2024 European Control Conference (ECC)
June 25-28, 2024. Stockholm, Sweden

Solving Mixed-Integer Optimization Problems in Microsecond Scale: A Scalable Real-Time Embedded Hardware Architecture

Nart Gashi[1], Victor Truong Thinh Lam[1] and Georgios Papafotiou[1]

***Abstract*— For the application of modern control and optimization techniques to power electronics applications, performing intensive computations on a microsecond-scale is of paramount importance. The optimization algorithms involved are in most of the cases NP-hard, involving integer variables and non-linear objectives; their real-time implementation is done on embedded control platforms in the form of field-programmable gate arrays (FPGAs). These enable fast computation but also feature limited hardware resources that constrain the size of the problems that can be tackled. In this paper, we present a scalable digital hardware (HW) architecture implemented on an FPGA for the real-time implementation of a branch-and-bound algorithm for mixed-integer optimization. The need for such a solver stems from the concept of converter modulation, where an optimal voltage path of a certain length has to be reconstructed by the appropriate choice of a combination of integer variables. We showcase how the geometrical properties of the optimization problem can be exploited to enable a solution in microseconds, and how the FPGA design can be modularized to ensure that the HW resources are adequate even when considering longer voltage reconstruction paths which translate to an exponential increase in problem complexity.**

I. Introduction

In the last decades, computation has revolutionized power electronics (PE) applications, where circuits employing networks of semiconductor power switches are converting electric energy from one form to another (e.g. dc to ac). Some 30 years ago, most of the control loops in PE applications were analog circuits. When more advanced control algorithms were reported, they were deployed on rudimentary digital control platforms due to hardware cost considerations [1]. In stark contrast with that time, field-programmable gate arrays (FPGAs) are nowadays commonly used in PE converters [2], [3]. This has allowed for the investigation and eventual application of more computationally expensive optimization-based control techniques, such as model predictive control (MPC) in power electronics systems [4].

The challenges in this field, however, remain formidable. Switching frequencies of the power switches in PE converters vary depending on the power level of the application; they have typically ranged from a few hundred of Hz for high-power (MW) applications utilizing IGCTs to a few 10s of kHz for medium power applications with IGBTs and a few 100s of kHz for low power applications employing power MOSFETs. The advent of wide band-gap power switches is pushing the numbers for medium power applications to the 100s of kHz and for low power into the MHz range. For optimization-based control techniques, this is limiting the time available for computing the optimal solution [5].

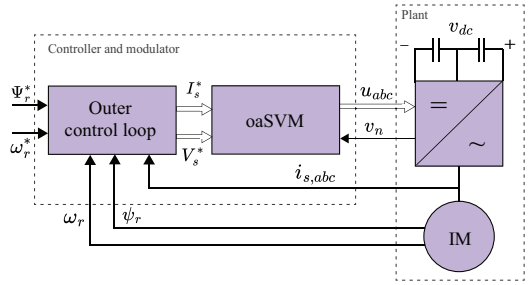


Fig. 1: The electric drive setup considered, consisting of outer control loops, oaSVM, and a three-level NPC inverter driving an induction machine (IM).

Motivated by this challenge, we present in this paper a scalable digital hardware architecture that can be implemented on an FPGA, dedicated to solving mixed-integer optimization problems, such as those encountered in PE applications, in real-time using branch-and-bound. The specific problem at hand is a mixed-integer non-linear optimization problem (MINLP) that originates from the example of an electric drive setup comprising a three-level neutral-point-clamped (NPC) inverter driving an induction machine (IM), see Fig. 1. In such a system, an outer control loop calculates the voltages (as continuous, real-valued variables) that the converter needs to apply to the motor to reach the desired performance. Subsequently, a modulator reconstructs these as time-average of the available discrete-valued voltages of the converter by choosing the appropriate switch on- and off-positions and their duration [6].

In previous work, we have investigated how posing the modulation problem as an online optimization process can bring performance benefits with respect to the state-of-the-art scheme. This new modulation scheme is called online adaptive space vector modulation (oaSVM) [7], and it is based on the search in real-time for the optimal switch positions which can reconstruct the voltages required by the converter's outer control loop, while also improving the trade-off between the switching losses in the converter semiconductors and current total harmonic distortion (THD) at the output. Furthermore, with oaSVM one can also compute the optimal switching sequence over an *optimization window* of L modulation cycles, when provided with a voltage path that needs to be followed into the future.

