Model Predictive Control of Buck Converter Using Easy FPGA Implementation

POORNIMA GULUR CHAKRAPANI[1], GUSTAVO CERDA-VILLAFANA[2], ROBERT S. BALOG[1], FRANCISCO JAVIER MORENO VAZQUEZ[2]
[1]Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX 77843-3127, U.S.A.
[2]Department of Electronics Engineering, University of Guanajuato, Salamanca, Gto. 36885, MEXICO.

Abstract: - Model predictive control of a DC-DC buck converter is implemented in a Field Programmable Gate Array (FPGA) and tested using a Hardware-in-the-Loop system. A step-by-step description is presented in this paper, showing the process of first developing the complete model of converter and control in MATLAB/Simulink then, converting the control into a Hardware Description Language (HDL) program, including the specifications of the FPGA target board used. Programming the FPGA includes the analog-to-digital converter and the constraints of the FPGA chip. Finally, the system is tested using a Typhoon HIL platform.

Key-Words: - Model Predictive Control, MATLAB simulation, FPGA implementation, Buck converter, Hardware in the Loop.

Received: April 21, 2025. Revised: August 24, 2025. Accepted: September 19, 2025. Published: December 5, 2025.

1 Introduction

Model predictive control (MPC) is an advanced control technique gaining more adoption in the field of power electronics. It has several advantages [1-4], for example:

- Can be adapted to a great variety of systems.
- Predicts future system behavior over a finite horizon to take defined actions according to predefined rules.
- Can manage multiple control objectives simultaneously.
- Can adapt to changing system dynamics.
- Can handle nonlinear systems.
- Constraints are easy to implement

MPC controllers have several classifications, according to different authors, explicit and implicit MPC [1,5,6] where the optimization problem of explicit MPC is solved offline and integrated as a look-up table containing the optimal solution as a function of the state of the system, while in implicit MPC the optimization problem is solved at each sampling instant. Direct and indirect control [1,7], where the direct control generates the modulating signal, without using a modulator, while the indirect control uses a fixed frequency, usually by a Pulse Width Modulation and the control action adjusts the duty ratio.

This work will present an implicit MPC with direct control, and its objective is to show a step-by-step description of implementing an MPC of a buck converter in an FPGA board and verify its behavior using a Hardware-in-the-Loop system.

2 Problem Formulation

MPC uses a dynamic model to predict the future behavior of a system, in this case, a buck converter, based on current and upcoming inputs [1,3,5]. The output variables are then evaluated using a cost function representing the desired behavior of the system, then, by minimizing a cost function, the optimal behavior is generated.

Implementation of an MPC controller in an FPGA requires expertise many electrical engineers do not possess, therefore, to guide them in the process, this work describes each step with a simple Buck converter. Once the implementation is achieved, the level of complexity of MPC controllers can be increased to any level, barred from the limitations of the hardware employed.

2.1 Buck converter

A buck converter is a dc-dc circuit used to convert a high DC input voltage to a lower DC output voltage, defined in this work as V_{ref}. A simple configuration is shown in Figure 1. The switching device generates two states, one when it is closed and the other when it is open.

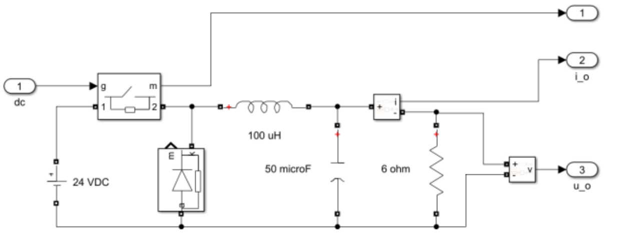

Figure 1. Buck circuit modeled in MATLAB/Simulink

For the first case, the equations for voltage and current governing the circuit are:

$$V_{in} = L\frac{di_L}{dt} + v_C \quad (1)$$

$$i_L = C\frac{dv_C}{dt} + \frac{v_C}{R} \quad (2)$$

The second case, when the switch is open:

$$0 = L\frac{di_L}{dt} + v_C \quad (3)$$

Using equation 2 in terms of dv_c/dt:

$$\frac{dv_C}{dt} = \frac{1}{C}i_L - \frac{1}{RC}v_C \quad (4)$$

Now using Euler's method to obtain a discrete representation of Eq. (4)

$$\frac{v_C(k+1) - v_C(k)}{T_S} = \\ = \frac{1}{C}i_{L(k)} - \frac{1}{RC}v_C(k) \quad (5)$$

