FPGA Implementation and System-Level Validation of Matrix Converter Space Vector Modulation Algorithm

J. ANDREU, U. BIDARTE, JL. MARTÍN, A. ASTARLOA, J. JIMÉNEZ
Department of Electronics and Telecommunications
University of the Basque Country
School of Engineering, Alameda Urquijo s/n., 48013, BILBAO
SPAIN

Abstract: - Matrix Converter (*MC*) presents several advantages in some power electronic applications, but yet it is not a mature option to be for industrial applications because some problems remain unsolved, for example because theoretical descriptions, still, are not implemented in real designs. This paper shows a Matrix Converter validation platform with the target of passing the control algorithm from *Matlab-Simulink* to *Modelsim* platform. By means of a unique block, the *MC* modulator synthesizes the Double Sided Space Vector Modulation algorithm. This modulator is used to control a Doubly Fed Induction Machine (*DFIM*). The proposed design methodology allows a hardware description of the algorithm to be validated at system-level. This way, once the functionality of the hardware circuit has been verified, the hardware design can be refined to produce synthesizable model that can be implemented in a *FPGA*. This methodology makes possible to achieve a hardware circuit that performs the functionality previously determined in the *Matlab-Simulink* environment.

Key-Words: - Matrix Converter, Space Vector Modulation, FPGA, System on Programmable Chip, Matlab-Simulink, AC/AC.

1 Introduction

The market of power electronics shows a clear tendency towards the following objectives: improving the interaction between power electronic converters and the grid, bidirectional power flow, high efficiency and high switching frequency, reduced size and high integration of complex solutions in a single power module. The Matrix Converter (*MC*) [1] presents an architecture with many of the before mentioned characteristics and overcomes a lot of problems of conventional power converters.

MC is an *AC/AC* converter, with an architecture based on "all silicon" solution with no significant reactive elements. It can operate in high and low pressure environments, and at high temperatures, such as space and submarine applications where the use of electrolytic capacitors is restricted. *MC* is inherently bidirectional and can operate in four quadrants, so it can deliver or take power from the grid [2].

Using appropriate modulation strategies, it is possible to achieve sinusoidal currents at the grid and sinusoidal voltages at the load [3], with a unity power factor with any type of load. On the other hand, *MC* control is a complicated task and has a large computational load. *Matlab-Simulink* is a very useful tool for the development of control algorithms for power converters. To carry out the control of a real prototype it is necessary to go beyond *Simulink* simulations and implement the code in a real integrated circuit. Up to now, *DSP*s have been widely used to embed the power converters control algorithms. Nowadays and, due to the arising of latest generation integrated circuits such as *FPGA*s [4], new possibilities for advanced control have emerged. These chances can be: design time reduction, the partial or total reconfiguration of the algorithm as a function of the control strategy, integration of a spy software to determine the constraints of the algorithm, hardware implementation of the code, design of *FPGA* hardware according to the needs of memory of the algorithm, buses interconnection as a function of the needs of information flow between the different hardware blocks, fast prototyping, etc.

Usually, digital circuits implemented in *FPGA*s are described using Hardware Description Languages like *VHDL* or *Verilog*, while a powerful simulation environment is needed to develop a proper test-bench where the design will be simulated and verified. First, this paper presents how a *MC* Space Vector Modulation (*SVM*) control algorithm has been, previously, described and tuned in *Matlab-Simulink* environment, where a complete model of the whole physical system has also been created. Second, the algorithm has been described in *VHDL* and both design platforms, *Matlab* and *Modelsim*, have been connected by means of the use of common data files. This way, the functionality of the hardware design

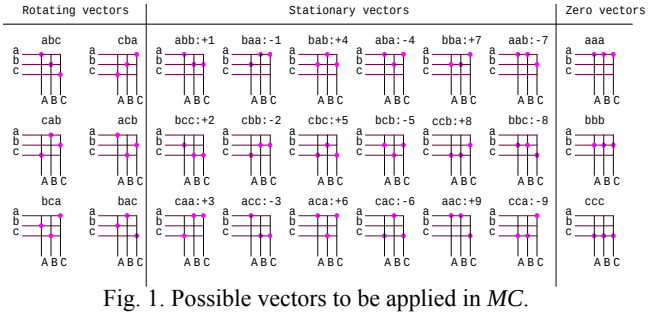

Fig. 1. Possible vectors to be applied in *MC*.

