_Shift_out1 <= TR_out1 srl 1;

  -- <S13>/Extract Bits
  Extract_Bits_out1 <= signed(Bit_Shift_out1(11 DOWNTO 0));

  Mc <= Extract_Bits_out1;

END rtl;
```

Fig. 2.16 Balance envelope modulation generated code.

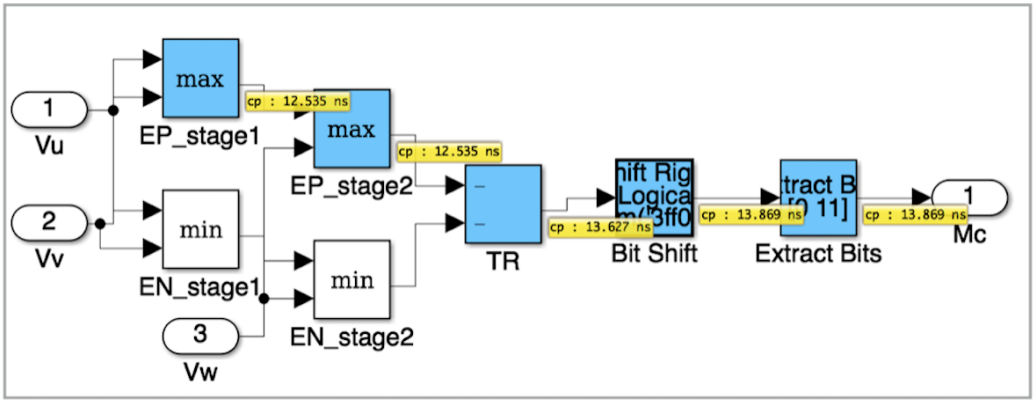

Fig. 2.17 eg. Most critical path in bem subsystem

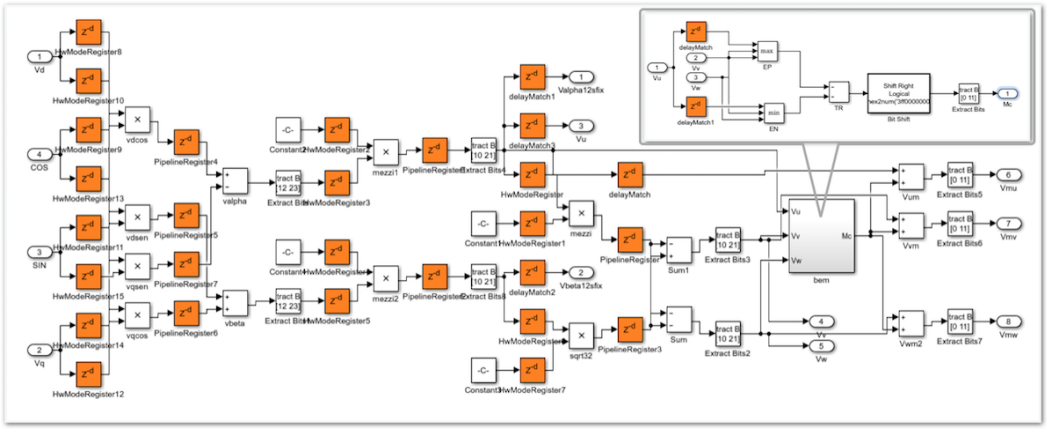

Fig. 2.18 Pipeline fully auto-delays generated

2.5 SIL: software in the loop.

Once the code has been produced, through the process described in "MIL to SIL", it only needs to be verified through the tools that allow the software in the loop. Verification and validation techniques applied throughout the development process enable you to find errors before they can derail your project. Most system design errors are introduced in the original specification but aren't found until the test phase. When engineering teams use models to perform virtual testing early in the project, they eliminate problems and reduce development time by as much as 50%. Activities for verification, validation, and test with Model-Based Design can be applied at every stage of the development process, as shown. SIL is a critical step of designing FPGAs and ASICs. Simulation allows the designer to stimulate his or her design and see how the code that they wrote reacts to the stimulus. A great simulation will exercise all possible states of the design to ensure that all input scenarios will be handled appropriately. Did you forget an if statement somewhere? Did you remember to give every possible case statement assignment? These are the types of errors that are very easy to make when you do not simulate your design. Mentor Graphics: ModelSim. the third part is needed. Digital waveforms are difficult to analyze, an application specific analysis methods are needed, so how to get test vectors to achieve 100% test coverage? ModelSim can be used to simulate VHDL-code, to determine whether it is "right" thinking. Mentor Graphics was

the first to combine single kernel simulator (SKS) technology with a unified debug environment for Verilog, VHDL, and SystemC. The combination of industry-leading, native SKS performance with the best integrated debug and analysis environment make ModelSim the simulator of choice for both ASIC and FPGA designs. Features:

- Fast time-to-debug, easy to use, multi-language debug environment.
- Powerful Waveformcompare for easy analysis of differences and bugs.
- Coupled with HDL Designer and HDL Author for complete design creation.
- High-level Resource Report
- For some platforms version of ModelSim is also integrated with a "database" with facts about chips, eg. Altera MAX-chips, so one can also do simulations that take into account the "time delay"and other phenomena within the intended target circuit.

However, for a graphic issue it was not possible to see all the variables in all the pipeline steps, so only the inputs and outputs are present, with a particular interest as usual in common mode, as show in figure 19.

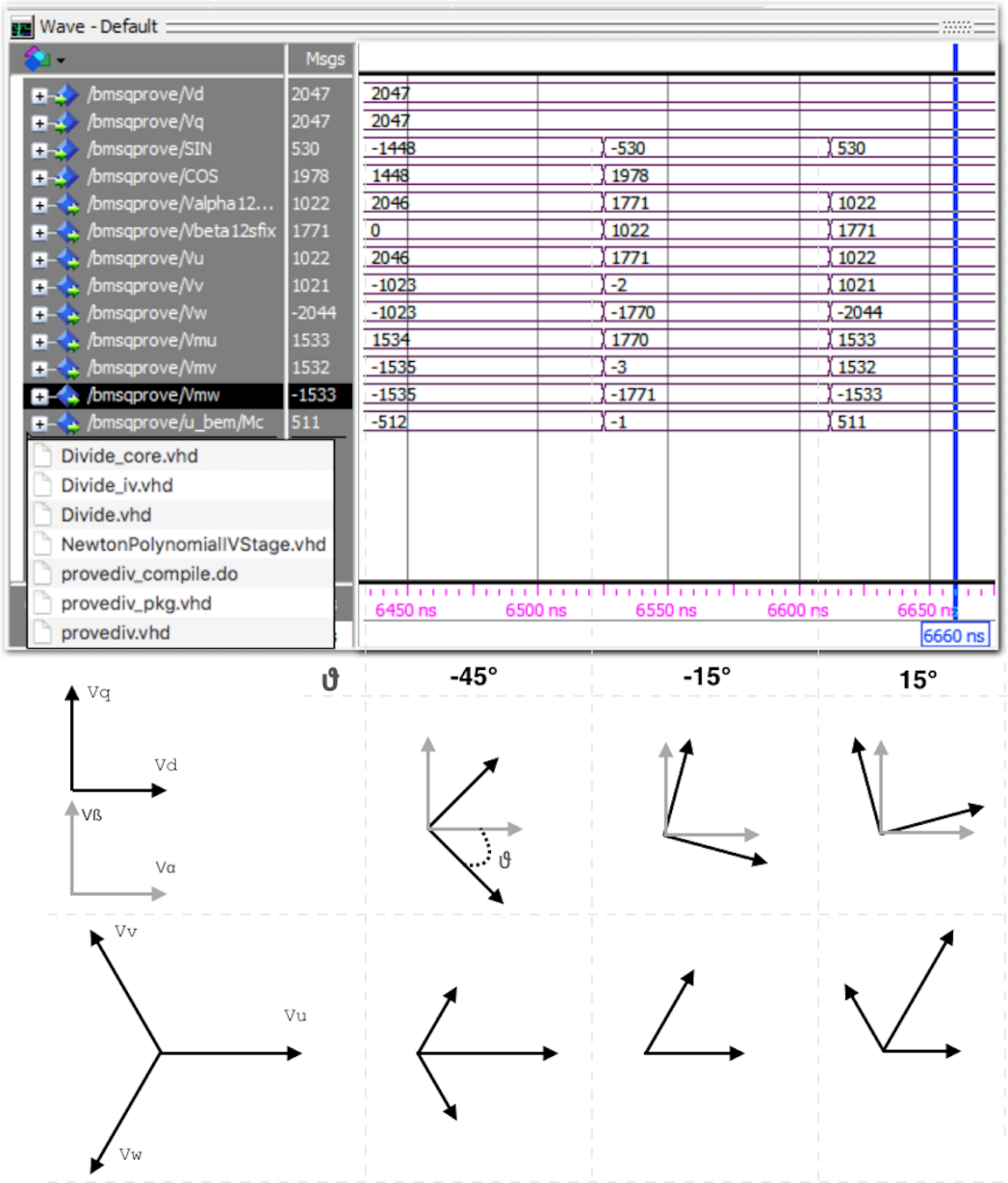

Fig. 2.19 Stimuli-Driven Test Bench in HDL Simulators

With HDL Verifier, again, third party simulation tools such as ModelSim will be required. HDL Verifier provides an elegant and integrated environment for running HDL co-simulations with Simulink and third party tools. Additionally, it also provides FPGA in the loop feature, enabling the developers to test the design on actual hardware from within the Simulink environment. HDL co-simulation with HDL Verifier lets one verify that the HDL code matches

theMatlabalgorithms andSimulinkmodels by providing visibility into the HDL code (see figure 20). One can assess how differences between expected results and HDL simulation could affect the design at the system level.