And solving for v_C(k+1) defines Eq. 6

$$v_C(k+1) = v_C(k) + T_S\left[\frac{1}{C}i_L(k) - \\ - \frac{1}{RC}v_C(k)\right] \quad (6)$$

v_C(k+1) is the voltage value one sampling time ahead. Now a cost function Cf is obtained subtracting v_C(k+1) from V_{ref} and it will define the state of the switch (on-off).

$$Cf(k) = V_{ref} - v_C \quad (7)$$

With equations (6) and (7) it is possible to implement a model predictive control, obtaining $v_C(k+1)$ from the present values of $v_C(k)$ and $i_L(k)$ and comparing it to the desired output voltage V_{ref} of the buck converter. The sequence of the algorithm is:

1) Measure of v_C and i_L.
2) Predict the future output (Eq. 6)
3) Obtain error from cost function
4) Turn on the switch when Cf >0 and turn it off when Cf≤0.

2.2 FPGA implementation

Hardware implementation of an MPC controller for power electronics applications can be achieved by microcontrollers, Digital Signal Processors (DSPs), and Field Programmable Gate Arrays (FPGAs) [7]. While microcontrollers and DSPs [8] can be programmed using C code, FPGAs are programmed with Hardware Description Language (HDL) like VHDL, Verilog and System Verilog. Because these languages require a skill set not many engineers are experts on, there are several tool sets developed to generate HDL code, like LabView G language, Xilinx System Generator and MATLAB HDL Coder.

There are examples in the literature of MPC controllers implemented in a FPGA [9-11], but usually, they are focused on describing the system and not the process of implementing it in the FPGA. [12] presents a detailed description of the FPGA implementation of a nonlinear finite control set model predictive control (FCS-MPC) but the reader still needs a comprehensive knowledge of HDL to implement the blocks and equations described.

FPGA main advantages are its parallel and fast processing, making it ideal for complex MPC systems where fast computation of two or more models and cost functions are necessary.

To facilitate the implementation of the buck converter, the MATLAB HDL Coder will generate the files needed to program the FPGA. Other files that are also required will be described in the proper section.

2.3 Typhoon Hill implementation

Typhoon Hill 4004 is a platform that allows Hardware-in-the-Loop (HIL) simulations of power electronics and e-mobility systems in a virtual environment [13-16], accelerating tests in secure conditions. The interface is similar to other software programs like MATLAB/Simulink, LabVIEW or

Proteus, allowing new users to implement circuits or complex systems easily and in a friendly interface. The interaction with external components is done through analog and digital inputs/outputs that can be configured to adapt to the external signals.
The buck converter is implemented in the Schematic Editor, shown in Figure 2. Here it is necessary to add a ground block, and, in the switching element, the driving signal is declared.

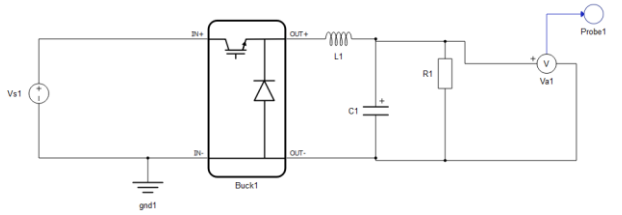

Figure 2. Buck converter circuit in Typhoon Hill software.

Once the circuit is verified and the HIL404 simulator hardware device is connected to the computer, the HIL SCADA is open, and the simulation can be executed.

3 Problem Solution

3.1 MATLAB/Simulink simulation

The Buck converter circuit is simulated in MATLAB/Simulink (version R2020b) as it is shown in Figure 1. Table 1 presents the parameters of the Buck converter:

Table 1. Parameters of the buck converter

Parameter	Symbol	Value
Input voltage	V_s	24V
Load	R	6Ω
Capacitance	C	50µF
Inductance	L	100µH
Sampling Freq	F_{sw}	100KHz

The control is written in a function using the equations described in the previous section:

```
function Dp = fcn(Vref, u_o, i_o)
% Switching frequency = 100 kHz
T_s = 0.00001;
C = 0.00005;   % Capacitance 50uF
R = 6;               % LoadResistance 6 Ohms
L = 0.0001;     % Inductance 100uF
Vin = 24; % Source voltage
% Simplified discrete model
Vp = u_o+(1-T_s/(R*C))*u_o+T_s*i_o/C;
CFon = (Vref -Vp);
if CFon > 0
    Dp = 1;
else
    Dp = 0;
end
```

The output value from the cost, function will make the output signal Dp equal to 1 (keeping the switch on) when it is greater than 0, if it is equal or less than 0, Dp will be 0 (turning the switch off).
Once the circuit and its control are tested, the control is converted to blocks from the HDL Coder toolbox in Simulink. Using basic blocks facilitates the generation of HDL code, therefore, the control (see Figure 3) is separated into three main blocks, according to their function.