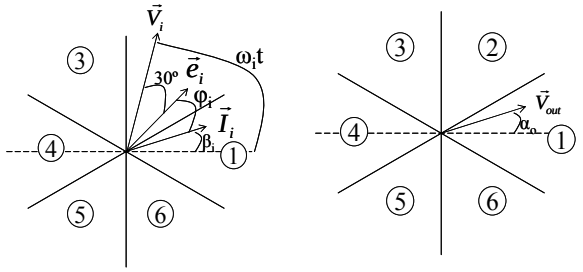

Fig. 2. Synthesizable reference vectors located in the sextant 1: input current I_{in} and output voltage V_{out}.

can be validated at system-level. The first behavioral *VHDL* description, once it is has been validated can now be refined to produce a register-transfer-level (*RTL*) model, from which the layout may be synthesized using commercial *FPGA* Place-and-Route tools.

2 MC Modulation Algorithm

Application of *SVM* [5] to *MC* allows an instantaneous control of the output voltage and the input power factor [6]. Eqs. (1) to (5) show the duty cycles of the applied voltage vectors in order to obtain the desired output voltage vector V_{out}, input current I_{in} and the desired power factor cos φ_i. The duty cycles are assigned to a determined switch combination following fig. 1 and are related with the time that four active vectors and one zero voltage vector are applied each switching interval.

$$duty\_a = \frac{2}{\sqrt{3}} \cdot \frac{v_{out}}{v_{in}} \cdot \frac{\cos(\beta_i - 60) \cdot \cos(\alpha_o - 60)}{\cos\varphi_i} \quad (1)$$

$$duty\_b = \frac{2}{\sqrt{3}} \cdot \frac{v_{out}}{v_{in}} \cdot \frac{\cos(\beta_i + 60) \cdot \cos(\alpha_o - 60)}{\cos\varphi_i} \quad (2)$$

$$duty\_c = \frac{2}{\sqrt{3}} \cdot \frac{v_{out}}{v_{in}} \cdot \frac{\cos(\beta_i - 60) \cdot \cos(\alpha_o + 60)}{\cos\varphi_i} \quad (3)$$

$$duty\_d = \frac{2}{\sqrt{3}} \cdot \frac{v_{out}}{v_{in}} \cdot \frac{\cos(\beta_i + 60) \cdot \cos(\alpha_o + 60)}{\cos\varphi_i} \quad (4)$$

$$duty\_0 = 1 - (duty\_a + duty\_b + duty\_c + duty\_d) \quad (5)$$

The possible switching states of *MC* (fig. 1) can be represented with vectors; in this way, each of the stationary vectors will be placed in the adjacent sides of each sextant of the complex plane (fig. 2). Duty cycles depend on the angles formed between vectors I_{in} and V_{out} and the corresponding bisectriz of the 60º sector where they are located. Vector angle β_i is linked to I_{in}, α_o is linked to V_{out} and φ_i is related with the input displacement factor (fig. 2).

Table 1. Examples of 14 vectors *DS SVM* sequences.

Sector		Double Sided Sequence							
I_{in}	V_{out}								
1	1	0_c	-3	6	0_a	-4	1	0_b	Symmetry
1	2	0_c	-3	9	0_a	-7	1	0_b	
1	6	0_c	-9	6	0_a	-4	7	0_b	
4	1	0_c	-6	3	0_a	-1	4	0_b	Symmetry
4	5	0_c	-3	9	0_a	-7	1	0_b	
4	6	0_c	-6	9	0_a	-7	4	0_b	
6	1	0_a	-4	1	0_b	-2	5	0_c	Symmetry
6	5	0_a	-1	7	0_b	-8	2	0_c	
6	6	0_a	-4	7	0_b	-8	5	0_c	

To obtain the low frequency average values of V_{out} and I_{in}, *SVM* only determines the duty cycles of the applied voltage vectors in a switching interval, but the sequence in which they must be applied is not determined. For this reason, there is a certain degree of freedom that allows implementing more complex techniques, such as Double Sided Modulation (*DS SVM*) [5]. With this technique, the voltage vectors are symmetrically distributed along the switching interval, zero vectors are applied every quarter of switching interval and only one switch state is changed at each step, minimizing losses and improving the quality of the waveforms controlled by the *MC*.

