Simulink Modeling and Design of an Efficient Hardware-constrained FPGA-based PMSM Speed Controller

Bogdan Alecsa, Marcian N. Cirstea, *Senior Member, IEEE*, Alexandru Onea

***Abstract*— The aim of this paper is to present a holistic approach to modeling and FPGA implementation of a permanent magnet synchronous motor (PMSM) speed controller. The whole system is modeled in the Matlab Simulink environment. The controller is then translated to discrete time and remodeled using System Generator blocks, directly synthesizable into FPGA hardware. The algorithm is further refined and factorized to take into account hardware constraints, so as to fit into a low cost FPGA, without significantly increasing the execution time. The resulting controller is then integrated together with sensor interfaces and analysis tools and implemented into an FPGA device. Experimental results validate the controller and verify the design.**

***Index Terms*—FPGA, PMSM, Simulink, System Generator.**

I. Introduction

MODERN FPGA devices offer a multitude of resources, thus moving forward from their original intended utilization: implementing glue logic in complex digital systems. Nowadays, the whole complex digital system can reside into the FPGA, leading to the concept of system on a programmable chip (SoPC). However, the main advantage FPGAs offer is the possibility to implement algorithms directly into hardware, maintaining the parallelism of the algorithm in the implementation and thus minimizing the execution time. Consequently, the FPGA utilization in industrial control applications became the subject of intensive research [1], [2].

There are many approaches regarding both the controller type (ranging from neural networks [3], [4], [5] and fuzzy logic [6], [7] to classical PID (proportional-integral-derivative) based control algorithms) and the implementation (ranging from pure hardware implementations [3] to combined hardware-software [6], [8] or pure software solutions using soft processor intellectual property (IP) cores [1], [9], [10]).

Another key factor, which contributed to the successful adoption of FPGA-based solutions, is the availability of a wide range of design tools [1], [11]. For example, the possibility to design the whole system in Matlab Simulink, at a high level of abstraction, and simulate it with bit and cycle accuracy, offers a high degree of confidence in the “correct first time” operation of the circuit [12].

A short overview of the literature regarding FPGA-based controllers for electric motors will be presented in the next paragraphs, aiming to highlight the space this paper tries to fill in. Although references [12], [11], [13], [14] partly review this domain, newer literature will be considered. In [3], a “holistic” approach is considered for modeling and FPGA implementation of a sensorless controller for the induction motor. Using a state-space observer and a controller based on neural networks, the authors present the modeling of the whole system (including the motor, thus the term “holistic”) using VHDL. After validation through simulation, the controller was experimentally verified. In [15], the algorithm for direct hardware implementation of neural networks is presented in detail.

In [6], the authors propose a speed control system for a PMSM based on hardware/software partitioning: an adaptive speed controller, based on fuzzy logic, was implemented by software running on a NIOS II soft processor, while the current controllers, with faster dynamics, were implemented in hardware. In [7], the same authors propose a hardware implementation of the adaptive fuzzy controller, this time applied to a permanent magnet linear synchronous motor. Another example of hardware/software partitioning is given in [16], where a multi-axis motion controller is implemented on a DSP (digital signal processor)/FPGA platform: the servo control loop (current and position/velocity control) was implemented in hardware on FPGA, while the trajectory generation was done by software on the DSP.

In [17], a sensorless controller for the induction motor is presented, using DTC (Direct Torque Control) and a state observer. The controller was designed in LabView FPGA and implemented on a National Instruments RIO PXI-7831R board, into a Virtex-II family FPGA.

In [18], a two axes motion control system is presented, partitioned between software and hardware: the PMSM current control loops are implemented in hardware, while the speed loops and the trajectory generation are based on software. A similar system is presented in [8], where PID speed controllers for DC motors were implemented as hardware modules, while the multi-axes trajectory generation was performed by software on a MicroBlaze soft processor. It is worth noticing the speed controllers, designed in VHDL, were validated by Simulink-ModelSim co-simulation.

In [19], a simple yet effective control method for BLDC (brushless DC) motors is presented. The speed is controlled by using only 2 values for the PWM duty cycle, leading to a very economic implementation. In [20], the stability of the proposed method is further analyzed.

In [21] and [22], a control system for a PMSM associated with an analog position resolver is presented. Hysteresis current controllers are implemented into the FPGA for the 3 phases of the motor, together with a module for resolver signals processing, based on the CORDIC (coordinate rotation digital computer) algorithm. It is worth noticing the target FPGA (AFS600 Fusion produced by Actel) integrates analog to digital conversion peripherals: ADC (analog to digital converter), analog prescalers, analog multiplexer. The emergence of such chips highlights a move of the FPGA vendors toward the embedded market, dominated by microcontrollers.

In [23], the authors present the design in Simulink, using the DSP Builder software from Altera, of a PMSM control system. The system employs hysteresis current controllers and a PI speed controller and uses 56 of the DSP blocks (9x9 bits wide) of an Altera Stratix II EP2S60F1024C4 FPGA.

In [24], a sensorless controller for a synchronous motor is presented, using PI current controllers and sinusoidal PWM. The rotor speed and position are estimated by using the extended Kalman filter, a very demanding algorithm due to the several matrix multiplication and inversion operations it requires. The algorithm has been optimized and factorized for efficient FPGA implementation, finally occupying 36 hardware multipliers (18x18 bits).

In [25], the authors present a sensorless control system for the PMSM, using high frequency signal injection to estimate the momentary stator inductance. A digital PLL (phase locked loop) is employed for signal processing. Both the PLL and the space vector modulation (SVM) algorithm use CORDIC to compute the needed trigonometric functions.

In [10], a comparison between a hardware implementation and a pure software implementation running on an ARM Cortex-M1 soft processor is presented, for the case of a PMSM hysteresis current controller. Coordinate transforms and a resolver signal processing unit were also implemented.

In [26], a reusable IP cores library for electrical vehicle (EV) propulsion control is presented. The library is organized hierarchically, having at the base an arithmetic unit for matrix-vector multiplication. As a case study, the control of an EV equipped with induction motors is presented.

In [27], a sensorless control system for a PMSM is presented, partitioned between hardware and software: the PI current controllers, the coordinate transforms, the SVM algorithm and the position sliding mode observer were implemented in hardware, while speed estimation and control are performed by software running on a NIOS II processor.