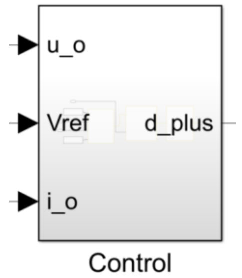

Figure 3. Control block in MATLAB/Simulink

Inside the Control block there are three blocks, as it is shown in Figure 4.

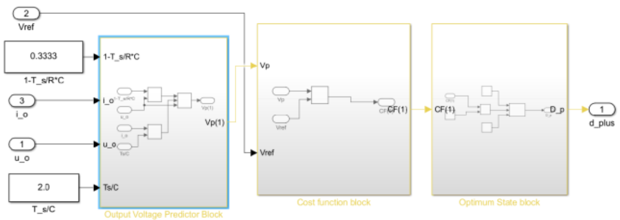

Figure 4. The three blocks inside the Control block

The first block inside the Control block is the output voltage predictor, shown in Figure 5.

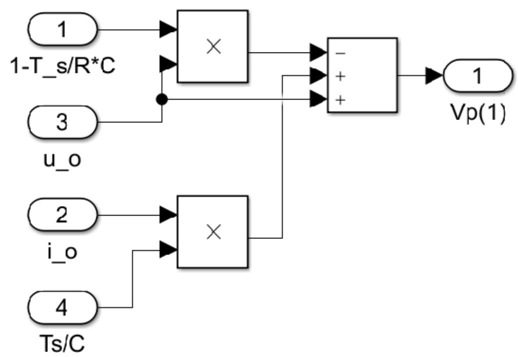

Figure 5. Output voltage predictor block

In this block, two inputs involve parameters that include a division. This operation is executed before sending the value to the control block to avoid performing it inside the FPGA because a division demands more resources, and it can be done externally because it is done with fixed value parameters.

The second block, shown in Figure 6 is the Cost function. In this case the only operation is the subtraction described by Equation 5.

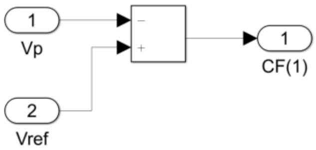

Figure 6. Cost function block

The third and last block is Optimum State (see Figure 7). It involves a relational operator to compare the CF(1) signal, from the Cost function to a V_{set}=0. If CF(1) is bigger to V_{set}, D_p will be 1, otherwise will be 0.

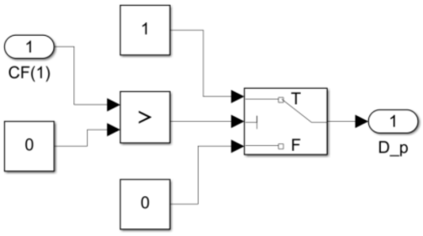

Figure 7. Optimum state block

3.2 HDL code generation

To generate the DHL code, in the pop-up window on the Control block, it is selected HDL Code->HDL coder properties->Target, and the specifications of the FPGA board are introduced, see Table 2. All signals entering the Control block must be fixed point, therefore fixdt(x,x,x) will define all the signals entering, leaving and inside the Control block. Because voltage and current from the circuit (u_o and i_o) are defined as double, a Data Type Conversion block is needed for each one. In this system all the signals range from 0.666 to 24 in the integer part, hence, the values selected for fixdt are 7 bits for the integer and 10 bits for the decimal part. The most significant bit is the sign bit, hence, fixdt is defined as fixdt(1,18,10).

Table 2. Parameters of Target Device

Synthesis tool	Altera Quartus II
Family	Cyclone V
Device	5CSEMA5F31C6
Target Frequency (MHz)	50

Before generating the HDL code, all the components inside the Control block that include Sample time should not have "inf" but "-1" (for inherited) or the sampling time of the project (in this case, 100μs). MATLAB/Simulink has the option to check the compatibility of the model/subsystem and fixing any issue.

Once the "Generate HDL code for subsystem" is selected five vhdl files are created:
Control.vhd – Global entity of the MPC
Control_pkg.vhd – Package defining some signals
The last three files are from each one of the blocks described in subsection 3.1:
Cost_function_block.vhd
Optimum_State_block1.vhd
Output_Voltage_Predictor_Block.vhd

These files are included in a project as described in the following section.