A critical question for this new scheme concerns the feasibility of its real-time implementation. The complexity

[1]Department of Electrical Engineering, Electromechanics and Power Electronics Group, Eindhoven University of Technology, Eindhoven, The Netherlands. E-mails: n.gashi@student.tue.nl. v.t.t.lam@tue.nl and g.papafotiou@tue.nl

of the underlying optimization problem is of the order of $\mathcal{O}(27^{3L})$, and the typical length of a modulation cycle for such a PE application, within which the optimal solution must be found, would be few 10s (for medium power) or mostly 100s (for high power) of microseconds. We are threfore aiming at a real-time implementation that allows for *(i)* solving the oaSVM problem in the permissible time intervals, and *(ii)* increasing the length of the optimization window without excessively increasing the FPGA resources required for the solution of the more complex problem that arises.

A number of past works have already reported on the use of FPGAs for the development of computationally intensive, optimization-based control techniques in PE applications. In [8], the authors are using an explicit MPC approach to the design of the outer control loop of a dc-dc converter and subsequently implement the evaluation of the control law in real-time on a FPGA. More common in the literature are direct MPC approaches, where the outer control loop selects the switch positions directly and the modulator is then removed. The authors in [9] are using an exhaustive search approach with a branch-and-bound algorithm; however, the prediction horizon is fixed in advance through simulations, and the presented FPGA implementation is not scalable. The use of sphere decoding algorithms has also gained significant attention, see [10], [11]. In these cases, the problem considered is based on a quadratic objective, rather than a non-linear one as in oaSVM; furthermore, the dynamics and need for control of the converter's neutral point potential are neglected. Finally, in [12], the authors are employing, and ultimately implement on an FPGA, a neural network that replaces the enumerative optimization search that direct MPC requires. To the best of our knowledge, this is the first time it is attempted to formulate the modulation problem of the three-level NPC converter as a non-linear mixed-integer optimization problem, including minimization of the semiconductor losses and control of the neutral point, and then develop a dedicated and scalable digital control hardware architecture to facilitate its real time implementation in the microsecond scale.

The rest of the paper is structured as follows. Section II introduces the application background. Section III explores the digital hardware architecture proposal, and Section IV presents the integration of oaSVM into it. Finally, Section V presents performance results, and Section VI concludes the paper.

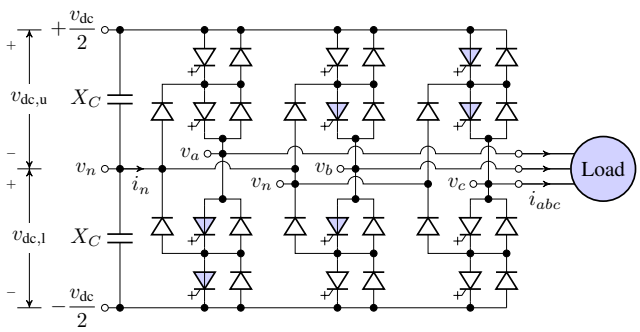


Fig. 2: The considered three-level neutral-point-clamped inverter.

II. Preliminaries

A. Three-level NPC inverter

The topology of the three-level NPC inverter is shown in Fig. 2 [13, Fig. 2.18, p. 55], which consists of IGCTs and diodes. Here, the dc-link voltages are $v_{\text{dc,u}} = \frac{v_{\text{dc}}}{2} - v_n$ and $v_{\text{dc,l}} = \frac{v_{\text{dc}}}{2} + v_n$, where v_n denotes the neutral-point voltage.

In each phase $x \in \{a, b, c\}$, the *on*- and *off*-states of the IGCTs can be modeled as a switch position that can take one of the three values $u_x \in \{-1, 0, 1\}$. In total, $3^3 = 27$ combinations of $u_{abc} = \begin{bmatrix} u_a & u_b & u_c \end{bmatrix}^T$ are possible and each can be mapped into the orthogonal $\alpha\beta$-reference frame as a so-called *voltage vector* v using the following relation [13, eq. 2.13, 2.77, p. 35, 56]

$$v = \frac{v_{\text{dc}}}{2} \underbrace{\frac{2}{3} \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix}}_{P} u_{abc}. \tag{1}$$