In [28], a control system for an induction motor supplied by an inverter bridge through a resonant circuit is presented. The system is described in AHDL (Altera HDL) and uses 76 hardware multipliers (9x9 bits).

From this short literature review, some conclusions can be drawn: (i) From [6], [8], [18], [27], where only parts of the controllers are implemented in software, results that hardware implementation of algorithms is the obvious choice for high demanding applications; software implementations are preferred for tasks with less stringent computation time constraints, or when reuse of existing code is desirable [9]. Moreover, in [10] a direct comparison is made between hardware and software implementations, highlighting the hardware advantages. In [29], comparison is made between an FPGA based hardware implementation and a DSP based software solution: the hardware solution is 11 times faster than the software, leading to a much higher controller bandwidth. In [1], the comparison is extended to a MicroBlaze soft processor implementation, which is even slower than the DSP implementation. (ii) The vast majority of the reviewed papers (except [3], [23]) lack a holistic modeling of the control system: there is a fracture between the design and simulation of the controller and drive, on one hand, and the design of the FPGA implementation, on the other hand. In [1], a hardware in the loop (HIL) validation step is proposed to fill in this gap. (iii) Most of the reviewed controllers employ a large quantity of the FPGA resources and need a serious revision for implementation in low cost devices, with a limited number of hardware multipliers. Only in [21], [22], [24], [26] are taken steps to apply the AAA (algorithm architecture adequation) optimization strategy, so as to minimize the usage of critical resources. (iv) It can be considered that the state of the art in FPGA hardware design is based on HDL (hardware description languages), as they are employed in most of the reviewed papers. As this design methodology resembles to software development, it has been proven to lead to a similar degree of faults in the implementation [30]. However, modern design tools, like LabView FPGA [17], [31], DSP Builder [23] or System Generator [32], [33] are gaining momentum. It has been proven [32] that System Generator can lead to comparable results in terms of obtained speed as HDL description for complex designs. In [33], both the VHDL and the System Generator designs of an adaptive filter show similar performance in terms of speed and area.

This paper will thus try to fill in the gap found in the existing literature: it proposes a holistic modeling of a permanent magnet synchronous machine (PMSM) speed control system in the Matlab Simulink environment, which allows validation by simulation of the controller model, as well as of the FPGA hardware; it also takes into account severe hardware constraints, leading to a very low cost FPGA implementation. The steps to be followed to get from a continuous time controller model to a discrete, FPGA synthesizable model, based on System Generator blocks, are outlined. The algorithm is refined to fit into a low cost FPGA, keeping its inherent parallelism. The short execution time is of paramount importance in order to use a low cost current measurement scheme. The system is experimentally tested and is proved to work correctly.

II. FPGA Controller Design

A. Motor and Controller Modeling

The PMSM is usually modeled in the rotor synchronous rotating frame (*q*/*d* frame), as this approach eliminates the time-varying inductances from the voltage equations in the stator frame. The motor model, as well as the controller derivation, has been presented in [34]. Suffice it to say here that the speed control system consists of two PI current controllers, a combined PI-P speed controller (double speed feedback loop) [35], an axes decoupling and back electromotive force (BEMF) compensation module, direct and inverse coordinate transforms.

A motor model from the SimPowerSystems Simulink library was used for simulation. The controller was designed in Simulink and the control parameters were validated by simulation. It is worth mentioning that the resulting current closed loop systems are described by first order transfer functions, while the speed closed loop system is described by a second order transfer function.

In reality the motor is fed by a three phase voltage source inverter (VSI). The VSI is controlled using space vector modulation (SVM) and the SVM algorithm is applied directly to the *α*/*β* voltage components. For the purpose of simulation, in a first phase, the SVM algorithm was replaced with an *α*/*β* to *a*/*b*/*c* coordinate transform and the resulting three phase voltages were fed to the motor model.

For the algorithm discretization, the SVM has a significant impact: it sets the maximum sampling rate, which is fixed by the SVM pulse width modulation (PWM) carrier frequency. This, in turn, is limited by the switching characteristics of the VSI power transistors. The carrier frequency was chosen in this case as 20*kHz*, meaning a sampling period $T_s = 50\mu s$.

B. Controller Discretization

After the simulation in "continuous time" has validated the controller, this must be discretized to enable digital implementation. The discrete operations to be performed are shortly described in the following.

The *a*/*b*/*c* to *q*/*d* transform consists of two consecutive transforms, *a*/*b*/*c* to *α*/*β* and *α*/*β* to *q*/*d*:

$$\begin{cases} i_\alpha[k] = \dfrac{1}{3}(2i_a[k] - i_b[k] - i_c[k]); \\ i_\beta[k] = \dfrac{\sqrt{3}}{3}(i_b[k] - i_c[k]), \end{cases} \tag{1}$$

$$\begin{cases} i_q[k] = \cos(\theta_e[k]) i_\alpha[k] + \sin(\theta_e[k]) i_\beta[k]; \\ i_d[k] = \sin(\theta_e[k]) i_\alpha[k] - \cos(\theta_e[k]) i_\beta[k]. \end{cases} \tag{2}$$

The axes decoupling block performs the following operations:

$$\begin{cases} v_q[k] = v_{q\_ref}[k] + \omega_e[k] i_d[k] L_d + \omega_e[k]\lambda_m; \\ v_d[k] = v_{d\_ref}[k] - \omega_e[k] i_q[k] L_q. \end{cases} \tag{3}$$

The *q*/*d* to *α*/*β* transformation is exactly the same as the *α*/*β* to *q*/*d* transformation, but is applied to voltages:

$$\begin{cases} v_\alpha[k] = \cos(\theta_e[k]) v_q[k] + \sin(\theta_e[k]) v_d[k]; \\ v_\beta[k] = \sin(\theta_e[k]) v_q[k] - \cos(\theta_e[k]) v_d[k]. \end{cases} \tag{4}$$

The PI controllers were discretized using the Tustin approximation. Two implementation choices were considered for the PI controllers. One is based on the PI transfer function seen as an infinite impulse response (IIR) filter, described by the equation:

$$u[k] = \left(\frac{K_I T_s}{2} + K_P\right)\varepsilon[k] + \left(\frac{K_I T_s}{2} - K_P\right)\varepsilon[k-1] + u[k-1] \tag{5}$$

where K_I is the integral gain, K_P is the proportional gain, ε is the controller input and *u* is the controller output.

