# A GUIDELINE FOR EMPLOYING PSIM ON POWER CONVERTER APPLICATIONS: PROTOTYPING AND EDUCATIONAL TOOL

A. M. S. Alonso[1], F. P. Marafão[1], D. I. Brandao[2], E. Tedeschi[3] and J. F. Guerreiro[4]
[1]Universidade Estadual Paulista (UNESP), Sorocaba – SP, Brazil
[2]Federal University of Minas Gerais (UFMG), Belo Horizonte – MG, Brazil
[3]Norwegian University of Science & Technology (NTNU), Trondheim, Norway
[4]Universidade Estadual de Campinas (UNICAMP), Campinas – SP, Brazil
e-mail: eng.amalonso@gmail.com, fmarafao@sorocaba.unesp.br, dibrandao@ufmg.br, elisabetta.tedeschi@ntnu.no, joel.engeletrica@gmail.com

***Abstract* – Fast prototyping tools for power electronics and control systems are becoming handful players on effortlessly translating computer simulated results into real experimental validations, not abdicating reliability. Among many alternatives, PSIM distinguishes itself from other platforms by providing fixed-step time simulations and quick access to DSP programming. This work aims at showing a starting point for the interaction with the PSIM SimCoder module, and how to take advantage of its features to specifically develop codes employed on power converter applications and for educational purposes. The off-the-shelf TMS320F28335 Experimenter Kit is commonly chosen as target. Example cases accompanied by experimental results are provided for the implementation of an IIR filter, a PLL algorithm, and the generation of three-level PWM signal for single-phase converters. Yet, some discussions regarding the constraints of such tool are pointed out.**

***Keywords*** **– IIR filter, F28335, PLL, PSIM SimCoder, three-level PWM prototyping.**

## I. INTRODUCTION

Prototyping is a cumbersome and time consuming task, and has been for a long time a burden on experimentally validating theoretical methodologies in the power electronics field. The evolution of studies in such area would certainly be more accentuated if easily accessible and fast prototyping tools offered means for translating software simulations into physical setups [1]. For instance, considering the use of microcontrollers and digital processors, code generation and hardware configuration are possibly two development steps that require most attention and demand much effort.

Having that in mind, PSIM offers a way of speeding those tasks up by automatically generating codes that can be immediately loaded onto digital signal processors (DSP), and specially offering support for TMS320F28335, which is extensively used on power converter applications. PSIM handles this code generation by creating C language projects from simulations with its SimCoder module, later compiling them into assembly instructions under a file with ".out" extension, which is thereafter available for loading onto DSPs for running on either RAM or Flash memory.

Although this tool was discussed on a general level in other studies [2]-[3], no mentions have been made regarding the guidelines for interacting with it, neither suggestions on the advisable initial documentations to be considered have been proposed. Yet, even with a few dispersed videos over the internet, there remains a significant gap that leads to an ineffective search for adequate literature.

Moreover, to the best knowledge of the authors, no real power electronic application examples running in a DSP using C generated projects from PSIM were found. Therefore, this work aims at experimentally presenting the implementation of some frequently used digital signal processing applications, such as infinite impulse response (IIR) filters, phase-locked loop (PLL) algorithms, and three level PWM generation for full-bridge inverters. Additionally, some experienced limitations of this tool, and their respective possible countermeasures, are discussed, trying to relieve prospective users of potential heavy efforts. Thus, under such considerations, the contributions of this work are settled.

This work is organized as follows. Section II shortly discusses the functioning of the PSIM tool, the possibilities of use, the suggestion of some initial documentation to be reviewed, and also an overview of the DSP kit chosen for the experimental examples. Section III focuses on the experimental case of digitally implementing an IIR filter, a PLL algorithm, and creating a three-level PWM following an internal or external reference signal. Section IV summarizes some encountered limitations and Section V the conclusions.

## II. STARTING WITH PSIM SIMCODER

The earliest step on getting along with auto-code generation tools is typically getting to know how powerful or limited the respective software is. PSIM is a simulation software that covers most of the major design applications on electrical and electronic circuits, as well as virtual modeling of renewable energy resources. Besides its typical use, it provides extension modules with focus on specific designs, such as motor drive, processor-in-loop, and FPGA implementation. Among those modules, the so-called SimCoder provides access to automatic code generation for DSPs by simply translating simulations into C language projects with configurations already set for embedding assembly instructions on hardware targets.

