Article

FPGA-Based Cost-Effective and Resource Optimized Solution of Predictive Direct Current Control for Power Converters

Deepa Sankar 1,*, Lakshmi Syamala 2, Babu Chembathu Ayyappan 1 and Mathew Kallarackal 2

1 Division of Electrical Engineering, Cochin University of Science and Technology, Ernakulam 682022, India; drcababu@gmail.com
2 Department of Electronics and Communication Engineering, Mar Athanasius College of Engineering, Kothamangalam 686666, India; laxmi.abhi@gmail.com (L.S.); kmathewmace@gmail.com (M.K.)
* Correspondence: deepa.eee@adishankara.ac.in; Tel.: +91-944-729-3774

Abstract: Recent advances in power converter applications with highly demanding control goals require the efficient implementation of superior control strategies. However, the real-time application of such control strategies demands high computational power that necessitates efficient digital controllers like field programmable gate array (FPGA). The inherent parallelism offered by FPGAs minimizes the execution time and exhibits an excellent cost-performance trade-off. In addition, rapid advancements in FPGA technology with a broad portfolio of intellectual property (IP) cores, design tools, and robust embedded processors resulted in a design paradigm shift. This article proposes a low-cost solution for the resource-optimized implementation of dynamic, highly accurate, and computationally intensive finite state-predictive direct current control (FS-PDCC). The challenges for implementing complex control algorithms for power converters are discussed in detail, and the control is implemented in Intel’s low-cost non-volatile FPGA-MAX®10. An efficient design methodology using finite state machine (FSM) is adopted to achieve time/resource-efficient implementation. The parallel and pipelined architecture of FPGA provides better resource utilization with high execution speed. The experimental results prove the efficiency of FPGA-based cost-effective solutions that offer superior performance with better output quality.

Keywords: field programmable gate array; intellectual property core; finite control set model predictive control; finite state machine; resource optimization; voltage source inverter; analog to digital converter

1. Introduction

The embedded system technology with powerful and low-cost processors permitted high-performance digital controllers in industrial electrical systems. The advancement in power converter technology and its control is driven by applications like renewable energy, automotive, drives, and distributed generation, to name a few. Application-specific control goals and functionalities necessitate extensive research in power converters’ control [1,2].

Among the power converter control strategies, model predictive control (MPC) [3] received more attention due to its ability to control multivariable systems and ease of including constraints and nonlinearities [4]. It became a popular time-domain control strategy in the process industry in the 1970s [5,6]. Initially, some investigations were carried out on its application in power electronics in the early 1980s [7,8]. However, the need for online optimization and high computational power limited its application in power electronics. The advent of digital technology with ever-increasing computational capability kindled the research on MPC for power electronics in the past decade [9–12]. MPC outpaces traditional control methods in handling time-domain constraints where the control task is devised as an optimization problem [13].

MPC techniques for power converters are classified as indirect and direct types [9]. Indirect MPC is a two-stage control with a predictive controller and modulator. Direct MPC unifies the control and modulation problem into a single-stage computational problem.
resulting in an integer vector \([9,14,15]\). The direct MPC is further categorized as MPC with hysteresis bound reference tracking and MPC with implicit modulator \([9]\). Finite control set MPC (FCS-MPC) \([15]\), a reference tracking-based technique, is the most commonly used control method for power electronic converters to solve an optimization problem online. A discrete model of the power converter helps reduce the algorithm’s computational burden \([16]\). FCS-MPC is advantageous over other control methods due to its simple algorithm structure, flexibility to address nonlinearities and constraints, ease of digital implementation, and fast dynamic response \([16,17]\). However, the computational complexity of the FCS-MPC method necessitates the use of powerful processors for its real-time implementation.

Digital technology is widely accepted for implementing complex control algorithms due to flexibility, noise immunity, and insensitivity to component variations \([18]\). However, Analog controllers offer rapidity and better bandwidth but lead to parameter drifting. For this reason, the execution time of the digital controllers must be of utmost importance for better performance \([19]\). MPC for power electronic converters can be implemented using digital controllers like microcontrollers and digital signal processors (DSPs) \([10,12,20]\). DSPs offer highly flexible software-based solutions and exhibit excellent performance for sequential algorithms. Their capability to manage floating-point arithmetic helps in solving complex mathematical problems. However, the main drawback of micro-programmable solutions is the computational delay in finding the optimal solution in one sampling period, especially in complex control techniques like FCS-MPC \([21,22]\). Various delay compensation techniques are addressed in literatures \([12,20,23]\). However, the design modifications may result in inferior performance compared to conventional control methods \([24]\).

Field programmable gate array (FPGA) is excellent for computationally intensive power converter control strategies \([21]\). They are a matrix of configurable logic blocks called logic elements (LEs) with programmable interconnects \([4,25]\). Application architecture can be either all hardware using logic cell (Hw) or all software using processor core (Sw) or a combination of these \([19]\). Nowadays, prominent FPGA manufacturers like Intel, Santa Clara, CA, USA, and Xilinx, Santa Clara, CA, USA offer a broad portfolio of FPGA solutions with better performance, energy efficiency, cost per function, density, and reliability. They offer the highest level of system integration with embedded processors, memory chips, intellectual property (IP) cores, and software tools in a single chip and are termed as “system on chip” (SoC) or field programmable system on chip (FPPSoC) \([26,27]\). There has been a spurt in the FPGA fabrication process, which has reached a level of 10 nm SuperFin process technology \([28]\). Its key features include high integration density, better performance, and reduction in power. The latest FPGA device from Intel (Intel Agilex) belongs to this category. However, the design complexity and lack of skillset required to interconnect the processors and FPGA limit its wide application \([29]\). Customized architecture utilization drastically reduced the design complexity and shortened the “time to market” for FPGA solutions.

FPGA-based control has been successfully implemented in many power electronic applications \([30,31]\). The main advantage of FPGAs over DSP is their deep pipelining and inherent parallelism that helps reduce the execution time of complex algorithms to a few microseconds \([1]\). In addition, they offer high performance, cost-effectiveness, and better reliability. FPGA includes high-quality hardware design and verification tools to implement the controller using hardware description languages (HDLs). Typical languages used for electronic design automation (EDA) synthesis in FPGA are Verilog HDL, very high-speed integrated circuit HDL (VHDL), System Verilog, and System C \([32]\). They permit the concurrent process design found in hardware \([33]\). However, the designer should have profound knowledge about the hardware design for optimized performance, limiting its widespread application in industrial systems \([34]\). High-level synthesis (HLS) tools emerged as an alternate enabling FPGA’s act as hardware accelerators to simplify the design process \([32]\). HLS tools convert high-level programming language, most commonly C, C++, and System C, to register-transfer-level (RTL) description of the control algorithm.
and finally into netlist to program FPGA [35]. A comprehensive review of the evolution of HLS tools and their success and failures are discussed in [36,37]. These tools exhibit great potential for implementing complex power converter control strategies with hardware in the loop (HIL). Among various alternatives, Xilinx Vivado, Altera OpenCL, Leg Up, Mentor Graphics Catapult C are the prominent HLS tools available in the market [38]. Major FPGA vendors currently offer proprietary solutions like Intel®HLS Compiler (Intel FPGA), Vitis HLS (Xilinx), to convert the high-level language to FPGA code. These vendors also provide conversion from Matlab/Simulink, Open CL (Intel FPGA, Xilinx), and LabVIEW (Xilinx) [39]. However, HLS tools present poor performance and high area consumption compared to direct HDL [32,36,39]. Despite the familiarity, C language lacks timing information, bit accuracy, hierarchy, memory bandwidth, and others critical for the hardware design [40].

