Embedded real-time simulators for electromechanical and power electronic systems using system-on-chip devices

Daniel Tormo, Lahoucine Idkhajine, Eric Monmasson, Ricardo Vidal-Albalate, Ramon Blasco-Gimenez

Embedded Real-Time Simulators for Electromechanical and Power Electronic Systems Using System-On-Chip Devices☆

Daniel. Tormo[a,*], Lahoucine. Idkhajine[a], Eric. Monmasson[a], Ricardo. Vidal-Albalate[b], Ramón. Blasco-Gimenez[c]

[a] *Laboratoire SATIE, Université de Cergy-Pontoise, 95031 Cergy-Pontoise, France*
[b] *Department of Industrial Systems Engineering and Design, Univeristat Jaume I, Castelló de la Plana, Spain*
[c] *Department of Systems Engineering and Control, Universitat Politècnica de València, 46022 València, Spain*

Abstract

This paper studies the suitability of System-on-Chip (SoC) devices to perform Embedded Real-Time Simulators (eRTS) of electrical systems. These new devices combine powerful ARM processors, a Field-Programmable Gate Array (FPGA) and other peripherals which make them very convenient for controlling, monitoring and emulating a complete electrical system. There are two main categories of elements with their own respective characteristics and constraints: electromechanical/electromagnetic systems, and switching elements. Accordingly, the proposed investigation will focus on a Doubly-Fed Induction Generator (DFIG) on one hand, and on a Modular Multi-level Converter (MMC) on the other hand. In addition to Real-Time (RT) simulation results, SoC time/resources evaluations will be presented for different implementation strategies: full-software, full-hardware based on High-Level Synthesis (HLS) tools, and using 64/32-bit floating- and fixed-point formats. The validity of the MMC implementation will be tested experimentally.

Keywords: Embedded Real-Time Simulator (eRTS), System-on-Chip (SoC), Fiel-Programmable Gate Array (FPGA), Modular Multi-level Converter (MMC), Doubly-Fed Induction Generator (DFIG), High-Level Synthesis (HLS)

☆The authors would like to thank Mr Miguel Albero and Mr Raúl Morillas for their help on building up the test bench to obtain the experimental results. Also thank the support of the Spanish Ministry of Economy and EU FEDER Funds under grants DPI2014-53245-R and DPI2017-84503-R.

*Corresponding author
Email addresses: daniel.tormo@u-cergy.fr (D. Tormo), lahoucine.idkhajine@u-cergy.fr (L. Idkhajine), eric.monmasson@u-cergy.fr (E. Monmasson), rvidal@uji.es (R. Vidal-Albalate), rblasco@upv.es (R. Blasco-Gimenez)

1. Introduction

Recent industrial controllers of electrical systems are becoming more sophisticated and demanding year after year [1, 2]. In addition to standard control strategies, these emerging controllers are integrating new functionalities such as Embedded Real-Time Simulators (eRTS). The latter can be advantageously deployed within the control closed-loop as estimators, observers, applied for diagnosis and health-monitoring, or also for sensorless and fault-tolerant control purposes [3, 4]. Moreover, these eRTS can be applied in the context of Hardware-In-the-Loop (HIL) so as to test virtually the system controller before its deployment on the real plant, avoiding the well-known experimental damage risks [3, 5, 6, 7, 8]. Then, when implementing such model-based eRTS, the main issue of interest is how to balance between the accuracy of the model, the system dynamics, the simulation time step (implicitly the execution time) and the needed mathematical computations to be executed on the chosen digital platform.

To face these implementation issues, a wide range of powerful devices are now available in the market. Among them, recent System-on-Chip (SoC) platforms are surely the most promising since they bring new design paradigm thanks to their heterogeneity and their computational power [9]. Heterogeneity because they include in the same chip hard processor cores, an FPGA fabric associated to several peripherals like analog/digital converters; and computational power thanks to the high level of available hardware resources and of course computational speed [10, 11]. Hence, the design of an eRTS can be achieved either with a full-software implementation using the processor cores, with a full-hardware implementation using the FPGA logic, or with a mixed software/hardware implementation by distributing the treatment between the two parts [12].

In addition to this technological evolution, the design tools have also been subject of important progress so as to make the development cycle more manageable. This is specifically true when dealing with full-hardware FPGA-based implementation where the limit is sometimes related to the VHDL programming [13, 14]. An example of design tool that bring a solution to this point is the *Vivado High-Level Synthesis* (HLS) provided by Xilinx [15]. Indeed, this tool generates a full-hardware design directly from a C/C++ or SystemC source code. Then, by inserting directives and constraints to the source code, this tool can produce different hardware implementations (full-combinatorial design, pipelined design, parallel or factorized design, etc.).

Having in mind all these potential features, the proposed paper can be seen as an extension of works [12, 16] and its aim is to evaluate these SoC platforms through the development of complex eRTS. The *Xilinx's Zynq-7000 All Programmable SoC* device has been chosen and two case studies have been maintained: the eRTS of a Doubly Fed Induction Generator (DFIG) and the one of a Modular Multi-Level converter (MMC). The idea is to evaluate the computational power, the resources utilization and of course the precision of the results. For the first study, since the DFIG has low/medium system dynamics (electrical and mechanical ones), both a full-software implementation using solely

the PS and full-hardware implementation using HLS to program the FPGA will be evaluated with different design optimizations and data formats (64/32-bit floating-point, 32-bit fixed-point). However, because of its harsh dynamics (switching dynamics), the MMC will be implemented only with a full-hardware approach coded using as well HLS tools. For the MMC application the fidelity of the eRTS will be tested experimentally.

The paper is then organized as follows: Section 2 gives more details about the chosen *Zynq* platform and its associated HLS tool used to program the FPGA fabric. The adopted evaluation methodology is also recalled. Section 3 and 4 are respectively dedicated to the DFIG and the MMC case study, followed by conclusions drawn in Section 5.

2. eRTS platform description and methodology

2.1. The Zynq

Recently, *Xilinx Inc.* provides a new reconfigurable device called *Zynq-7000 All Programmable SoC*, which integrates the programmability of an ARM-based hardwired processing system (PS) with the hardware flexibility of an FPGA [10]. Additionally, other useful peripherals like Analog-to-Digital Converters (ADC), external Direct Memory Access (DMA) controllers, Floating-Point Units (FPU), and I/O peripherals and interfaces are integrated in the same device as well. All of them are interconnected using *AXI* point-to-point channels for communicating addresses, data, and response transactions between master and slave clients [17]. All the system components can be accessed using physical addresses except for the CPU cores and their *L1* instruction caches. The memory map of the *Zynq* device defined in [18] indicates the address range of each logic block.

The *Dual-Core ARM® Cortex™-A9* processor comes with two embedded hardware accelerators: The *NEON* SIMD (Single Instruction, Multiple Data) and the *VFPv3* which is a Floating-Point Unit (FPU). The *NEON* technology is a 128-bit SIMD architecture extension designed initially to provide flexible and powerful acceleration for consumer multimedia applications [19]. It has 32 registers, 64-bits wide (dual view as 16 registers, 128-bits wide), which are shared with the FPU. It can process multiple operations of the same type (e.g. multiplications, divisions, subtractions or additions) simultaneously on different registers, which increase considerably the performance of an algorithm. However, 64-bit floating-point operations are not supported but 32-bit are. Regarding the *VFPv3* FPU, it provides hardware support for floating-point operations in 16-, 32-, and 64-bit arithmetic. It supports basic instructions such as addition, subtraction, multiplication, division, multiplication-addition, and square root [20]. The use of these hardware accelerators is done automatically by setting the compiler optimization to *-O3* and adding the corresponding compiler options stated in [21].

2.2. HLS

The traditional way of programming an FPGA is by using Hardware Description Languages like VHDL or Verilog to describe the connections between

the different internal components of the Programmable Logic (PL). However, in this paper an HLS tool is used which translates the function coded in C/C++ or SystemC into Register Transfer Level (RTL) and then, it is synthesized in order to program the FPGA fabric. The *Vivado HLS* software provided by *Xilinx Inc.* includes several libraries giving support for arbitrary precision data types, data streaming, math, linear algebra, or data signal processing among others [15].

This platform allows the use of test bench files written in C or C++ which eases significantly the debugging and verification of the proper functioning of the whole algorithm, avoiding the necessity of creating complicated HDL test bench files commonly used in hardware designs.

Moreover, the C synthesis process can be controlled through optimization directives which allow to generate different hardware implementations from the same source code. Among the endless possibilities these directives provide there are some which are more likely to be used, like *pipelining* the data paths, *inlining* external functions, reduce BRAM usage by *resizing* and *rearranging* arrays, or *unrolling* loops to execute them in parallel and reduce the total latency. A good study of different HLS tools available in academia and the market can be found in [22].