If I_{in} is in the 4th sector and V_{out} in the 5th, the corresponding switching times are: *t_0c, t_c, t_a, t_0a, t_b, t_d, t_0b, t_0b, t_d, t_b, t_0a, t_a, t_c, t_0c*, where:

$$t\_x = \frac{duty\_x}{2} \cdot T_{sw} \qquad x \in \{a,b,c,d\} \quad (6)$$

$$t\_0_y = \frac{duty\_0}{6} \cdot T_{sw} \qquad y \in \{a,b,c\} \quad (7)$$

$$0_y = Vector(yyy) \qquad y \in \{a,b,c\} \quad (8)$$

where T_{sw} is the space vector modulator period ($f_{sw} = 1/T_{sw}$).

Each of the zero vectors: 0_a, 0_b, 0_c (8) are applied for a time interval t_0_y (7), because zero vectors are applied 6 times between the active vectors. In turn, because of the symmetry of *DS SVM*, active vectors are applied during half of their corresponding switching time at each side of the symmetry axis.

Table 1 shows some possible combinations of the 14 vectors necessary for *DS SVM* when the measured input voltage is the line voltage (the rest of sequences are described in [7]). If the input voltage is the phase voltage a new sequence Look-Up table must be obtained.

3 Platform Validation

3.1 Matlab-Simulink Platform

The platform (fig. 3), in which matrix converter *DS SVM* algorithm has been integrated, controls a *DFIM* [8]. It includes the next seven main blocks: Reference Generator, Control Loop, Modulator, *MC*, Grid, Input Filter and *DFIM*.

The modulator algorithm has been synthesized in a single functional block simplifying the implementation and the debugging time of the platform. The algorithm must calculate the duty cycles (1)-(5) in each modulator period T_{sw} and so, during the next period the corresponding vectors of fig. 1 will be applied during the times associated to these duty cycles. The selection of the vectors depends on the sextant (fig. 2) in which I_{in} and V_{out} are located. The vectors sequences are described in table 1 and V_{out} and I_{in} are imposed by the control loop in which the *DFIM* is connected.

The conventional approach in *Matlab-Simulink* is the use of previously defined *Simulink* blocks, which is a very tedious task, because a high number of parameters must be correctly set and there is no proper debug tool. All these problems are overridden with the use of a single "*S- Function*" block (the algorithm is described in detail in [7]). With this method, the user has an entire control of the execution timing of the *SVM* algorithm (which is very important in the modulation technique). Here, *S-Function* block is called with a fixed period (T_{res}) defined by the clock *resolution* of the duty cycles. The *resolution* parameter defines the number of times in which the modulator period T_{sw} is divided. So the *resolution* clock signal (*clk_res*) that defines the high speed clock for the hardware circuit, has got a frequency of f_{res}= f_{sw} **resolution* and a period of T_{res} = T_{sw} /*resolution*. By means of the *S-Function*, the number of simulation blocks is reduced drastically, the simulation speed of the entire platform is faster than with discrete blocks, the implementation of the algorithm is simplified and high level programming language can be used (it can be coded in C or in *Matlab* programming language). The synthesis of the algorithm in an integrated circuit (microcontroller, *DSP* or *FPGA*) is more straightforward, debugging is easier using the *Matlab* incorporated debugger (it is not possible in the other methods) and any additional function can be easily included in the modulation algorithm.

Fig. 4 shows the *Simulink* and *Modelsim* platform validation diagram. The *S-Function* in the *Simulink* platform, on one hand performs the modulation and, on the other hand, writes in each *DS SVM resolution*

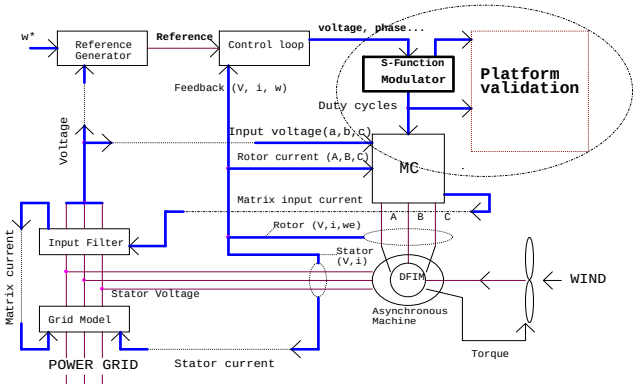

Fig. 3. *Matlab-Simulink* power system model and validation platform.