As FPGA technology is experiencing a great revolution in packaging, integration level, design tools, and methodologies, it poses a challenge for the designers as the design complexity grows faster than the productivity of the designers, also known as the Design productivity gap [41]. Moreover, the industries always focus on the economic benefits and show reluctance to adopt new control techniques due to their higher investment cost. Although complicated, control techniques like FCS-MPC provide better performance; for the industries to adopt such control methods, the investment cost for the power electronic system needs to be reduced without compromising the system’s performance [24]. The computational burden and more resource utilization of the predictive control is another aspect that needs more attention. A suitable design optimization method is essential for the better utilization of resources and to fit the design in the target FPGA.

A detailed survey on recent research trends and challenges of MPC is discussed in detail in [9], and the design guidelines to maximize system performance are given in [24]. The HIL-based FCS-MPC current control using MATLAB/Simulink integrated Xilinx System Generator (XSG) is explained in [42]. This intuitive solution allows the designer to generate automatic HDL code without an extensive understanding of HDL manual coding. FPGA-based FS-MPC implementation for PMSM drives is given in detail in [43]. XSG environment generates the HDL code and implements it in a Xilinx Vivado platform. However, in both these cases, the resource utilization will be more compared to manual coding. The MATLAB HDL Coder extends support to FPSoC devices for IP Core generation. However, this is limited to specific FPSoCs from Xilinx and Intel [44]. A Xilinx Zynq-based FPSoC with ARM core is utilized for FCS-MPC prototyping in [45]. A Hw/Sw-based approach is applied, and the results are compared with a DSP-based solution. Since a low-cost solution is one of the primary objectives, FPSoC-based implementation is not adopted for the proposed work.

This paper discusses a low-cost solution for efficiently implementing a computationally intensive predictive direct current control technique in detail. A design optimization method is adopted to ensure better resource utilization with faster computation. A low-cost new generation Intel Altera MAX 10® FPGA is utilized for the control implementation, and the implementation process is discussed in detail to address the design productivity gap. Since control performance is of prime concern in power converter applications, low-level HDL workflow is preferred over HLST, which also helps utilize the FPGA device’s full potential. This article also highlights single-chip integration benefits with the increased processing capability of the FPGA device with inbuilt ADC functionality, intellectual property cores (IP cores), and memory controllers. The predictive direct current control is employed in a two-level three-phase VSI to focus on the power SoC-based algorithm implementation utilizing the Quartus® Prime design tool.

The rest of the paper is organized as follows; Section 2 describes the finite set-predictive direct current control (FS-PDCC) algorithm. Section 3 explains the design methodology, and Section 4 elaborates the realization of the control algorithm in the FPGA device. Section 5 discusses the simulation results. Finally, experimental validation of the control strategy is elaborated in Section 6, and Section 7 summarizes the conclusions with future scope.
2. FS-PDCC Control Strategy

FS-PDCC, a current control method based on FCS-MPC, utilizes power converters’ discrete nature to apply control action in a single-stage without intermediate modulators. The algorithm predicts the system’s future behavior based on its model over a horizon. It utilizes only a fixed number of inverter switching states and evaluates a cost function [21]. First, the cost function evaluates the error between each voltage vector’s reference and predicted currents [46]. Then, the voltage vector with the lowest value is selected, and the corresponding switching signals are applied at the next sampling interval [12]. Online implementation of FS-PDCC involves three significant steps: estimation, prediction, and optimization [47]. During the first phase, the optimal switching state from the previous period is applied at an update instant “k,” and phase currents are measured at t_k. The current switching state is maintained till the end of t_{k+1}, where the new switch state is applied. In the prediction phase, the value of the controlled variables for the next sampling instant is predicted for all the finite switching states covering all the future instants. The number of such sampling periods is the prediction horizon (N). When horizon length increases, the number of possible switching sequences becomes quite large, making its implementation quite tricky. Finally, the optimal voltage vector is selected in the optimization phase, and the corresponding switching state is applied to the inverter [48].

A three-phase VSI has eight switching states with six active and two zero vectors. Two switches per phase operate in a complementary manner. The switching function S_x represents “1” and “0” (x = A, B, C) for the closed and open states of the switch. A two-level three-phase VSI is shown in Figure 1.

![Circuit diagram of a two-level three-phase VSI.](image)

The voltage vectors generated by the VSI is given by the expression [12]:

\[ V = \frac{2}{3} \left( v_{AN} + v_{BN}e^{j\frac{2\pi}{3}} + v_{CN}e^{j\frac{4\pi}{3}} \right) \]  

(1)

where \( v_{AN}, v_{BN}, \) and \( v_{CN} \) are the phase to neutral voltages of the VSI. The following expression relates the voltage vector and the switching state sequence and is given by:

\[ V = V_{in} \times S_i \]  

(2)

\( V_{in} \) represents the DC input and \( S_i \), the switching state for \( i = 0,1,2 \ldots 7 \).
The balanced three-phase quantities are converted to two-phase orthogonal reference quantities by Clarke’s transformation, for simplifying the analysis of three-phase circuits, as given by:

$$
\begin{bmatrix}
I_a \\
I_\beta
\end{bmatrix} = \frac{2}{3} \begin{bmatrix}
1 & -\frac{1}{2} & -\frac{1}{2} \\
0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2}
\end{bmatrix}
\begin{bmatrix}
I_A \\
I_B
\end{bmatrix}
$$

(3)

The load dynamics represented in the stationary reference frame for the inverter circuit with RL load is given by the differential equation [46]:

$$V_{a\beta} = R i_{a\beta} + L \frac{di_{a\beta}}{dt}$$

(4)

where \(V_{a\beta}\) and \(i_{a\beta}\) are the load voltage and current vectors, \(R\) and \(L\) are the load resistance and inductance, respectively.

The finite switching states with voltage vectors are illustrated in Figure 2.