A fundamental literature for starting with SimCoder is its manual [4], where one can understand how hardware targets are established for simulations, the circuit elements that might be used for such approach, as well as the means for digitally implementing desired circuits for the code generation. For a faster learning regarding of how properly set up the environment and circuit elements, it is important to already have the specifications of the experimental hardware (e.g., ADC and General Purpose Input/Output (GPIO) voltage levels, clock capacities, etc.) in mind. Digital implementation

of the simulated circuit is a fundamental need, since it is where the digital processor functioning converges to. Considering an educational view, this tool therein incorporates the learning of analog-to-digital conversion methodologies.

Focusing on power converters, the most interesting possibilities are given, for instance, by the use of circuit elements for the following purposes:

- Digital Input and Output: used for control actions, sensoring or as interrupt channels;
- ADC Converter: required for making physical sensored analog current and voltage signals available for the digital processing on the DSP;
- Serial Communication Interface (SCI) and Serial Peripheral Interface (SPI): provide data transfer between host computer and DSP, and allow devices to communicate among themselves (e.g., for data transfer), respectively;
- PWM Generator: allow the creation of modulation signals based on a given reference, allowing switches (e.g., transistors) to be controlled. Different switching frequencies, dead times, and carrier wave types are available from single- to three-phase cases;
- C blocks: C codes may be used for handling data, employing algorithms, and controller implementation.

For the particular case of this paper, the chosen hardware target was the TMS320F28335 Experimenter Kit [5], since it provides support for floating-point F28335 DSPs, which are highly employed in power converter applications. Besides, it presents accessible cost, and has a friendly docking station which eases prototyping and motivates educational uses. The key features of this kit are summarized in Table I, along with the major F28335 controlCARD [6] specifications.

To learn how to use auto-generated codes of simulated circuits on DSPs, it is advisable to have a previous knowledge about how these processors can have their memories loaded with programs. For such task, the Code Composer Studio (CCS) software needs to be utilized, and [7] may be a useful reference. Once one knows how to accomplish that task, it will be clear how PSIM is able to generate C projects from simulations, considering DSP memory registers allocation and other set up configurations.

By understanding the procedures discussed in [8], it is possible to learn how to completely integrate this fast prototyping tool with a real hardware target, from the most basic task of creating a digital circuit in PSIM, up to the loading and running of a programme with F28335 DSPs. The work presented in [2], in spite of being generic regarding the explanation of this entire process, is an interesting resource for assimilating the attractiveness of auto-code generation in converter applications and educational resources.

The methodology here adopted, taking into consideration the aforementioned goals, structures the code generation operation with PSIM following the scheme in Fig. 1. From that, one can note that PSIM is able to configure the minimum elements responsible for the prototyping of a power converter, from receiving inputs, to the processing of data, and finally the creation of hardware output control commands. Outputs at this point mean PWM signals, or any other kind of digital command within the range of the DSP hardware limitation. It is highlighted the possibility to both use digital circuits or C code blocks to create the desired logic for the output generation.

**TABLE I - Features of the Hardware Target**

| F28335 Experimenter Kit + ControlCARD | | |
|---|---|---|
| **Feature** | **Description** | **Peripherals** |
| Clock | 150MHz | 18 PWM outputs |
| CPU | 32-Bit | 8 32-Bit timers |
| Memory | 256KB Flash<br>68 KB RAM | 12-Bit ADC with 16 channels<br>Anti-aliasing filter at ADC inputs |
| Power Supply | 5V | **Docking Station** |
| GPIO Pins | 88 | 5V and 3.3V Pins<br>UART Communication through USB<br>Wire-wrap and soldering prototype area |

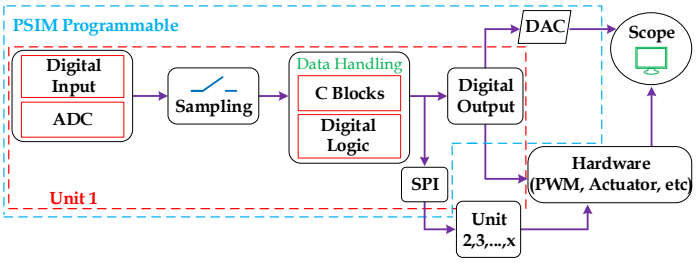


Fig. 1. Range of PSIM auto-code generation for the F28335.

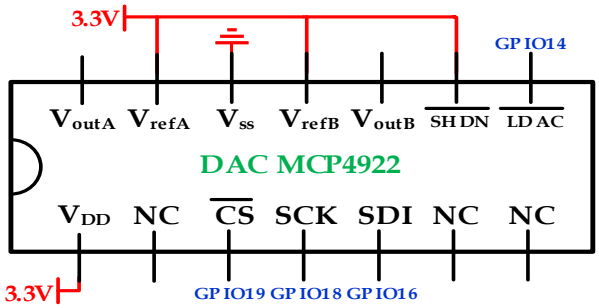


Fig. 2. DAC MCP4922 proposed wiring connection.