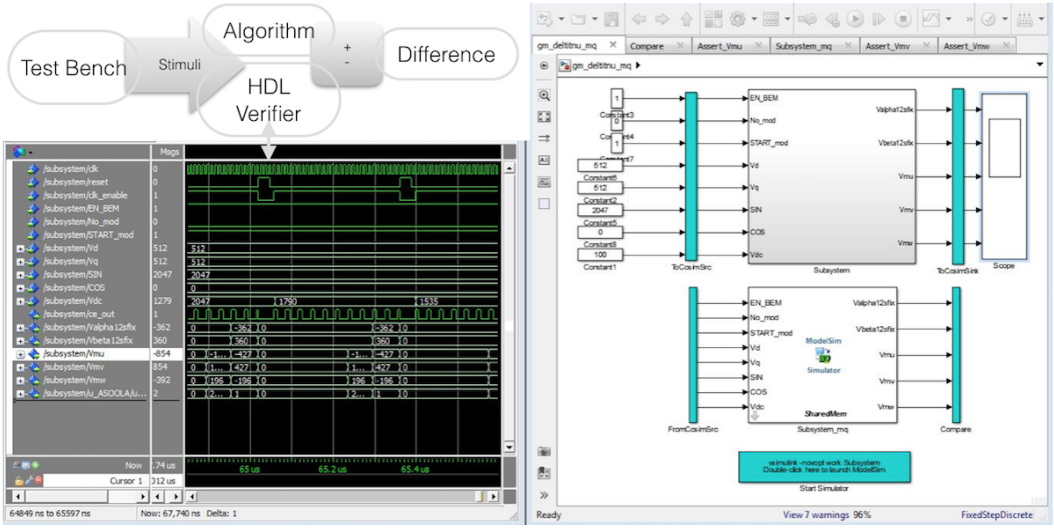

Fig. 2.20 Control block stimuli-driven by HDL Verifier+ModelSim Co-simulation.

2.6 SIL to FIL, synthesizer tools.

Once the code is uniquely generated by the template and verified through the processes described so far, it remains to give it a real meal to the FPGA board in the analysis. To do this, you need third-party tools, based on the FPGA manufacturer. In our case, Xilinx offers a complete synthesis package that includes ISE Design Suite and Vivado Design Suite. The first one is essential because it allows full access to hardware resources, while the second allows the interface with the Mathworks environment through the Xilinx System Generator tool, a detailed discussion later. By analyzing purpose, that is interior permanent magnets motor control, not just the block in development is to be considered. There is a need for SPI communication, an implementation of the modulation, as well as an orchestra conductor who says the timings and system states. To meet these requirements for DQ to ph-medium lock, other subsystems owned by the Politecnico di Torino, handwritten by Eric Armando, were used to complete the motor control. The situation under consideration is, therefore: in the .vhdl pFARAD kart engine control project, the BEM block

of the Politecnico will be deleted, replacing the subsystem generated by the HDL coder for the generation of modulators. The types of blocks used and their functionalities are described in the following bulleted list and shown as ISE schemes in Figure 21:

- System time manager: Is the orchestra conductor, takes the nature of the oscillator and transforms it into a complete architectural time handler. Time management (clk, en, flag, etc) of the various subsytem is entrusted to him.
- State manager, options: start up, AD offset, AD end of offset, stop, GO or error. Select which kind of step are running into motor control.
- SPI communicaton block: Serial Peripheral Interface with double channel, 5 data each channel and a slave role.
- DQ to ph-medium, generated by HDL coder: take the (d,q) and (sin,cos) from SPI and elaborate the modulations index send to PWM modulators.
- Three phase PWM: modulation of the indexes is sent by the block, transforming them into the 6 leg commands.
- Manager IPM: Takes the leg commands and according to the status of control processed by the State Manager, whether or not the commands sent by the modulator are compressed.

In Figure 19 you can also see the operations that can be performed with the ISE design suite tool:

- Synthesize and generate post-synthesis simulation model.
- Implement design, mapping resources and routing the path. Optimization process.
- Generate the .msc file to program FPGA.
- Configure target device: pop up tool that allow to erase and program the selected fpga.

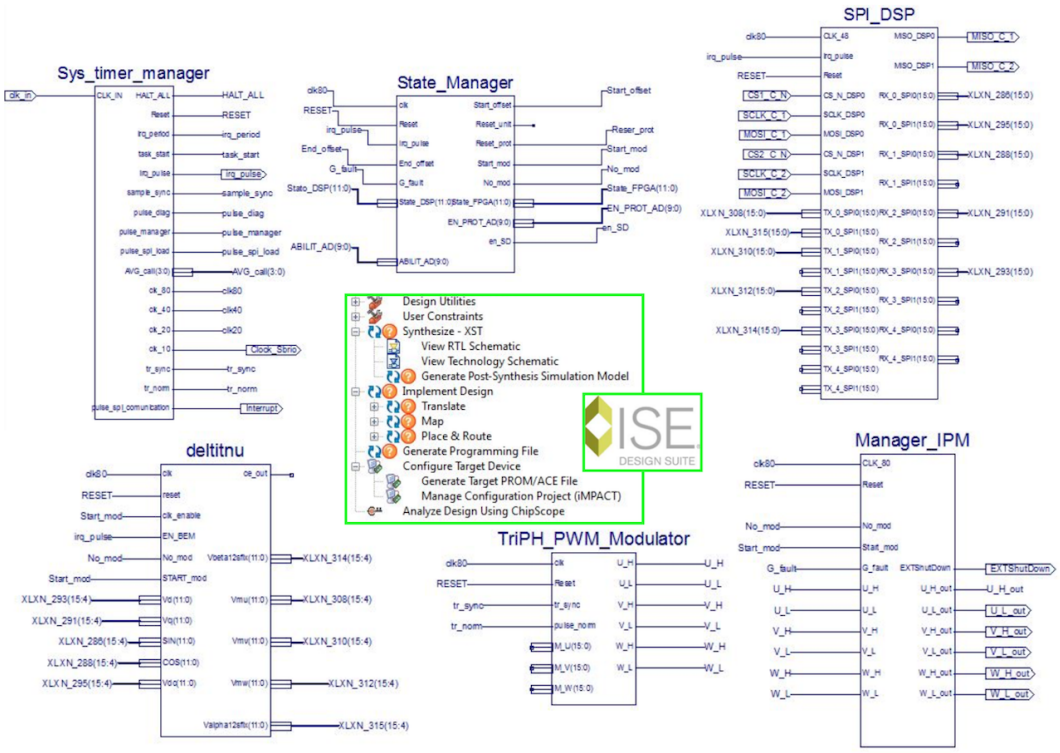

Fig. 2.21 ISE main schematic blocks.

Once synthesized, optimized and programmed, the hardware has nothing but to verify what has been implemented through the next step in product development: FPGA in the loop. To do this, Xilinx offers its Xilinx System Generator tool. In addition to bringing the programmer from SIL to FIL, in a single Mathworks interface, there are other interesting features to focus on. Xilinx System Generator is an FPGA programming tool specifically focussed on Xilinx FPGAs, enabling the developers to work in the Simulink environment and to generate parameterized cores particularly optimized for Xilinx FPGAs. The tool need:

- Xilinx ISE Design Suite (System Edition)
- Xilinx Vivado HL (System Edition).

By default, the Xilinx Blockset contains over 90 DSP blocks:

- Adders
- Multipliers

- Memories
- FFTs
- Filters

Moreover, the System Generator also includes the .m code and Black Box blocks, which can be used to integrate .m code and "handwritten" VHDL codes, respectively, directly into the Simulink design environment. For those who have long programmed FPGA, perhaps before model-based design, this interesting feature allows you to validate and test your handwritten .vhdl modules. As an example in Figure 20, you can see the implementation of the BEM.vhdl written by E. Armando, stimulated with the various DSP blocks proposed by the Xilinx System generator. Obviously, it is possible to create continuity with what is said in the previous paragraphs with the ModelSim token, even if ISE provides a personal analysis tool.

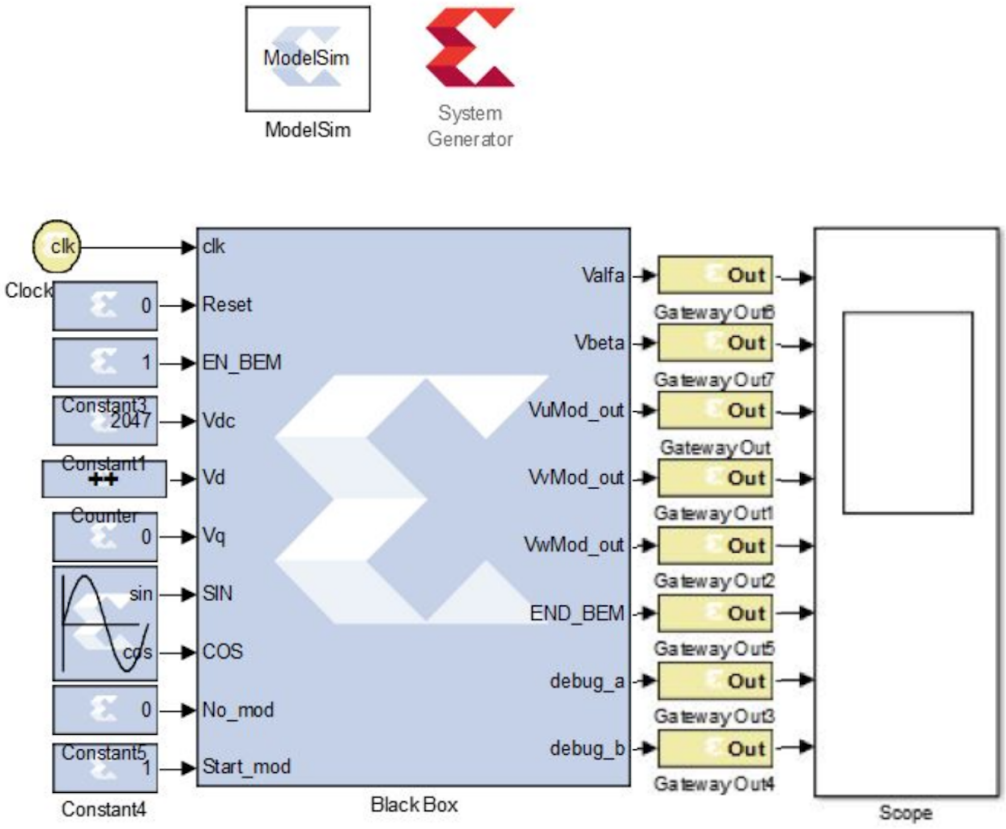

Fig. 2.22 BEM.vhdl by E.Armando, testing by black-box + ModelSim Co-simulation.