3.3 Quartus II Implementation

Quartus II is an intuitive and easy to follow program by Intel. Besides that, the manual of the FPGA board gives instructions on how to create a project, add the FPGA, files and define tools. In this project a DE1-SoC FPGA board is used to implement the control of the buck converter.

In this board the analog-to-digital converter is LTC2308. It has a maximum of 500ksps with a 12-bit resolution. The communication between the ADC and the FPGA is via SPI and the ADC requires a maximum conversion time of 1.6 μs. After that, the FPGA starts sending 12 clock pulses and through SDI the 6-bit configuration and, at the same time, receiving the 12 bits from the conversion. A diagram from [17] is presented in Figure 8.

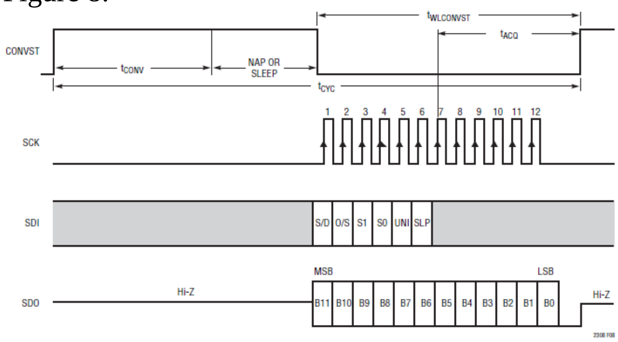

Figure 8. LTC2308 ADC conversion timing

The signals from the ADC are 12-bit and must be adjusted to the 18-bit signals in the control block. It is done with the following lines:

```
Temp1 <= "000" & vol1 & "000";
Voltage1 <= std_logic_vector(signed(Temp1));

Temp2 <= "000" & curr1 & "000";
Current1 <= std_logic_vector(signed(Temp2));
```

vol1 and curr1 are the 12-bit signals coming from the ADC. The most significant 5 bits are left in the integer part because they will contain a maximum value of 24 (for the input voltage) and the rest is left in the decimal part. The global file, including the five components for the files generated by the HDL Coder are included in a project as shown in Figure 9.

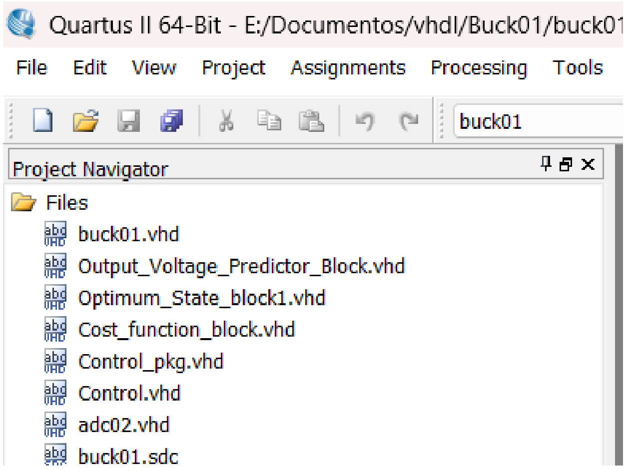

Figure 9. VHDL files in a Quartus II project.

The inputs and outputs of the signals (ports in VHDL) are defined according to the FPGA board manual in the Pin Planner window, and for the DE1-SoC board, a section of this window is shown in Figure 10.

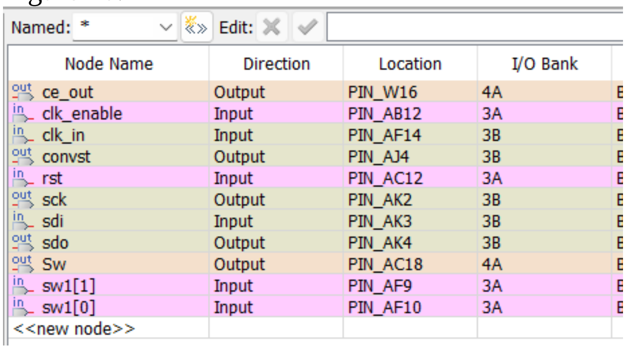

Figure 10. Pin assignment in the Pin Planner

Table 3 shows a description of each signal.