Once the algorithm is finished and ready to be tested, an Intellectual Property (IP) is generated which will be added to the hardware design using *Vivado IP Integrator*. Then, instantiating a *Zynq* processing system, *Vivado Design Suite* preconfigures the platform with the correct peripherals, drivers, and memory map to use both software and hardware IPs like the ones just developed using HLS.

2.3. Implementation

When implementing an eRTS in a heterogeneous system like the *Zynq*, and considering that the models are coded in C/C++, one can think of different approaches:

- A software implementation using the ARM. The mathematical model is a function that is called from within the main program.
- A hardware implementation using HLS to program the FPGA fabric using C/C++, which allows to modify easily the size and format of the variables in order to study different implementations in either fixed- as floating-point.
- A combination of both where some parts can run in software and some others in hardware. This is known as hardware accelerators and for some applications they can help reducing the total algorithm execution time. When this option is going to be implemented, a shared memory space has to be used in order to send data between the PS and the PL. This approach was covered in [12] and evaluates two different implementations, one using BRAM and the other using the *On-Chip Memory* (OCM).

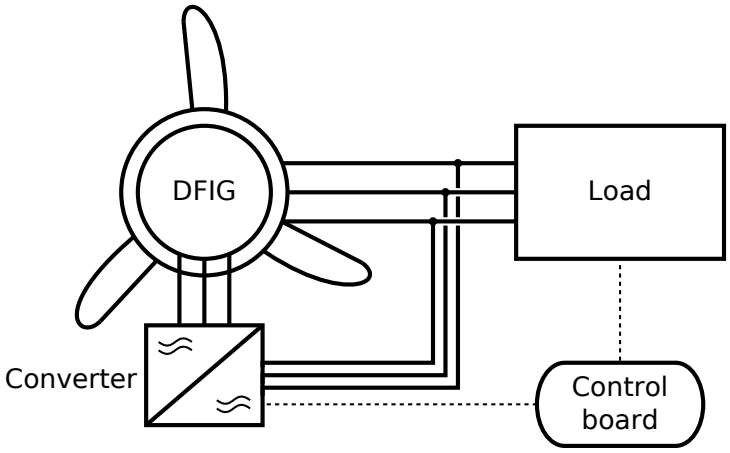

Figure 1: DFIG connected to an isolated load

Considering the characteristics of the two systems studied in this paper, different scenarios are used to evaluate the capabilities of the SoC when implementing the corresponding eRTS. In Section 3, the case of an electromechanical system is explained, whereas in Section 4, the case of a power electronics system is elucidated.

3. eRTS for electromechanical systems

In this section, the capabilities of the *Zynq-7020* mounted in the *ZedBoard* are evaluated and analysed through the implementation of a DFIG. This evaluation board is an economical platform for developing controllers broadly used in academia and industry. Regarding the machine, the decision has been taken because it is a quite popular generator used for windgeneration and it has a representative complexity compared to other electromechanical systems. Two implementation approaches have been adopted: (i) a full-software ARM-based implementation, and (ii) a full-hardware one using HLS. The device performance is evaluated in terms of execution time, resources utilisation and precision of the calculations.

3.1. The DFIG

A schematic of the complete system where the generator under study is used can be seen in Figure 1. It is composed of the DFIG, a power converter, and a three-phase *RL* load. This is a typical setup for an islanded generator feeding an isolated load [23, 24].

The dynamic behavior of a DFIG can be described using equations 1 to 4. Though, these equations have to be converted to the *dq* reference frame [25] before being programmed on the *Zynq*. Some simplifications are adopted like neglecting the saturation, hysteresis and iron losses, and not considering the

temperature effect on the windings. More information about the model, the *dq* reference frame equations, and the discretization method utilised can be found in previous works of the same authors [16].

$$v_s = R_s i_s + \frac{d}{dt}\varphi_s + j\omega_s \varphi_s \quad (1)$$

$$v_r = R_r i_r + \frac{d}{dt}\varphi_r + j\omega_r \varphi_r \quad (2)$$

$$\varphi_s = L_s i_s + M_{sr} i_r \quad (3)$$

$$\varphi_r = M_{sr} i_s + L_r i_r \quad (4)$$

It has to be mentioned that only the DFIG *dq* model is evaluated in this study. Therefore, neither the converter, nor the load, nor the *dq* reference frame conversions are taken into account. The inputs of the model are v_{sd}, v_{sq}, v_{rd}, v_{rq}, and T_m; and their outputs i_{sd}, i_{sq}, i_{rd}, i_{rq}, and T_e.

3.2. DFIG implementation

Four different solutions will be evaluated which can be grouped into two:

- **Full-Software:** One version featuring 64-bit floating-point variables, and another one featuring 32-bit floating-point variables. It has to be pointed out that this is the easiest and most straightforward implementation. It is supposed that all the system modules are implemented in the PS and hence, all the system variables are stored in the same memory space which facilitates and reduces the implementation time considerably.
- **Full-hardware:** In this case, the two versions are one using 32-bit floating-point variables and the other 32-bit fixed-point variables. The design and development of these versions comprise managing variables and communications between the different system blocks, which can be more time consuming than the computation itself if care is not taken.

All these four different versions will be compared in terms of accuracy of the results with a *Simulink* implementation of the continuous time model equations, setting a $100ns$ fixed time step, using Runge-Kutta ODE4 solver and double-precision (64-bit) floating-point variables. This solver was chosen to produce the reference values because of its high accuracy. Moreover, these implementations are studied considering two common case scenarios: (i) one when the speed of the shaft change between ±30% around the synchronous value; and (ii) another one when the load pass from consuming 50% to 85% of the nominal power at constant speed. The variation of the input signals on each of these experiments can be seen in Figures 2 and 3, along with the speed of the shaft which is an internal variable. Currents and voltages are shown in per unit whereas omega and torque are in $^{rad}/_{s}$ and Nm respectively.

It has to be taken into account that the reference data has the same sampling time of the shortest implementation: $100ns$. The other are multiple of this value in order to be coherent with the sampling time of the calculations.

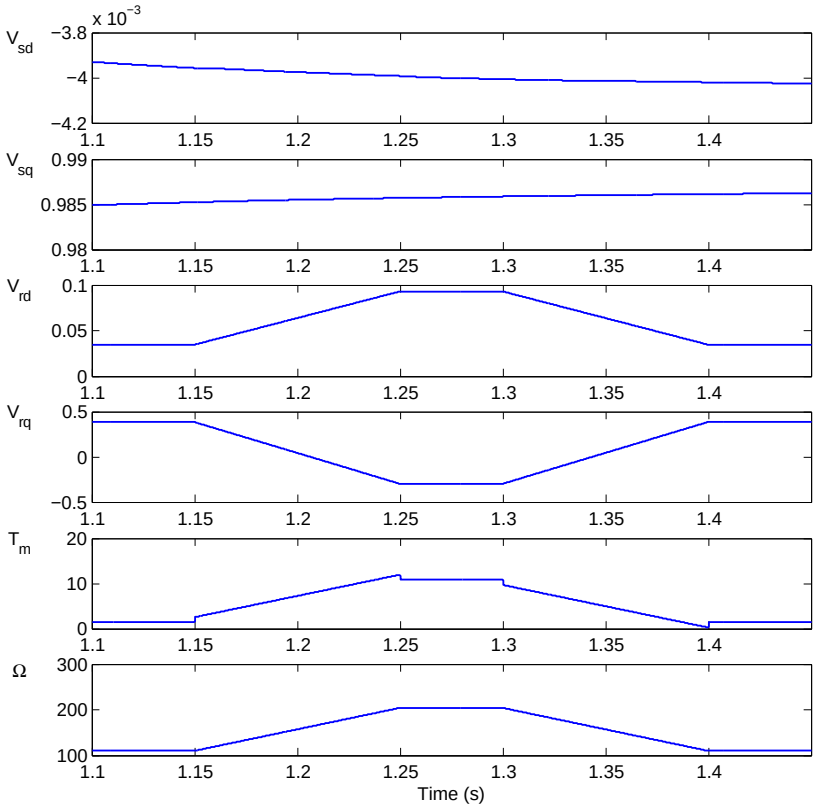

Figure 2: DFIG input signals from the mechanical load and controller - Speed change test

The error is calculated using equation 5.

$$e_{avg} = \frac{\sum |x_{ref} - x_{imp}|}{N} \tag{5}$$

where x_{ref} is the reference signal obtained in Simulink, x_{imp} is the actual implementation result, and N the number of samples.

3.2.1. Full-Software implementation

As mentioned before, this is the easiest and most straightforward implementation. Compiler optimization options are used to infer automatic vectorization which implies arranging data and operations in a proper way to be executed by the NEON SIMD and the VFPv3 FPU at a CPU clock frequency of $667MHz$. In some cases it reduced the execution time by 80% while maintaining the same precision in the results [16].