System Generator works quite similar to HDL Coder, users drag and drop various blocks into the Simulink environment in order to design the overall system. Users can then use MATLAB scopes and other sinks as well as elements of Sources library to check the design results. Since System Generator is already part of Xilinx ISE or Vivado HS, no additional synthesis tools are required and the users can generate the bitstream directly from within the Simulink environment. To this end, the System Generator token needs to be added to the design. The System Generator token provides the users with functionality quite similar to HDL Coder Workflow Advisor, in that it allows users to select specific target workflow and platform. In order to check and verify the HDL codes through simulations, the System Generator can be used to invoke the default HDL simulator from Xilinx, which comes as part of the design suite and here again, unlike HDL Coder and HDL Verifier; no third-party HDL simulation tool is required. Nevertheless, System Generator is also compatible with third party simulation tools such as ModelSim, thus providing an integrated flow. Lastly, the System Generator also supports hardware in the loop feature, which is the equivalent of FPGA in the loop feature of HDL Coder and HDL Verifier. To this end, the Hardware Co-Simulation drop-down menu within the System Generator token enables the users to perform tests on real hardware directly from within Simulink. To summarize Matlab HDL Coder and Xilinx System Generator both enable rapid prototyping of the FPGA design by utilizing MATLAB and Simulink environment. Both of these packages come with their associated pros and cons. The HDL Coder provides a complete integrated environment for the design flow. It supports a larger number of MATLABFunctions and Simulink Blocks. It also automatically converts the floating point numbers to fixed-point. Furthermore, HDL coder provides the means to generating target-independent Verilog and VHDL codes, which can then be synthesized for various FPGA platforms. On the downside, the HDL Coder alone is insufficient for synthesizing, simulating and verifying the HDL codes. Additional MATLAB toolboxes and third-party software are required to achieve these goals. The Xilinx System Generator comes as part of Xilinx design suits and is specifically tailored for Xilinx products. It also supports a sufficiently large number of DSP blocks, including those that support .m code, HDL codes, and floating-point DSP. Other than MATLAB itself, no third party software is needed, neither for generating HDL codes from Simulink blocks

nor for synthesizing and verifying the generated codes. On the downside, it is only limited to Xilinx products and requires manual conversion of Simulink floating-point numbers to fixed-point, which can be a time to consume and error-prone process.

2.7 FIL: FPGA in the loop.

Not having a Xilinx System Generator compatible developer board, switching the stimulus directly from the Simulink environment via JTAG or Ethernet to the board is not possible. The need to stimulate FPGAs in order to validate the architecture produced motivates the presence in controlling a dual SPI channel. In fact, in the validation process in question, it will be an external DSP to stimulate, via SPI, the FPGA engine control board.

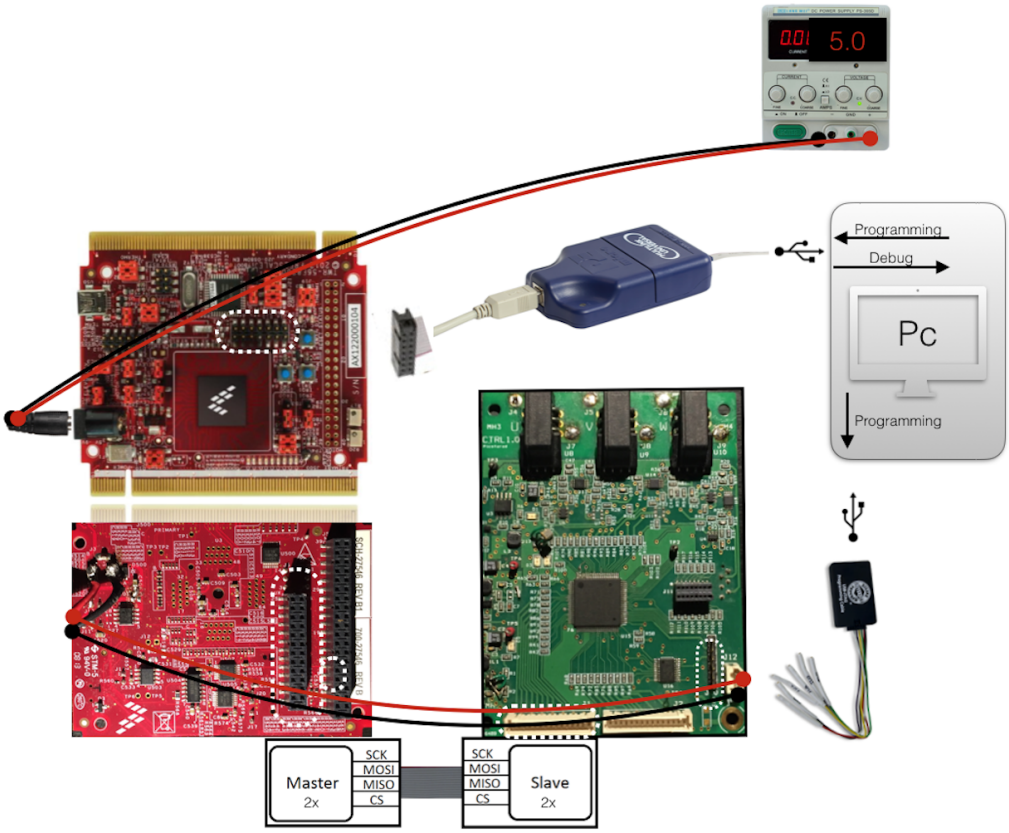

Fig. 2.23 Hardwere Control Unit: summary schematic about control and debug the fully integrated system FPGA + DSC.

A simple laboratory test banch, figure 32, composed of:

- Laboratory 5V power supply.
- FPGA pFarad Motor-control board.
- TWR-56F8200 (MCU) by Freescale.
- Multilink and JTAG cable.
- PC and oscilloscope.

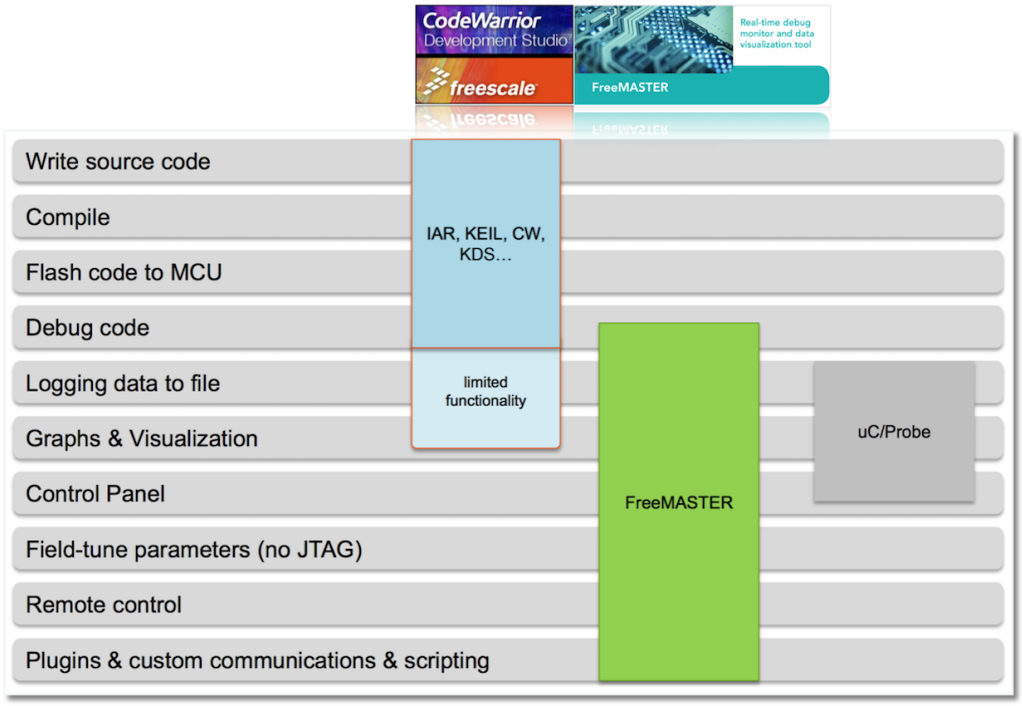

Fig. 2.24 Freescale Freemaster and Code Worrior key skills.

Why do all this? Failing to stimulate Simulink, the Freescale board is a tool that allows easy programming to give INPUT to the board, and easy debugging, in order to validate the processes that take place in tasks handled by the FPGA. Some features of Freescale's programming (CW) and debugging (FM) products are shown in Figure 33, in addition to their free-licenses, they provide a truly user-friendly environment, one of the most intuitive MCU products.