![Figure 2. Switching states and voltage vectors of the three-phase VSI.](image)

The load current derivative is based on the forward Euler equation with sampling interval \(T_s\) expressed as [46]:

$$\frac{di_{a\beta}}{dt} = \frac{i_{a\beta}(k+1) - i_{a\beta}(k)}{T_s}$$

(5)

Equation (4) is substituted in Equation (2) to obtain the expression for the load current prediction at instant \(t_{k+1}\) and is given as [46]:

$$i_{a\beta}^p(k+1) = 1 - \frac{RT_s}{L} i_{a\beta}(k) + \frac{T_s}{L} (V_{a\beta}(k))$$

(6)

“\(p\)” denotes the predicted load current value for all the voltage vectors generated by the inverter at an instant \(k+1\). Therefore, the absolute value of cost function (\(G_n\)) defined to indicate the error between the reference and predicted values of load current is expressed as [46]:

$$G_n = \left| i_a^r(k+1) - i_{a\beta}^p(k+1) + i_\beta^r(k+1) - i_{a\beta}^p(k+1) \right|$$

(7)

where \(i_a^r(k+1), i_\beta^r(k+1)\) are the load current references and \(i_{a\beta}^p(k+1), i_{a\beta}^p(k+1)\) are the load current predictions, respectively. The cost function is determined for \(n = 0\)–7, and the switching state corresponding to the minimum \(G_n\) is applied in the subsequent sampling interval. The orthogonal components of reference current are obtained from an external loop and \(i^r(k+1) \equiv i^r(k)\) during a particular sampling interval.
3. Design Methodology

As mentioned earlier, FPGA controllers can drastically reduce the computational effort of complex control algorithms. However, for implementing a predictive control algorithm, prediction and optimization stages present a significant computational load. By exploiting the parallel and pipelined architecture of FPGA, computation time can be reduced but at the expense of consumed resources. In this regard, the implementation aspect of the control algorithm needs to be well designed to improve the time-area efficiency. An efficient design methodology will enable the designer to optimize resource utilization. FPGA permits the use of a specific architecture to implement the control algorithm in a flexible environment with available memory blocks, multipliers, and logic elements. However, the designer must follow specific rules and steps to make it more manageable. Simple and less intuitive design methodology will ensure optimization and reusability of the available resources with proper sharing and streaming.

Furthermore, the architecture should be scalable enough to include future modifications and constraints. The modular architecture of the FPGA with IP core improves the control performance of the entire system with a shorter execution time. MATLAB/Simulink provides a friendly environment to generate the FPGA architecture using dedicated toolboxes like HDL coder for Altera and System Generator for Xilinx. However, the control performance may deteriorate due to unoptimized architecture leading to more usage of FPGA resources. So, in this article, Verilog HDL coding is adopted for FPGA architecture. The initial system design followed by algorithm formulation, FPGA architecture optimization, and implementation forms the entire design process [4, 18, 19]. The design process overview is given in Figure 3 [19].

![Figure 3. The design process of the FPGA controller.](image)

3.1. Algorithm Design

Preliminary system design includes source and load considerations, selection of digital controllers, sensors, and finalizing the hardware specifications. The algorithm modification process to adapt to the available FPGA resources is very important in the design stage. First, the algorithm is split into modules directly executable by finite state machines (FSM). The module partitioning can be done based on the concepts of hierarchy and reusability, whereby the modules can be divided into smaller ones that are more manageable and reusable [49]. The designer has the freedom to remodel the algorithm to reduce the number of operations to limited hardware resources. Some reusable modules for controlling electrical systems with different hierarchy levels are specified in [18]. A continuous-time functional validation of the algorithm using MATLAB/Simulink tools is performed during this stage, making it ready for digital implementation.
The following important task is the choice of fixed-point format. Usually, FPGAs handle fixed-point calculations effortlessly. Intel’s newest generation 10 FPGAs like Arria® 10 and Stratix® 10 possess IEEE-754 single-precision floating-point units (FPU) inside the DSP block. However, the sequential operation of the FPU may slow down the entire calculation process, and capital cost is a matter of concern for the proposed system. MathWorks has a Fixed-point Designer™ tool that provides data types and optimization tools for implementing fixed/floating-point algorithms in embedded systems. The tool will enable the designer to implement the data in a floating-point with the same efficiency and performance as a fixed point. However, the result may consume more FPGA resources, and the design will not fit the target FPGA.

3.2. Architecture Optimization and Scheduling

Optimization of the algorithm and architecture is another vital stage involved in the design process. The designer can opt for automatic code generation using an HDL coder, but it may result in an unoptimized solution regarding the available FPGA resources. So, for better resource allocation and architecture optimization, the designer has to perform HDL coding by himself with the help of the Algorithm Architecture Adequation (AAA) methodology [50].

Application of AAA methodology in FPGA processors aims to prototype real-time applications with potential factorization quickly. This optimization methodology takes care of the size and timing constraints of the algorithm. Potential factorization leads to maximum operations with minimum operators. The process consists of three stages: Design of data flow graph (DFG), data dependency validation, and design of factorized DFG (FDFG) [19]. However, a compromise between hardware resources and computation time should be made as factorization leads to more computation time and less consumed resources. Each elementary module consists of a data and control path coded in Verilog HDL, and they are synchronized with a clock (CLK) signal. The data path consists of various operators like multipliers, adders, and registers, and the data transfer is managed by a global control unit-FSM. The module control units are triggered via a start signal and generate an end signal upon process completion. The architecture scheduling is done by FSM, which sequences the operation of different modules via start and end signals. As a result, the AAA methodology can achieve better time/area performance [19]. The methodology is applied to sine wave generation, ABC-αβ transformation, and predictive control to reduce hardware resources. The repetitive patterns in the data flow graph can be optimized using FDFG, and a sample of the methodology is represented in Figure 4.

System validation can be performed using both software simulation tools and hardware testing. The FPGA controller used in the proposed work is Intel® MAX® 10 FPGA. They are low-cost, non-volatile, small form factor, single-chip programmable logic devices with advanced processing capabilities. The device highlights include integrated Analog to digital converter, DSP blocks, Nios II Gen 2 Embedded soft processor support. In addition, Intel FPGAs are equipped with highly efficient design tools like Intel Quartus® Prime Lite edition, Nios® II Embedded Design Suite (EDS), and Platform Designer.

The functionality and performance validation of the Verilog coded architecture is performed using the EDA tool Modelsim*-Intel® FPGA Starter Edition Software available with Quartus Prime. Furthermore, a relevant set of test bench inputs is provided for validating the architecture. In addition, Intel Quartus prime provides System Console, a fast and efficient real-time debugging tool, to realize the communication interface between the host PC and the target platform. System Console helps to efficiently debug the design while the design runs at full speed. The entire design and implementation process is summarized in Figure 5.
4. FPGA Implementation Process

The basic procedure involved in the FPGA implementation of the control algorithm is depicted in Figure 6.

---

**Figure 4.** FDFG-based design methodology.

**Figure 5.** Summary of design and implementation process.

**Figure 6.** FPGA implementation process.

FPGA design starts with design entry with RTL using Verilog/VHDL in Quartus II environment. GUI-based Platform Designer tool or an HLS Compiler can also be used to create the logic. Altera provides optimized and verified IP cores and third-party IP cores that can be integrated into the design to improve the design performance and efficiency. IP Catalogue and the parameter editor generate customized IP cores for a specific design.
The parameter editor customizes and creates a Quartus IP (.qip) file representing the IP core in the design. The Platform Designer (Standard), a system integration tool available in Quartus®, integrates customized IP components into the FPGA design. The Platform Designer improves the device performance with shorter design cycles and enables design reuse in a GUI.