An interesting alternative related to SPI communication with PSIM, beyond the primary one, which is transferring data among devices, is the ability to program DAC hardware. This application is very helpful since it allows the analysis of digital reference signals that are used inside the DSP program, supporting troubleshooting of the circuit's logic. In this paper, a 12-Bit MCP4922 DAC was programmed and is used for explaining this claim through real examples. The material found in [9], along with the respective example found within PSIM's installation directory, is sufficient for rapidly implementing such component.

The schematic depicted in Fig. 2 may be used as reference for the connection of this particular DAC model with the Experimenter Kit. Under such wiring, this model provides two analog outputs ($V_{outA}$ and $V_{outB}$), that range from $V_{ss}$ to $V_{DD}$. $V_{refA}$ and $V_{refB}$ are the reference inputs for channels A and B, and the shutdown pin (SHDN) needs to be high when the DAC is active. LDAC is the synchronization input, while SDI is the SPI data input pin, and SCK is the SPI clock input. The Chip Select pin is CS, and NCs are not connected pins.

Accordingly, examples can be described with the intention of showing the feasibility of this tool and explaining possibilities for fast implementations in power converter prototypes or the offering of educational material with practical and theoretical content.

## III. EXAMPLE CASES WITH PSIM AND F28335

Here the suitability of the auto-code generation with PSIM is proved to be feasible through the experimental evaluation of educational examples. Such cases focus on the digital implementation of filters, and single-phase inverter applications (synchronism algorithm and PWM generation).

### *A. IIR Filter*

Digital filters are interesting alternatives for separating desired frequency components in audio and image processing, filtering voltage and current signals under nonsinusoidal conditions to be used in control references for active power filters [10], and many other applications [11].

As described in [12], IIR filters are as notch filters, giving infinite gains in selected frequencies, and allowing the filtering in determined bands. For instance, considering the purpose of identifying components in a polluted signal, it would be enough to use a band-pass approach [13].

Thus, for this example it is considered a band-pass filter with desired center frequency ($\omega_0$) defined to be 60 Hz, with a passing-band ($\omega_c$) of 4 Hz. In a digital implementation, we must account for the discretization of the filter, which may be obtained by a Bilinear Transformation, resulting in the difference equation given in (1).

$$a_0.y(k) = b_0.x(k) + b_1.x(k-1) + b_2.x(k-2) - a_1.y(k-1) - a_2.y(k-2) \quad (1)$$

Where, "y" and "x" are the respective output and input of the filter at a "k" sample. Their aggregated coefficients, "$a_n$" and "$b_n$", were calculated considering a sampling frequency ($f_s$) of 12 kHz, being limited by the 32-bit precision of the F28335, and are shown in Table II.

The difference equation (1) was implemented within a Simplified C Code block, and the PSIM schematic was developed as presented in Fig. 3. One can note that this scheme faithfully follows the methodology discussed in Fig. 1, in which the IIR filter output is viewed in an oscilloscope by means of a DAC. Depending on the features of one's data acquisition circuit, the ADC block from SimCoder module gives the flexibility on accepting such measurements as AC or DC inputs.

In AC mode, if the input signal presents any offset level, it is removed automatically. On the opposite, if operating in DC mode, any undesired offset value should be removed by means of an additional routine. Here it is presented the latter case, reinforcing the goal of educational contribution.

The simulation result of such circuitry is depicted in Fig. 4. That is the expected outcome, considering that the Fourier Series [12] decomposes a square wave as the sum of infinite sinusoidal signals of odd multiple frequencies. Finally, the code automatically generated by PSIM from the simulation in Fig. 3 was loaded into the DSP through CCS.

By using a function generator, a unitary square wave signal with offset was connected to the selected ADC input pin, and the output processed by the IIR filter running on the physical DSP was watched by pinching an oscilloscope probe in the respective DAC output pin. From Fig. 5 it is possible to see the input signal with offset (1-blue), the input after the offset removal algorithm (2-cyan) and the filter output (4-green) giving the expected result. Note that the experimental result matches the simulated one.