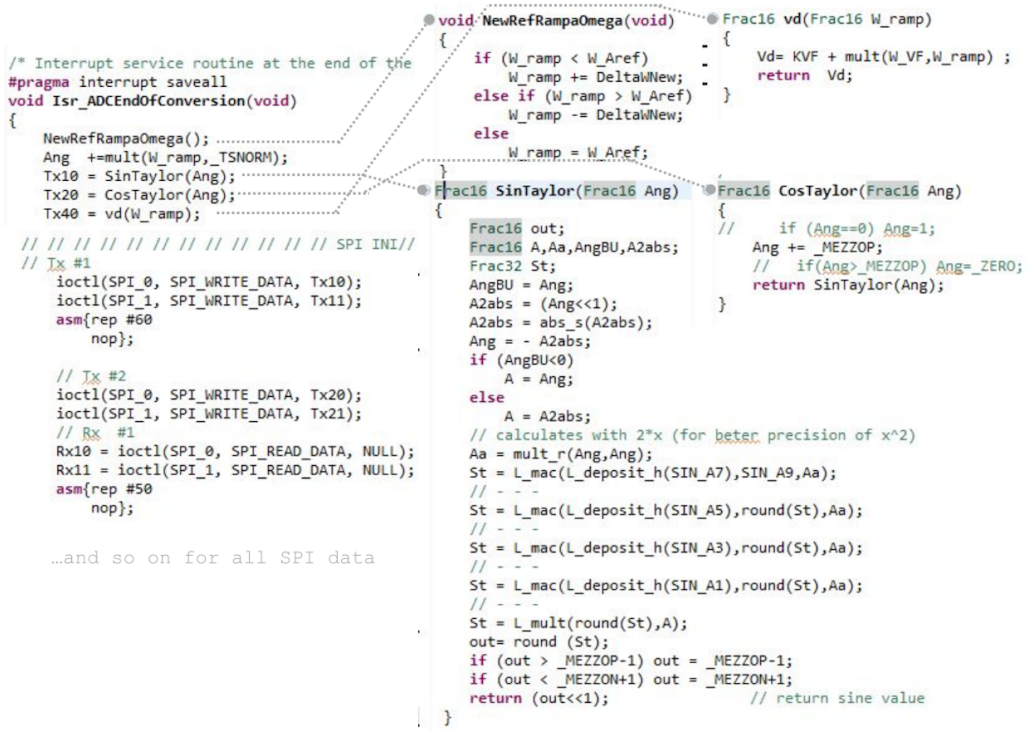

Fig. 2.25 MCU ISR and main called function.

Through Code Worrior then we will implement tasks for the INPUT generation, which will be transmitted to the engine control card via SPI. Using the ISR, called by the interrupt sent in turn by the FPGA slave board, we generate all the required variables for the control:

- Vd: come from V/Hz control, it has linear dependence on velocity through a specific coefficient.
- Vq: freezing at zero.
- Vdc: to simulate the info from the voltage partition for measuring the dc bus.
- Sin (Angle) : using a Taylor series, is a representation of a function as an infinite sum of terms that are calculated from the values of the function's derivatives at a single point.
- Cosin (Angle) : add 90 degree at Angle variable, and call the same generating sin-function.

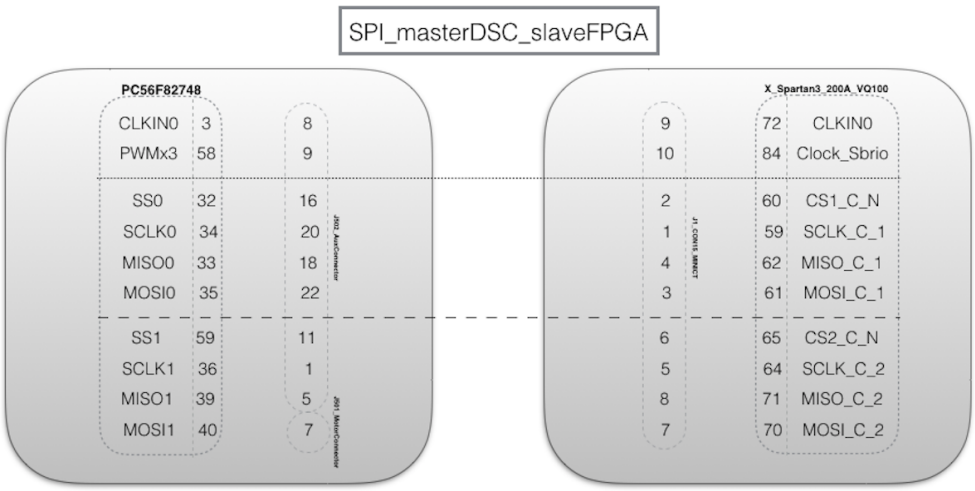

Fig. 2.26 SPI CORE to CORE pin-out.

Some of the functions that are called during the routine service interruption are explained in Figure 34, for the complete display of MCU programming sources C, refer to Appendix 4.

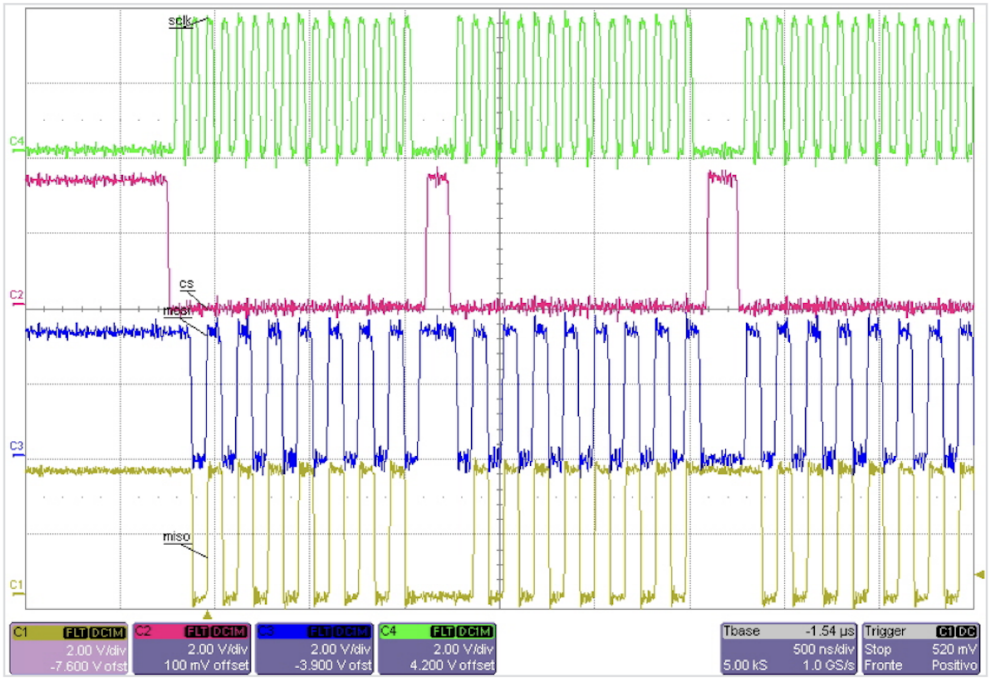

Fig. 2.27 SPI test sending in repetition h5555 and hAAAA, one negates the other at binary level.

Particular attention should be paid to the SPI line, as the source of communication between the two cards. In the diagram of Figure 26, it is possible to see a complete pinout of the two communication channels, starting from the core of the MCU to its connector, passing through the pinout of the FPGA board AMP 15 CT Wire-to-Board Connector and finally to the FPGA core. To test SPI channel h5555 and hAAAA data are sent alternating and repetitive, not having any particular problems as shown in Figure 27. It is clearly indispensable to see in the actual reality the response to the stimuli imposed by the MCU master, where the V / Hz task is processed each interrupt stroke mandated by the FPGA slave board. In order to debug the appropriate scope on Free-master can show the sent and received variables after slave-board processing. In figure 28 are show the follow tasks-results:

– Velocity control, 16-bit. Implemented in MCU.

- V/Hz control: Vd, Vq, sin and cosine generation. Implemented in MCU. Vdc set in free master as variable. All send by CH1.
- DQ to ph-medium results, implemented in FPGA board, then sends to MCU by CH1.