B. oaSVM algorithm

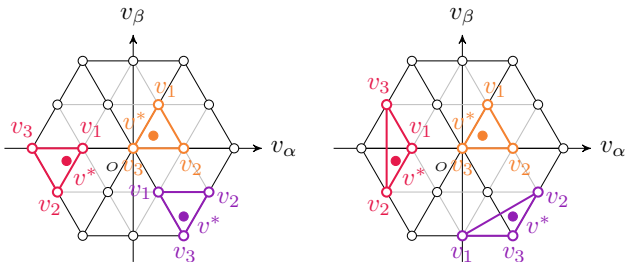
The modulation method oaSVM is based on space vector modulation (SVM). In each p-th modulation cycle T_s, SVM chooses the three (out of the available 27) voltage vectors $v_{p,1}, v_{p,2}, v_{p,3}$ which form the closest triangle that contains the reference voltage v_p^* in the $\alpha\beta$-reference frame, see Fig. 3a. Then, v_p^* is reconstructed by applying each vector for $t_{p,1}, t_{p,2}, t_{p,3} \geq 0$ seconds, respectively, i.e.

$$\begin{aligned} t_{p,1}v_{p,1} + t_{p,2}v_{p,2} + t_{p,3}v_{p,3} &= T_s v_p^* \\ t_{p,1} + t_{p,2} + t_{p,3} &= T_s. \end{aligned}$$

By defining $\bar{t}_{p,q} = \frac{t_{p,q}}{T_s}$ for $q = 1, 2, 3$, one can see that

$$\begin{aligned} \bar{t}_{p,1}v_{p,1} + \bar{t}_{p,2}v_{p,2} + \bar{t}_{p,3}v_{p,3} &= v_p^* \\ \bar{t}_{p,1} + \bar{t}_{p,2} + \bar{t}_{p,3} &= 1. \end{aligned} \tag{2}$$

From (2) and $\bar{t}_{p,q} \geq 0$, the reference voltage v_p^* is a convex combination of the three voltage vectors $v_{p,1}, v_{p,2}, v_{p,3}$. Therefore, oaSVM considers any triangle formed by the three voltage vectors that contains v_p^*, see Fig. 3b, and chooses the one that gives the best loss-THD trade-off.



(a) SVM: the *closest* triangle. (b) oaSVM: *any* triangle.

Fig. 3: Illustrating the concept of how oaSVM has the freedom to choose any triangle that satisfies hardware imposed constraints, while SVM is limited to choosing the nearest triangle that contains the requested voltages.

As mentioned before, L is the number of upcoming modulation cycles we consider over which the reference voltage path is assumed to be known and needs to be reconstructed. The voltage and current paths refer to sequences of reference voltages, denoted as $V^* = \begin{bmatrix} v_1^* & \cdots & v_L^* \end{bmatrix}$, and load currents, denoted as $I = \begin{bmatrix} i_1 & \cdots & i_L \end{bmatrix}$, respectively. These sequences are defined in the $\alpha\beta$-reference frame and are generated by the outer control loop.

By minimizing a cost function, oaSVM tries to improve the trade-off of the switching losses and current THD, while also balancing the neutral point voltage in the three-level NPC inverter [14]. For each p-th modulation cycle, i.e. triangle, the following cost function is used

$$f_p = w_{v_n,p} v_{n,p}^2 + w_{\text{loss},p} \left(\sum_{q=1}^{3} \sum_{r=a,b,c} \bar{\mathcal{E}}_{p,q,r} \right)^2 + w_{\text{THD},p} \left(\sum_{q=1}^{3} \left(v_{p,q,\alpha} - v_{p,\alpha}^* \right)^2 + \left(v_{p,q,\beta} - v_{p,\beta}^* \right)^2 \right),$$

Here, $\bar{\mathcal{E}}_{p,q,r}$ denotes the switching energy losses using [13, tab. 2.5, p. 61], $v_{n,p}$ denotes the neutral-point voltage after applying the three voltage vectors and $w_{v_n,p}, w_{\text{loss},p}, w_{\text{THD},p}$ denote the weights which can be tuned.