DFIG 64-bit Floating-point Software Version. When using double-precision floating-point variables, the time needed by the DFIG function to return the values was

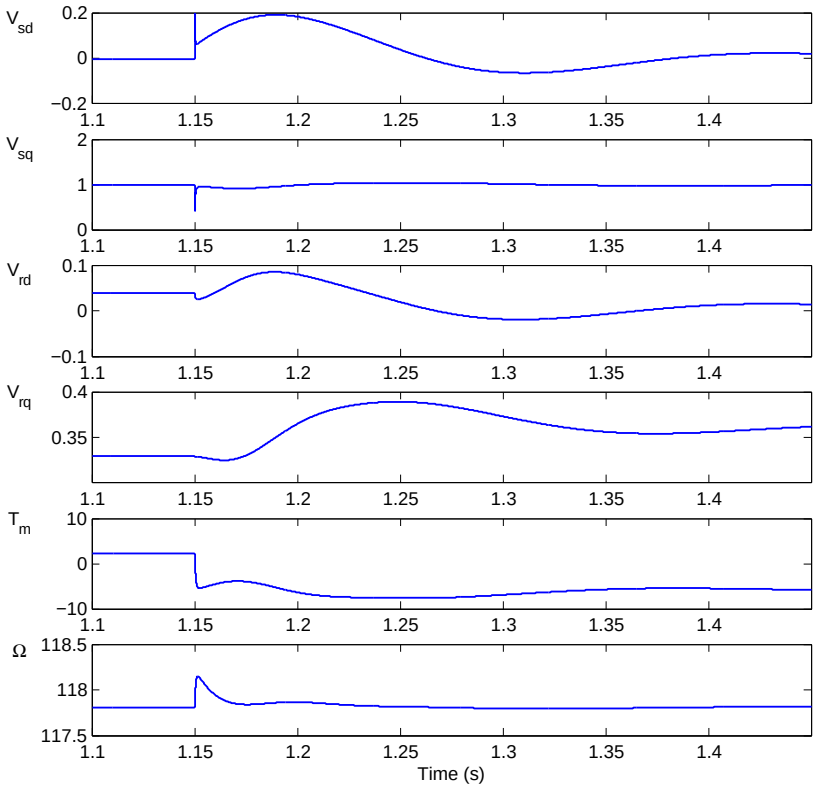

Figure 3: DFIG input signals from the mechanical load and controller - Load change test

Table 1: 64-bit Floating-point average errors (SW version)

	Speed change test	Load change test
i_{sd}	$7.302e^{-8}$	$1.739e^{-6}$
i_{sq}	$6.947e^{-8}$	$2.064e^{-6}$
i_{rd}	$7.328e^{-8}$	$1.599e^{-6}$
i_{rq}	$4.724e^{-8}$	$1.884e^{-6}$
T_e	$1.451e^{-6}$	$4.289e^{-5}$

Table 2: 32-bit Floating-point average errors (SW version)

	Speed change test	Load change test
i_{sd}	$1.136e^{-3}$	$1.986e^{-3}$
i_{sq}	$7.959e^{-4}$	$2.057e^{-3}$
i_{rd}	$1.222e^{-3}$	$1.896e^{-3}$
i_{rq}	$7.554e^{-4}$	$1.925e^{-3}$
T_e	$1.674e^{-2}$	$4.485e^{-2}$

$372ns$.

Then, in order to evaluate the results obtained by this version, a comparison with the *Simulink* continuous-time model equations is performed. Results of both experiments can be seen in Table 1. The values are in p.u.

DFIG 32-bit Floating-point Software version. On the other hand, when using single-precision floating-point variables, the execution time needed by the IP was not significantly smaller: $367ns$. This gives us an idea about how optimized are these CPUs to perform 64-bit operations, even though their architecture is based on 32-bit.

The error results when comparing this version with the *Simulink* model can be seen in Table 2.

Opposing the errors of both software implementations and taking into account that the difference in execution time is barely noticeable, it is obvious that using double-precision floating-point variables is the most convenient choice.

3.2.2. Full-hardware implementation

During the development of this version, it was found that the best way to make the full-hardware treatment faster than the full-software one, was to realize a full combinatorial treatment (i.e. zero latency). By default, HLS uses a $100MHz$ clock. So to boost the treatment, the fabric clock was reduced to $5MHz$ in order to ensure a full combinatorial treatment.

DFIG 32-bit Floating-point Hardware Version. The HLS execution time estimation of this version was $172ns$. This means that during one clock cycle ($200ns$), the full combinatorial simulation step is achieved after $172ns$. Conversely, when mantaining the clock to $100MHz$, the tool adds pipelining registers in order

to optimize the combinatorial paths which leads in computing the model in 76 clock cycles, thus $760ns$ which is larger than full-software version. Table 3 presents the corresponding consumed resources.

Regarding the accuracy of the results in the Speed and Load tests, they do not differ significantly from the 32-bit Floating-point version. See Table 3 for more details.

However, empiric evaluations showed that the time estimation is not what the IP really needs to stabilize the combinatorial output. After running the place and route, the timing analysis summary says that the longest data path is $256ns$. Notwithstanding, the model was tested empirically in the *ZedBoard* and looking at the waveform the results showed that the outputs were stabilized between 70 and $80ns$. Even though the results were correct, care should be taken when considering the smaller times as there is a data path that exceeds the $200ns$.

To end with, a 64-bit floating-point version of the DFIG model was also examined. However, the amount of DSP blocks available in the *Zynq-7020* was not enough to obtain a hardware solution comparable to the software version in terms of execution time.

DFIG 32-bit Fixed-point Hardware Version. The aim to study a Fixed-point version was because the logic involved in the mathematical operations is simpler than its Floating-point counterpart, and hence, the execution may be shorter and the area utilisation should be reduced significantly. As the model uses p.u. variables, it was easy to define the format of the fixed-point word for voltages and currents: $32Q30$, which uses 1 sign bit, 1 integer bit, and 30 fractional bits. However, for the torque and angular speed the format used was $32Q23$, which uses 1 sign bit, 8 integer bits, and 23 fractional bits. This allows the current and voltage signals to be between the range $[-2, 2[$, whereas the torque and omega can vary between $[-256, 256[$.

Here again, a long clock period has been defined in order to let HLS synthesize the IP in full combinatorial. After the HLS tool finished the synthesis, estimations said that the fixed-point version executes all the necessary operations in $110ns$ using the resources shown in Table 5. Again, these results did not match after executing the place and route, which showed that the longest path needed only $76ns$. Moreover, after programming and running the model in the real system, the waveform captured by the *Integrated Logic Analyzer* (ILA) showed that the output results were valid and stable after $20ns$ letting the DFIG model to be executed in a very short time. But once again, experimental results should be considered with care as according to the timing summary, which takes into account worst-case scenario (highest temperature and lowest device supply voltage) there is a critical path which needs $76ns$ to output a valid value. Notwithstanding, taking into account a system of these characteristics where the time constant is much slower than those $100ns$, executing the calculations very fast can be seen as a great advantage. The model could be improved adding some other characteristics like aging, losses, etc., and having enough time to be able to follow the system's dynamics.

Table 3: 32-bit Floating-point absolute average errors (HW version)

	Speed change test	Load change test
i_{sd}	$4.957e^{-4}$	$1.848e^{-3}$
i_{sq}	$5.937e^{-4}$	$1.837e^{-3}$
i_{rd}	$5.788e^{-4}$	$1.745e^{-3}$
i_{rq}	$4.595e^{-4}$	$1.705e^{-3}$
T_e	$5.877e^{-4}$	$2.132e^{-3}$

Table 4: 32-bit Fixed-point absolute average errors (HW version)

	Speed change test	Load change test
i_{sd}	$4.391e^{-4}$	$5.657e^{-4}$
i_{sq}	$8.225e^{-4}$	$4.351e^{-4}$
i_{rd}	$4.706e^{-4}$	$6.015e^{-4}$
i_{rq}	$7.842e^{-4}$	$3.971e^{-4}$
T_e	$8.680e^{-2}$	$5.165e^{-3}$

The average absolute errors compared with the *Simulink* 64-bit Floating-point version can be seen in Table 4.

4. eRTS for power electronics systems

Some might think that it would have been more appropriate to use the power converters connected to the DFIG to test the device in a power electronics application. However, this work has already been done and can be found extensively in the literature [3, 8, 7]. The aim of this paper is however to use a scalable power converter with a more complex structure in order to exploit the device at maximum coping with the eRTS requirements. Hence, an MMC was chosen as the application to be studied [26]. The idea is to have a block which estimates the arm current based on the total DC voltage, the voltages of the Half-Bridge (HB) capacitors, and the PWM signals. This block could be used in the context of a fault-tolerant embedded controller where this estimated current will be used in case of a current sensor fault.