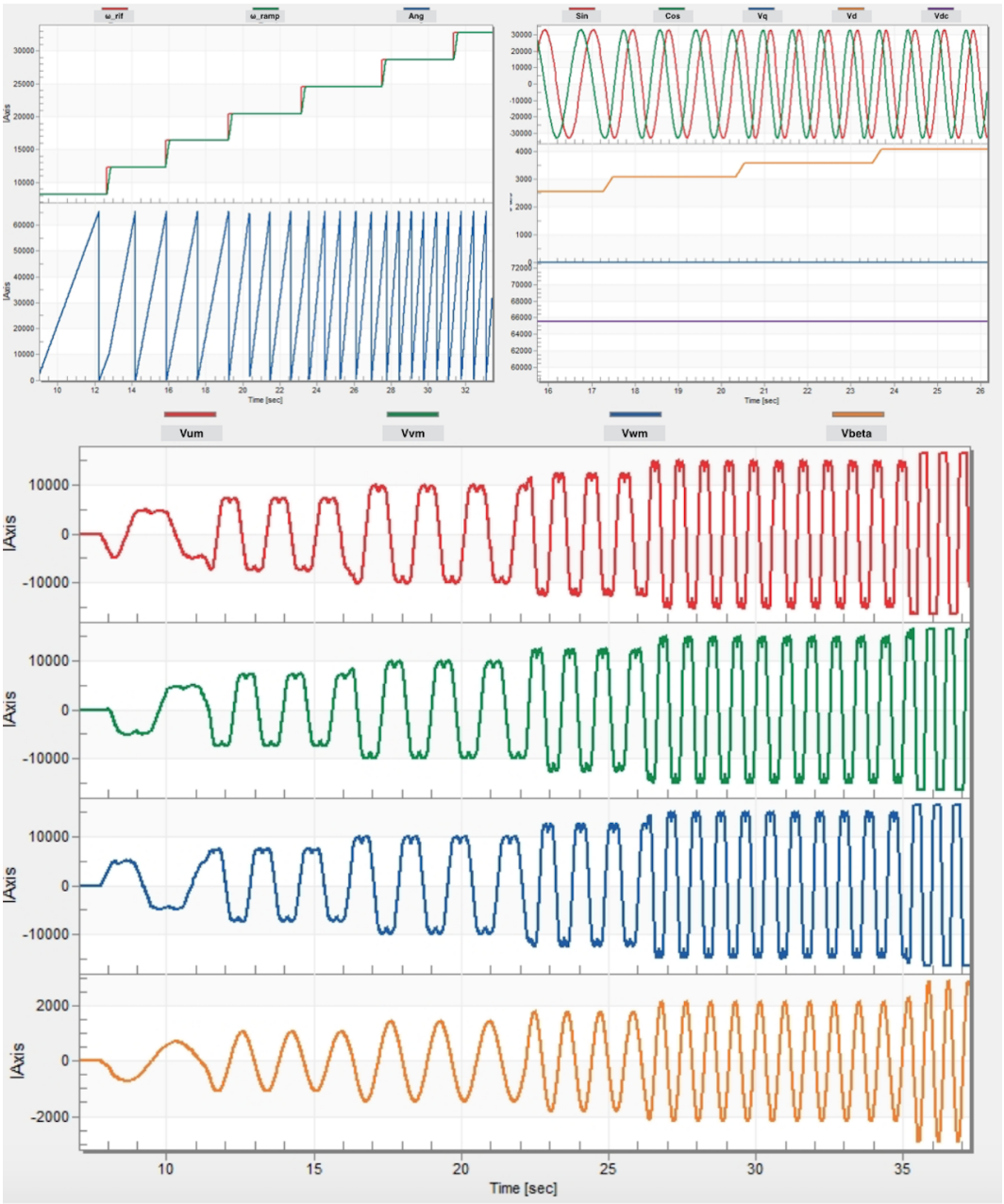

Fig. 2.28 DQ to phase-medium results, stimulated by V/Hz control running in MCU master. Maximum hFFFF costant DC bus voltage.

MCU task:

$$\left.\begin{aligned} \omega_{ramp} &= \omega_{ramp} + \Delta \\ \omega_{ramp} &= \omega_{ramp} \\ \omega_{ramp} &= \omega_{ramp} - \Delta \end{aligned}\right\} \quad \text{According to the set speed} \tag{2.28}$$

$$\left.\begin{aligned} \theta += T_s \cdot \omega_{ramp} \\ sen(\theta) \\ cos(\theta) \end{aligned}\right\} \quad \text{Set the angle and trigonometry} \tag{2.29}$$

$$\left.\begin{aligned} V_d = V_o + K \cdot \omega_{ramp} \\ V_q = 0 \end{aligned}\right\} \quad \text{V/Hz control, to set voltage.} \tag{2.30}$$

So far, DC bus voltage has not been taken into account, since it has been considered constant. Now that part of the algorithm is also validated, simulating a variable DC bus acquisition as in the case of automotive power source: where the DC bus range can vary a lot, percent speaking. It is considered one of the main problems in the electric vehicle. It has solutions like that of a BOOST stage, before the inverter, at the expense of bulk and weights. Obviously, when the Vdc drops, the modulators will grow, although their frequency will remain constant, as the voltages Vd and Vq: will remain unchanged! The results of the drop in bus voltage are shown in Figure 27.

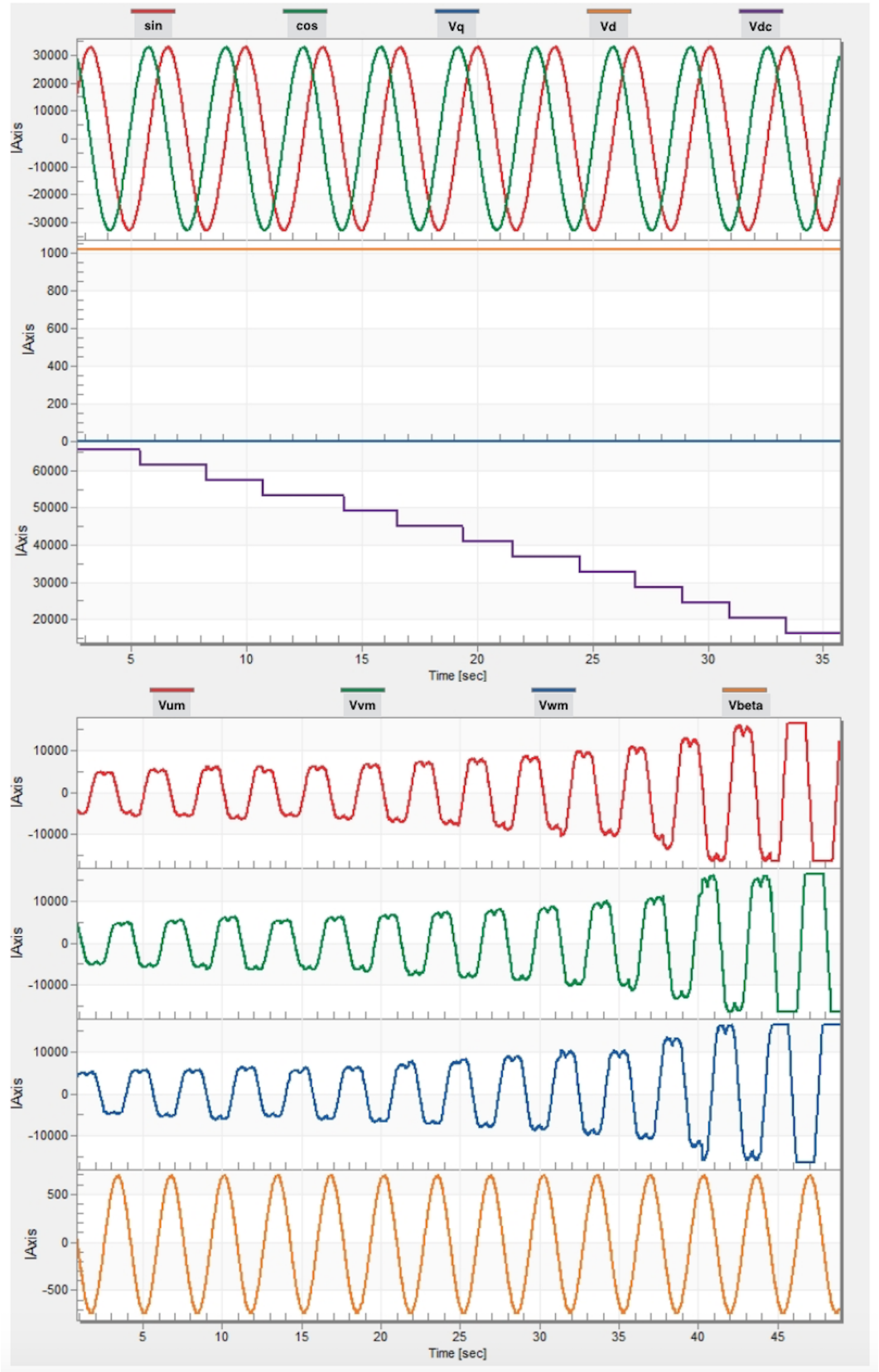

Fig. 2.29 DQ to phase-medium results, stimulated by V/Hz control running in MCU master. Variable DC bus voltage.

2.8 FIL to HIL: road to power stage.

Given the comforting results of the virtual world, the process can continue in its entirety, verifying its effective applicability in the physical world. The purpose to date has been to obtain the modulants for the three legs of inverters. Once generated, the modulants must be implemented through the known process under the name of PWM, literally pulse with modulation. The three phase modulator block, along with IPM manager, has no other task than discretizing a continuous function in order to achieve the same average value. To understand if the pre-existing subsystems interface optimally with the modulator generator, derived from the model, the method is always the same as used in the loop's FPGA paragraph. With only one exception, now debugging the leg controls generated, it will only be obtainable via oscilloscope due to bandwidth. In fact, while the FPGA board will be powered by the usual digital signal controller, the leg commands will only be available in the 15 pin connector called J2 (see schematics in Appendix):

- Phase 1: High pin 1, Low pin 2.
- Phase 2 High pin 4, Low pin 5.
- Phase 3 High pin 6, Low pin 7.

The debug gives the desired results as shown in Figures 30,31 and 32. Underlining the fixed 20kHz carrier frequency and the equivalent Space Vector Modulation label and sector.