The decision variable U is given as

$$U = \begin{bmatrix} u_1 \\ \vdots \\ u_L \end{bmatrix}, u_p = \begin{bmatrix} u_{p,1} \\ u_{p,2} \\ u_{p,3} \end{bmatrix}, u_{p,q} = \begin{bmatrix} u_{p,q,a} \\ u_{p,q,b} \\ u_{p,q,c} \end{bmatrix},$$

where u_p with $p = 1, ..., L$ denotes the set of three switch positions $u_{p,q}$, which correspond to three voltage vectors $v_{p,q}$ using (1) with $q = 1, 2, 3$. Note that, only u_1 is sent to the inverter every T_s. Then, oaSVM finds the optimal solution U_{opt} such that

$$\min f = \sum_{p=1}^{L} f_p \quad \text{s.t. (2) and } \bar{t}_{p,1}, \bar{t}_{p,2}, \bar{t}_{p,3} \geq 0.$$

C. Solving MINLP using branch-and-bound (BnB) method

The problem introduced above is a Mixed-Integer Nonlinear Program, owing to the presence of integer decision variables and the nonlinear terms included in the cost function. Finding solutions to MINLP problems typically requires considerably more computational effort, compared to continuous optimization problems [15], and *branch-and-bound (BnB)* is commonly used for a more efficient search of the solution space [16, ch. 1, p. 6]. .

The BnB method constructs a search tree over all the possible integer combinations, and starts the search for the optimal solution from the top root node. The algorithm searches until the last nodes have been reached (which are known as *leaf nodes*). BnB will start searching nodes on a branch, and when a node satisfies the constraints, its cost is computed, and it will then be evaluated by a bounding function whether to branch into the node (unless it is a leaf node), into a new set of nodes that comes from the node before it. In the case that a node does not satisfy the constraints, then the branches of that node are *pruned*, and do not need to be searched further since they cannot contain the optimal solution. BnB also keeps track of the best-found selection of nodes so far, and uses it as a bounding function. This means the selection of nodes that have a cost higher than the best-found selection are directly pruned, and not searched further as they are guaranteed to not contain the optimal solution [16, ch. 1, p. 6]. In this way, the search for an optimal solution can be done more efficiently without having to explore all the nodes on a tree. This concept is illustrated in Fig. 4

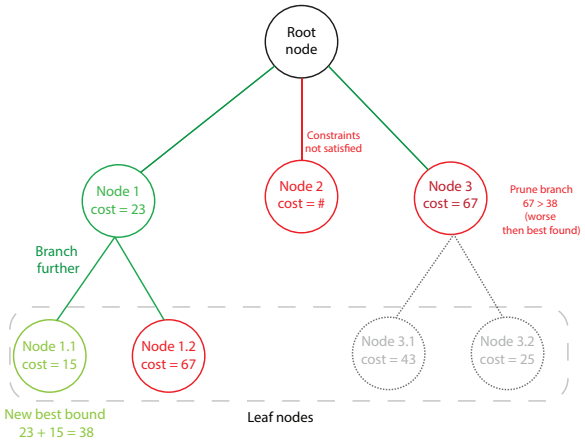


Fig. 4: Branch-and-bound method visualized. The search starts from node 1, it then branches further and finds a best-known bound with a total cost of 38. It then backtracks and proceeds to node 2. However, node 2 does not satisfy the constraints, so it gets pruned without computing its cost. Finally, node 3 satisfies the constraints, but its cost exceeds the bound found, and it gets pruned as it cannot contain an optimal solution.

III. Proposed Hardware Architecture

The proposed hardware architecture comprises three main parts: the process control unit (PCU), the node computation unit (NCU) and the branch-and-bound unit (BnB). We abstracted the architecture in such a way that the NCU does not have information about the tree depth or the search but rather only evaluates nodes, for which the relevant data is provided from the PCU. Nevertheless, the BnB unit requires information on tree depth in order to compute new bounds. The reason for this abstract structure is to allow for easy scalability of the tree depth and enable the possibility of searching multiple nodes in parallel to speed up the search (which is beyond the scope of this paper).

A. The process control unit (PCU)

The process control unit manages the entire search process, controls tree exploration, and coordinates the operation of other units. To enable tree searching, as shown in Fig. 4, the process control unit utilizes two finite-state machines (FSMs): one to track tree depth exploration, see Fig. 5a, and another to coordinate unit operation for searching nodes at a specific tree depth, as shown in Fig. 5b.