Table 3 Signal description for the FPGA implementation

ce_out	Output equal to clk_enable
clk_enable	Enables the operation of the controller
clk_in	Board clock input (50MHz)
convst	Activates ADC data transfer
rst	Reset
sck	Clock signal for ADC
sdi	Data in from ADC
sdo	Data out for ADC
sw	Signal for switching element (d_plus)
sw1[1]	Voltage level selector 1
sw1[0]	Voltage level selector 2

sw1[1] and sw1[0] are signals connected to switches in the FPGA board to select four specific voltage levels: “00” 2V, “01” 6V, “10” 12V and “11” 20V. The project is compiled, and the FPGA is programmed by using the Programmer. Some FPGA boards are not automatically recognized by the computer. If this is the case, just update the drivers from the *drivers* folder inside the *quartus* folder in the installation folders.

3.4 Typhoon Hill verification

The circuit implementation in Typhoon Hill is shown in Figure 2 and the connections between the HIL404 simulator hardware device and the DE1-SoC FPGA are shown in Figure 11.

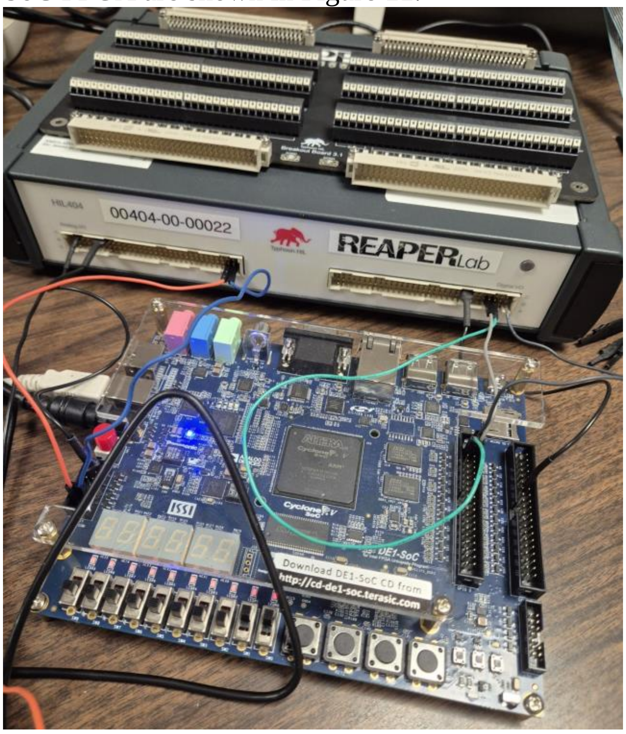

Figure 11. Wire connections between the HIL404 and the FPGA board.

For the communication between the FPGA board and the HIL404, 5 connections are required, and Table 4 shows the pins on each side.

Table 4. Pin connection between HIL404 and FPGA board.

Signal	FPGA board	HIL404
u_o	2 ADC_IN0	A1 Analog connector
i_o	4 ADC_IN2	A2 Analog connector
GND	10 GND	C31 Analog connector
d_plus	GPIO [0]	A1 Digital connector
GND	GPIO [11]	C6 Digital connector

Once the simulation starts, the plots for voltage and current can be visualized in HIL SCADA, which shows lower sampling curves, therefore for visualization the Capture/Scope widget is selected and the current and voltage are shown in Figures 12 and 13 respectively.

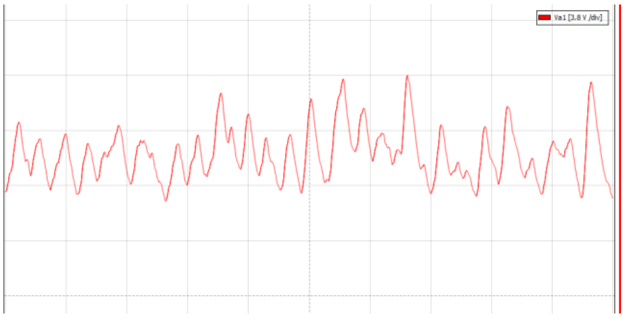

Figure 12. Voltage plot with the Capture/Scope widget.

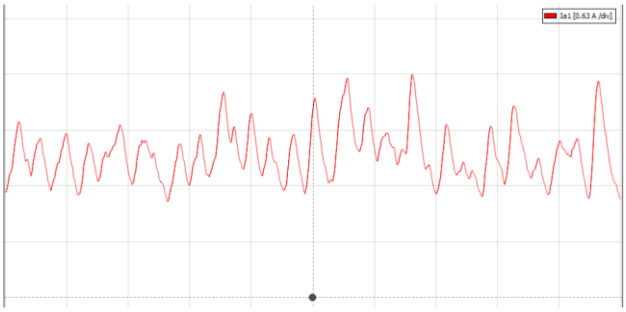

Figure 13. Current plot with the Capture/Scope widget.