The other implementation considers the P and I components of the controller in parallel. This is expressed as:

$$u[k] = K_P \varepsilon[k] + \frac{K_I T_s}{2}(\varepsilon[k] + \varepsilon[k-1]) + i[k-1] \tag{6}$$

where *i* is the output of the integral part and the same notations as in (5) are kept for the rest of the symbols. Equation (6) has the advantage that it offers the possibility to implement an anti-windup strategy for the integral part. The first term on the right hand side of (6) corresponds to the P part, while the rest corresponds to the I part.

Considering the controller gain factors and sampling period fixed, the coefficients for multiplications in (5) and (6) can be pre-computed. Comparing (5) and (6), it is clear that (5) is easier to implement, as it needs only 2 multiplications and 2 additions. Equation (6) needs 2 multiplications, but 3 additions. If the result of each operation is registered, this also means an additional step in computation. However, integral wind-up can be very inconvenient.

There are two commonly used anti-windup strategies: conditional integration and tracking back calculation. In [36], [37] specific strategies for PI (PID) speed controllers are proposed. However, for the double loop PI-P controller [35] used here, conditional integration with the condition for integration obtained from the second (P) loop shows very good behavior.

The discrete equations were implemented using System Generator blocks and fixed point arithmetic. The 18 bit precision was used for all multiplications (to make use of the 18x18 embedded multipliers of the low cost FPGA), whereas additions were limited to 24 bits. All limitations were applied using additional hardware for saturation on overflow, thus ensuring stable system behavior even when the representation limits are reached. The fixed point position is different in each block, depending on the magnitude of the signals the block works with. This is another great advantage of the FPGA implementation over a DSP (or any other processor) implementation: the computing architecture is not fixed, it can be tailored in any point to accommodate the task at hand.

The design was verified by simulation at this point. The simulation was performed with a sample time of $50\mu s$ for the System Generator blocks. The discrete controller was simulated together with the continuous one. The differences between signals of the continuous model and the ones of the discrete model were computed and analyzed in Simulink, thus validating the algorithm. The errors introduced by the discretization process (quantization errors, errors due to integrators Tustin approximation, errors due to zero order hold outputs) were evaluated and proved acceptable. For example, for a $200 rad/s$ step increase of the speed and a simulation length of $0.05s$, which is enough to reach steady state, the speed root mean square deviation (RMSD) between the continuous and the discrete model is around $0.25 rad/s$ and the q axis current RMSD is around $0.008A$. Simulations with different word lengths were performed. No significant improvement of the RMSD was observed for higher word lengths, so the 18 bits precision was kept. The decrease of the sampling period has a much higher impact on RMDS improvement, but it is not an option for the system discussed: the sampling frequency is limited by the VSI power transistors switching characteristics.

Although the System Generator blocks are directly synthesizable in FPGA hardware, the algorithm can not be implemented in this form for two main reasons: (i) The algorithm would need a $20kHz$ clock, derived from the $50MHz$ system clock, and the results are ensured to be valid synchronously to this clock. Of course, the results are available much earlier, but the rest of the system should take care of valid results reading. Additionally, it is not good design practice to have multiple clock signals, especially if derived by combinational means. (ii) The algorithm in this form would use independent hardware resources for all operations. While this is not an issue for logic or add/subtract operations, it is certainly a problem when limited hardware resources come into account, like multipliers or RAM (Random Access Memory) blocks. Additionally, many operations depend on the result of other operations, so they could use the same hardware sequentially.

For these reasons, the algorithm was factorized and transformed into a sequential automaton driven by the system clock, keeping a high degree of parallelism to ensure a very short execution time.

C. *Control Algorithm Implementation*

Through the factorization process, the algorithm was re-organized to employ only 4 multipliers. The other operations will be performed by dedicated hardware, but multipliers are a scarce resource on low cost FPGAs. Analyzing the algorithm, it was observed that it only needs 4 multipliers, while keeping its inherent parallelism [34].

The problem with sharing the multipliers between several functional blocks resides in the increased complexity of the datapath and the datapath controller. Each input of the four multipliers is fed by a 6 inputs multiplexer. The inputs correspond to the utilizations of the multiplier. The datapath controller performs the multiplexer selection and saves the selected values into the multiplier input registers. The output of each multiplier is distributed to all the functional blocks that use it. The datapath controller ensures the multiplication result is saved into the correct functional block by enabling the corresponding register. The Simulink implementation of the control algorithm is presented in Fig. 1. All the functional blocks were implemented using System Generator components. Only the datapath signals are explicitly shown on the figure, signals flowing from one functional block to the other. There are links between each functional block that needs multiplication operations and the multiplication engine. Also, there are links from the datapath controller to each functional block. These links were implemented using "Goto" and "From" Simulink signal routing blocks.

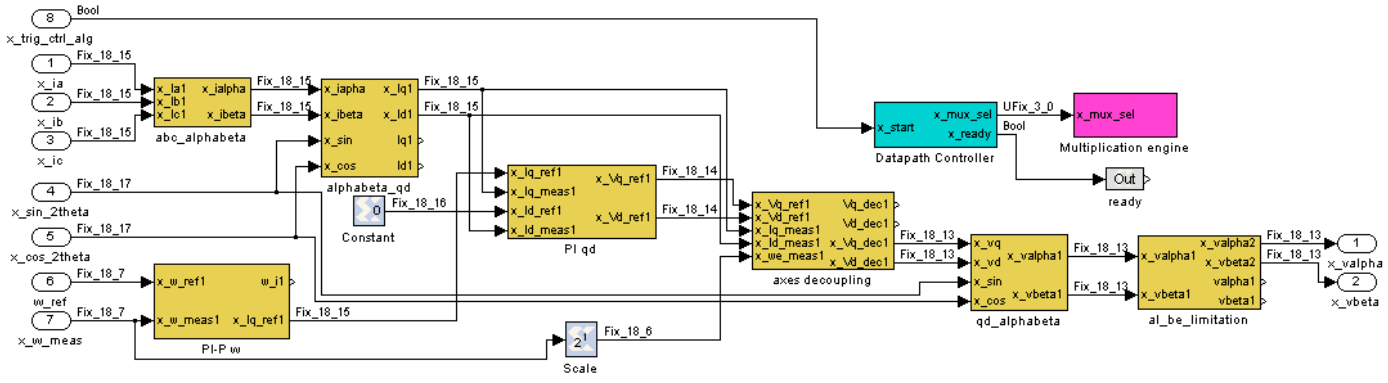

Fig. 1. Simulink implementation of the control algorithm.