A broad comparison changing the number of cells per arm and using different variable formats is made as well, but conversely to the DFIG evaluation, in this

Table 5: FPGA hardware resources utilisation

Resource type	BRAM	DSP48E	FF	LUT
Available	280	220	106400	53200
Floating-point version	0	179	10832	24505
Usage (%)	0	81	10	46
Fixed-point version	0	128	353	1441
Usage (%)	0	58	˜0	2

case the selected device is the *Zynq-7045*. It has much more resources than the mid-range *Zynq-7020* which will be needed if the goal is to keep low the execution time when increasing the number of submodules (SM) per arm.

4.1. The MMC

The basic structure of an MMC is shown in Figure 4. It is formed by N HB SM per arm capable of producing a line-to-neutral voltage waveform of $N+1$ levels [26]. An inductor L_{arm} is added on each arm to limit current harmonics and the fault current in the event of a DC fault. Each SM includes a capacitor and two IGBTs with antiparallel diodes as shown in the enlarged part of Figure 4.

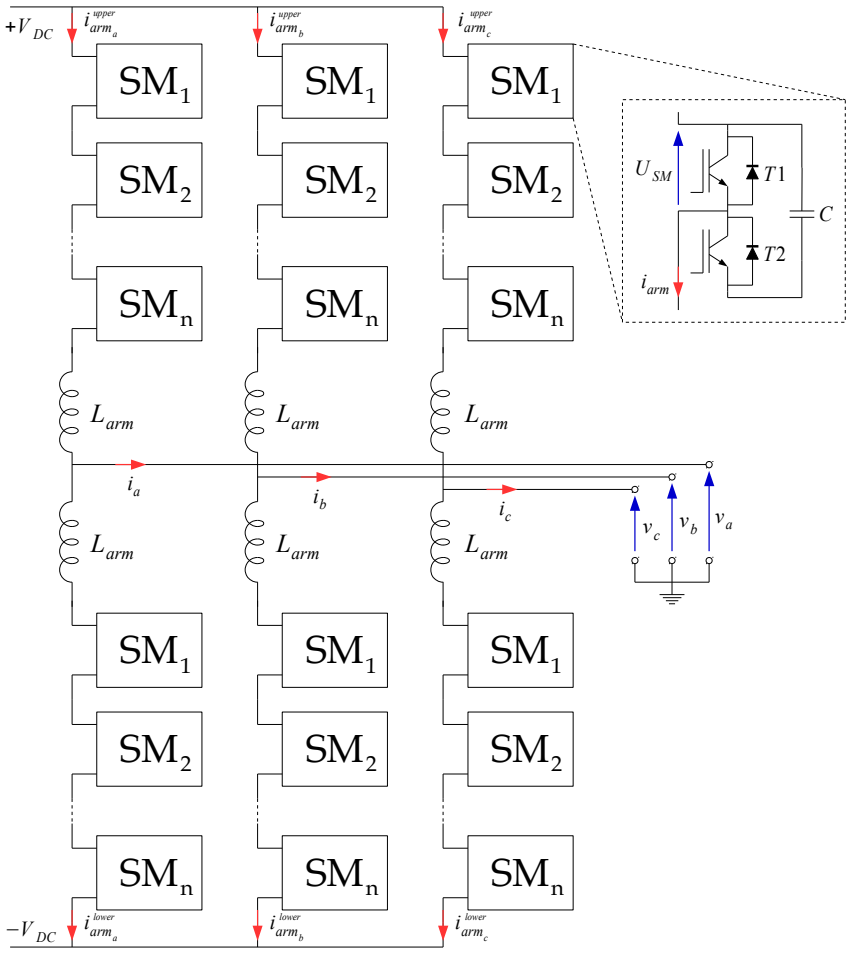

Figure 4: Structure of the MMC

The two IGBTs of each SM can be controlled through gate signals which

allow two different states. In the ON state $T2$ is fired and $T1$ is blocked and accordingly, the SM voltage U_{SM} is equal to the capacitor voltage. Depending on the arm current (I_{arm}) direction, the capacitor will be charged through the IGBT or discharged through the diode (equations 6 and 7 respectively). In the OFF state $T1$ is fired and $T2$ is blocked, resulting in $U_{SM} = 0$. The capacitor voltage remains constant whatever the direction of I_{arm} is, but in one case the current flows through the IGBT and in the other through the diode (equations 8 and 9 respectively). These four possibilities depicted in Figure 5 show graphically these four current paths.

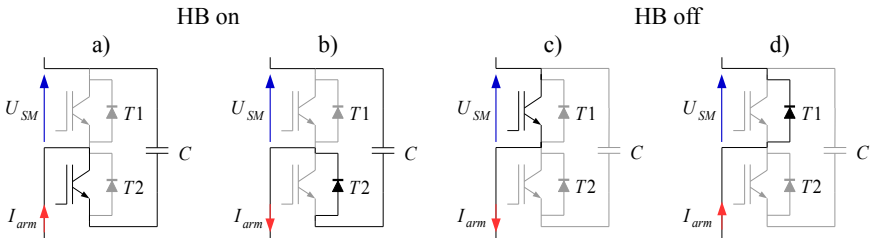

Figure 5: HB functioning modes

Taking into account these four options, the output voltage of each SM can be calculated using the following equations:

$$U_{SM} = R_{ON}^{IGBT} I_{arm} + V_{cap} \quad (6)$$
$$U_{SM} = R_{ON}^{diode} I_{arm} + V_{cap} \quad (7)$$
$$U_{SM} = R_{ON}^{IGBT} I_{arm} \quad (8)$$
$$U_{SM} = R_{ON}^{diode} I_{arm} \quad (9)$$

Concerning the digital implementation of the converter, a solver adapted to the requirements of the application has to be chosen. Due to its simplicity and low computational burden it was decided to use the Euler Backward's discretization method. Its accuracy when the step time is sufficiently low does not differ significantly from other solvers with much more operations to be performed.

A diagram of the circuit used to validate the model in simulation and in the experiments is the one shown in Figure 6. It is basically composed of three HB, an inductor L_{arm}, and a resistor R_{load}. The rest of the circuit includes a voltage source V_{ac}, an autotransformer AT, a diode based rectifier DB, and a filter circuit formed by an inductor L and two capacitors C_1 and C_2 to smooth the input DC voltage. The behavior of this circuit can be modeled using equation 10, being S_i the switch signals for each SM.

$$V_{DC}(t) = \sum S_i(t) V_{cap_i}(t) + L_{arm} \frac{di(t)}{dt} + R_{load} i(t) \quad (10)$$

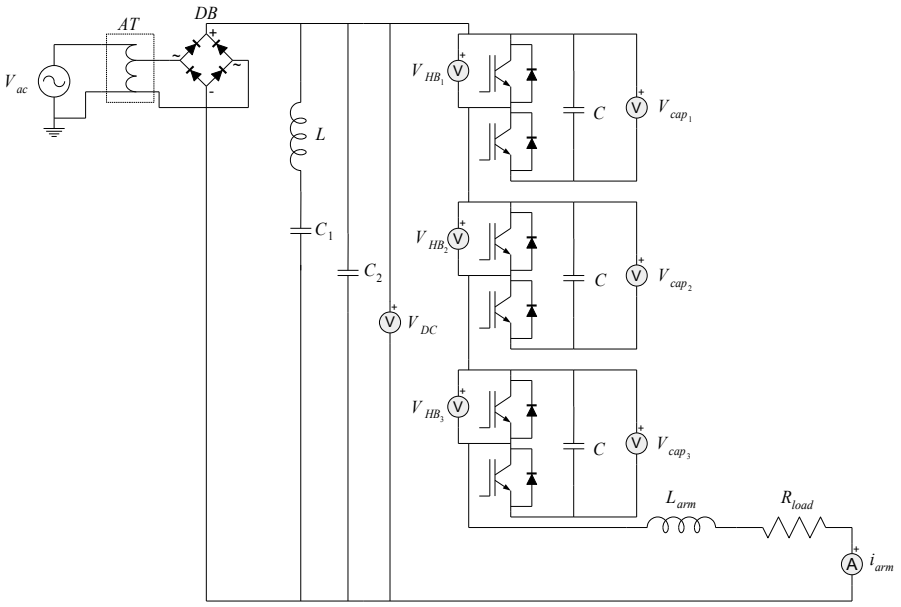

Figure 6: The MMC circuit diagram

Applying Euler Backward's discretization method, equation 11 is obtained, where $x\,(k)$ is the value of x in the actual time step, $x\,(k-1)$ is the value of x in the previous time step, and $\triangle T$ is the sampling time. Then operating, equation 12 is solved for $\hat{i}\,(k)$, which is the one programmed on the FPGA fabric for its use as eRTS.

$$V_{DC}\,(k-1) = \sum S_i\,(k-1)\,V_{cap_i}\,(k-1) + L_{arm}\frac{i\,(k) - i\,(k-1)}{\triangle T} + R_{load} i\,(k-1) \tag{11}$$

$$\hat{i}\,(k) = i\,(k-1) + \frac{\triangle T}{L_{arm}}\left[V_{DC}\,(k-1) - \sum S_i\,(k-1)\,V_{cap_i}\,(k-1) - R_{load} i\,(k-1)\right] \tag{12}$$

4.2. MMC implementation

As previously said, the idea is to develop a full-hardware IP using HLS which has as inputs the capacitor voltages V_{cap_i}, the PWM signals S_i, and the input DC voltage V_{DC} as depicted in Figure 7. The only output is the estimated arm current $\hat{i}_{arm}$.