Following the design entry, the Assignment Editor and Pin Planner interfaces help to constrain the design. The Assignment Editor permits the specification of different options and settings in the design logic and attempts to match the resource assignments with the available resources. Individual and group pin assignments are made using the Pin Planner. The design fitting stage (compiler stage) includes analysis and synthesis, fitter (place & route), assembler, timing analysis, and EDA Netlist Writer. The design analysis and synthesis stage check the design source file for errors, build the database, synthesize and optimize the design, and map the design logic to the device resources. Fitting the design logic by utilizing fewer resources is the principal design challenge in an FPGA implementation. The fitter stage includes Placement and Routing. Placement ensures optimum location for the design logic, and Routing connects the nets between the logic. Quartus II provides powerful tools to view and analyze different design and synthesis results through RTL Viewer, State Machine Viewer, and Technology Map Viewer [51]. In addition, the Chip Planner displays the utilization of the design resources. These design analysis tools can be used throughout the design, debug, and optimization stages [51].

Synchronization and timing analysis is crucial for a successful FPGA design. The clock is the synchronizing signal, and timing analysis checks for timing violations relative to the clock. Quartus II is equipped with a Timing Analyzer tool to analyze the critical paths in the design and view the violations using an industry-standard SDC (Synopsys Design Constraints) package. A .sdc file is created, which specifies the constraints and validates the timing performance of the design logic. After the compilation stage, the design validation and timing simulation are performed using Modelsim. A test bench is created to specify the parameters of all the signals in the design. This tool will help us to test and understand the operation of the design logic.

In the configuration stage, the design is loaded into the FPGA using the Programmer. Programmer prompts to specify the correct JTAG device, USB Blaster. SRAM Object File (.sof) is transferred into the FPGA using the USB Blaster. The debugging tools provide concurrent verification of the design at system speed. The signals are routed to the debugging logic, and debugging tools utilize a combination of logic, memory, and routing resources [52]. Signal Tap logic analyzer is used for the design verification of the algorithm in hardware. The signals are routed to the Quartus environment through a JTAG interface for analysis. The performance of ADC in the MAX® 10 device is analyzed using ADC Toolkit available in System Console.

4.1. Complete Hardware Architecture of the FS-PDCC Controlled Three-Phase VSI

Figure 7 depicts the complete hardware architecture of predictive current-controlled two-level three-phase VSI. The experimental setup consists of a three-phase voltage source inverter with RL load. It includes ACS722 current sensor for sensing the load current and an ACPL-C870 voltage sensor for sensing the input DC voltage to the inverter. Intel®MAX® 10 (10M08SAE144C8G) FPGA containing 8 K logic elements with a 50 MHz clock oscillator was used as the target device for the control implementation. The architecture consists of six functional blocks: ADC IP core, Dual-Port RAM IP core, the reference current generation block, ABC to αβ conversion block, prediction and optimization block, and switching state generation block. The data flow between these blocks is controlled via a global control unit (FSM).
source inverter with RL load. It includes ACS722 current sensor for sensing the load current and ACPL-C870 voltage sensor for sensing the input DC voltage to the inverter.

Intel® MAX® 10 (10M08SAE144C8G) FPGA containing 8K logic elements with a 50 MHz clock oscillator was used as the target device for the control implementation. The architecture consists of six functional blocks: ADC IP core, Dual-Port RAM IP core, the reference current generation block, ABC to αβ conversion block, prediction and optimization block, and switching state generation block. The data flow between these blocks is controlled via a global control unit (FSM).

Figure 7. Complete hardware architecture of the FS-PDCC controlled two-level three-phase VSI.

Finite state machine (FSM) controls the transition among a limited number of internal states, as determined by the current state and external input [53]. It gets triggered via a start signal and controls all the modules over a sampling period $T_s$. A Start_ADC signal is provided to start ADC conversion and waits for the End_ADC signal. Once the conversion gets over, the prediction and optimization block is activated via Start_Pred, and the End_Pred signal indicates the completed process. End signal marks process completion, and optimized switching states are applied to the inverter. The system clock is 50 MHz (sampling period 20 ns), and controller performance is greatly influenced by the computation time. A signal latency is considered in each block. The finite state machine of the control algorithm is shown in Figure 8.
Figure 8. Finite state machine of the FS-PDCC control.

4.2. ADC IP Core Implementation

The ADC IP core converts the analog voltage and current sensor outputs to digital data for predictive control implementation in FPGA. It consists of hard IP blocks and soft Modular ADC Core Intel® FPGA IPs for logic implementation. The FPGA used in the design is a 12-bit SAR ADC with a sampling rate of 1 MSPS. It has one dedicated analog input channel and eight dual function channels with an input voltage range of 0–2.5 V. The voltage range can be extended to 3–3.3 V using the ADC Prescaler function, and the full-scale voltage is full scale-1 LSB. The Quartus® software consists of a Modular ADC Core Intel® FPGA IP to create, configure, and compile the ADC design. Modular ADC Core Intel® FPGA IP is a soft controller to instantiate on-chip ADC hard IP blocks, and PLL provides a 50 MHz input clock to the ADC. Each ADC block can use internal or external voltage references [54].

ADC IP Core design is performed using the Platform Designer Interface in Quartus II. Modular ADC Core Intel® FPGA IP controls the Hard IP Block in the ADC. Using the parameter editor in Modular ADC Core Intel® FPGA IP, ADC clock, sampling rate, analog channel selection, and sequencing of the channels are made. Modular ADC Core IP Core has four configuration alternatives for various ADC applications [54]. Among them, Standard Sequencer with External Sample Storage configuration is utilized in the proposed system. In this configuration, the ADC design exports the ADC conversion data to the core for post-processing, and the ADC Toolkit monitors and analyzes the ADC data when the design is running. This data can be accessed through the debugging tool-System Console, and the analog performance of each channel is verified [55].

4.3. Dual-Port RAM IP Configuration

The 12-bit output data from the ADC block is provided to an internal memory block. Several IP cores are available in the Quartus environment to implement various memory modes. The proposed system uses a simple dual-port RAM IP core to perform simultaneous read and write operations. Simple dual-port RAM selected from the IP catalog is customized using the parameter editor. The number of ports, memory size, the width of the input data bus, memory block type, clocking method, and options for output file generation is specified in the parameter editor, and a .qsys file is generated representing the IP core [55]. The simple dual-port RAM IP core output is analyzed using the system debugging tool, Signal Tap Logic Analyzer. The tool utilizes on-chip memory for the
functional verification of the design. The test nodes are sampled to display in the Quartus environment for analysis. The available resources can be estimated using the logic analyzer interface before its compilation into the design. Add an instance in the signal configuration window with sample depth and RAM type. The memory buffers and communicates the data to the analyzer interface [55].

4.4. Reference Current Generation Block

The three-phase sinusoidal reference current is generated using a Single port ROM memory IP core. The IP Catalog provides the ROM:1-PORT IP Core, and the parameters are customized using the parameter editor. It has one port for read-only operations, and the initial content of the memory is specified using a .mif file. Three 1-PORT ROM IP cores are generated for each of the three-phase reference currents $i_A^*$, $i_B^*$, and $i_C^*$ and instantiated in the top module. The ADC outputs 12-bit data of actual load current, and the reference generator block also generates 4096 samples of sinusoidal reference output.