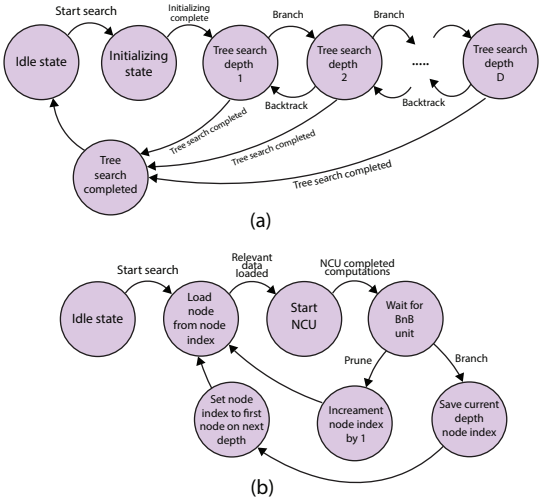


Fig. 5: Finite state machines of the process control unit: (a) the tree depth exploration FSM. FSM (b) is used to perform the searching at a tree depth, hence iterating through the nodes at a tree depth.

Initially, the FSMs start with initializing states. Some problems may require some procedures before the tree searching starts, such as processing the input data (which is the case with oaSVM as will be seen later). Nevertheless, the initialization states may be skipped for the case where no initialization is needed, and it can proceed directly to tree searching. After initialization is complete (or skipped) and the system is ready for tree searching, the FSM transitions to the tree search depth states (on the tree depth exploration FSM, see Fig 5a). These states keep track of which tree depth the current search is happening. Moreover, each node in the tree-searching process is assigned an index number, so that there is information as to which node is currently being evaluated. Before evaluating a node, first the relevant data for the node is loaded (or generated). After loading is done, the NCU starts computing the cost of the node and checks if constraints are satisfied. Once the NCU has finished, it will signal to the PCU, and then the PCU activates the BnB unit to check if the node should be pruned or branched further. If it results in branching further, then the tree-depth exploration FSM transitions to the next depth level, otherwise no branching happens, and the tree-depth exploration FSM stays in the same state. Moreover, when a branch is finished and all nodes have been searched, the tree-depth exploration FSM jumps one state down, and the search proceeds on that tree depth level from where it was left of. When the whole tree has been evaluated, the PCU will signal to the BnB to output the optimal solution

By abstracting the search process in this manner, expanding the tree depth only requires adding more states and other recurring elements to the PCU and the BnB unit associated with the increase in tree depth, such as more registers, without modifying the NCU. Essentially, the NCU operates solely on the current node iteration, unaware of the tree depth or process state, while the PCU manages the tree search. An example of recurring elements per each tree depth may be registers which store the value of the node before it transitions to the next depth.

The choice to add more states to the FSM and other recurring elements in order to increase the maximum tree search depth, instead of having one register that counts the tree depth and using memory such as BRAM or RAM for recurring elements, is made to minimize latency. A recurring element such as a basic register, can be accessed instantaneously. While, if these recurring elements would be dynamically allocated on BRAM, accessing and writing the data may take a minimum of two clock cycles or more, which is undesired due to increasing the latency of basic data accesses.

B. *The branch-and-bound (BnB) unit*

The BnB unit's main responsibility is to signal the PCU when to branch further and when to prune the branch. It performs so by utilizing registers that store the best cost found, and the node selection that has given the best cost. The BnB unit takes as input the current tree depth (which comes from the PCU), the evaluated node's cost, and whether the node satisfies the optimization constraints (which come from the NCU). When the tree searching starts, the BnB initially has a high cost (if the objective is to minimize) or a zero cost (if the objective is to maximize). During the tree exploration, when a selection of nodes has been done such that it has reached the leaf node, its total cost is first compared to the best cost found. In case it is better than the best cost, then the best cost is updated with this selection, hence this refers to a new bound being computed in the branch-and-bound method. Moreover, if a selection of nodes has a worse cost than the best found, even though the leaf node has not been reached, the BnB unit signals for the branch to be pruned to the PCU.

C. *Node computation unit (NCU)*

The node computation unit has two main tasks: *(i)* checking if the current node satisfies optimization constraints and *(ii)* calculating its cost. The unit's structure may vary depending on the optimization problem's cost function. It takes node-specific input data and provides the node's cost and constraint satisfaction as output.