**TABLE II - IIR Filter Coefficients**

| *a's* | *Value* | *b's* | *Value* |
|---|---|---|---|
| $a_0$ | 1.000000000 | $b_0$ | 0.001045930 |
| $a_1$ | -1.996923387 | $b_1$ | 0.000000000 |
| $a_2$ | 0.997908139 | $b_2$ | -0.001045930 |

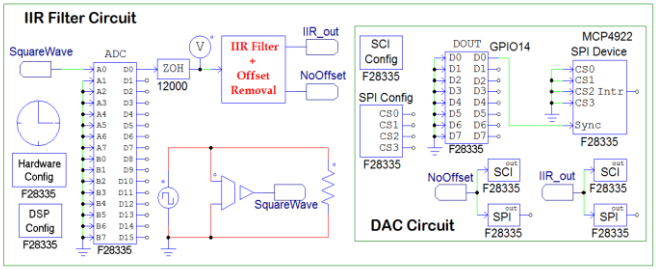


Fig. 3. PSIM circuitry for the IIR filter.

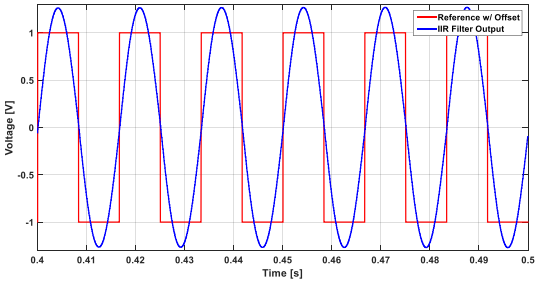


Fig. 4. Simulation result for the IIR filter in PSIM.

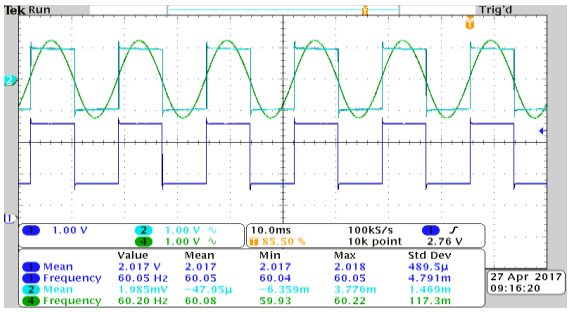


Fig. 5. Experimental result of the IIR filter.

### *B. PLL*

PLL synchronism algorithms have been playing a key role on grid connected DC-AC power converters. They are significantly important to regulate energy injection into grids, when compliance requirements need to be met, as well as for providing adequate power quality interventions, which is the case of active power filters and multifunctional inverters. The PLL methodology proposed in [14] is here adopted for the exemplification of how other phase detection algorithms might be implemented in DSPs using PSIM generated codes.

Looking at single-phase inverters, this methodology consists on synthesizing an orthogonal signal from the fundamental frequency of a reference voltage, as depicted in Fig. 6. Such operation lies on the idea of attaining the dot product ($d_p$) between the orthogonal ($x_\perp$) and the reference signal ($x$), comparing it with a null reference ($d^*_p$), and using the respective error into a PI controller for adjusting the angular frequency ($\omega_x$) of this orthogonal unitary signal, aiming at zero error. For educational purposes, or even fast learning for prototyping use, [13] presents a very accessible

explanation for this PLL methodology and facilitated directions for its implementation in PSIM.

Employing such methodology, and also following a similar circuitry approach as the IIR filter, the PSIM schematic depicted in Fig. 7 was built. Once again a squared wave is chosen as input signal, aiming to show that, even under very nonsinusoidal voltage conditions, the PLL should be able to synthesize an orthogonal component of the fundamental frequency from the mentioned input.

An interesting feature of such algorithm is the ability to provide a unitary orthogonal reference that can be used to synthesize any other multiple signal, meaning variant in multiple frequencies or even with phase displacements and amplitude variations. In Fig. 8 it is possible to notice that, from a square wave reference (in green), the PLL detects a unitary sinusoidal orthogonal signal (in red), and also allows the generation of a signal in-phase with the fundamental frequency of the input (in blue).

The experimental implementation is made likewise, generating the code from PSIM and loading onto the hardware. The result is shown in Fig. 9, where one can note the in-phase (2-cyan) and quadrature (orthogonal) (4-green) signals created from a square wave reference with offset (1-blue). Yet, in Fig. 10 it is shown how the in-phase reference may be handled for creating a 3rd harmonic signal (4-green). This functionality given by the flexibility of some PLL algorithms, for instance, is highly useful when generating control references for an active power filter responsible for selectively mitigating harmonic pollution in a grid.