For the datapath signals, the number format can also be observed in Fig. 1. The signals between functional blocks are 18 bits wide, represented in signed fixed point format. For example, “Fix_18_15” for signal x_ia means the current in the *a* axis (flowing in the *a* stator coil) is represented as a signed value on 18 bits, using 15 bits for the fractional part and 3 bits for the integer part (including the sign bit). Although, as it will be discussed later on, the currents are measured using 12 bit ADCs, the ADC value must be processed to obtain the current value, and the processing is performed on 18 bits. The multiplier engine multiplexer selection signal, x_mux_sel, is unsigned integer, 3 bits wide: “Ufix_3_0”. The datapath controller trigger signal is a logic signal, 1 bit wide: “Bool”. As already stated, the representation format varies along the datapath according to the value range of the results of various operations. For example, the motor speed may vary from around -750*rad/s* to +750*rad/s*, requiring 11 bits for the integer part, while the motor currents are limited to ±2*A*, needing only 3 bits for the integer part.

The datapath controller is a Moore type finite state machine (FSM). The FSM remains in zero state until the algorithm is triggered. Afterwards, the FSM passes unconditionally from one state to the next, in each state activating one state variable (“one hot” encoding). The FSM sequences in fact the operations in the functional blocks shown in Fig. 1. The FSM has 26 states, so the algorithm needs 26 clock cycles to complete. Fig. 2 presents the FSM state diagram. The functional blocks to which the variables refer in each state are codified in the figure by circles with different fill patterns and listed below. Some overlapping of operations in different functional blocks can be observed, accounting for parallel execution by independent hardware (for example, in state 11, multiplier cells 1 and 2 have been used by the speed PI block and multiplier cells 3 and 4 have been used by the decoupling block, and the multiplication results are saved in different registers in these blocks by the state variable signal). Although “one hot” encoding was used, in each of the overlapping states the state variable was given two names, for clarity, each one corresponding to the usage of the variable (for example, the state variable of state 11 is named PI_P_w_step4 and axes_dec_step2). Besides the state variables, the FSM has an additional 3 bit output, msel, used to select the appropriate input to the multiplier cells. This is the x_mul_sel signal from Fig. 1. The msel value is changed in the next state after it was used, specifically after the save_mul_sel signal was asserted. The save_mul_sel signal is employed to save the input values to the multiplier input registers. All state variable names (except save_mul_sel, which is recurring) reflect their connection to a functional block (trans1 – the *abc* to *αβ* transform, trans2 – the *αβ* to *qd* transform, PI_P_w – the PI-P speed controller, axes_dec – the axes decoupling block, PI_qd – the *q* and *d* axes current controllers, trans3 – the *qd* to *αβ* transform). In the last state, a ready signal is asserted, to signal the algorithm execution has finished.

The datapath controller was implemented using a Moore

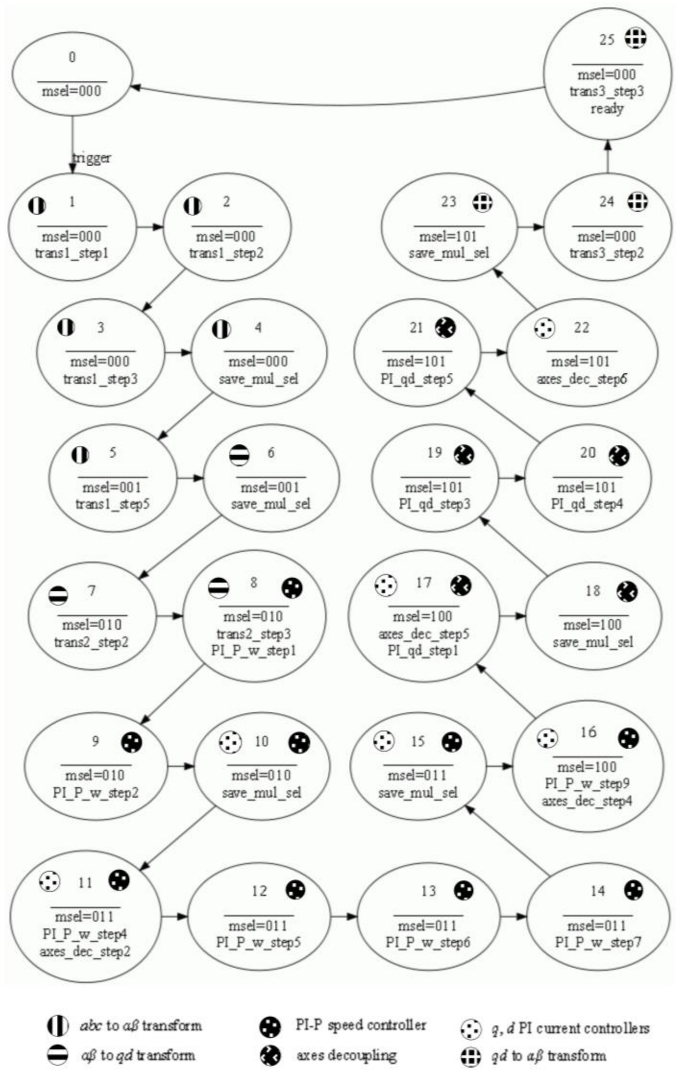

Fig. 2. State diagram of the FSM.

State Machine from the Xilinx Reference Blockset of System Generator. The state machine is described by the transition and output matrixes, which are translated to read-only memories (ROM). For implementation, distributed RAM is employed.

The approach presented here is different from others by the fact that the whole datapath of the algorithm is controlled by the same FSM. While a modular approach with each functional block controlled by its own FSM (as in [14], [22], [24]) may offer more chances for reuse, it eliminates the possibility of execution overlapping. As it can be seen in Fig. 2, this overlapping can be significant (more than 20% of the FSM states).