4.5. ABC to $\alpha\beta$ Conversion Block

The $ABC$ to $\alpha\beta$ conversion, also known as Clarke’s transformation, transforms balanced three-phase quantities to two-axis orthogonal reference quantities for simplifying the analysis of three-phase circuits. The transformation equation is given in Equation (3). The conversion is coded in Verilog in fixed-point format. Both the reference and actual values of three-phase currents are converted to the $\alpha\beta$ reference frame. Nios II Floating-Point Hardware 2 (FPH2) component can be utilized for the floating-point operations in MAX 10. However, only single-precision VHDL coding is supported in FPH2 [56].

4.6. Prediction and Optimization Block

The prediction and optimization block is the most computationally intensive stage during the implementation. A well-designed architecture with optimized use of resources (multipliers, multiplexers, adders, static RAM) and appropriate calculation time will reduce the computational burden of the system. For long prediction horizons ($N > 1$), the computational burden is addressed by a branch-and-bound algorithm [57]. However, for short prediction horizons ($N < 2$), the optimization problem can be solved via exhaustive search enumerating all the possible switching states of the inverter [58]. Resource sharing and streaming is another approach to ease the computational burden of the system. In the proposed system, only a short prediction horizon is evaluated using an exhaustive search approach. The FPGA is well suited for parallel and pipelined architecture implementation, forming the exhaustive search core. By utilizing the concurrent nature of FPGA, prediction of future states and cost function calculation is performed in parallel, and the resources can be decoupled. The cost function evaluation is then pipelined, and cost function values are obtained for different inputs. Only sequential operation is possible for cost function evaluation which gets modified at every input. The optimum switching state corresponding to the minimum cost function is then applied to the inverter. This operation is performed sequentially with the proper scheduling of the calculation core [48]. The data flow among different modules is controlled by FSM shown in Figure 8. The measurement, prediction, and switching state generation must be complete in the same sampling interval.

5. Simulation Results of the FS-PDCC

The simulation of the FS-PDCC is performed in MATLAB/Simulink to verify the performance of the control algorithm. The output power of 200 W is considered for the design. The LC filter parameters are designed based on the load current rating and cut-off frequency. The RL load parameters are fixed based on the three-phase VSI design. DC input voltage of 30 V is provided for the analysis. The parameters used for the experimental setup are utilized for the simulation analysis. The simulation results for a sampling time of $T_s = 20 \mu s$ are presented here. The simulation parameters are given in Table 1.
Table 1. Simulation parameters.

| Parameters                  | Values          |
|-----------------------------|-----------------|
| Input voltage ($V_{in}$)    | 30 V            |
| Load inductance ($L$)       | 18 mH           |
| Load resistance ($R$)       | 5 Ω             |
| Filter inductance ($L_f$)   | 5 mH            |
| Filter capacitance ($C_f$)  | 1 μF            |
| Sampling time ($T_s$)       | 20 μs           |
| Switching frequency ($f_s$) | 20 kHz          |
| Output frequency ($f_o$)    | 50 Hz           |

Figure 9 shows the switching pulses to drive the MOSFETs generated by the proposed FS-PDCC technique. The optimum switching state is applied to the upper and lower switches of the three-phase inverter bridge circuit in a complementary manner. The three-phase output voltages, $V_{AN}$, $V_{BN}$, and $V_{CN}$ (phase-to-neutral) of the proposed system are shown in Figure 10. Figure 11 represents the load current waveform of FS-PDC controlled two-level three-phase VSI for a sampling time of $T_s = 20$ μs.

![Figure 9](image1.png)

**Figure 9.** Switching pulses to drive inverter switches.

![Figure 10](image2.png)

**Figure 10.** The output voltage (phase-neutral) waveform of the FS-PDCC controlled three-phase VSI for $T_s = 20$ μs.
Figure 11. The load current waveform of the FS-PDCC controlled three-phase VSI for $T_s = 20 \mu s$.

Figure 12 shows the phase “A” load current waveform with the actual current $i_{A_{\text{act}}}$ tracking the reference $i_{A_{\text{ref}}}$ for a sampling period of $T_s = 20 \mu s$. The dynamic response of the load current under transient conditions is given in Figure 13. A step-change in reference is provided, and the load current immediately tracks the reference value.

Figure 12. The waveform showing the actual load current $i_A (i_{A_{\text{act}}})$ tracking the reference current ($i_{A_{\text{ref}}}$) for $T_s = 20 \mu s$.

Figure 13. The waveform representing the dynamic response of the load current for a step-change in the reference current for $T_s = 20 \mu s$. 
The FPGA enables the control implementation at a very high sampling rate. The FPGA under consideration has a clock frequency of 50 MHz (20 ns). Large sampling intervals result in high ripples at the output resulting in poor output quality. Figure 14 shows the load current waveforms for different sampling intervals. The waveform quality is much lower for a sampling interval of $T_s = 50 \mu s$, as shown in Figure 14a. For a small sampling interval of $T_s = 5 \mu s$, the load current exhibits better output quality with reduced ripples represented in Figure 14b. However, this results in high switching frequency and can be addressed by adding constraints in the cost function. Figure 15 depicts variation in load current THD (ITHD) for different sampling intervals, and it can be observed that THD is much lower for small sampling intervals but at the expense of more resources.

![Waveform graph](image1)

**Figure 14.** The load current waveforms for different sampling periods: (a) $T_s = 50 \mu s$; (b) $T_s = 5 \mu s$. 
The resource utilization of the predictive control algorithm is another impeding factor for employing very small sampling intervals in the range of nanoseconds. Many resources will be utilized, and the design will not fit the target device. The cost/performance trade-off is one of the primary objectives for the complex predictive current control discussed in the article. Intel MAX 10 device is highly cost-effective compared to its conventional counterparts from Xilinx. Hence, a sampling time of $T_s = 20 \mu s$ is used for the experimental analysis, and its load current spectrum is given in Figure 16a. Figure 16b shows the load current spectrum for $T_s = 5 \mu s$.

Control strategies for power electronic converters include linear and non-linear techniques. Among them, sinusoidal pulse width modulation (SPWM) and hysteresis current control are the prominent ones. The FS-PDCC is also a non-linear control technique, and hence a comparison is made between FS-PDCC and hysteresis current control for validating the predictive current control using MATLAB/Simulink. Figure 17a shows the load current waveform of a hysteresis current controlled three-phase VSI. The load current waveform exhibits high distortion than the load current waveform of FS-PDC-controlled three-phase VSI ($T_s = 20 \mu s$) in Figure 17b. The comparison chart representing THD vs. sampling rate for different sampling intervals is shown in Figure 15.

**Figure 15.** Comparison results of ITHD and resource utilization for different sampling intervals.

**Figure 16.** The load current spectrum of FS-PDCC controlled three-phase VSI: (a) $T_s = 20 \mu s$; (b) $T_s = 5 \mu s$.