### *C. Three-level PWM Generation*

Concerning operation control of power converters, the pulse width modulation (PWM) is one of the most used techniques. SimCoder elements can generate this type of digital signals based on a given reference. Taking the SimCoder "1-ph PWM" element as example, it allows full-bridge single-phase inverters [15] to be controlled by the respective output modulation signals.

This PWM generation is done by comparing a carrier wave, which may be configured as a sawtooth or triangular signal, with the control reference. However, the SimCoder PWM elements can create a two-level modulation, which results in inverter outputs synthesized by $+V_{DC}$ and $-V_{DC}$ voltage levels. The work in [2] controls a single-phase inverter through PSIM generated code based on this type of modulation. Although effective, with a three-level PWM, also called unipolar modulation [15], lower harmonic content is generated, being more suitable for active filtering applications. The difference between the above mentioned modulation and the two-leveled one, consists in generating as output, three voltage levels ($+V_{DC}$, 0, $-V_{DC}$).

The proposed open loop scheme in Fig. 11 is utilized as example of unipolar PWM generation using SimCoder blocks. Note that two single-phase PWM blocks (I and II) are used, being each one configured for different PWM modules. For such topology, the modulation signal "m" generates digital complementary commands (A and B ports) for the first leg of the inverter (switches: upper S1_PWM and lower S4_PWM), and for the second leg (upper S3_PWM and lower S2_PWM). Note that the modulation signal of the second leg goes through

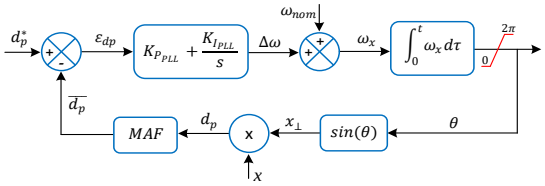


Fig. 6. Methodology of PLL algorithm.

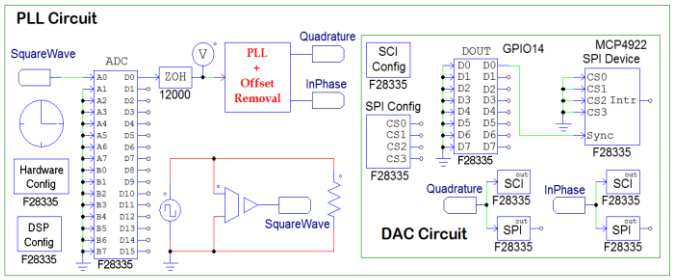


Fig. 7. PSIM circuitry for the PLL synchronism algorithm.

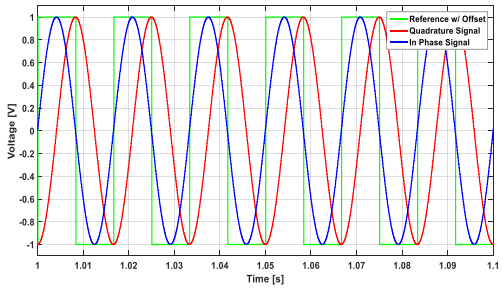


Fig. 8. Simulation result for the PLL in PSIM.

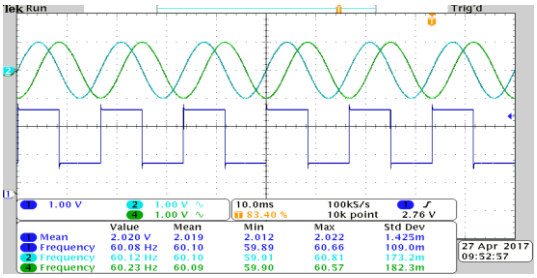


Fig. 9. Experimental result of the PLL algorithm.

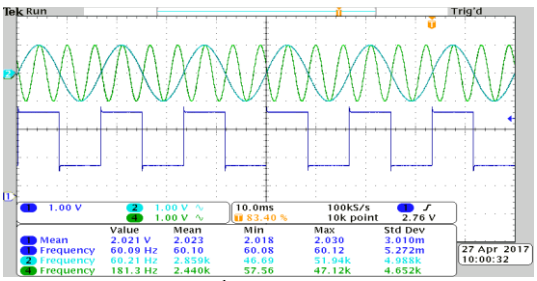


Fig. 10. DSP generating a 3rd harmonic signal from the PLL output.

a negative unitary gain to generate the 180° phase displacement required for the unipolar modulation, as shown in Fig. 12. It is reinforced that using C blocks in auto-code generation provides scalability of the tool, for instance, allowing easier manipulation of variables for several applications. The mentioned scalability is justified by the capability of providing a more flexible way to handle a reference signal used in the PWM generation. The explanations of the following two examples allows a better understanding of this statement.