Fig. 3 presents the internal structure of the PI-P speed controller block, as an example. The links to multiplier cells 1 and 2 can be observed, as well as the sequencing registers and the datapath controller signals. The upper part of Fig. 3 presents the PI controller, implementing (6). The error input signal is calculated from the reference speed and the measured speed. It is then multiplied by K_P. At the same time, it is added to the previous sample time error and multiplied by $K_I T_s / 2$. This result is then conditionally added to the

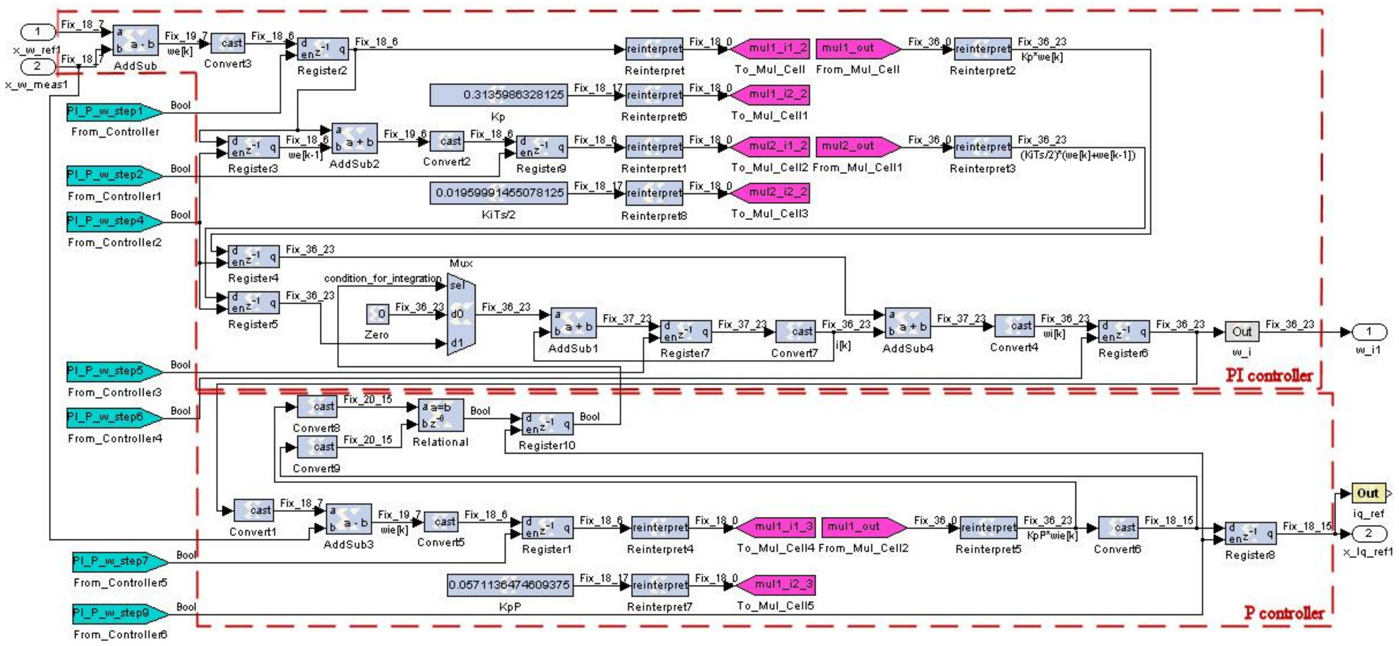

Fig. 3. The PI-P speed controller with anti-windup mechanism.

previous value of the integral part of the controller. At the end, the resulted integral part and the proportional part are added together, giving an intermediate speed value to be used by the second P controller. For multiplication, the multiplier cells 1 and 2 are used. Reinterpret blocks are used to make the number representation transparent for the multiplier cells, because different utilizations employ different formats. At the input in the multiplier cell, the fixed point is forced to 0, and at the output is replaced in the right position.

The lower data flow in Fig. 3 is the P controller. It computes the intermediate speed error, using the measured speed and the output of the PI controller, then multiplies it by the gain factor and saves it to the Reg8 register. The registered value is limited (with saturation) to 18 bits. In case saturation is reached at this point (the unsaturated and the saturated values do not match), the integration in the PI controller is deactivated, to avoid windup. Because the feedback for the conditional integration comes from the second loop, the behavior of the whole speed controller is much improved (as will be shown in the experimental results). The P controller uses also the multiplier cell number 1. All the operations are sequenced by the datapath controller signals, shown on the left side of Fig. 3. The same principle is used for all the functional blocks in Fig. 1.

The current PI controllers were implemented using (5), because it is easier to implement. As the output values of these controllers are saturated on overflow, the saturation acts also as an anti-windup mechanism: the command is limited to the maximum value allowed by the format. This strategy is not usable for the speed controller, because the number format allows values much larger than the maximum obtainable speed.

D. *Space Vector Modulation*

As already stated, the voltage is applied to the motor using SVM. A geometric version of the SVM algorithm was used, which needs only simple comparisons and 3 sets of formulae to compute the PWM threshold values (it does not need trigonometric functions). The algorithm, presented in [38], was redesigned to use only 2 multipliers. It is a “5 step” version of the SVM, making use of only one of the null vectors, the 000 vector. Although this puts more stress on the low side transistors of the VSI, it has the great advantage it prolongs the period in which the stator currents can be measured. This is due to the fact that a low cost current measurement scheme was employed, measuring in fact the voltage drop over shunt resistors connected in series to the low side VSI transistors. In order to get correct current measurement, at the instant of measurement, the current flowing in the low side transistors must be the current flowing in the stator coils. That is, the

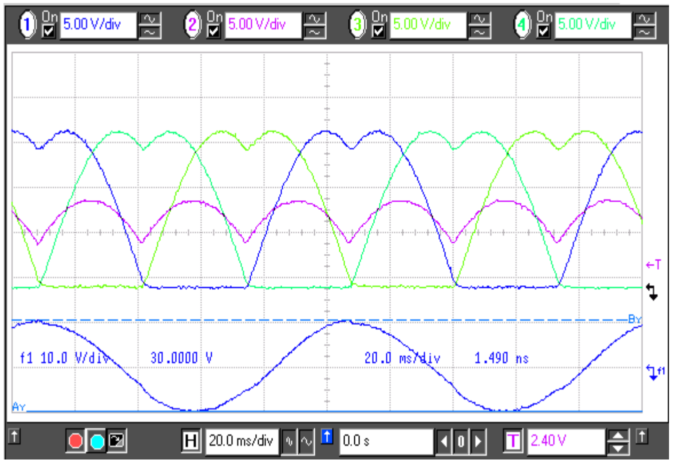

Fig. 4. VSI filtered signals obtained by SVM.

current measurement must be synchronized with the 000 null vector application by the SVM.