IV. Integrating oaSVM into the proposed HW architecture

A. Triangles as nodes

In order to implement oaSVM in hardware, a table of all possible voltage space vectors that form a triangle is generated through MATLAB. Since, on a three-level inverter there are 27 possible voltage space vectors, the total number of triangles is $27^3 = 19683$. Nevertheless, we only consider the triangles that satisfy the switching constraints, which are given in [13, ch. 2.4.1.6, p. 59-60]. Moreover, we also remove the triangles that have an area close to 0, as they are degenerate triangles and cannot contain any point inside [17, ch. 5.2, p. 175].

After applying these constraints, the total list drops down to just 612 triangles. Each of these triangles is a node in a tree and has a specific index, which corresponds to a memory address inside the BRAM from 0 to 611. Hence, now the total number of nodes to be searched is 612^L instead of 19683^L.

Furthermore, we can exploit the symmetry of the space vector diagram. A fine grid of reference voltages (v_s^*) is generated for each sector. The presence of each grid point within triangles is examined, revealing that 161 triangles contain at least one point in this sector. Subsequently, the 612 triangles are grouped into sets of 161 triangles, as depicted in Fig. 6, and stored in block random access memory (BRAM). In this way, only the triangles on the sector where the requested voltage v_s^* is located are iterated through, reducing even further the search space from 612 triangles to just 161 triangles. To enable this, before starting the tree search, the sector in which the requested voltage lies has to be decoded first (which is an example of initializing states used in oaSVM). The triangles in the BRAM can be rearranged to provide a warm start to the search, without any additional hardware, which can speed-up the search. In each set of the 161 triangles, by firstly placing equilateral triangles in the beginning, and then the rest of the triangles, essentially the search starts with computing traditional SVM (hence the equilateral triangles), which can be assumed as a warm-start, and then proceeds to triangles which traditional SVM does not consider, but oaSVM does.

B. oaSVM integration

The specifics of the oaSVM integration into the proposed architecture is illustrated in Fig. 7. As it can be observed, it contains the triangle LUT, which is the triangle table discussed in the previous subsection, stored in BRAM. The PCU provides the triangle index input (hence the BRAM address memory) to load the data for the triangles. Inside the NCU, there is the constraints checker module, which makes sure that the constraints are satisfied, and the rest of the three modules compute the switching loss cost, THD cost and neutral-point voltage balance cost. These sub-units operate in parallel to speed up the computation, and their results are all added together as the total cost for that node. It is important to note, that the accumulation of the cost, between tree depth 0 and tree depth L is chosen to be done on the BnB unit, such that NCU stays independent of tree depth information.

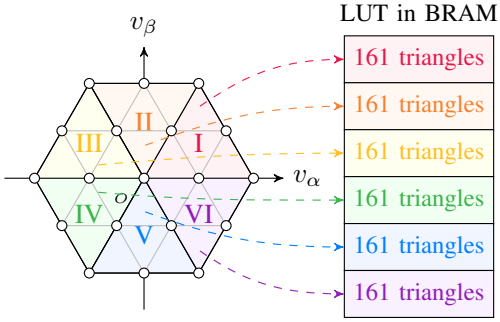


Fig. 6: Organization of the triangles which contain at least a point in a sector.

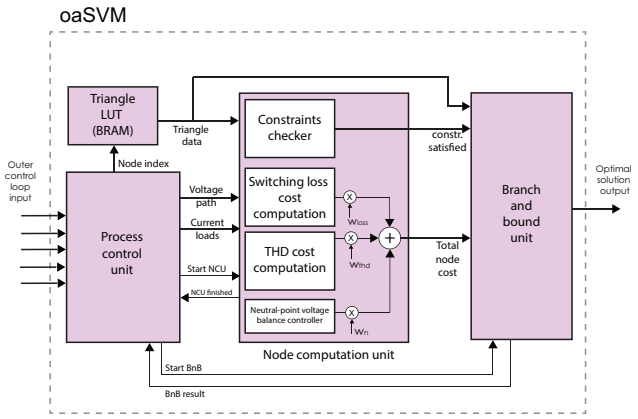


Fig. 7: Illustration of oaSVM implemented in the proposed architecture.

Finally, the BnB module will have stored the best selection of nodes, i.e. triangles, which give the optimal switching positions at the output.