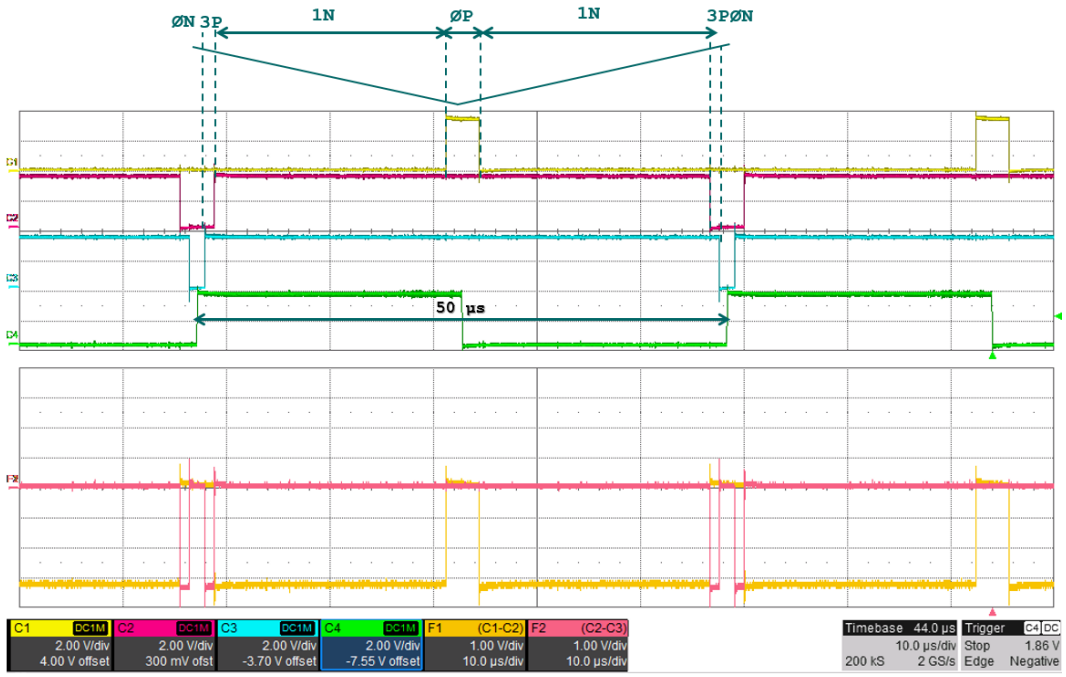

Fig. 2.30 SVM SECTOR 1 - C1: U High, C2: V High, C3: W High, C4: SPI int, F1: differential mode Uh-Vh, F2: differential mode Vh-Wh

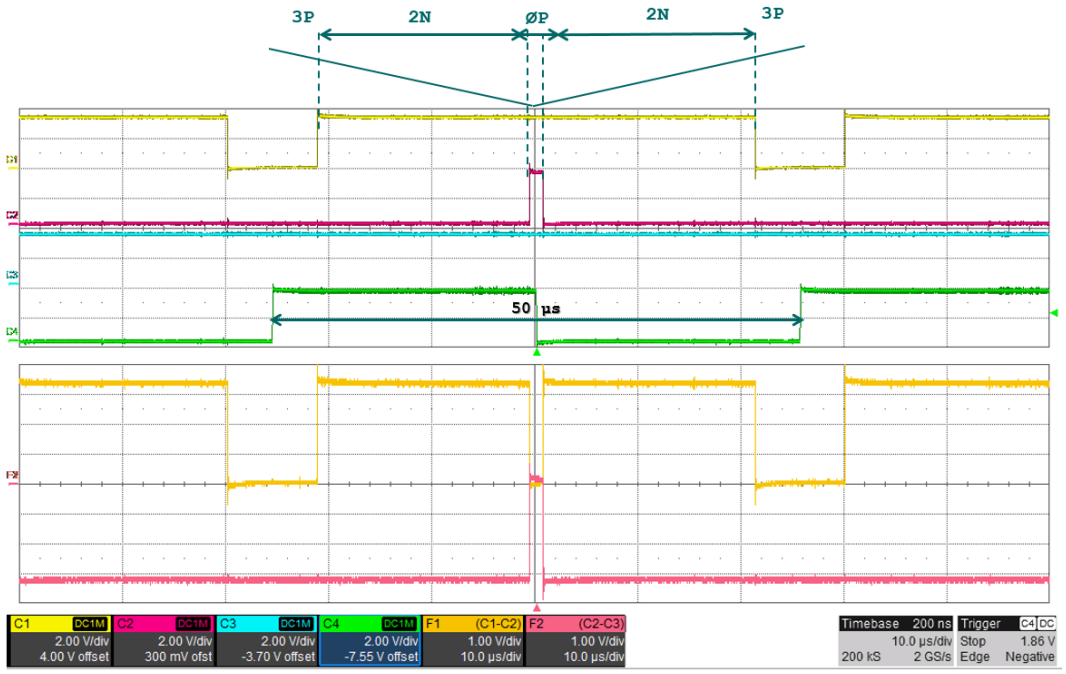

Fig. 2.31 SVM SECTOR 5 - C1: U High, C2: V High, C3: W High, C4: SPI int, F1: differential mode Uh-Vh, F2: differential mode Vh-Wh.

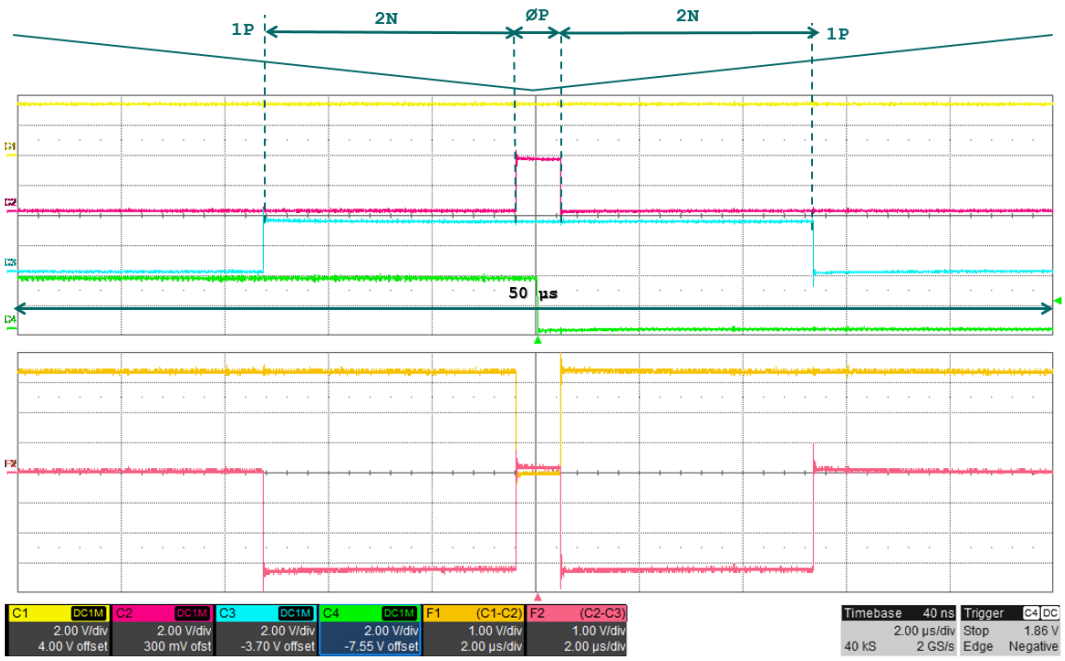

Fig. 2.32 SVM SECTOR 6 - C1: U High, C2: V High, C3: W High, C4: SPI int, F1: differential mode Uh-Vh, F2: differential mode Vh-Wh.

Moreover, dead-times is interesting to observe in figure 33., speech is very much in vogue in literature, especially to reduce it and their compensation in feed forward control. After validating the 6 leg commands, they must be galvanically isolated, via optocouplers, in order to be able to run them through power stage drivers safely.

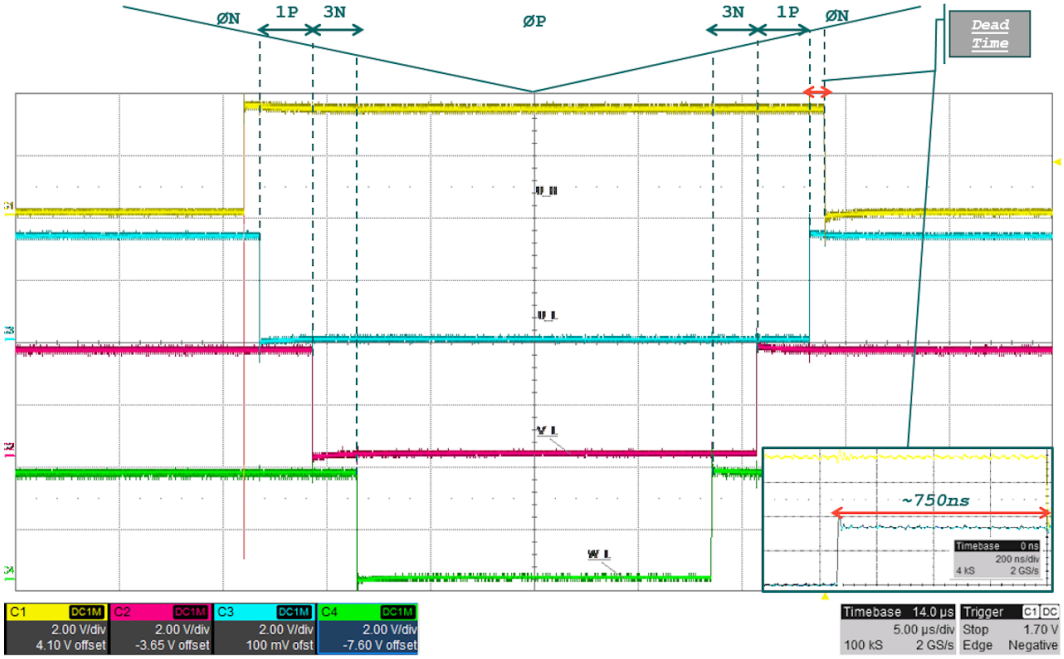

Fig. 2.33 U high (pin1), U Low (pin2), V Low (pin 5), W Low(pin 7).