The current and voltage signals generated by the HIL404 can be shown with an oscilloscope, like the screenshot showing the current curve in Figure 14.

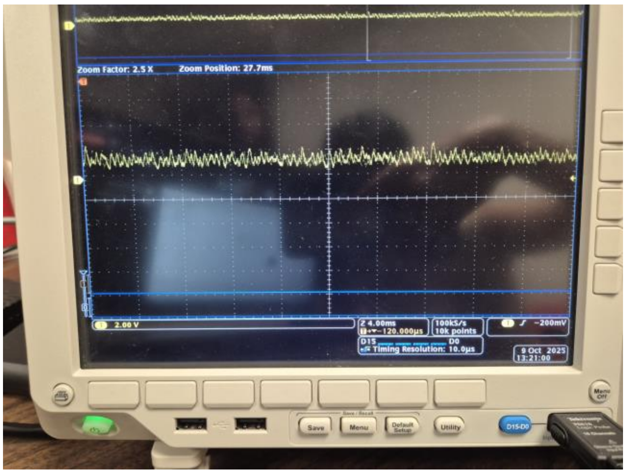

Figure 14. Current plot on an oscilloscope.

Finally, for comparison purposes, the current curve generated in MATLAB/Simulink is presented in Figure 15.

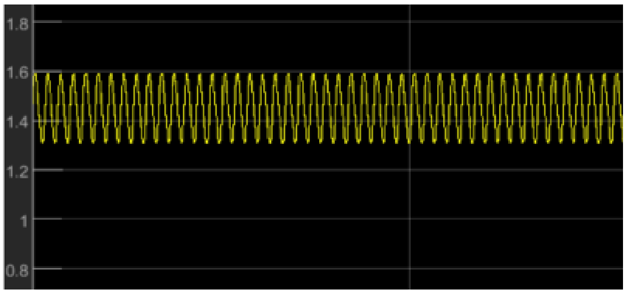

Figure 15. Current curve generated in MATLAB/Simulink.

4 Conclusion

An easy-to-follow implementation of an MPC controller is presented, describing the steps, from simulation to generating HDL code, programming an FPGA board and testing the system in a hardware-in-the-loop platform. A simple case of a Buck converter was chosen to focus on the process rather than the details of a complex MPC system. First is presented the simulation in MATLAB/Simulink, observing the behavior of the system and adjusting the parameters to reach the desired outcome, allowing the generation of HDL code to program an FPGA. Finally, the verification of the system with a Hardware-in-the-Loop tool demonstrated the feasibility and robustness of the MPC controller and its operational robustness.
Once the process is complete, the complexity of the control algorithm can be scaled up to more demanding MPCs with multiple control objectives.

References:

[1] Rodriguez J. and Cortes, P., *Predictive Control of Power Converters and Electrical Drives*, Wiley-IEEE Press, 2012. ISBN-13: 978-1119942641

[2] Augustine, M. T. (2023). Model predictive control using matlab. *arXiv preprint arXiv:2309.00293*.

[3] P. Karamanakos, E. Liegmann, T. Geyer and R. Kennel, Model Predictive Control of Power Electronic Systems: Methods, Results, and Challenges, *IEEE Open Journal of Industry Applications*, vol. 1, pp. 95-114, 2020, doi: 10.1109/OJIA.2020.3020184.

[4] P. Karamanakos, T. Geyer and S. Manias, "Direct model predictive current control of DC-DC boost converters," 2012 15th International Power Electronics and Motion Control Conference (EPE/PEMC), Novi

Sad, Serbia, 2012, pp. DS2c.11-1-DS2c.11-8, doi: 10.1109/EPEPEMC.2012.6397294.

[5] Bemporad, A. Explicit Model Predictive Control. In *Encyclopedia of Systems and Control*; Baillieul, J., Samad, T., Eds.; Springer International Publishing: Cham, Switzerland, 2021; pp. 744–751.