clock sample period (T_{res}) the input parameters of the *Modelsim* platform. These parameters are written in "*modelsim_in.m*" text file and are: the absolute time of the simulation (*time_abs*), the relative time in each modulation period T_{sw} (*time_Tsw*), the amplitude of V_{in} and V_{out} of the *MC* and, at last, the angles αo and βi (fig. 2) of I_{in} and V_{out} vectors respectively. With these parameters *Modelsim* platform is able to calculate for each instance the corresponding duty cycles. So, these duty cycles will be written in "*modelsim_out.m*" file and they will be validated with the duty cycless calculated in *Simulink* platform, which have been introduced by this platform previously.

3.2 ModelSim Platform

In order to make possible the interaction between the two design platforms (*Matlab-Simulink* and *VHDL* design tools) early in the overall design flow, a top-down design methodology for the hardware design have been followed. The basic idea is to iteratively refine a high level description down to the layout level by systematically developing a hierarchy of *VHDL* models [9].

As a first step, *VHDL* may be used to model and simulate a fully functional description of the system, allowing the *FPGA/ASIC* specification to be validated prior to the detailed design.

This may be a partial description that abstracts certain properties of the system, such as a performance model to detect system performance bottle-necks [10]. This is called system-level verification. The first goal has been to validate a functional (also called behavioural) *VHDL* description. That is to say, we want to verify that the *VHDL* design and the *Matlab* design perform exactly the same algorithm [11]. Fig. 5 shows a very simplified block diagram of the hardware implementation.

In the first *VHDL* functional description the real *VHDL* signal type has been used for digital signals associated to input voltages, currents and vector

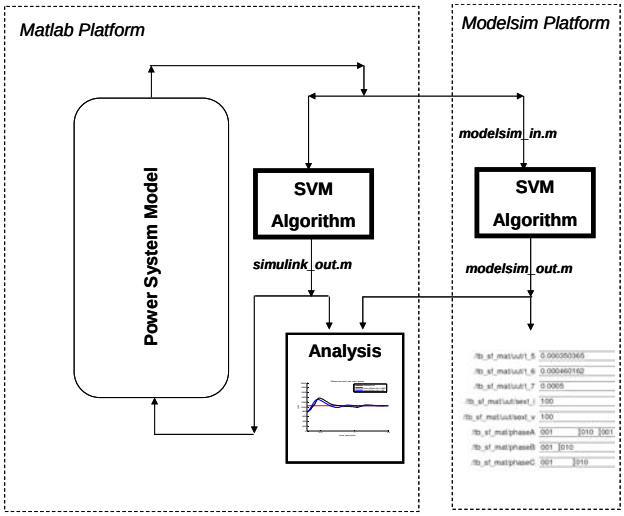

Fig. 4. *Matlab-Simulink* and *Modelsim* validation platform.

angles. The *math_real VHDL* package must be used to manage real type signals, because it contains common real constants and common real functions [4]. Although the real type can not be synthesized by actual *FPGA* software tools, it allows the system-level simulation in order to compare results in the *Modelsim* platform with results in the *Simulink* platform (where real type signals are also used).

Although hardware description refinement is not the aim of this paper, functional blocks must be refined to produce a register-transfer-level (*RTL*) model, from which the layout may be synthesized.

Fig. 6 shows the *VHDL* test environment that has been developed to facilitate the interaction between *Matlab-Simulink* and the hardware design tools. The same *VHDL* platform can be used to verify models at successive levels of refinement, from the first functional model to the synthesizable *RTL* model. It is composed of the next elements:

1. Text files: these files allow the interconnection between both platforms. An input file, "*modelsim_in.m*" in fig. 6, contains simulation input vectors generated in the *Matlab* platform (fig. 4) and an output file, "*modelsim_out.m*" in fig. 6, contains the simulation results, which will be analyzed in the *Matlab* environment (fig. 4).