The SVM algorithm was designed as an independent module, also by the datapath-controller paradigm. It needs 10 clock cycles to complete. The modulator was tested stand-alone, fed with sinusoidal voltages. The VSI outputs were filtered by passive low pass filters and observed using an oscilloscope. Fig. 4 presents an oscilloscope capture with the VSI filtered signals: the 3 phases and the neutral point. The obtained characteristic signals are equivalent to sinusoidal signals with minimum magnitude signal injected to the neutral point [39].

The PWM circuit used by SVM was designed to provide 15 bits resolution (1.3*ns* time resolution) using the phase shift possibility of the digital clock managers (DCM) present on the FPGA [40]. This way, the controller is not in danger of limit cycling, as the resolution of the command signals is higher than the resolution of the feedback signals (the ADCs have 12 bits resolution).

III. Experimental Results

The presented design was synthesized and implemented in a low cost XC3S500E Spartan-3E FPGA, produced by Xilinx. Besides the control algorithm and sensors interfacing hardware, a ChipScope virtual logic analyzer core was inserted in the FPGA and used to capture internal signals. The ChipScope core is driven by a 20*kHz* clock signal. This way, the sensor interface was debugged and the controllers were verified. The controller was tested with a 19.1V 3441 Pittman PMSM, fed by a PM50 Technosoft three phase inverter. The motor characteristics are given in Table I. It is a low power motor, but has the advantage of a very low inertia. Thus, it makes a good case study for a high bandwidth controller.

The control system is synchronous, with a system clock of 50*MHz*. As previously stated, the control algorithm execution needs 26 clock cycles. The SVM algorithm takes 10 clock cycles to complete, while the current acquisition using the ADCS7476 device (ADC with serial interface) needs 74 clock cycles for data transfer. So the whole execution takes 110 clock cycles, meaning 2.2μs at 50*MHz* clock rate. The high computation speed is essential for the system to be able to acquire the currents in the same sample period in which it derives the command signal. For the current loop experimental verification, refer to [34].

Fig. 5 and Fig. 6 present speed signal captures from the experiments using the Chipscope core. This way, a comparison can be made between the controller without anti-windup mechanism and the one with anti-windup mechanism, for a step reference change on speed from 0*rad/s* to 400*rad/s*. In both cases, the rise time is set by the motor dynamics (the torque to inertia ratio) to about 10*ms*. In Fig. 5, the overshoot due to integral windup is significant. However, the response in Fig. 6 exhibits no overshoot whatsoever.

The presented controller occupies only 10 of the 20 FPGA embedded multipliers, 36% of the logic resources and 1 RAM block. So a much cheaper FPGA device could be used, or a higher resolution in computation can be achieved.

TABLE I
Pittman 3441 PMSM Parameters

Symbol	Meaning	Value and units
r_s	Stator resistance	2.625Ω
L_q	*q* axis equivalent stator inductance	$0.00046H$
L_d	*d* axis equivalent stator inductance	$0.00046H$
λ_m	Voltage constant	$2.62V/krpm$
J	Moment of inertia	$9.9\cdot10^{-7}\,kg\cdot m^2$
F	Friction factor	$0.175\cdot10^{-6}\,N\cdot m\cdot s$
P	Number of pole pairs	2

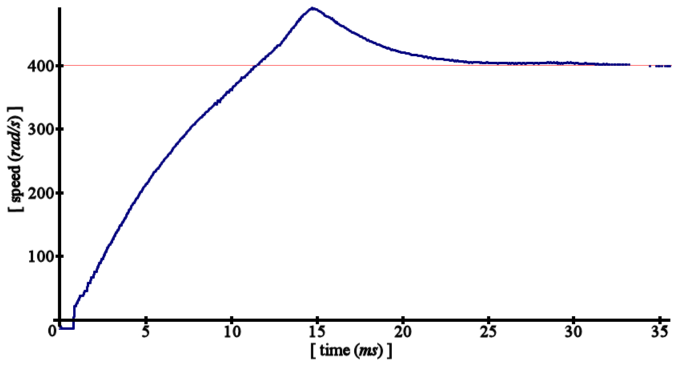

Fig. 5. Speed controller step response (without anti-windup mechanism).

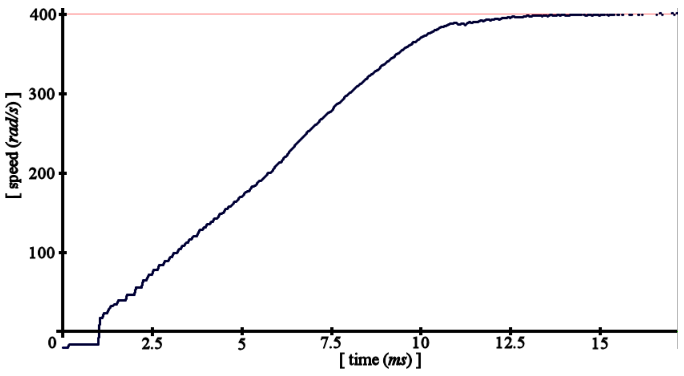

Fig. 6. Speed controller step response (with anti-windup mechanism).

IV. Conclusion

A new holistic modeling of an FPGA speed controller for PMSM was presented, using Matlab Simulink and System Generator. The approach presented allows the modeling of the controller and the controlled system in the same environment, leading to a real time FPGA implementation. A clear methodology for controller design in System Generator was proposed, and the steps followed in order to obtain a synchronous factorized design from a first iteration are presented.