Chapter 3

pHIL, power hardware in the loop

3.1 Introduction to pHIL.

Since signal validation has met all specifications, it only needs to validate the power part. To do this, the leg controls generated by the FPGA board, through isolation and dedicated drivers for the intelligent power module in question, have to be implemented. For a safety reason, in addition to the leg commands, a feedback status from the power board is sent by the J2 (power-control connector) to activate the faults strategies on the control-board. Moreover, as control with a simple open-loop Volt-Hertz, no electrical feedback is monitored by the sensors. This further increases the need for a good diagnosis strategy to avoid breakdowns. For this reason, the states of the parts in question must be defined, which allow the leg commands to be used:

- 1 Startup.
- 2 Offset
- 3 Stop.
- 4 GO.
- 15 Error.

Basically, the V-Hz control is nothing more than a rotating vector that increases in amplitude and rotation speed without any feedback of the electrical and mechanical magnitudes: that is, in the open loop.

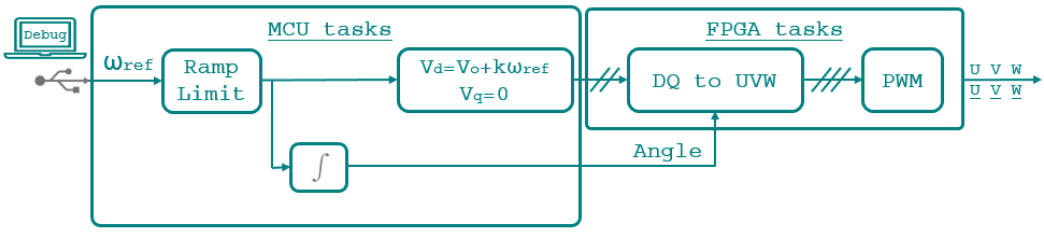

Fig. 3.1 Implemented Volt - Hertz control strategy.

To select the correct speed reference ramp the user starts from slow ramp and measure the currents. The ramp is selected so that the current remains within the limits and it is constant during full flow acceleration. Obviously, control will not start to modulate until both DSP and FPGA status are set to GO. In addition to these conditions on control electronics states, for switching: the power-board signal with SD label (shout down) must also not be active. The power stage validation has been divided into two phases:

- pHIL without load..
- pHIL with induction motor.

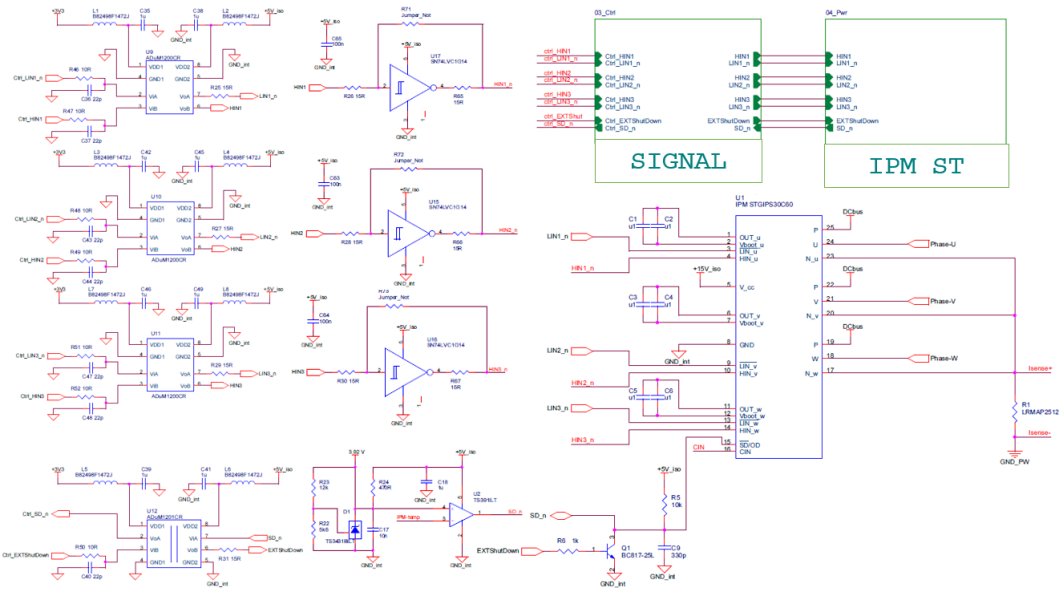

Fig. 3.2 Power stage board schematics.

3.2 pHIL with no load.

3.2.1 Test bench for testing.

- 1 Variable DC power supply for Vbus sets to 30V.
- 1 Variable DC power supply for 12V.
- 1 Variable DC power supply for 5V.
- 3 Differential voltage probes.
- 1 Oscilloscope.
- 1 Laptop with Freemaster.

3.2.2 Results.

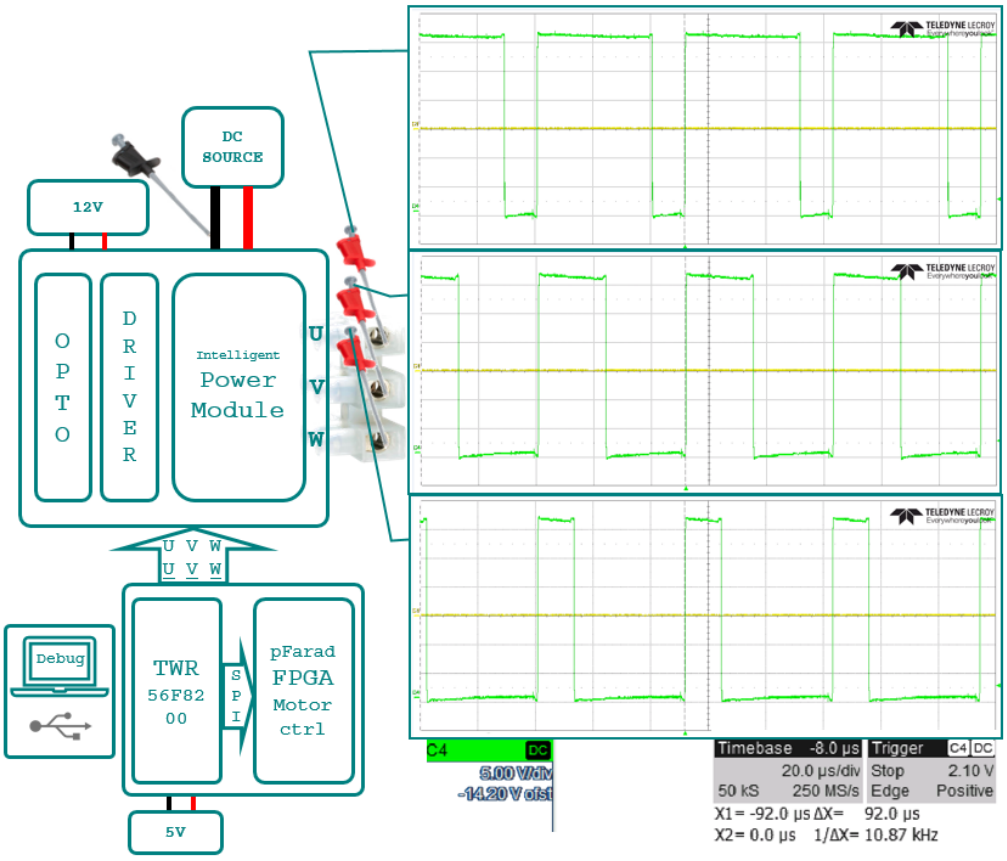

Fig. 3.3 pHIL test bench, 30Vdc without load: three phases legs voltage referred to DC bus.

3.3 pHIL with IM load.

3.3.1 Test bench for testing.

- 2 Variable DC power supply for Vbus sets to 30V, in series.
- 1 Variable DC power supply for 12V.
- 1 Variable DC power supply for 5V.
- 3 Current probes.
- 1 Oscilloscope.
- 1 Laptop with Freemaster.
- 1 3ph Induction motor.