2. File management and data conversion: it is necessary to manage the files containing simulation input vectors and output results using valid *VHDL* sentences. The standard *textio* package allows complex file management. Fig. 7 includes some of the file management procedures available in this package. Fig. 8 summarizes the *VHDL* process that reads the input values from the "*modelsim_in.m*" file: each *resolution* clock period (T_{res}) a new line containing the values for every input signal is read, then converted to digital signal in order to apply a new simulation vector. On the other side, output

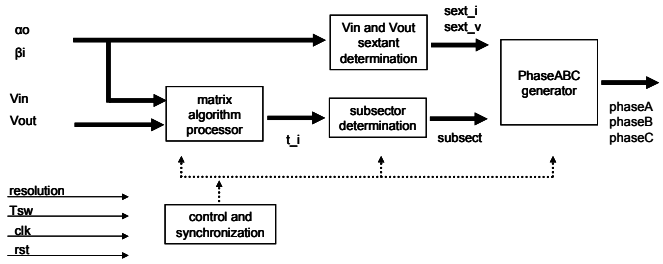

Fig. 5. Hardware design block diagram.

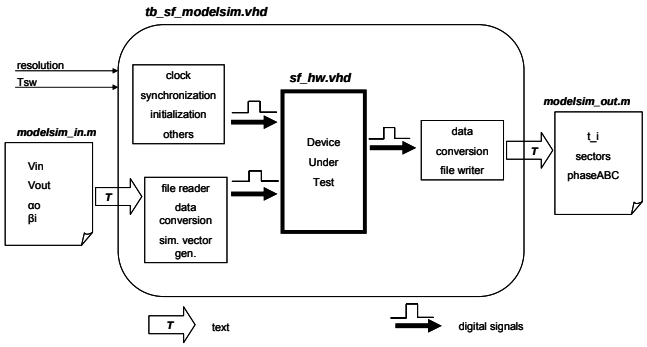

Fig. 6. *VHDL* test environment.

digital signals follow the inverse scheme and are finally stored in "*modelsim_out.m*" file. This file will be analyzed in the *Simulink* platform and compared with the results on that platform.

3. Initialization, synchronization and others: these processes control the simulation initialization and the synchronization between all modules.

4. Device Under Test (DUT): this is the *VHDL* description of the matrix controller that we want to verify. The name in the figure, sf_hw.vhd, means that all the functionality contained in the Matalb *S-Function* block has been included in this *VHDL* hardware design.

Simulation results can be verified by means of the *Modelsim* waveform editor, where usual timing analysis can be performed. Fig. 9 shows a timing diagram that contains the most representative input and output signals. This environment is necessary to study the correctness of the hardware design, but this is not enough. The output results file "*modelsim_out.m*" also contains the simulation results, but with the same file format used in the *Matlab* environment, in order to validate the whole hardware system performance against the same algorithm but described as a *Matlab S-Function*.

3.3 Methodology summary

The main steps of the described *MC* control algorithm validation methodology are summarized in the next lines:

1. The *MC DS SVM* algorithm, implemented in the *S-Function*, is executed sampling the parameters associated to the duty cycles (1)-(5). The sampling

period is determined by the *resolution* clock signal *clk_res*. For each sample, fprintf instruction is executed, this way, the afore mentioned parameters are included in a text file ("*modelsim_in.m*"). At the same time, for each period T_{sw} the calculated duty cycles are written in "*simulink_out.m*". The format of these files is one column per parameter and one file per *resolution* clock period (T_{res}).

2. "*modelsim_in.m*" is read by *Modelsim* environment and then, the same algorithm (now described in *VHDL*) takes as input parameters the data that synthesizes the duty cycles (1)-(5).

As the model of the whole power system is not needed in the *VHDL* environment, the simulation in the *Modelsim* platform is greatly accelerated.

3. The *DS SVM* algorithm is executed in *Modelsim* to generate its duty cycles (besides of other outputs such as sextant of V_{out} and I_{in}, number of the vector in *DS SVM* 14 vectors sequence, activated switches, etc.) which will be written in "*modelsim_out.m*" text file.

4. Finally, the result of both environments (located in "*simulink_out.m*" and "*modelsim_out.m*" files) are analyzed and represented graphically to view if there is any difference between the calculated duty cycles.

4 Results

The design of *Matlab-Simulink* platform is shown in fig. 3. *S-Function* algorithm behavior has been simulated taken into account transitory and stationary conditions of the *DFIM*. The parameter to control by the *Simulink* is the speed of the *DFIM* rotor. Fig. 10 contains the line to line *MC* modulated output voltage and its filtered low frequency component. Fig. 11 shows how the output voltage of the *MC* follows the low frequency reference voltage.