Several HLS implementations were evaluated: changing the number of SM per arm; and similarly to the DFIG study, changing the format of the variables

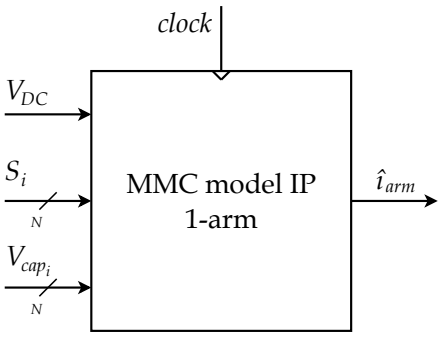

Figure 7: The MMC model IP

between 64-bit floating-point, 32-bit floating-point, and 32-bit fixed-point. In order to be fair between all the implementations, the same $100MHz$ clock was utilised to drive the IPs, which was the best frequency in average for most of the cases. The different results regarding FPGA area used can be seen in Figure 8. The results are shown in percentage of the total amount of every resource available in the *Zynq-Z7045*, which are 1.090 BRAM blocks, 900 DSP units, 437.200 Flip-Flops, and 218.600 Look-up Tables (LUTs).

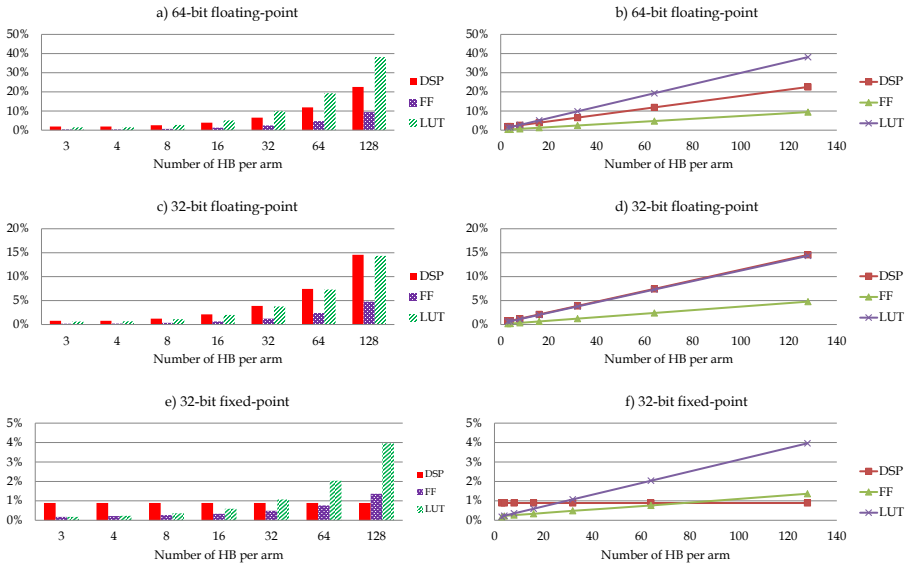

Figure 8: MMC FPGA resources usage per arm

It can be seen that in all data formats the resources utilisation increase linearly with different slopes though, but the number of DSP blocks in the fixed-point case remains constant and equal to eight. This is because the sub-

tractions of equation 12 are performed using LUTs instead of DSP units like in the floating-point cases. It has to be kept in mind that these results are per arm. Therefore, if the purpose is to simulate the whole converter, these results have to be multiplied by the number of arms, hence 6. Consequently, the 128 and 64 HB solutions for the 64-bit floating-point version will not be possible to be implemented on the selected device because of lack of resources if the idea is to execute all the IPs in parallel.

Regarding the execution time, in Figure 9 can be seen that the difference between the two floating-point versions is not significantly big. However, the execution time of the fixed-point version is considerably low. This is due to the low complexity of a fixed-point operation compared to a floating-point one. The logarithmic-like shape visible in Figure 9b is due to the implementation of an *adder tree* to perform the subtractions of the ON capacitor voltages of equation 12, i.e. $\sum S_i\,(k-1)\,V_{cap_i}\,(k-1)$. The execution time increases as a logarithmic function of the number of SM, $t_{calc} = f\,(log_2\,(N))$. Consequently, attention has to be taken in order to not overflow the result of the *adder tree* in the fixed-point case. This can be done either by increasing the number of integer bits at expense of the fractional bits hence loosing accuracy, or by expanding the overall word length according to the number of levels of the *adder tree*.

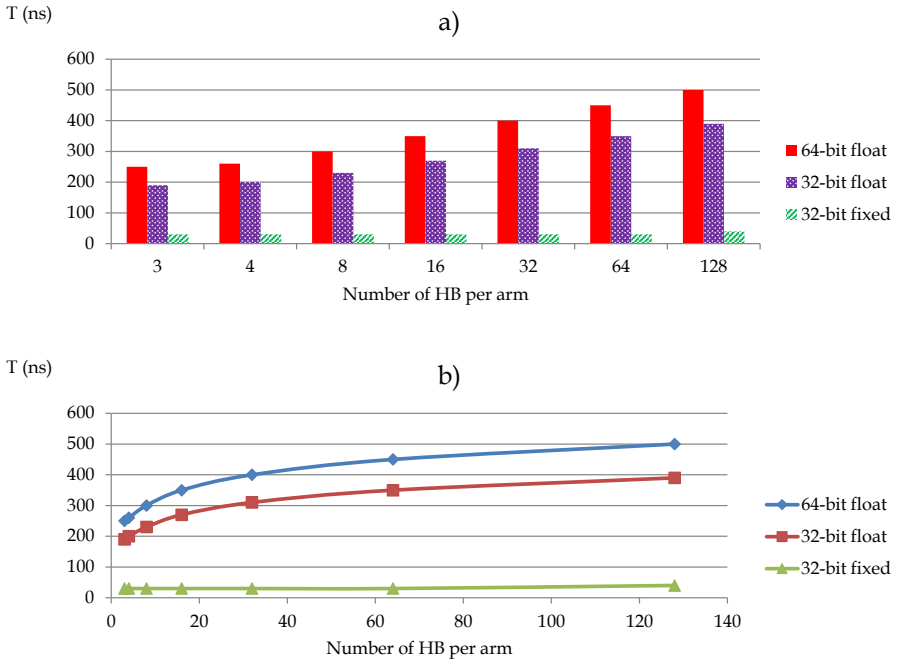

Figure 9: MMC execution time evaluations

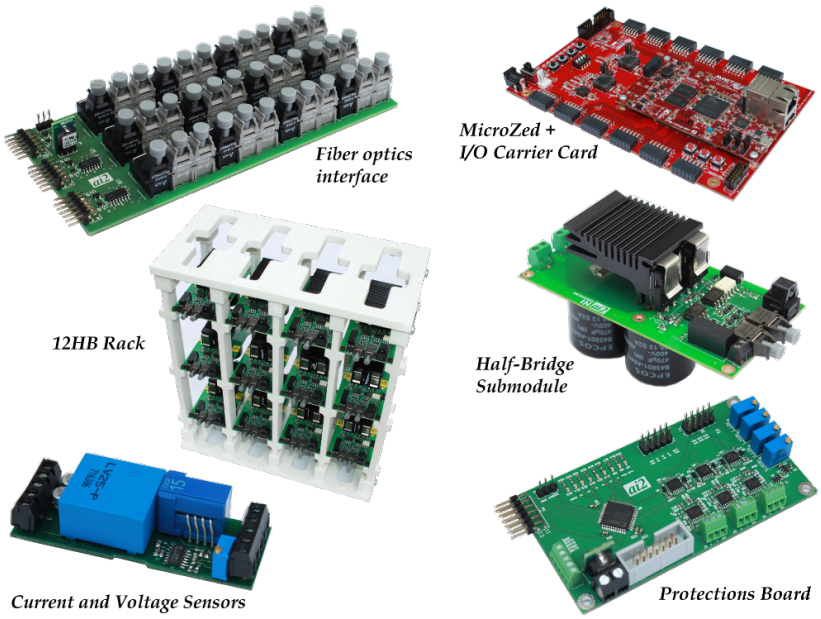

Figure 10: Experimental test bench subsystems

4.3. Experimental verification

All the subsystems that form the test bench are shown in Figure 10. They were all built and assembled at the Institute of Automation and Industrial Informatics (AI2) of the Universitat Politècnica de València, Spain.