[6] Alessio, A., Bemporad, A. (2009). A Survey on Explicit Model Predictive Control. In: Magni, L., Raimondo, D.M., Allgöwer, F. (eds) Nonlinear Model Predictive Control. *Lecture Notes in Control and Information Sciences*, vol 384. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-01094-1_29

[7] Purraji, M.; Zamiri, E.; de Castro, A. Easy and Straightforward FPGA Implementation of Model Predictive Control Using HDL Coder. *Electronics* 2025, 14, 419. https://doi.org/10.3390/electronics14030419

[8] M. Metry, N. Guler, W. Rohouma, and R. S. Balog, "Performance Comparison of dSPACE and DSP MPC Implementation in PV Power Optimizer," in *IEEE Energy Conversion Congress and Exposition (ECCE)*, Nashville, TN, 29 Oct 29 - 02 Nov, 2023, pp. 6266-6272. doi: 10.1109/ECCE53617.2023.10362081

[9] T. Kosan, J. Talla, S. Janous and V. Blahnik, "FPGA-Based Accelerator for Model Predictive Control of Induction Motor Drive," *2018 18th International Conference on Mechatronics - Mechatronika* (ME), Brno, Czech Republic, 2018, pp. 1-6.

[10] R. Babojelić, B. V. Belina, Š. Ileš and J. Matuško, "FPGA Implementation of Set-based Model Predictive Control," *2022 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO)*, Opatija, Croatia, 2022, pp. 147-152, doi: 10.23919/MIPRO55190.2022.9803512

[11] V. Spinu, A. Oliveri, M. Lazar and M. Storace, "FPGA implementation of optimal and approximate model predictive control for a buck-boost DC-DC converter," *2012 IEEE International Conference on Control Applications*, Dubrovnik, Croatia, 2012, pp. 1417-1423, doi: 10.1109/CCA.2012.6402444.

[12] J. Escárate, R. López, A. L. Cedeño, J. C. Agüero, C. Silva and G. Carvajal, "FPGA Implementation of ADMM for Model Predictive Control in a DC/AC Converter," *2022 IEEE International Conference on Automation/XXV Congress of the Chilean Association of Automatic Control (ICA-ACCA)*, Curicó, Chile, 2022, pp. 1-6, doi: 10.1109/ICA-ACCA56767.2022.10005973.

[13] Zehra, S.S., Carnì, S., Hemmati, M., Faranda, R., Gruosso, G., Mussetta, M. (2025). Voltage Tracking of PV with a Nonlinear Controller in DC Microgrid: An Experimental Verification Using Hardware-in-the-Loop. In: Belenguer, E., Beltran, H. (eds) *ELECTRIMACS 2024. ELECTRIMACS 2024. Lecture Notes in Electrical Engineering*, vol 1275. Springer, Cham. https://doi.org/10.1007/978-3-031-73921-7_58

[14] Gunjal, M., Mathew, L., Shimi, S.L. (2025). Optimizing Battery Efficiency: Design and Implementation of a Fuzzy Logic Based Controller for Hybrid Energy Storage System in Electric Vehicles. In: Nanda, S.J., Yadav, R.P., Gandomi, A.H., Saraswat, M. (eds) *Data Science and Applications. ICDSA 2024. Lecture Notes in Networks and Systems*, vol 1237. Springer, Singapore. https://doi.org/10.1007/978-981-96-1185-0_14

[15] Zeb, K., Busarello, T. D. C., Ul Islam, S., Uddin, W., Raghavendra, K. V. G., Khan, M. A., & Kim, H.-J. (2020). Design of Super Twisting Sliding Mode Controller for a Three-Phase Grid-connected Photovoltaic System under Normal and Abnormal Conditions. *Energies*, *13*(15), 3773. https://doi.org/10.3390/en13153773

[16] Siza, J., Llanos, J., Ortiz-Villalba, D., Cajas, M., Moya, A. (2025). Bidirectional Control for Battery Considering SOC, Implemented in Typhoon HIL. In: Olmedo Cifuentes, G.F., Arcos Avilés, D.G., Lara Padilla, H.V. (eds) *Emerging Research in Intelligent Systems. CIT 2024. Lecture Notes in Networks and Systems*, vol 1347. Springer, Cham. https://doi.org/10.1007/978-3-031-87704-9_10

[17] Linear Technology, “Low Noise, 500ksps, 8-Channel, 12-Bit ADC” LTC2308 Data sheet. Jun. 2016 [Revised Oct. 2016]

Contribution of Individual Authors to the Creation of a Scientific Article (Ghostwriting Policy)

The authors equally contributed to the present research, at all stages from the formulation of the problem to the final findings and solution.

Sources of Funding for Research Presented in a Scientific Article or Scientific Article Itself

No funding was received for conducting this study.

Conflict of Interest

The authors have no conflicts of interest to declare that are relevant to the content of this article.