*Comparison with Conventional Control Techniques*

Control strategies for power electronic converters include linear and non-linear techniques. Among them, sinusoidal pulse width modulation (SPWM) and hysteresis current control are the prominent ones. The FS-PDCC is also a non-linear control technique, and hence a comparison is made between FS-PDCC and hysteresis current control for validating the predictive current control using MATLAB/Simulink. Figure 17a shows the load current waveform of a hysteresis current controlled three-phase VSI. The load current waveform exhibits high distortion than the load current waveform of FS-PDC-controlled three-phase VSI ($T_s = 20 \mu s$) in Figure 17b. The comparison chart representing THD vs. sampling rate for different sampling intervals is shown in Figure 15.
is given in Figure 18, and it can be identified that FS-PDCC exhibits better output quality compared to hysteresis current control.

**Figure 17.** The load current waveforms of the two-level three-phase VSI: (a) using hysteresis current control (b) using FS-PDCC control.

**Figure 18.** THD vs. sampling time comparison for FS-PDCC and hysteresis current control.
6. Experimental Implementation of FS-PDCC in FPGA

6.1. ADC IP Core Implementation Results

In the ADC IP core window, the clock input is set to 10 MHz and the internal reference voltage to 2.5 V. Standard Sequencer with External Sample Storage core variant is selected. Debug path should be enabled to provide the data to ADC Toolkit for analysis. ALTPLL Intel FPGA IP core is added to the design to provide an ADC input clock. The frequency of inclk0 is set to 50 MHz in the ALTPLL Intel FPGA IP core parameter editor. ALTPLL outputs a set of input clock frequencies for a predefined ADC sampling rate ranging from 1 MHz to 25 kHz. PLL output clock frequencies varying from 2 MHz to 80 MHz can be provided as ADC inputs. A sampling rate of 1 MSPS with a 10 MHz clock is chosen in the proposed design. ALTPLL IP core and Modular ADC IP are connected. JTAG to Avalon Master Bridge is selected from the IP catalog and links all the CLK signals in the design. Finally, create Avalon Slave Interface to start the ADC [54]. Verilog HDL files will be created in the design. More design files are made for the selected ADC configuration for sequencing, sample storage, and post-processing with a top instantiation file to connect them. The implemented ADC design for the FS-PDCC is shown in Figure 19. The data information about all the four ADC channels can be analyzed using the ADC Toolkit.

![Schematic of the ADC design implementation.](image)

**Figure 19.** Schematic of the ADC design implementation.

6.2. Simple Dual-Port RAM IP Core Implementation

Three ADC outputs, \( i_A, i_B, \) and \( i_C \), are provided to three simple dual-port RAMs. From ADC, generate a write signal, store the data each time in consecutive memory locations in the RAM, and get the data out from the memory locations for each clock. The Signal Tap Logic Analyzer provided in the Quartus displays the data, waddress (write address), rdaddress (read address), and a clock input to the memory as .stp file as shown in Figure 20.

![Signal Tap Logic Analyzer window displaying simple dual-port RAM process.](image)

**Figure 20.** Signal Tap Logic Analyzer window displaying simple dual-port RAM process.
6.3. RTL View of the Implemented Prediction and Optimization Stage

The RTL view of the prediction and optimization phase implemented in FPGA is shown in Figure 21. The ADC outputs are provided to the RAM blocks. RAM outputs the load current values to the prediction and optimization block for each clock and evaluates the switching state.

![Figure 21. RTL view of the prediction and optimization stage.](image)

6.4. Design Considerations and Results: Resource Utilization

The target device Intel MAX®10 FPGA 10M08SAE144C8G consists of 8 K logic elements with a clock frequency of 50 MHz (clock period of 20 ns). The rapidity offered by the FPGAs often comes at the expense of more usage of resources. Therefore, the system should be designed carefully with limited multipliers and adders since they utilize considerable resources. With the help of the AAA methodology, area optimization without compromising rapidity in the control execution is achieved. The data transfer between multipliers, adders, and multiplexers is through FSM, as shown in Figure 4. For optimizing the resources for predictive control implementation, DFG is scheduled based on resources and constraints. The FPGA has 24 dedicated multipliers, and the optimization strategy reduces its usage. A lower number of resources results in better time-area efficiency.

The algorithm is implemented in fixed-point arithmetic, which speeds up the process and reduces resource usage. The integer values are provided with 5 bits to avoid truncation, and fractional values with 12 bits to prevent data loss. The clock cycles required for finding the optimal switching state are obtained from the Modelsim testbench with a counter. The Timing Analyzer generates the timing information of all the paths in the design. FSM generates the starting signal, and the inverter switching sequences are applied within a sampling time of 20 μs (Ts). As discussed in Section 6, the PLL is fed with a 50 MHz clock signal and clocks the ADC at 10 MHz. 12-bit ADC samples the input signals at 1 MHz (1 μs) and is stored in dual-port RAM. The computation of the control algorithm with all three stages is completed in the sampling period. The total clock cycles are given by Ts × fCLK, where fCLK is the clock frequency of the current control. With a clock frequency of 50 MHz and a 20 μs sampling period, 1000 clock cycles are available with a total latency of 160 clock cycles to complete the process. Therefore, the number of samples taken during each sampling period is 1000. The algorithm’s entire execution time, together with ADC conversion, is 3.2 μs (160 clock cycles), much lower than the sampling period Ts, and the inverter switching frequency does not exceed half of the sampling period.
RTL simulation is performed by creating a test bench in Modelsim®. The test bench parameters are set in the parameter window, and waves for each signal are generated. Next, Modelsim® runs the test bench and generates a simulation report. The RTL simulation of the FS-PDCC using Modelsim® is shown in Figure 22.

![RTL simulation output of FS-PDCC using Modelsim®.](image)

Table 2 summarizes resource utilization in the MAX® 10 FPGA for predictive current control implementation, including ADC conversion, reference generation, prediction, optimization, and switching state generation, for a short predictive horizon of N = 1.

| Resources                     | Used/Available | Percentage of Utilization |
|-------------------------------|----------------|---------------------------|
| Logic elements                | 2261/8064      | 28%                       |
| Registers                     | 274/8542       | 3%                        |
| Memory bits                   | 768/387,072    | <1%                       |
| Embedded Multiplier 9-bit elements | 22/48          | 46%                       |
| PLL                           | 1/1            | 100%                      |
| ADC                           | 1/1            | 100%                      |

Owing to the concurrent and pipelined structure of FPGA, only 28% of logic elements are utilized for the entire implementation. The memory utilization is less than 1%, and the number of 9-bit multiplier elements used is only 22. The worst-case slack is 10.193 ns, and the data delay is 9.112 ns. Only a few resources are used, and the extra resources indicate that predictive control with long prediction horizons (N > 1) can be accommodated in the low-cost FPGA MAX® 10. Intellectual property (IP) cores and architecture optimization methods enhanced the efficient execution of the complex control algorithm.