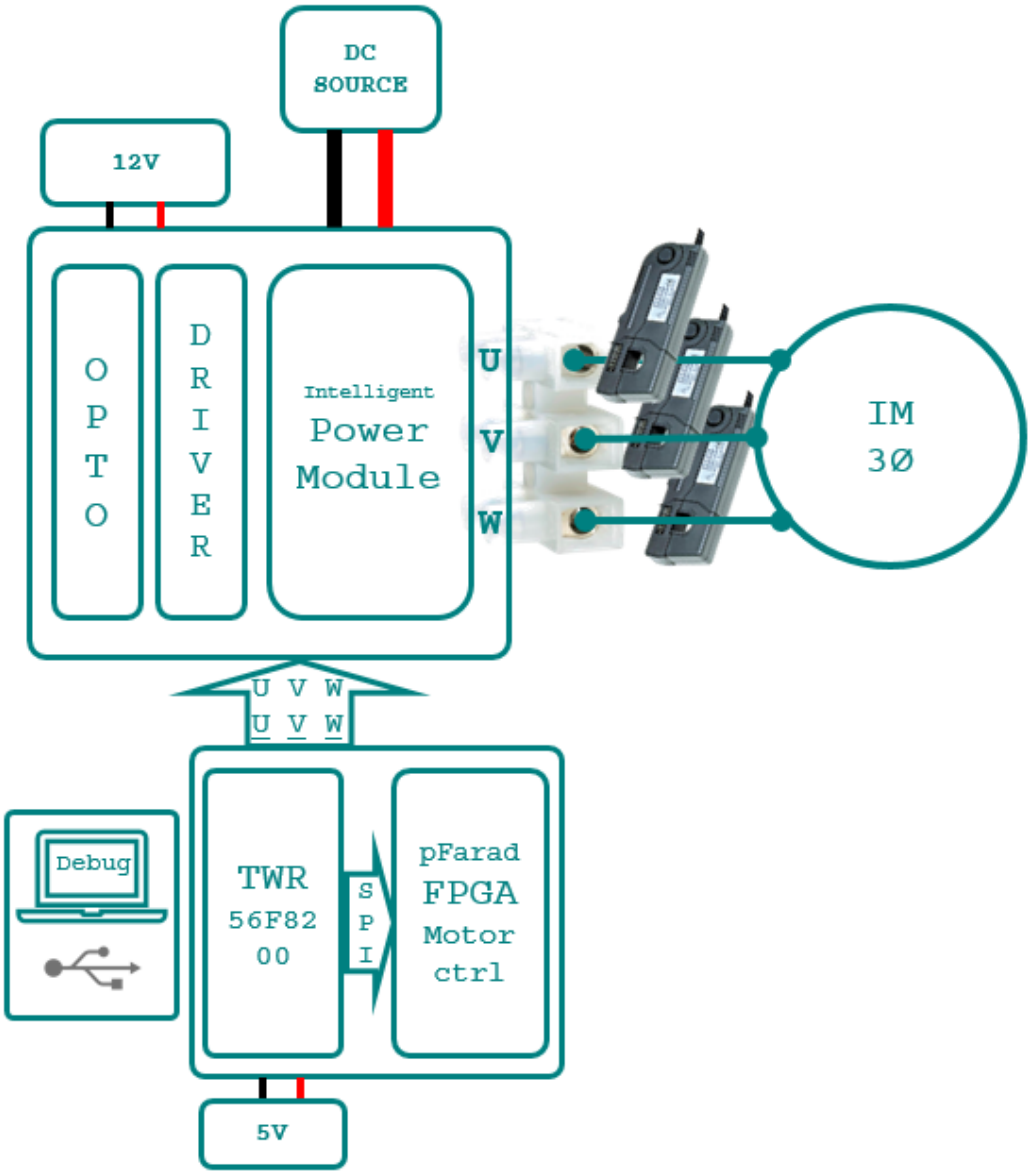

Fig. 3.4 pHIL test bench with load.

3.3.2 Results.

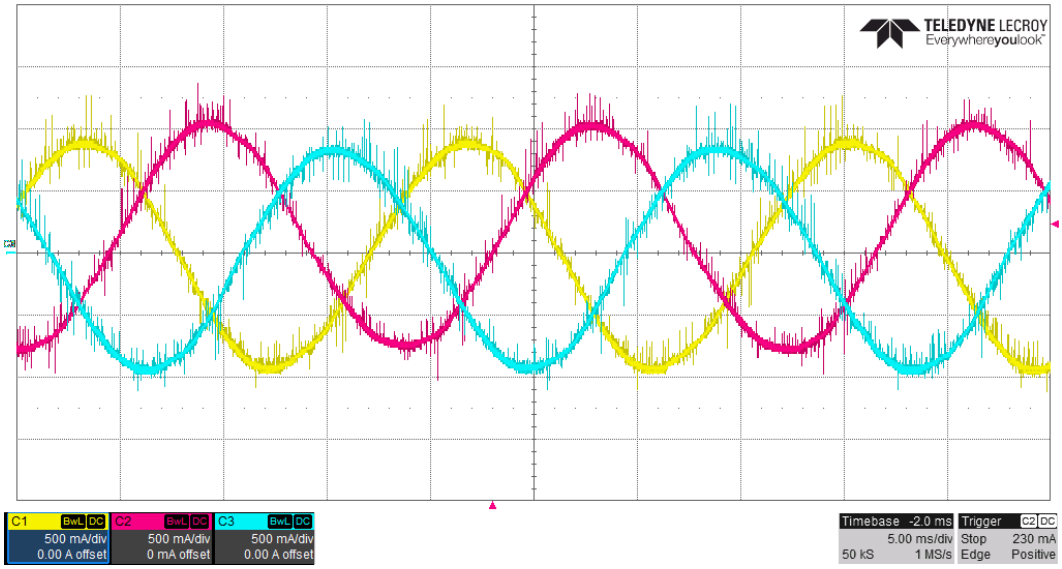

Fig. 3.5 pHIL test bench, Bus 60Vdc, modulation frequency 20kHz, with load and three phases legs currents probes: Ipk-pk 2A at 50Hz, without mechanical load.

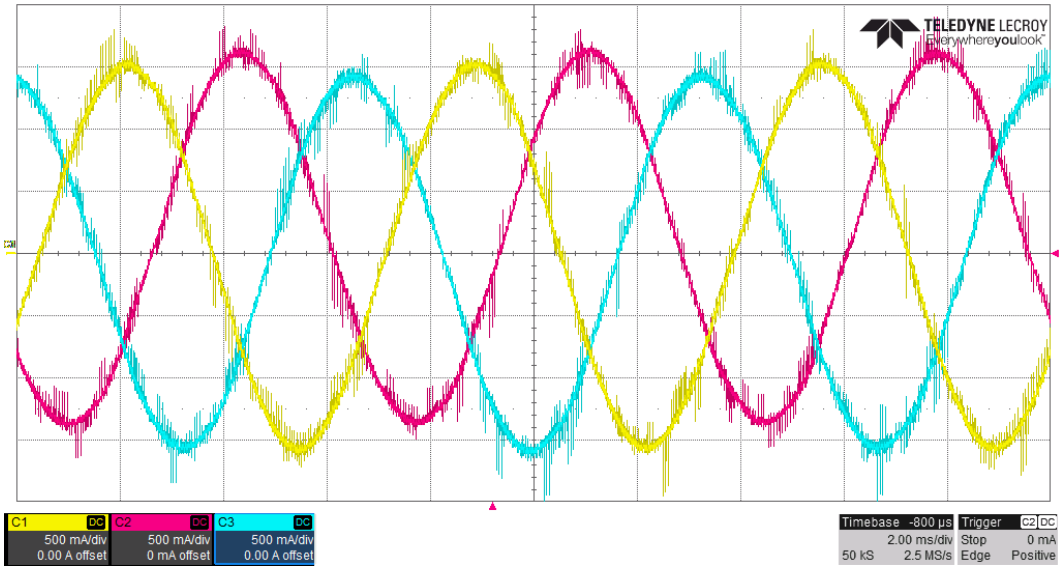

Fig. 3.6 pHIL test bench, Bus 60Vdc, modulation frequency 20kHz, with load and three phases legs currents probes: Ipk-pk 3A at 150Hz, with mechanical load.

3.4 In Closing

The purpose of the elaborate is to give a clear view of the state of the art in the field of product development, in the specific case firmware development. By accompanying the developer from the beginning, starting from a preliminary model, to the actual implementation of the hardware, passing through the generation of model-based code. Through the passage of the various development and validation phases, many errors can be corrected to avoid delays or recalls of the final product. Nowadays, moving from the world of signals (virtual) to real mode, it is possible thanks to the vastness of the commercially available tools - all this safely and quickly. There is a lot of investment in inverter development and validation, as it will be the key element of current and future BEV and PHEV powertrain. The major companies are working to provide increasingly sophisticated equipment for Testing and Developing in order to pass stringent regulations and to guarantee the OEM duty cycle specifications. Of course, further improvements in the process can be obtained by using in pHIL:

- Battery emulator: which allows the setting of derating strategies and chemical features.
- Motor emulator: which allows you to simulate faults in a realistic and safe way.

References

[1] NXP Quick Start Guide, “TWR 56F8200 Tower System Module for MC56F823xx and MC56F827xx”, Freescale.

[2] Mathworks, “HDL Coder Users guide”, Matlab.

[3] Graham Reith, MathWorks, “Design and Verification of FPGA and ASIC Applications,” 2014.

[4] Tabrez Khan and Vidya Viswanathan, “Accelerating FPGA/ASIC Design and Verification,” Matlab Expo 2017.

[5] Xilinx, “System Generator for DSP User Guide,” (v11.4) December 2, 2009.

[6] Auon Akhtar, “An Overview of MATLAB HDL Coder and Xilinx System Generator,” https://www.nutaq.com/matlab-hdl-coder-xilinx-system-generator.

[7] www.mathworks.com/.../tagteam/72320hdl-coder.pdf

[8] “http://www.pe-ip.com/why-fpgas-are-better-than-dsps-for-motor-control/”.

[9] Intel, “Top 7 Reasons to Replace Your Microcontroller with a MAX 10 FPGA”.

[10] Intel, “Five Ways to Build Flexibility into Industrial Applications with FPGAs”.

[11] DOE/EE-0218, "Assessment of High-Performance, Family-Sized Commercial Clothes Washers".

[12] Monmasson E., Cirstea, M., "FPGA Design Methodology for Industrial Control Systems" - A review, IEEE Trans. on Industrial Electronics, Vol. 54, No. 4, August 2007.

[13] Le Roux, W., Harley, R.G., Habetler, T.G., "Detecting faults in rotors of PM drives." Industry Applications Magazine, IEEE, vol.14, no.2, pp.23-31, March-April 2008.

[14] Parker, M., FPGA vs. DSP Design Reliability and Maintenance, Altera white paper, May 2007.

[15] Intel, “FPGAs Provide Programmability and Performance for Next Generation Motor Control”.

[16] Shelley Gretlein, Gerardo Garcia and Joel Sumner, "DSPs, Microprocessors and FPGAs in Control", National Instruments.

[17] Julian Felix Wolfle, Jorg Roth-Stielow, "A hybrid discontinuous modulation technique to influence the switching losses of three phase inverters."

[18] Antonino Fratta, Static conversion lessons.

Appendix A

Schematics FPGA Board.

Appendix B

Model in the loop results.

Appendix C

HDL coder generated code.

Appendix D

HIL .C test-bench.

Appendix E

Schematics ST IPM board.