It should be noted that the proposed implementation allows for the online adjustment of the parameters of the optimization problem, such as the cost function weights or the length of the opimization window, online during run time. This is an important feature as it renders the applicability of the method more general, providing flexibility to tune the behavior of the new modulator not only at the time of application deployment but also during run-time.

V. Performance Results

The proposed architecture, with oaSVM integrated into it, is implemented on an AMD Xilinx ZYNQ 7020 (xc7z020clg400-1) system-on-a-chip (SoC), which consists of an FPGA part, but also has more features integrated to it, such as two ARM-A9 Core CPUs [18].

The performance evaluation of the proposed scheme is done in terms of computational performance, as well as performance of the power electronics system's physical variables in a closed-loop simulation. For this, we employ HIL (Hardware-in-the-loop) simulations with MATLAB using the electric drive setup presented in the introduction, in order to validate the design and measure its performance on an FPGA. The oaSVM on the FPGA receives as input the required data from MATLAB, such as the requested voltage path and current loads, for L modulation cycles. It then computes and outputs the optimal switching positions for the next modulation cycle, as well as the number of clock cycles that were required for finding the solution. The rest of the electric drive setup is modeled in MATLAB.

In order to achieve the communication between MATLAB and the FPGA, we make use of the universal serial bus (USB) protocol and the ARM-CORTEX A9 CPU inside the ZYNQ 7000 SoC. The CPU acts as an intermediate step between taking the input data coming from MATLAB via the serial bus, parsing it to the necessary format, and delivering it to the FPGA via the AXI4 interface [19]. Then, the CPU starts the solver in the FPGA and waits until the solver is finished. Once it is finished, the CPU takes the optimal switching

TABLE I: The minimum, maximum, average amount of clock cycles N_{clk} and the problem size with respect to optimization window. Used abbreviations are: min.: minimum; max.: maximum; avg.: average.

Optimization window	**Problem size (total nodes)**	**Min.** N_{clk}	**Avg.** N_{clk}	**Max.** N_{clk}
1	161	1128	1170	1226
2	25921	2340	9032	32318
3	4173281	1172	106600	2396200
4	671898241	1180	3656200	43776000

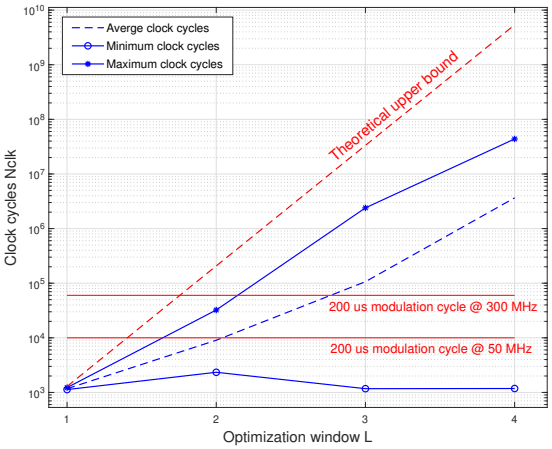


Fig. 8: Computational performance of the oaSVM implementation: Number of clock cycles needed vs optimization window.

positions and the number of clock cycles it took to solve and delivers this data to MATLAB for the next modulation cycle. This repeats for every sample of the simulation.

A. *Computational performance*

As a metric for measuring the computational performance, we use the number of clock cycles needed for the algorithm to provide the solution. In Table I, one can see the problem size in terms of number of search nodes, and the (minimum, maximum and average) clock cycles that were required to reach a solution, as observed in the simulations, with respect to the length of the optimization window L. These are also graphically depicted in Fig. 8, where, for completeness, the theoretical upper bound of clock cycles that relates to the full enumeration of all permissible triangles is also presented. Moreover, in order to put these numbers into the context of the application, we show the number of clock cycles available within a typical modulation cycle interval of 200 μs, corresponding to two different FPGA speeds: one running at a clock speed of 50 MHz (standard commercial speed grade) and the other at 300 Mhz (speed limit of existing commercial FPGAs).