The bar graph in Figure 22 summarizes the timing performance of the data paths in the control algorithm obtained using Timing Analyzer in the Quartus software. The time taken for ADC conversion (TADC) is 1 μs with a latency of 50 clock cycles. The conversion from ABC to αβ takes place in 0.24 μs, and the time for estimation (TEst) is 0.1 μs. The time taken for the prediction and optimization (TPred&Opt) phase is 1.86 μs, and the control algorithm’s total execution time (TEX) is only 3.2 μs. The corresponding value of cycle time for each stage is also represented in Figure 23.
The target device’s resource use and cost are critical for FS-PDCC implementation. FPGA implementation of FCS-MPC is discussed in [42,43]. In [42], HIL-based co-simulation of FCS-MPC using Xilinx System Generator and implementation in ZedBoard Zynq evaluation kit is discussed. Xilinx VIVADO Design suite is utilized in [43] for FCS-MPC implementation. Since capital cost is a deciding factor, the MAX®10 device is chosen for the proposed implementation. The FS-PDCC implementation in MAX®10 is compared with a Xilinx Spartan 6 FPGA (xc6slx4-3tqg144), and the number of slice LUTs utilized for the implementation was 5348 (2400 available). The resource utilization was 222%, and the design did not fit the target. Moreover, it lacks inbuilt ADC and embedded processors.

6.6. Hardware Implementation Results

The FS-PDCC technique is implemented in a two-level three-phase VSI with RL load. The test parameters used for the experimental analysis are the same as the simulation parameters given in Table 2. The PCB of the inverter circuit is designed using Altium Designer 17 and fabricated for a power rating of 1 kW. For the experimental analysis, an output power of 50 W is considered. The power circuit consists of six IRFP150N Power MOSFET switches and three Si8233 high/low side MOSFET Driver ICs. The control algorithm is implemented in MAX10 FPGA, and the control signals corresponding to the optimum switching states are provided to the driver IC inputs.

Three ACS-722 (5 A) current sensors measure the output current and are fed to the ADC input. Allegro’s DC voltage sensor ACPL-C870 senses the DC input. The DC input should be scaled down since 2.5 V is the maximum permitted input voltage to the ADC. The RL load is designed for load resistance value R = 5 Ω, 50 W, and inductance L = 18 mH. The load current reference is a 50 Hz sine wave with a peak value of 1.7 A generated by the FPGA. The three sinusoidal references are generated using the ROM:1-PORT IP Core available in Quartus II. The actual and the reference values are compared, and the switching state with the optimum cost function is applied to the inverter. The sampling period has a significant impact on the performance parameters of the system, like THD and switching frequency. The whole execution of the control algorithm takes place in 3.2 μs, and a sampling time of 20 μs is provided since longer sampling times are the better choices for low switching frequency applications. The hardware setup of the entire system is displayed in Figure 24.
Figures 24 and 26 show the experimental results of predictive current-controlled 2 level three-phase VSI in steady-state for N=1. The experimental setup consists of three current sensors (ACS 722) for each phase. Figure 25a,b shows the sensed output current waveform of phases A and B with a maximum value of 1.7 A. The load current waveforms for phases A and B are measured using Keysight 1146B AC/DC current probe presented in Figure 26a,b.

The voltage sensor output is shown in Figure 27a. In Figure 27b, the experimental output voltage of the inverter (with RC filter R = 10 Ω, C = 22 µF) is given. The hardware results of the control algorithm validate better functionality of the controller architecture. The predictive controller achieves steady-state performance with fast dynamic response and excellent output quality. THD, measured using the HIOKI power quality analyzer, is given in Figure 28. The waveform mode representing three phase output voltage and current is shown in Fig.28a and Fig.28b shows the harmonics mode representing harmonic waveform measurements of voltage and current. THD is a critical performance parameter of the control algorithm and is 1.3%, 1.5%, 1.3%, very close to the simulation result of 2.1% shown in Figure 16a. THD may further reduce for higher switching frequencies and longer prediction horizons.
Figure 26. The load current waveforms with an amplitude of 1.7 A and a frequency of 50 Hz for $T_s = 20 \, \mu s$ measured using Keysight 1146 B current probe: (a) Phase “A” load current, (b) Phase “B” load current (Scale: Y-axis: 500 mA/div; X-axis: 10 ms/div).

The voltage sensor output is shown in Figure 27a. In Figure 27b, the experimental output voltage of the inverter (with RC filter $R = 10 \, \Omega$, $C = 22 \, \mu F$) is given. The hardware results of the control algorithm validate better functionality of the controller architecture. The predictive controller achieves steady-state performance with fast dynamic response and excellent output quality. THD, measured using the HIOKI power quality analyzer, is given in Figure 28. The waveform mode representing three phase output voltage and current is shown in Fig.28a and Fig.28b shows the harmonics mode representing harmonic waveform measurements of voltage and current. THD is a critical performance parameter of the control algorithm and is 1.3%, 1.5%, 1.3%, very close to the simulation result of 2.1% shown in Figure 16a. THD may further reduce for higher switching frequencies and longer prediction horizons.

Figure 27. The experimental voltage waveforms of the FS-PDCC-based three-phase VSI: (a) The output of the voltage sensor IC ACPL-C870, (b) the experimental output voltage waveform of the predictive current-controlled three-phase VSI (Scale: Y-axis: 10 V/div; X-axis: 5 ms/div).
Figure 28. The HIOKI Power Quality Analyzer output of the FS-PDCC based two-level three-phase VSI: (a) waveform mode showing output voltage and current, (b) Harmonics mode.

7. Conclusions

This paper presents the time/resource optimized implementation of a computationally efficient predictive direct current control in a very low-cost SoC-based FPGA. An efficient AAA design methodology is adopted to minimize the use of multipliers and adders in the design hence minimizing the computational burden of the algorithm. The FS-PDCC algorithm is verified using MATLAB/Simulink for different sampling rates ranging from 1 μs to 50 μs. For a sampling rate of 1 μs, the FFT analysis provides a THD value of 0.03%, and for 50 μs, it is 6.33%. However, very high sampling rates result in more resource utilization, and the design will not fit the target. Hence, a trade-off between the sampling rate and resource utilization is maintained by selecting a sampling period of $T_s = 20 \mu s$ for the experimental validation.

Inbuilt ADC IP core permits a faster digital conversion at the rate of 1 MSPS. Only 28% of the logic elements were utilized, with very low memory utilization. A quick execution time of 3.2 μs is achieved using FSM-based implementation, resulting in high dynamic performance. While preserving excellent control quality, experimental results show a better output quality. The THD measured using a power quality analyzer exhibits a low THD value of 1.3%, 1.5%, and 1.3% for the load currents $i_A$, $i_B$, and $i_C$. The optimization problem can be extended for long predictive horizons of $N > 1$ with a high sampling rate and fast execution speed. Implementing HLS tools and utilizing floating-point IP cores without compromising the control quality is another research possibility. Optimizing the predictive control algorithm for multilevel inverters and other power electronic converters like Z-source converters, without much computational power for longer horizons, is also a promising area of research.