DFIM must accelerate until the reference speed of the machine is reached. Fig. 12 shows the transitory and stationary state of the rotor speed in two different cases. As it can be seen, the profile depends on the modulation period (T_{sw}) and the *resolution* clock period (T_{res}).

It is important to take into account that the *resolution* defines the number of times in which the modulator period is divided and so, it will define the accuracy of the time in which one vector is applied. If the overshoot and the time in which the reference speed is reached are taken into account, it is preferable to apply a greater modulation frequency, even if not all the vectors are applied due to lack of accuracy in the *resolution* clock. In the two cases, the reference speed of the machine is finally reached.

The time needed by the *Simulink* platform to simulate the 1.5 secs. depends on, among others, T_{sw} and T_{res}. If f_{sw} =5*KHz* ($1/T_{sw}$) and *resolution*=100 are used,

```
INPUT FILE MANAGEMENT
procedure READLINE(file modelsim_in.m: TEXT;
L_in: out LINE);

procedure READ(L_in:inout LINE; VALUE_in: out
type);

OUTPUT FILE MANAGEMENT
procedure WRITE(L_out:inout LINE; VALUE_out: out
type);

procedure WRITELINE(file modelsim_out.m: TEXT;
L_out: out LINE);

type = bit_vetor, character, string, integer,
real, etc.
```

Fig. 7. File management resources used from *textio VHDL* package.

```
-- process to read all coefficients in the input
file
load_coe: process(clk)

file coe_file: text open read_mode is
"modelsim_in.m ";
variable curLine_v : line;
variable curLineNum_v : natural := 0;
variable read_ok_v : boolean;
-- declaration of all variables to be read

begin
if (clk'event and clk = '1') then
       if (curLineNum_v <= 100000) and not
       (endfile(coe_file) ) then
       -- Read the current line
       readline(coe_file,curLine_v);
       -- Read all the coefficients in the
       current line
       read(curLine_v,abs_timer,read_ok_v);
       if not read_ok_v then
       report "fircplx_tb: Error reading real
       coefficient from line: "& curLine_v.all
       severity warning;
       ...
       -- assign local variables to glabal
       signals

       else report "File reading end"&
curLine_v.all
       severity failure;
       end if;
end if;
end process load_coe;
```

Fig. 8. File reading process.

the elapsed simulation time is 0.91 hours, meanwhile if f_{sw} =1*KHz* and *resolution*=1000 are used the time will increase to 1.95 hours.

These times are so long because of the large computational load required by the entire platform (*MC*, Control Loop, *DFIM*, etc.). For these simulations, a PC with an Intel Pentium μP, 1 Gbyte RAM and 3.2 *GHz*, has been used.

The time required by *Modelsim* platform is insignificant (compared to the previous times). The reason is that the test-bench developed in *Modelsim* only includes the algorithm processor and it does not need the rest of the blocks because they are replaced by the entries contained in "*modelsim_in.m*" file.