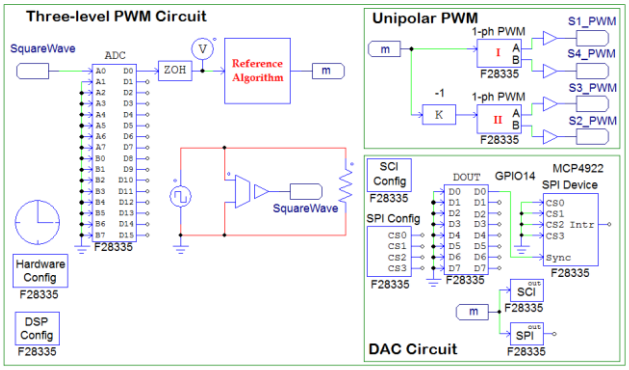


Fig. 11. PSIM circuitry for the PWM generation example.

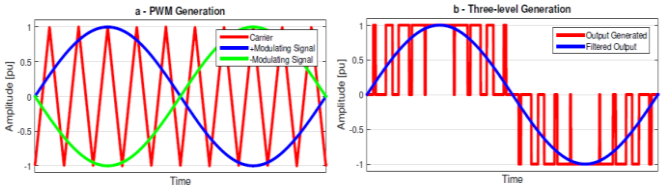


Fig. 12. Three-level PWM generation.

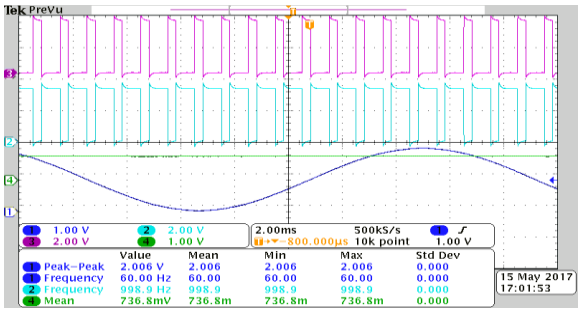


Fig. 13. PWM generated with a constant reference.

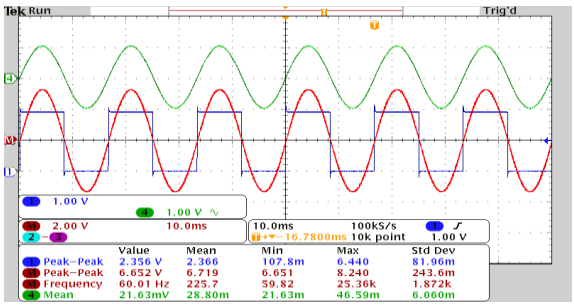


Fig. 14. 12kHz three-level PWM over a low-pass filter.

For educational purposes, an example of PWM generation is given by using the PWM block "I" from Fig. 11. For this case, it is considered a triangular carrier wave with 1V peak-to-peak, 0V offset and 1kHz of switching frequency. For the reference "m", it is proposed the implementation of a RMS (root mean square) algorithm responsible for calculating this respective value from a sinusoidal input, which is generated by using a function generator set for creating a 60Hz sinusoidal signal, with 2V peak-to-peak, as shown in the experimental result of Fig. 13. The RMS calculation is performed inside the C block with very few code instructions.

The mentioned RMS value is then used as the reference signal for the PWM. For such case, the respective RMS value (4-green) is close to the 0.707V expected. Thus, the duty cycle of the PWM is around 70%, as depicted in the PWM digital pin output (2-cyan) of the "port A", and its complementary "port B" (3-purple).

Furthermore, consider the circuitry of Fig. 11 without modifications: a three-level PWM generation is used to command a full-bridge inverter [16], as also shown in Fig. 15. The AC output voltage of the inverter is given by the combination of the modulation levels, having positive voltage supply when "S1_PWM" is closed, and negative when "S3_PWM" is closed. For such case, in both PWM blocks (I and II), a carrier wave with 2V peak-to-peak, -1V offset, and 12kHz as switching frequency is considered. Just as previously, the earlier PLL algorithm is employed and its unitary output signal is set as the reference "m" of the PWM.

Since the output of the PWM digital pins are pulsed signals, on a such high frequency, it is not very straightforward to present those results on a didactic way. Therefore, the AC output signal generated under a three-level PWM is shown in Fig. 14 by passing the difference of the digital outputs pins (S1_PWM and S3_PWM) on a low pass filter implemented on the oscilloscope. One can note that, giving an input reference with offset (1-blue) for the ADC, the PLL synthesizes the in-phase signal (4-green), and the three-level PWM output, which is passed through the filter (M-red), adequately follows its control reference.