Hence, in order to validate the developed eRTS, the test rig was configured as depicted in Figure 4. It is formed of three HB connected in series to a $5mH$ inductance L_{arm}, a 40Ω resistor R_{load}, and to an AC voltage source through an autotransformer and a diode based rectifier using a simple LC filter at the output. Both inductance and resistor were identified using common identification methods and the final values used in the model were 42.89Ω for the resistor, and $3.9mH$ for the inductance instead of their nominal values. It has to be said that even though the inductance was working below its nominal value, its behavior is not linear. Notwithstanding, the eRTS results were acceptable in terms of accuracy and the estimated current did not diverge.

The SM were all fired with the same PWM signal at a switching frequency of $5KHz$ to force the maximum current oscillation possible in order to be sure that the current estimation does not diverge in harsh conditions.

There were performed two main tests setting the data acquisition of the scopecorder at two different sampling ratios: One (i) sampling the voltage and current signals at $10ns$ during $10ms$ for a first validation of the model; and another (ii) sampling the signals at $2\mu s$ during $5s$ to validate that the model does not diverge in mid-term simulations. The model is also validated for different execution times which confirms that it can work when the number of cells per

arm is big and the calculations need more time to be executed. In the same way as in the DFIG study, three different data formats (64/32-bit floating-point and 32-bit fixed-point) are also compared for a three HB case. All these verifications were performed using *MATLAB* and the *Vivado Design Suite* to confirm that this model can be used as an eRTS.

4.3.1. Model validation

The proper functioning of the model was verified offline, executing the model at the same sampling rate as the data acquisition, i.e. at $10ns$. The eRTS was configured to use 64-bit floating-point variables and its results are shown in Figure 11. It can be seen that the estimation follows the curve quite well. The small difference is due to the non-linearities of the inductance and the non consideration of the R_{ON} voltage drop in the IGBTs and diodes, but it does not affect at all the stability of the results.

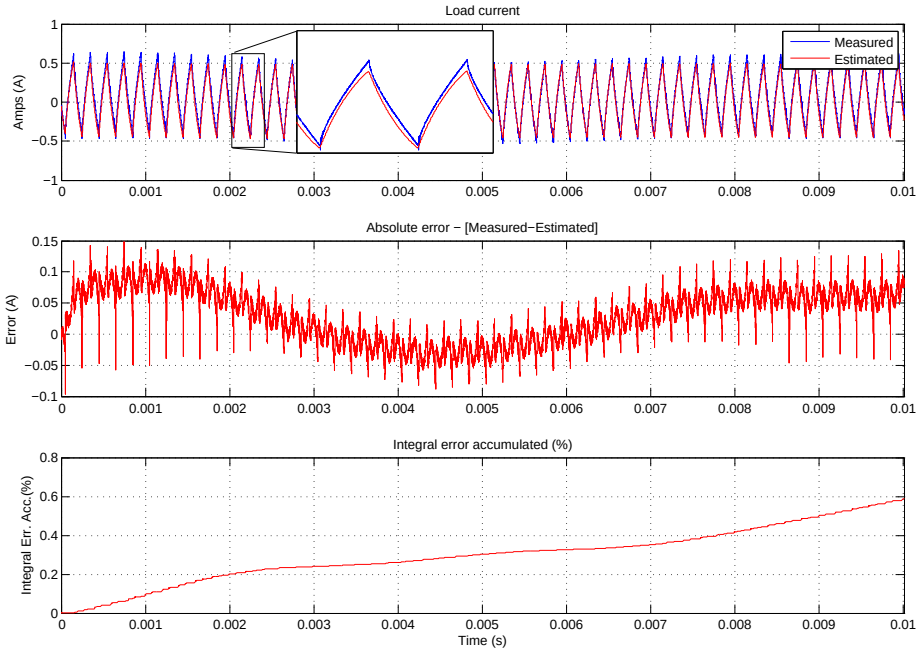

Figure 11: MMC Model validation

The equations used to generate the two error plots are equation 13 and 15:

$$\begin{aligned} e_{abs}(k) &= i(k) - \hat{i}(k) \\ e_{rel}(k) &= \left| \frac{i(k) - \hat{i}(k)}{i(k)} \right| 100 \end{aligned} \tag{13}$$

Table 6: MMC execution time (in ns)

no. HB	64-bit float	32-bit float	32-bit fixed
3	250	190	30
4	260	200	30
8	300	230	30
16	350	270	30
32	400	310	30
64	450	350	30
128	500	390	40

Table 7: Mean integral errors for different execution times

	e_{int} (%)
64fl @ $10ns$	$5.587e^{-7}$
64fl @ $100ns$	$8.116e^{-6}$
64fl @ $200ns$	$1.708e^{-5}$
64fl @ $500ns$	$4.644e^{-5}$
64fl @ $1\mu s$	$7.502e^{-5}$

$$e_{int}(k) = \int_{k}^{k+1} e_{rel}(k)dx \simeq \triangle Ts\left(\frac{e_{rel}(k) + e_{rel}(k-1)}{2}\right) \quad (14)$$

$$e_{acc}(k) = \sum_{k=1}^{k} e_{int}(k-1) \quad (15)$$

Authors decided to use the mean value of the integral of the relative error (equation 14) to obtain more easily comparable results. Hence, the mean value of the integral error e_{int} obtained for the whole $10ms$ period is $5.87e^{-7}$%. It can be said that the model performs satisfactorily when running at $10ns$.

Nevertheless, in none of the versions the model can be executed in less than $10ns$. It is then necessary to execute the model at the same period required by the IP to perform all the calculations to see if the model still emulating the real system properly. The eRTS is then executed at $100ns$, $200ns$, $500ns$, and $1\mu s$ to be coherent with the execution time needed by the different models when increasing the number of cells as seen in Table 6. Figure 12 shows the estimated current of the different versions along with their absolute errors e_{abs} in amperes and the integral relative error accumulated e_{acc} in %.

It can be seen that even executing the model at $1\mu s$, the estimation does not differ significantly compared to the version when it runs at $10ns$. In Table 7 the mean integral relative errors are shown for comparison. It can then be firmly said that the simple Euler Backward's model chosen in this paper can be utilised as an eRTS.

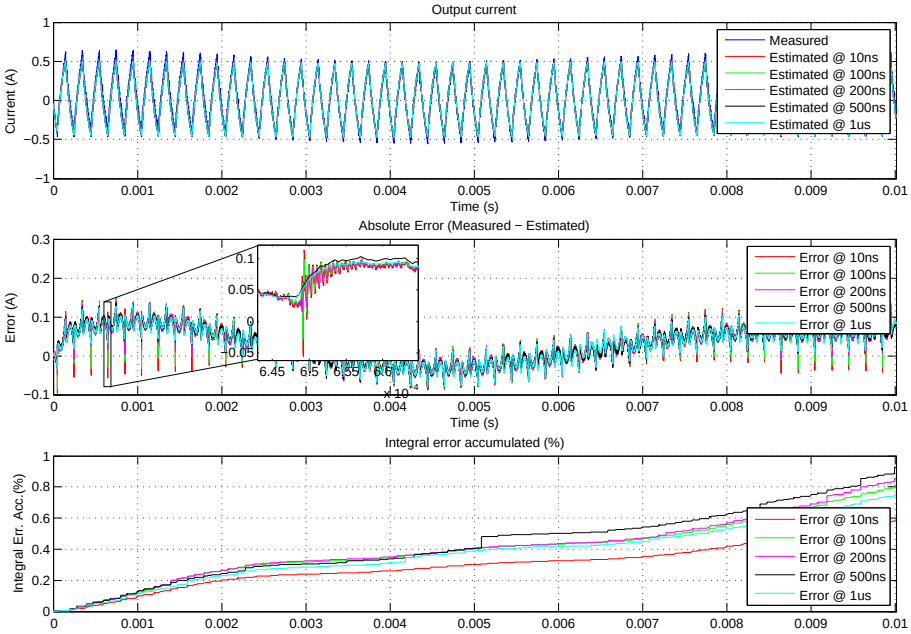

Figure 12: MMC execution time comparisons

Table 8: Mean integral errors for different data formats

	e_{int} (%)
64fl @ $10ns$	$5.587e^{-7}$
64fl @ $250ns$	$5.879e^{-7}$
32fl @ $190ns$	$5.872e^{-7}$
32fx @ $30ns$	$5.862e^{-7}$

4.3.2. Data format comparison

In this subsection, the three proposed data formats are compared with the measured current in an offline simulation. The execution rate of every IP is set to their corresponding execution time for the three HB configuration (see Table 6), i.e. $250ns$ for the 64-bit floating-point, $190ns$ for the 32-bit floating-point, and $30ns$ for the 32-bit fixed-point. It can be seen in Figure 13 that the errors differ very little from each other. Moreover, when calculating the mean of the integral error for the four different solutions it can be verified that the errors are almost equal.