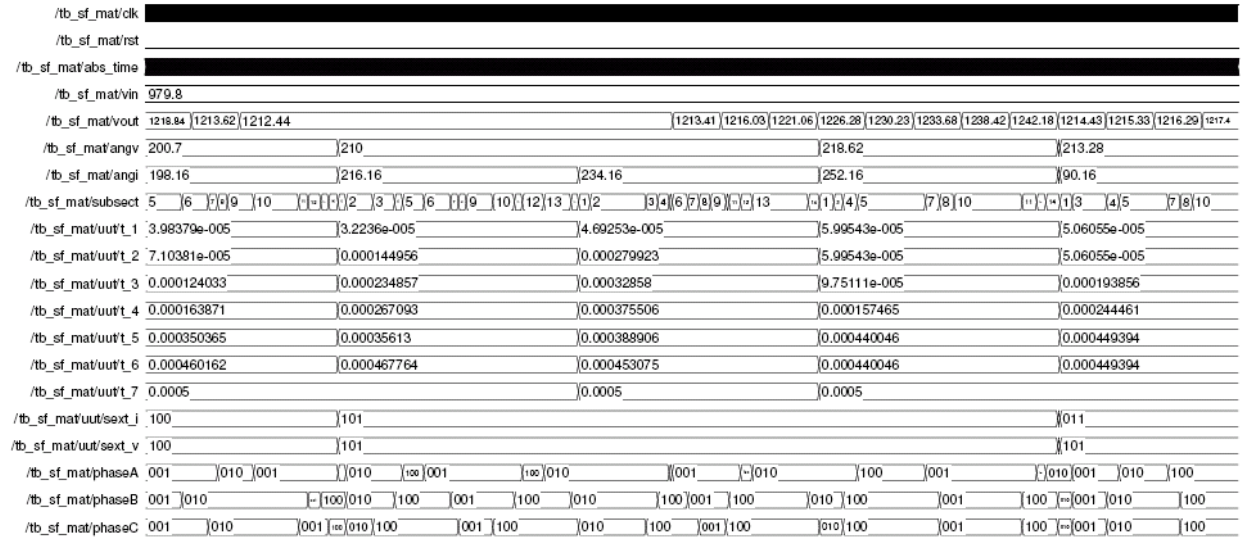

Fig. 9. Example of timing diagram from *Modelsim*.

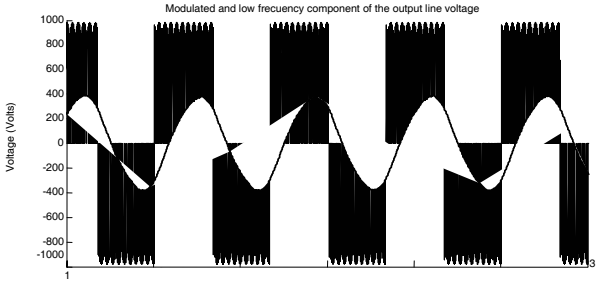

Fig. 10. *MC* line to line output modulated voltage and its filtered low frequency component.

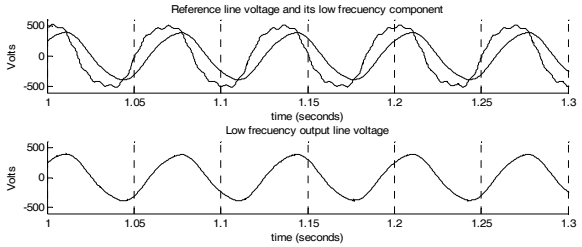

Fig. 11. Reference and its low frequency voltage (upper); low frequency output voltage (lower).

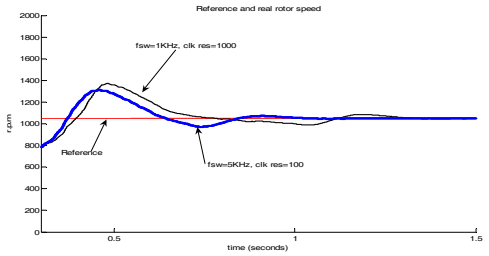

Fig. 12. Reference and real *DFIM* rotor speed.

In this way, this file replaces the control loop, *DFIM*, etc.

In principle, the fact that *DS SVM* calculates the 14 vectors to be applied does not ensure these vectors will be applied. Because of this, a *resolution* clock, with a frequency of $f_{res} >> f_{sw}$, must be used (for example 1000 times higher). So, the algorithm must check if the elapsed time since the beginning of the T_{sw} is included between the time intervals of *DS SVM* sequence. In this way, the vector corresponding with the interval in which the *resolution* clock is sited will be applied. If this timer has not enough *resolution* and the calculated duty cycles are sufficiently small (for example, the vector 3 in fig. 13), the timer will not pass through the interval associated to this vector, and then, it will not be applied. So the average value of V_{out} and I_{in} will not coincide throughout this period with the value of the reference dictated by the control.

Fig. 13 shows comparative results of the duty cycles calculated by each platform, *Simulink* and *Modelsim*. The results of the upper side correspond with T_{sw} = 5*KHz* and *resolution*=100, while the lower figure shows the absolute time in which each vector must be applied (the last data have been calculated employing T_{sw} = 1*KHz* and 1000 samples per modulator period). In each graphic, each pair of columns show the results obtained in each platform (left: *Simulink* and right: *Modelsim*). As it can be seen, the calculated duty cycles and times are equal for both platforms, so the validation of the algorithm codified in *VHDL* is satisfactory. In this way, the task of synthesizing into a *FPGA* can be performed.