It is restated that the most adequate way to evaluate the PWM generation, by auto generating code from PSIM, would be watching the real voltage and current outputs of a physical inverter. However, since an experimental inverter prototype is still under development, these particular results will be presented in a future work. Moreover, the designer must just keep in mind that, for more elaborated cases, memory and processing capacity are limited in the F28335. Therefore it is advisable, for instance, to create a "set bit – clear bit" interruption routine [17] within the CCS compiled code to ensure correct operation.

## IV. DISCUSSIONS ON CONSTRAINTS

For the purpose of acquainting beginner users before committing effort to learn how to use this tool, the following points are risen:

- Elements availability: due to the fact that the SimCoder environment only accepts digital domain simulation, there is a limited number of elements that can be used. For instance, only a few digital filters for fast design are provided, forcing the user to create his/her own, through Z domain functions or C blocks, as in the example earlier shown here;
- C blocks: one must keep in mind that, if any C codes are desired to be used for handling data, only simplified blocks are available. In such blocks, under such application, the available code prompt works as a "main()" function. Therefore, if one defines any other C functions inside it, those will not be accessible for the sequential running of the DSP code. That happens because PSIM's automatic generation allocates such codes inside interruption calls. Hence, one would have a function definition inside such interruptions, which are also functions. A possible way out of that could be through the adequacy of the code inside the CCS;
- PWM Generators: this may be one of the most critical constraints of this tool, and it is directed to users who wish to employ closed loop control schemes for

converters. It is very important for the user to know that all the respective PWM generation blocks, as the "1-ph PWM" aforementioned, have an inherited digital unit delay ($Z^{-1}$) in their construction [4]. The impact of that relates to likely unstable conditions of the system on simulations. For example, Fig. 15-a presents a simulation scheme to show the condition of such instability.

Looking at this last particular PWM simulation, the inverter was supposed to act as a multifunctional unit, injecting all the active and non-active currents drawn by the nonlinear load. Disregarding the above-mentioned delay, Fig. 15-c shows the outcome of the unstable condition. On the opposite, when this delay is accounted for in the controller design, the inverter current (in blue) follows the reference (in red), resulting in the desired behavior depicted in Fig. 15-b. In Fig. 16 the control scheme adopted for the inverter is presented, where "$C_i(s)$" is the controller transfer function, "$G_i(s)$" is the inverter model, and "$K_i$" is the current transducer gain. "PWM(s)" stands for the dynamic of the modulation, and "$V_{DC}$" is the inverter static gain.

Since remodeling controllers is not a trivial task, it is also mentioned that another possibility to work around that issue would be to simulate with a higher carrier frequency, multisampling the output and making the delay irrelevant in a steady-state condition [16]. Finally, when it comes to applications where faster and more precise PWMs are needed there is the possibility to use external PWM modules connected to the I2C (Inter-Integrated-Circuit) bus of the F28335 Kit. However one must keep in mind that there are no SimCoder elements for that, requiring I2C codes to be implemented with C blocks.

Currently, some ongoing work is focusing on effectively controlling a single-phase inverter in a closed loop scheme by using DSP codes generated from PSIM.

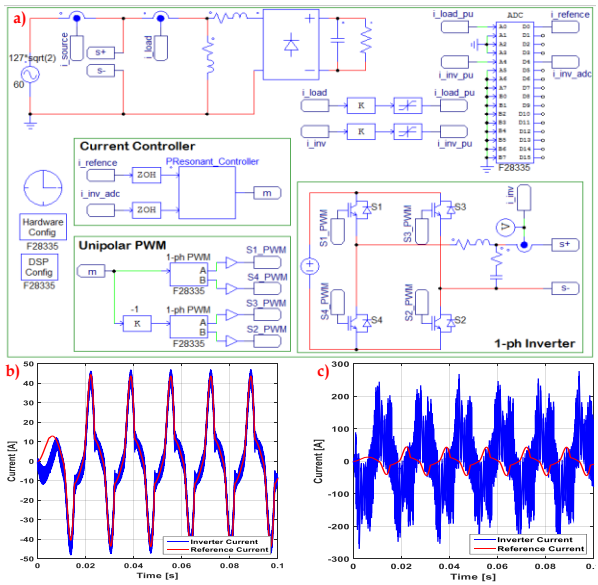


Fig. 15. Accounting for the delay of PWM generation block.