The key achievements are related to the effective use of the on-chip hardware multipliers, by the original design of the control algorithm to match the hardware resources, keeping its inherent parallelism. Thus, high speed of control signal processing is possible. Further contributions are related to the integration of the sensor interfaces and logic analyzer tools together with the controller, those enabling the holistic hardware verification of the system.

The controller was implemented in a low cost FPGA and was able to execute in only a fraction of the sample period (the whole execution takes 110 clock cycles, meaning 2.2μs at 50MHz clock rate), thus enabling a cost effective current measurement scheme. An efficient anti-windup strategy was also defined, allowing effective motor control limited only by the mechanical part dynamics.

Experimental results have proven the correct operation of the controller, thus validating the viability of the design method. One drawback of the design method is given by the simulation requirements: after algorithm factorization and redesign using the datapath/controller paradigm, it must be simulated in Simulink with a fixed step, given by the clock period. Specifically, the simulation must be performed with a step of 20ns. This leads to a very costly simulation in terms of processing time and required memory on the host computer. However, the advantage of validation by simulation is significant. Other validation techniques suffer from the same drawback: in HIL, even though the simulation is performed on the FPGA, the input and output data must be generated by and analyzed on a host computer, the time to transfer the huge amount of data being comparable to the simulation time for the proposed method.

It is expected that this methodology can be adapted for future use to a range of drive systems. A Simulink library based on System Generator will be created, containing ready to use configurable modules for drives control, as current/speed/position controllers, SVM modules (different zero vector allocation schemes), sinusoidal PWM modules (different zero sequence signal injection schemes). Also, future work will target computationally more intensive control algorithms, like predictive controllers [41], [42], that will take full advantage of the execution speed-up by parallelization that FPGAs can offer.

REFERENCES

[1] E. Monmasson, L. Idkhajine, M.N. Cirstea, I. Bahri, A. Tisan, M.W. Naouar, "FPGAs in industrial control applications", *IEEE Transactions on Industrial Informatics*, vol. 7, no. 2, May 2011, pp. 224-243.

[2] E. Monmasson, L. Idkhajine, M.W. Naouar, "FPGA-based controllers", *IEEE Industrial Electronics Magazine*, vol. 5, no. 1, Mar. 2011.

[3] M.N. Cirstea, A. Dinu, "A VHDL holistic modeling approach and FPGA implementation of a digital sensorless induction motor control scheme", *IEEE Transactions on Industrial Electronics*, vol. 54, no. 4, Aug. 2007.

[4] D. Zhang, H. Li, "A stochastic-based FPGA controller for an induction motor drive with integrated neural network algorithms", *IEEE Transactions on Industrial Electronics*, vol. 55, no. 2, Feb. 2008.

[5] N.Q. Le, J.-W. Jeon, "Neural-network-based low-speed-damping controller for stepper motor with an FPGA", *IEEE Transactions on Industrial Electronics*, vol. 57, no. 9, Sep. 2010.

[6] Y.-S. Kung, M.-H. Tsai, "FPGA-based speed control IC for PMSM drive with adaptive fuzzy control", *IEEE Transactions on Power Electronics*, vol. 22, no. 6, Nov. 2007.

[7] Y.-S. Kung, C.-C. Huang, M.-H. Tsai, "FPGA realization of an adaptive fuzzy controller for PMLSM drive", *IEEE Transactions on Industrial Electronics*, vol. 56, no. 8, Aug. 2009.

[8] A. Astarloa, J. Lazaro, U. Bidarte, J. Jimenez, A. Zuloaga, "FPGA technology for multi-axis control systems", *Mechatronics*, vol. 19, no. 2, Mar. 2009.

[9] A. Das, K. Banerjee, "Fast prototyping of a digital PID controller on a FPGA based soft-core microcontroller for precision control of a brushed DC servo motor", *Proceedings of the 35th Annual Conference of the IEEE Industrial Electronics Society, IECON 2009*, Porto, Portugal, Nov. 2009.

[10] I. Bahri, E. Monmasson, F. Verdier, M.E.-A. Ben Khelifa, "SoPC-based current controller for permanent magnet synchronous machines drive", *Proceedings of the 2010 IEEE International Symposium on Industrial Electronics, ISIE 2010*, Bari, Italy, July 2010.

[11] J.J. Rodriguez-Andina, M.J. Moure, M.D. Valdes, "Features, design tools and application domains of FPGAs", *IEEE Transactions on Industrial Electronics*, vol. 54, no. 4, Aug. 2007.

[12] E. Monmasson, M.N. Cirstea, "FPGA design methodology for industrial control systems – a review", *IEEE Transactions on Industrial Electronics*, vol. 54, no. 4, Aug. 2007.

[13] R. Dubey, P. Agarwal, M.K. Vasantha, "Programmable logic devices for motion control – a review", *IEEE Transactions on Industrial Electronics*, vol. 54, no. 1, Feb. 2007.

[14] M.-W. Naouar, E. Monmasson, A.A. Naassani, I. Slama-Belkhodja, N. Patin, "FPGA-based current controllers for AC machine drives – a review", *IEEE Transactions on Industrial Electronics*, vol. 54, no. 4, Aug. 2007.

[15] A. Dinu, M.N. Cirstea, S.E. Cirstea, "Direct neural-network hardware-implementation algorithm", *IEEE Transactions on Industrial Electronics*, vol. 57, no. 5, May 2010.

[16] X. Shao, D. Sun, "Development of a new robot controller architecture with FPGA-based IC design for improved high-speed performance", *IEEE Transactions on Industrial Informatics*, vol. 3, no. 4, Nov. 2007.

[17] J. Lis, C.T. Kowalski, T. Orlowska-Kovalska, "Sensorless DTC control of the induction motor using FPGA", *Proceedings of the 2008 IEEE International Symposium on Industrial Electronics, ISIE2008*, Cambridge, UK, June-July 2008.

[18] Y.-S. Kung, R.-F. Fung, T.-Y. Tai, "Realization of a motion control IC for x-y table based on novel FPGA technology", *IEEE Transactions on Industrial Electronics*, vol. 56, no. 1, Jan. 2009.

[19] A. Sathyan, N. Milivojevic, Y.-J. Lee, M. Krishnamurthy, A. Emadi, "An FPGA-based novel digital PWM control scheme for BLDC motor drives", *IEEE Transactions on Industrial Electronics*, vol. 56, no. 8, Aug. 2009.