Watching at the results, it can be said that all three versions perform similarly for the 3 HB case. The choice between one of them will be depending on the number of SM per arm and the area available for the implementation.

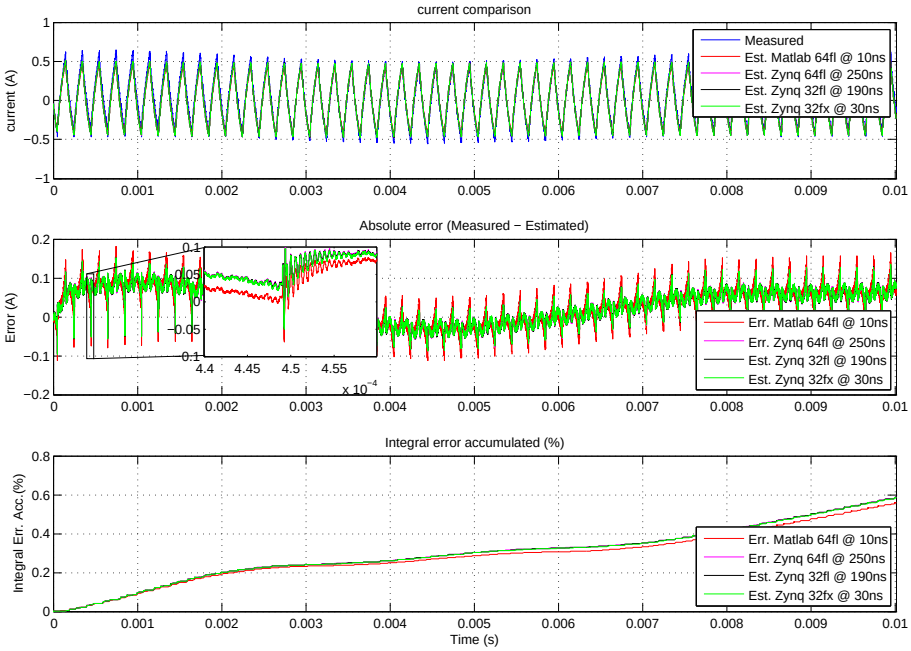

Figure 13: Data format comparison ($10ms$)

4.3.3. Longer horizon simulation

The last test performed is a $5s$ simulation sampling the data at $2\mu s$ and the purpose is to verify that the current estimation does not diverge. Thus, the second plot of Figure 14 shows that the absolute error (equation 13) remains constant for the whole simulation period. Moreover, the last plot of the same figure shows that the integral relative error accumulated (equation 15) increases linearly and not exponentially as it would if the absolute error increased overtime, corroborating the proper functioning of the model.

Figure 15 shows a more detailed view at different times of the simulation ($0s$, $2.5s$ and $5s$), which allows to visually verify that the estimation follows the measured current throughout the whole period. The mean value of the integral of the relative error (equation 14) for the whole simulation running the model at $2\mu s$ is $3.371e^{-4}\%$, which still remains a quite satisfactory result.

5. Conclusions

The objective of this paper was to evaluate how suitable is the *Zynq* SoC platform to be used as eRTS of electromechanical and power electronic systems. This study was achieved through the implementation of a DFIG and an MMC both coded in C/C++.

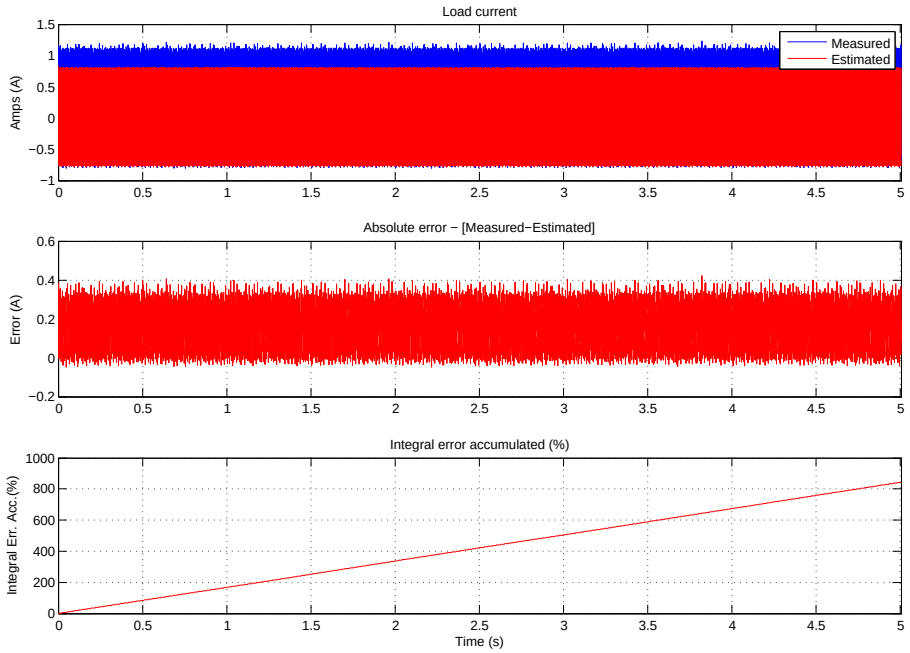

Figure 14: $5s$ simulation

For the DFIG case, four digital implementations have been made and analysed: Two full-software designs (32-bit and 64-bit floating-point) and two full-hardware (32-bit floating-point and 32-bit fixed-point) generated by the HLS tool. For each version, time/resources analyses were presented and discussed. In addition, a *Simulink* continuous-time model using $100ns$ of simulation time step and a Runge-Kutta ODE4 solver was taken as a reference to compare the precision of the results in two different scenarios: (i) a $\pm 30\%$ progressive change in the speed of the shaft; and (ii) a change in the load from 50% to 85% of the nominal power of the generator.

For this first case study, results showed that in terms of accuracy, having a model running with a time step four times shorter but half the precision did not obtain better results than the 64-bit floating-point software implementation. Furthermore, using a 32-bit floating-point software implementation did not reduce significantly the execution time compared with its 64-bit counterpart. Hence, if the system is not execution-time critical, a software double-precision version is recommended. However, if the best performance in terms of rapidity of the calculations is needed, the fixed-point version implemented in the PL is the most convenient solution.

As far as the MMC is concerned, authors were focused exclusively on full-hardware implementations, still using the HLS tool. Then, there were compared three different word formats: a 64-bit floating-point, a 32-bit floating-point, and

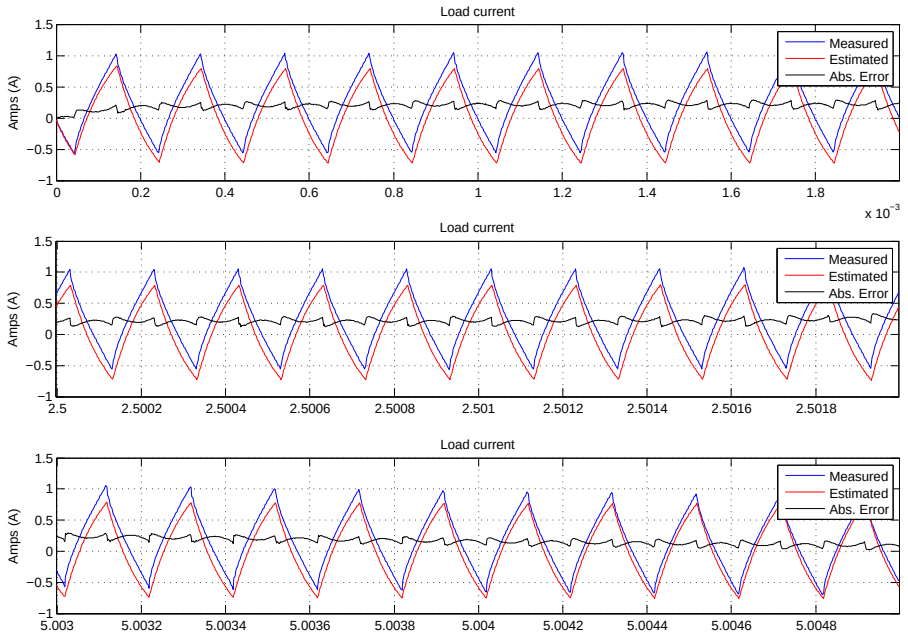

Figure 15: 5s simulation enlarged

a 32-bit fixed-point; for 3, 4, 8, 16, 32, 64, and 128 HB cells per arm. Here again, they were analysed in terms of resources usage, execution time, and precision compared with values obtained from an experimental test bench.