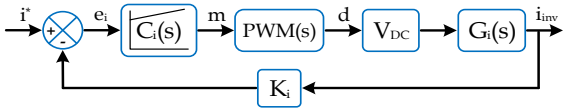


Fig. 16. Block diagram of current control with three-level PWM.

## V. CONCLUSIONS

Auto-code generation for DSP programming using PSIM is an interesting tool and can provide a fast prototyping alternative for the project of power converters. Moreover, it was shown that it could be well suited for educational purposes, once it allows easy experimental handling. Nonetheless, several limitations, such as the unit delay inherited on PWM generation elements, and the implications of Simplified C blocks with functions, were pointed out. Thus, it is finally concluded that, by taking those issues into consideration when building PSIM simulations, the employment of the SimCoder module offers a reliable and accessible way to program DSPs.

## ACKNOWLEDGEMENTS

The authors are grateful to FAPESP (2016/08645-9) and CAPES Brazilian agencies, and the Research Council of Norway (f261735/H30) for supporting the NB_POCCREI project and the related international collaboration.

## REFERENCES

[1] Jacobs, J.; Detjen, D.; Karipidis, C.; Doncker, R. "Rapid Prototyping Tools for Power Electronic Systems: Demonstration with Shunt Active Power Filters", IEEE Trans. Power Electron. vol. 19, pp. 500-507, March 2004.

[2] Rodrigues, M.; Silva, N.; Nunes, W. "Aplicação do software PSIM para o uso em prototipagem rápida através de um processador digital de sinais", COBENGE, 2014. (*in Portuguese*)

[3] Morkoc, C.; Onal, Y.; Kesler, M. "DSP based embedded code generation for PMSM using sliding mode controller", 16th IEEE PEMC, Sep. 2014.

[4] POWERSIM, "SimCoder User's Guide", Powersim Inc., v. 11.0, Sep. 2016. Available at: https://powersimtech.com/drive/uploads/2016/12/SimCoder-v11-User-Manual.pdf.

[5] Texas Instruments, "TMS320C2000 Experimenter Kit Overview", TI - Quick Start Guide, Feb. 2011. Available at: http://www.ti.com/lit/ug/sprufr5f/sprufr5f.pdf.

[6] Texas Instruments, "C2000 Real-Time Microcontrollers", C2000 Microcontrollers Brochure, June 2016. Available at: http://www.ti.com/lit/sg/sprb176ad/sprb176ad.pdf.

[7] Texas Instruments, "TMS320C2xx/C24x Code Composer User's Guide", TI - Manuals, Oct. 2000. Available at: http://www.ti.com/lit/ug/spru490/spru490.pdf.

[8] POWERSIM, "Auto Code Generation for F2833X Target", PSIM Tutorial, Oct. 2016. Available at: https://powersimtech.com/drive/uploads/2016/12/Tutorial-Auto-Code-Generation-for-F2833x-Target.pdf.

[9] POWERSIM, "Using SPI in F2833X/F2803X Target", PSIM Tutorial, Oct. 2016. Available at: https://powersimtech.com/drive/uploads/2016/12/Tutorial-Using-SPI-in-F2833x-F2803x-Target.pdf.

[10] Sozanski, K. "Selected Active Power Filter Control Algorithms", Digital Signal Processing in Power Electronics Control Circuits, pp. 145-204, Ed. Springer, 2013.

[11] Smith, S. "The Scientist and Engineer's Guide to Digital Signal Processing", 2nd Ed., Analog Devices, 1999.

[12] Oppenheim, A. "Discrete-Time Signal Processing", 3rd Ed., Prentice-Hall, 2009.

[13] Simoes, M.; Farret, F. "Modeling Power Electronics and Interfacing Energy Conversion Systems", 1st Ed., Wiley-IEEE Press, 2017.

[14] Marafao, F.; Deckmann, S.; Pomilio, J.; Machado, R. "Metodologia de Projeto e Análise de Algoritmos de Sincronismo PLL", Eletrônica de Potência – SOBRAEP, vol. 10, nº 1, June 2005. *(in Portuguese)*

[15] Rashid, M. H. "Power Electronics Handbook", 3rd Ed., 2011.

[16] Buso, S.; Mattavelli, P. "Digital Control in Power Electronics", 2nd Ed., Morgan&Claypool, 2015.

[17] Texas Instruments, "TMS320x2833x,2823x System Control and Interrupts", Reference Guide, Sep. 2007. Available at: http://www.ti.com/lit/ug/sprufb0d/sprufb0d.pdf.