**Author Contributions:** All authors have equally contributed to this work. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Conflicts of Interest:** The authors declare no conflict of interest.
References

1. Curkovic, M.; Jezernik, K.; Horvat, R. FPGA-based predictive sliding mode controller of a three-phase inverter. IEEE Trans. Ind. Electron. 2013, 60, 637–644. [CrossRef]

2. Kouro, S.; Cortés, P.; Vargas, R.; Ammann, U.; Rodríguez, J. Model predictive control—A simple and powerful Method to control power converters. IEEE Trans. Ind. Electron. 2009, 56, 1826–1838. [CrossRef]

3. Camacho, E.F.; Bordons, C. Model Predictive Control, 2nd ed.; Springer: London, UK, 2007; ISBN 978-1-85233-694-3.

4. Monmasson, E.; Cirstea, M.N. FPGA design methodology for industrial control systems—A review. IEEE Trans. Ind. Electron. 2007, 54, 1824–1842. [CrossRef]

5. Richalet, J.; Rault, A.; Testud, J.L.; Papon, J. Model predictive heuristic control. Applications to industrial processes. Automatica 1978, 14, 413–428. [CrossRef]

6. Morari, M.; Lee, H.J. Model predictive control: Past, present and future. In Proceedings of the Computers and Chemical Engineering; Elsevier Science Ltd.: Amsterdam, The Netherlands, 1999; Volume 23, pp. 667–682.

7. Holtz, J. A predictive controller for the stator current vector of ac machines fed from a switched voltage source. In Proceedings of the IEE of Japan IPEC-Tokyo, Tokyo, Japan, 21–27 March 1983; pp. 1665–1675.

8. Schröder, D.F.; Kennel, R. Model-Control PROMC-A New Control Strategy with Microcomputer for Drive Applications. IEEE Trans. Ind. Appl. 1985, 21-2, 1162–1167. [CrossRef]

9. Karamanakos, P.; Liegmann, E.; Geyer, T.; Kennel, R. Model Predictive Control of Power Electronic Systems: Methods, Results, and Challenges. IEEE Open J. Ind. Appl. 2020, 1, 95–114. [CrossRef]

10. Kennel, R.; Linder, A.; Linke, M. Generalized Predictive Control (GPC)—Ready for use in drive applications? In Proceedings of the 2001 IEEE 32nd Annual Power Electronics Specialists Conference, Vancouver, BC, Canada, 17–21 June 2001; Volume 4, pp. 1839–1844. [CrossRef]

11. Geyer, T.; Papafotiou, G.; Morari, M. On the optimal control of switch-mode DC-DC converters. In International Workshop on Hybrid Systems: Computation and Control; Springer: Berlin/Heidelberg, Germany, 2004; Volume 2993, pp. 342–356. [CrossRef]

12. Rodríguez, J.; Pontt, J.; Silva, C.A.; Correa, P.; Lezana, P.; Cortés, P.; Ammann, U. Predictive Current Control of a Voltage Source Inverter. Ind. Electron. IEEE Trans. 2007, 54, 495–503. [CrossRef]

13. Stellato, B.; Geyer, T.; Goulart, P.J. High-Speed Finite Control Set Model Predictive Control for Power Electronics. IEEE Trans. Power Electron. 2017, 32, 4007–4020. [CrossRef]

14. Rodríguez, J.; Kazmierkowski, M.P.; Espinosa, J.R.; Zanchetta, P.; Abu-Rub, H.; Young, H.A.; Rojas, C.A. State of the Art of Finite Control Set Model Predictive Control in Power Electronics. Ind. Inform. IEEE Trans. 2013, 9, 1003–1016. [CrossRef]

15. Cortés, P.; Kazmierkowski, M.P.; Kennel, R.M.; Quevedo, D.E.; Rodríguez, J. Predictive control in power electronics and drives. IEEE Trans. Ind. Electron. 2008, 55, 4312–4324. [CrossRef]

16. Young, H.; Perez, M.A.; Rodríguez, J. Assessing Finite-Control-Set Model Predictive Control: A Comparison with a Linear Current Controller in Two-Level Voltage Source Inverters. IEEE Ind. Electron. Mag. 2014, 8, 44–52. [CrossRef]

17. Kouro, S.; Perez, M.A.; Rodríguez, J.; Llor, A.M.; Young, H.A. Model Predictive Control: MPC’s Role in the Evolution of Power Electronics. IEEE Ind. Electron. Mag. 2015, 9, 8–21. [CrossRef]

18. Naouar, M.W.; Monmasson, E.; Naassani, A.A.; Slama-Belkhodja, I.; Patin, N. FPGA-based current controllers for AC machine drives—A review. IEEE Trans. Ind. Electron. 2007, 54, 1907–1925. [CrossRef]

19. Monmasson, E.; Idkhajine, L.; Bahri, I.; Naouar, M.W.; Charaabi, L. Design methodology and FPGA-based controllers for power electronics and drive applications. In Proceedings of the 2010 5th IEEE Conference on Industrial Electronics and Applications,ICIEA 2010, Taichung, Taiwan, 15–17 June 2010; pp. 2328–2338. [CrossRef]

20. Abu-Rub, H.; Guzinski, J.; Krzeminski, Z.; Toliyat, H.A. Predictive current control of voltage-source inverters. IEEE Trans. Ind. Electron. 2004, 51, 585–593. [CrossRef]

21. Martin Sanchez, P.; Machado, O.; Bueno Pena, E.J.; Rodriguez, F.J.; Meca, F.J. FPGA-based implementation of a predictive current controller for power converters. IEEE Trans. Ind. Inform. 2013, 9, 1312–1321. [CrossRef]

22. Kukrer, O. Discrete-time current control of voltage-fed three-phase PWM inverters. IEEE Trans. Power Electron. 1996, 11, 260–269. [CrossRef]

23. Cortes, P.; Rodriguez, J.; Silva, C.; Flores, A. Delay compensation in model predictive current control of a three-phase inverter. IEEE Trans. Ind. Electron. 2012, 59, 1323–1325. [CrossRef]

24. Karamanakos, P.; Geyer, T. Guidelines for the Design of Finite Control Set Model Predictive Controllers. IEEE Trans. Power Electron. 2020, 35, 7434–7450. [CrossRef]

25. Rodriguez-Andina, J.J.; Moure, M.J.; Valdes, M.D. Features, design tools, and application domains of FPGAs. IEEE Trans. Ind. Electron. 2007, 54, 1810–1823. [CrossRef]

26. Altera Corporation. What is an SoC FPGA? Architecture Brief; Altera Support Resources: San Jose, CA, USA, 2014.

27. Iyer, S.S. Heterogeneous Integration for Performance and Scaling. IEEE Trans. Compon. Packag. Manuf. Technol. 2016, 6, 973–982. [CrossRef]

28. Intel. Intel FPGA Product Catalog. Version 20.1; Intel: Santa Clara, CA, USA, 2021.

29. Fujieda, N.; Ichikawa, S.; Ishigaki, Y.; Tanaka, T. Evaluation of the hardwired sequence control system generated by high-level synthesis. In Proceedings of the IEEE International Symposium on Industrial Electronics, Edinburgh, UK, 19–21 June 2017; pp. 1261–1267.