[20] N. Milivojevic, M. Krishnamurthy, Y. Gurkaynak, A. Sathyan, Y.-J. Lee, A. Emadi, "Stability analysis of FPGA-based control of brushless DC motors and generators using digital PWM technique", *IEEE Transactions on Industrial Electronics*, vol. 59, no. 1, Jan. 2012.

[21] L. Idkhajine, A. Prata, E. Monmasson, M.-W. Naouar, "System on chip controller for electrical actuator", *Proceedings of the 2008 IEEE International Symposium on Industrial Electronics, ISIE2008*, Cambridge, UK, June-July 2008.

[22] L. Idkhajine, E. Monmasson, M.W. Naouar, A. Prata, K. Bouallaga, "Fully integrated FPGA-based controller for synchronous motor drive", *IEEE Transactions on Industrial Electronics*, vol. 56, no. 10, Oct. 2009.

[23] S. Carbone, V. Delli Colli, R. Di Stefano, G. Figalli, F. Marignetti, "Design and implementation of high performance FPGA control for permanent magnet synchronous motor", *Proceedings of the 35th Annual Conference of the IEEE Industrial Electronics Society, IECON2009*, Porto, Portugal, Nov. 2009.

[24] L. Idkhajine, E. Monmasson, A. Maalouf, "Fully FPGA-based sensorless control for AC drive using an Extended Kalman Filter", *Proceedings of the 35th Annual Conference of the IEEE Industrial Electronics Society, IECON 2009*, Porto, Portugal, Nov. 2009.

[25] G. Maragliano, M. Marchesoni, L. Vaccaro, "FPGA Implementation of a Sensorless PMSM Drive Control Algorithm Based on Algebraic Method", *Proceedings of the 2010 IEEE International Symposium on Industrial Electronics, ISIE 2010*, Bari, Italy, July 2010.

[26] R. de Castro, R.E. Araujo, D. Feitas, "Reusable IP cores library for EV propulsion systems", *Proceedings of the 2010 IEEE International Symposium on Industrial Electronics, ISIE 2010*, Bari, Italy, July 2010.

[27] Y.-S. Kung, C.-C. Huang, L.-C. Huang, “FPGA-realization of a sensorless speed control IC for IPMSM drive”, *Proceedings of the 36th Annual Conference of the IEEE Industrial Electronics Society, IECON 2010*, Phoenix, AZ, USA, Nov. 2010.

[28] J. Kedarisetti, P. Mutschler, “FPGA based control of quasi resonant DC-link inverter and induction motor drive”, *Proceedings of the 2011 IEEE International Symposium on Industrial Electronics, ISIE 2011*, Gdansk, Poland, June 2011.

[29] L. Idkhajine, E. Monmasson, A. Maalouf, “Extended Kalman filter for AC drive sensorless speed controller - FPGA-based solution or DSP-based solution”, *Proceedings of the 2010 IEEE International Symposium on Industrial Electronics, ISIE 2010*, Bari, Italy, July 2010.

[30] F. Salewski, S. Kowalewski, “Hardware/software design considerations for automotive embedded systems”, *IEEE Transactions on Industrial Informatics*, vol. 4, no. 3, Aug. 2008.

[31] T. Orlowska-Kovalska, M. Kaminski, “FPGA implementation of the multilayer neural network for the speed estimation of the two-mass drive system”, *IEEE Transactions on Industrial Informatics*, vol. 7, no. 3, Aug. 2011.

[32] L. Costas, P. Colodron, J.J. Rodriguez-Andina, J. Farina, M.-Y. Chow, “Analysis of two FPGA design methodologies applied to an image processing system”, *Proceedings of the 2010 IEEE International Symposium on Industrial Electronics, ISIE 2010*, Bari, Italy, July 2010.

[33] A. Rosado-Munoz, M. Bataller-Mompean, E. Soria-Olivas, C. Scarante, J.F. Guerrero-Martinez, “FPGA implementation of an adaptive filter robust to impulsive noise: two approaches”, *IEEE Transactions on Industrial Electronics*, vol. 58, no. 3, Mar. 2011.

[34] Blind, “Holistic modeling and FPGA implementation of a PMSM speed controller”, *Proceedings of the 37th Annual Conference of the IEEE Industrial Electronics Society, IECON 2011*, Melbourne, Australia, Nov. 2011.

[35] W. Naouar, A. Naasani, E. Monmasson, I. Slama Belkhodja, “FPGA-based speed control of synchronous machine using a P-PI controller”, *Proceedings of 2006 IEEE International Symposium on Industrial Electronics, ISIE’06*, Montreal, Canada, July 2006.

[36] J.-W. Choi, S.-C. Lee, “Antiwindup strategy for PI-type speed controller”, *IEEE Transactions on Industrial Electronics*, vol. 56, no. 6, June 2009.

[37] H.-B. Shin, J.-G. Park, “Anti-widup PID controller with integral state predictor for variable-speed motor drives”, *IEEE Transactions on Industrial Electronics*, vol. 59, no. 3, Mar. 2012.

[38] Blind, “An optimized FPGA implementation of the modified space vector modulation algorithm for AC drives control”, *Proceedings of 21st International Conference on Field Programmable Logic and Applications, FPL2011*, Chania, Greece, Sep. 2011.

[39] K. Zhou, D. Wang, “Relationship between space-vector modulation and three-phase carrier-based PWM: a comprehensive analysis”, *IEEE Tansactions on Industrial Electronics*, vol. 49, no. 1, Feb 2002.

[40] Blind, “High resolution 6 channels pulse width modulator for FPGA-based AC motor control”, *Proceedings of 2011 International Conference on Applied Electronics, AE2011*, Pilsen, Czech Republic, Sep. 2011.

[41] H. Liu, S. Li, “Speed control for PMSM servo system using predictive functional control and extended state observer”, *IEEE Transactions on Industrial Electronics*, vol. 59, no. 2, Feb. 2012.

[42] J. Rodriguez, R.M. Kennel, J.R. Espinoza, M. Trincado, C.A. Silva, C.A. Rojas, “High-performance control strategies for electrical drives: an experimental assessment”, *IEEE Transactions on Industrial Electronics*, vol. 59, no. 2, Feb. 2012.