5 Conclusions

The power electronics community needs to make an effort to commercialize advanced converters, such as *MC*. In this sense, this article shows a methodology to validate a *MC* control algorithm. Initially *Matlab-Simulink* Space Vector Modulator environment

(which includes *MC*, Control Loop, *DFIM*, etc.) has been described. Then, a validation platform is presented. Its objective is to validate the behavioral description of the same algorithm, but now using a Hardware Description Language (*VHDL*). So, it can now be refined to produce a register-transfer-level (*RTL*) model, from which the layout may be synthesized using commercial *FPGA* Place-and-Route tools.

Besides of implementing safely the *SVM* algorithm in *Modelsim* platform, this method looks for other benefits such as: the use of arising integrated circuits like *FPGA*s (which offer new possibilities like reconfiguration, high speed execution, user defined design, etc.) and contribution to the development of Integrated Power Modules (*IPM*s).

6 Acknowledgement

The present paper has been financed by "Ministerio de Educación y Ciencia" and "FEDER" within the research project ENE 2004-07881-C03-01/ALT.

References:

[1] P. Wheeler, J. Clare, L. Empringham and M. Bland, "Matrix converters: the technology and potential for exploitation," in The Drives and Controls Power Electronics Conference, Vol. 5, 2001.

[2] H. Keyuan; H. Yikang, "Investigation of a matrix converter-excited brushless doubly-fed machine wind-power generation system," Conference on Power Electronics and Drive Systems (PEDS), Vol. 1, Nov. 2003, pp. 743-748.

[3] L. Neft and C. D. Shauder, "Theory and design of a 30-hp matrix converter," IEEE Trans. on Industry Applications, Vol 28, No.3, May / Jun. 1992, pp.546–551.

[4] H. Hajimowlana, "Design verification and debugging FPGA implementations," EDN (US Edition), Vol. 48, No. 25, Nov. 2003, pp. 69-72.

[5] D. Casadei, G. Serra, A. Tani and L. Zarri, "Matrix converter modulation strategies: a new general approach based on space-vector representation of the switch state," IEEE Trans. on Industrial Electronics, Vol. 49, No. 2, April 2002, pp. 370-381.

[6] L. Huber and D. Borojevic, "Space vector modulation with unity input power factor for forced commutated cycloconverters," in Proc. of IEEE Industry Applications Society (IAS), Vol 1, 1991, pp. 1032-1041.

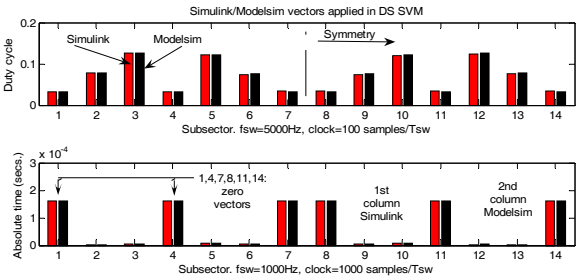

Fig. 13. Comparison of calculated dutys and times in *Simulink* and *Modelsim* platforms.

[7] J.Andreu, I. Mtz. de Alegría, J.L. Martín, S. Ceballos, I. Gabiola, "Matrix Converter Double Sided Space Vector Modulation: a fast way to synthesize via S-Function," in Proc. of the IEEE International Symposium on Industrial Electronics (ISIE), 2006, pp. 779-784.

[8] I. Mtz. de Alegría, J. Andreu, P. Ibañez, JL. Villate, I. Gabiola, "Novel power error vector control for wind turbine doubly fed induction generator," in Proc. of Industrial Electronics, Control and Instrumentation (IECON), Vol. 2, Nov. 2004, pp. 1218-1223.

[9] S. Meiyappo, J. Steele, "VHDL constructs and methodologies for advance design verification," EDN, Vol. 44, Nov. 1999, pp. 65-86.

[10] G. Peyrot, "Behavioral modeling in VHDL simulation," EDN (US Edition), Vol. 44, No. 22, Oct. 1999, pp. 49-66.

[11] M. Benmohamed, S. Menriz, "VHDL specification methodology from high-level specification," Journal of Computer Sciences, Vol. 1, No. 2, Apr-Jun 2005, pp. 270-275.