For the MMC evaluation, an IP block containing the power converter equation necessary to estimate its arm current was programmed where the circuit parameters, the number of HBs and the format of the variables can be changed easily.

Regarding the area used in the floating-point versions, the resources utilisation increase linearly with the number of cells but the area used for the 32-bit version is around the half of the 64-bit. This was not the case for the DSP blocks used by the fixed-point version which remained fixed and equal to eight due to the realization of the additions using LUTs instead of DSPs.

With respect to the execution time, both floating-point implementations do not differ significantly. This is not the case of the fixed-point version, where the execution time is way below their floating-point counterparts because is specially benefitting from the implementation of an *adder tree* which performs the HB voltage subtraction operations using LUT.

It has to be said as well that when using HLS tools is quite straightforward to change the format and width of the variables, letting the tool to deal with all the hardware operations automatically. The choice between them will be made depending on the precision of the results needed, and the area available in order

to best represent the behavior of the real system.

Taking the MMC IP as an example and considering possible real applications of these eRTS, if the purpose is to monitor continuously the system in order to evaluate if diagnostic actions need to be performed, a controller evaluating windows of the integral of the relative error periodically could be included. Therefore, a sudden error increase is detected could be caused by a system fault. Different error scenarios with their corresponding actions should be studied. An overcurrent could be caused by a short circuit in the load, or by a faulty IGBT. Conversely, a decrease in the current could be caused due an IGBT failure or a capacitor explosion, etc. Another plausible use of these eRTS could be in the context of sensorless or fault-tolerant control. Regarding the latter, a current sensor malfunction could be overcome by retrieving the current estimated by the IP instead of the measured one.

References

[1] C. Townsend, T. Summers, J. Vodden, A. Watson, R. Betz, J. Clare, Optimization of switching losses and capacitor voltage ripple using model predictive control of a cascaded h-bridge multilevel statcom, IEEE Transactions on Power Electronics 28 (7) (2013) 3077–3087. doi:10.1109/TPEL.2012.2219593. 1

[2] C. Townsend, D. Tormo, R. Baraciarte, Y. Yu, H. Z. de la Parra, G. Demetriades, V. Agelidis, Heuristic model predictive modulation for high-power cascaded multi-level converters, IEEE Transactions on Industrial Electronicsdoi:10.1109/IECON.2015.7392131. 1

[3] M. Dagbagi, A. Hemdani, L. Idkhajine, M. W. Naouar, E. Monmasson, I. Slama-Belkhodja, ADC-based embedded real-time simulator of a power converter implemented in a low-cost FPGA: Application to a fault-tolerant control of a grid-connected voltage-source rectifier, IEEE Transactions on Industrial Electronics 63 (2) (2016) 1179–1190. doi:10.1109/TIE.2015.2491883. 1, 4

[4] J. Sawma, F. Khatounian, E. Monmasson, R. Ghosn, L. Idkhajine, Evaluation of the new generation of system-on-chip platforms for controlling electrical systems, Industrial Technology (ICIT), 2015 IEEE International Conference on (2015) 1570–1575doi:10.1109/ICIT.2015.7125320. 1

[5] A. Schmitt, J. Richter, U. Jurkewitz, M. Braun, FPGA-based real-time simulation of nonlinear permanent magnet synchronous machines for power hardware-in-the-loop emulation systems, in: Industrial Electronics Society, IECON 2014-40th Annual Conference of the IEEE, IEEE, 2014, pp. 3763–3769. doi:10.1109/IECON.2014.7049060. 1

[6] A. Hasanzadeh, C. Edrington, N. Stroupe, T. Bevis, Real-time emulation of a high-speed microturbine permanent-magnet synchronous generator using multiplatform hardware-in-the-loop realization, IEEE Transactions on Industrial Electronics 61 (6) (2013) 3109–3118. doi:10.1109/TIE.2013.2279128. 1

[7] F. Montano, T. Ould-Bachir, J.-P. David, An evaluation of a high-level synthesis approach to the FPGA-based sub-microsecond real-time simulation of power converters, IEEE Transactions on Industrial Electronicsdoi:10.1109/TIE.2017.2716880. 1, 4

[8] C. Liu, X. Guo, F. Gao, E. Breaz, P. Damien, F. Gechter, FPGA based real-time simulation of high frequency soft-switching circuit using time-domain analysis, in: Industrial Electronics Society , IECON 2016 - 42nd Annual Conference of the IEEE, 2016. doi:10.1109/IECON.2016.7793227. 1, 4

[9] J. Rodriguez-Andina, M. D. Valdes-Pena, M. J. Moure, Advanced features and industrial applications of FPGAs - a review, IEEE Transactions on Industrial Informatics 11 (4) (2015) 853–864. doi:10.1109/TII.2015.2431223. 1

[10] Xilinx Inc., DS190: Zynq-7000 All Programmable SoC Overview (January 2016). 1, 2.1

[11] Altera Inc., Altera ARM-Based SoC Overview (2016). 1

[12] D. Tormo, L. Idkhajine, E. Monmasson, R. Blasco-Gimenez, Embedded real-time simulator implementations of electromechanical systems using system-on-chip devices, in: ELECTRIMACS 2017, 2017. 1, 2.3

[13] E. Monmasson, M. N. Cirstea, FPGA design methodology for industrial control systems - a review, IEEE Transactions on Industrial Electronics 54 (4) (2007) 1824–1842. doi:10.1109/TIE.2007.898281. 1

[14] I. Bahri, L. Idkhajine, E. Monmasson, M. E. A. Benkhelifa, Hardware/software codesign guidelines for system on chip FPGA-based sensorless AC drive applications, Industrial Informatics, IEEE Transactions on 9 (4) (2013) 2165–2176. doi:10.1109/TII.2013.2245908. 1

[15] Xilinx Inc., UG902: High-Level Synthesis with Vivado HLS (2012). 1, 2.2

[16] D. Tormo, L. Idkhajine, E. Monmasson, R. Blasco-Gimenez, Evaluation of SoC-based embedded real-time simulators for electromechanical systems, in: The 42nd Annual Conference of IEEE Industrial Electronics Society, 2016. doi:10.1109/IECON.2016.7793185. 1, 3.1, 3.2.1

[17] ARM Holdings, AMBA AXI and ACE Protocol Specification (2013). URL http://infocenter.arm.com 2.1

[18] Xilinx Inc., UG585: Zynq-7000 Technical Reference Manual (2016). 2.1

[19] ARM Holdings, Cortex-A9 NEON Media Processing Engine - Rev: r4p1 (2012). 2.1

[20] ARM Holdings, Cortex-A9 Floating-Point Unit - Rev: r4p1 (2012). 2.1

[21] J. Wu, NEON/VFPU hardware acceleration (2012). 2.1

[22] R. Nane, V.-M. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. T. Chen, H. Hsiao, S. Brown, F. Ferrandi, J. Anderson, K. Bertels, A survey and evalutaion of FPGA high-level synthesis tools, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 35 (10) (2015) 1591 – 1604. doi:10.1109/TCAD.2015.2513673. 2.2

[23] R. Pena, J. Clare, G. Asher, A doubly fed induction generator using back-to-back pwm converters supplying an isolated load from a variable speed wind turbine, in: IEE Proceedings - Electric Power Applications, Vol. 143, IEEE, 1996, pp. 380–387. doi:10.1049/ip-epa:19960454. 3.1

[24] M. E. Achkar, R. Mbayed, G. Salloum, N. Patin, S. L. Ballois, E. Monmasson, Modeling and control of a stand alone cascaded doubly fed induction generator supplying an isolated load, in: Power Electronics and Applications (EPE'15 ECCE-Europe), 2015 17th European Conference on, 2015. doi:10.1109/EPE.2015.7309125. 3.1

[25] R. Park, Two-reaction theory of synchronous machines generalized method of analysis, Transactions of the American Institute of Electrical Engineers 48 (3) (1929) 716–727. doi:10.1109/T-AIEE.1929.5055275. 3.1

[26] A. Lesnicar, R. Marquardt, An innovative modular multilevel converter topology suitable for a wide power range, in: Power Tech Conference Proceedings, 2003 IEEE Bologna, 2003. doi:10.1109/PTC.2003.1304403. 4, 4.1