APPENDIX A - ADC code

```
entity adc2 is
Port (
clk :      in std_logic;
reset:     in std_logic;
sdi_x :    in std_logic;
sdo_x :    out std_logic;
sck_x :    out std_logic;
convst :   out std_logic;
sal1:      out std_logic_vector(11 downto 0);
sal2:      out std_logic_vector(11 downto 0)
 );
end adc2;

architecture Behavioral of adc2 is
-- S/D     1
-- O/S     0
-- S1      0
-- S0      0
-- UNI     0
-- SLP     0
signal temp, Flag: STD_LOGIC := '0';
signal counter : std_logic_vector(5 downto 0) := (Others => '0');
signal reg1, reg2 : std_logic_vector(11 downto 0) := (Others =>
'0');
signal Index: integer range 0 to 11 := 11;
signal Config1: std_logic_vector(5 downto 0) := "100000";
signal Config2: std_logic_vector(5 downto 0) := "100100";
begin

general: process (reset, clk)
begin
    if (reset = '1') then
      temp <= '1';
      counter <= (Others => '0');
      convst <= '1';
    elsif rising_edge(clk) then
      counter <= counter + '1';
      if (counter < 24) then
        temp <= '1';
        convst <= '1';
        Index <= 11;
      elsif (counter < 36) then
        temp <= NOT temp;
        convst <= '0';
        if (counter(0) = '0') then
          if (Flag = '0') then
            sdo_x <= Config1(Index-6);
            reg1(index) <= sdi_x;
          else
            sdo_x <= Config2(Index-6);
            reg2(index) <= sdi_x;
          end if;
          index <= index - 1;
        end if;
      elsif (counter < 48) then
        temp <= NOT temp;
        convst <= '0';
        if (counter(0) = '0') then
          if (Flag = '0') then
            reg1(index) <= sdi_x;
          else
            reg2(index) <= sdi_x;
          end if;
          index <= index - 1;
        end if;
      else
        counter <= (Others => '0');
        convst <= '1';
         if (Flag = '0') then
            sal1 <= reg1;
          else
            sal2 <= reg2;
          end if;

        Flag <= NOT Flag;
      end if;
    end if;
  end process;

  sck_x <= temp;

end Behavioral;
```

APPENDIX B - Global entity

```
entity buck01 is
Port (
clk_in:     in std_logic;
clk_enable:   in std_logic;
rst :       in std_logic;
sw1:      in std_logic_vector(1 downto 0);
sdi :       in std_logic;
sck :       out std_logic;
sdo :       out std_logic;
convst :    out std_logic;
ce_out :    out std_logic;
Sw :            out std_logic
 );
end buck01;

architecture Behavioral of buck01 is

component Control IS
  PORT( clk : IN    std_logic;
        reset :   IN    std_logic;
        clk_enable :   IN    std_logic;
        u_o :   IN    std_logic_vector(17 DOWNTO 0);
        Vref :   IN    std_logic_vector(17 DOWNTO 0);
        i_o :   IN    std_logic_vector(17 DOWNTO 0);
        ce_out :   OUT   std_logic;
        d_plus :   OUT   std_logic
        );
end component;

component F_divider is
Port (
clk_in : in  STD_LOGIC;
reset  : in  STD_LOGIC;
clk_out: out STD_LOGIC
 );
end component;

component adc2 is
Port (
clk :       in std_logic;
reset:      in std_logic;
sdi_x :     in std_logic;
sdo_x :     out std_logic;
sck_x :     out std_logic;
convst :    out std_logic;
sal2:       out std_logic_vector(11 downto 0);
sal1:       out std_logic_vector(11 downto 0)
 );
end component;

signal clk_x, sdo_x : std_logic;
signal R : std_logic_vector(17 downto 0) := "000001100000000000"; --6 ohms
signal Vref : std_logic_vector(17 downto 0);
signal Voltage1, Current1 : std_logic_vector(17 downto 0);
signal vol1, curr1 : std_logic_vector(11 downto 0);
signal Temp1, Temp2 : std_logic_vector(17 downto 0);

begin

--Section for selecting four different Vref
Vref <=   "000000100000000000" when sw1="00" else --2
          "000001100000000000" when sw1="01" else --6
          "000011000000000000" when sw1="10" else --12
          "000101000000000000";                   --20


Temp1 <= "000" & vol1 & "000";
Voltage1 <= std_logic_vector(signed(Temp1));

Temp2 <= "000" & curr1 & "000";
Current1 <= std_logic_vector(signed(Temp2));

u1: adc2 port map (clk_x, rst, sdi, sdo_x, sck, convst, vol1, curr1);
u2: Control port map (clk_x, rst, clk_enable, Voltage1, Vref, Current1, ce_out, sw);

sdo <= sdo_x;


end Behavioral;
```