As can be observed, the proposed oaSVM implementation is feasible for the case of $L = 1$. This is already a much more versatile, powerful but also complex modulation concept compared to standard SVM. For the case of $L = 2$, the solution is within the range of the capabilities of the

Resource name (FPGA)	$L = 1$	$L = 2$	$L = 3$	$L = 4$	**Util.** $N_m = 4$
LUT	3186	4010	4382	4636	8.71%
LUTRAM	62	62	62	62	0.36%
FF	2316	3146	4558	5240	4.92%
DSP	64	64	64	64	29.0%
BRAM	15.5	15.5	15.5	15.5	11.07%

TABLE II: FPGA Implementation resources as L increases. The last column contains the resource utilization (%) (for $L = 4$) based on available resources inside ZYNQ-7020 (xc7z020clg400-1) FPGA fabric. Used abbreviations are: util.: utilization.

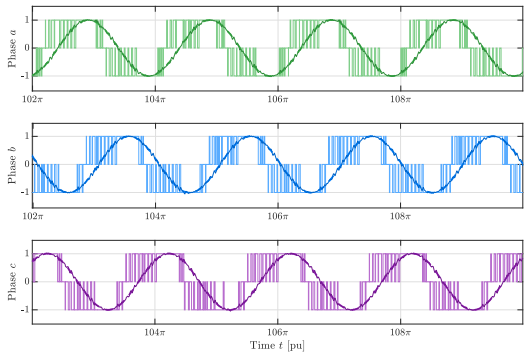


Fig. 9: Switching position u_x generated from the FPGA and normalized stator currents $i_{s,x}$ plotted per phase x, for $L = 2$, $T_l = 1$ pu, $\omega_r = 1$ pu and $T_s = 300$ μs.

existing computational HW - although not for the very conservative theoretical upper bound. For $L = 3$, algorithmic improvements in the order of magnitude are required for feasibility. These observations motivate the future directions of work. Firstly, the development of solution strategies that can mathematically guarantee a feasible, even if sub-optimal, solution within a given time that is shorter than the modulation cycle. This would enable the deployment of the method on real power electronics HW, where a good enough solution within the modulation cycle needs to be guaranteed to avoid destroying the semiconductors. Secondly, the development of suitable parallelization strategies that would allow for a more than 10-fold performance improvement to render the real-time solution for $L = 3$ feasible – without, however, exhausting the resources of the FPGA and annulling the scalability properties of the design.

The FPGA resource usage for increasing L can be found in table II. Due to the way that the proposed architecture is built, increasing L only requires extending the tree-depth exploration FSM, which requires adding LUTs and FFs. Moreover, it avoids the addition of DSP elements which are usually very limited on the FPGA. This demonstrates the scalability of the proposed architecture for various optimization window sizes (L) without imposing resource restrictions.

B. *Closed-loop drive performance*

Fig. 9 shows the switching positions and the currents per phase, of a HIL simulation, consistent with the results in [7]. Furthermore, Fig. 10 shows the closed loop performance

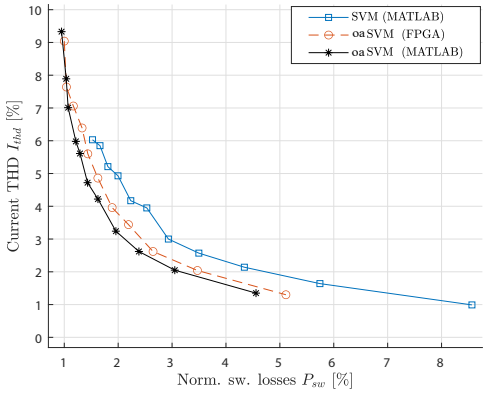


Fig. 10: Trade-off between current THD and normalized switching losses P_{sw}, between SVM and oaSVM running in oaSVM and FPGA, with $N_m = N = 1$. Simulated with T_s from $600\mu s$ to $100\mu s$ with steps of 50 μs for $T_l = 1$ pu and $\omega_r = 1$ pu. Abbreviations are: norm.: normalized; sw.:switching.

of the proposed oaSVM implementation, in terms of the observed trade-off between switching losses and harmonic content of the current – the standard way to evaluate the performance of a modulator in power electronics applications. oaSVM on the FPGA behaves similarly to the ideal version of MATLAB and better than SVM. Nevertheless, there is a discrepancy between the implementation on MATLAB and the real-time version due to the following reasons: *(i)* The FPGA version uses a different method for balancing the neutral point of the inverter, which is computationally faster but alters the switching behavior, and *(ii)* the FPGA limited numerical precision can have an impact on the choice of switch positions. Nevertheless, the impact is not significant.

VI. Conclusions

In this paper, a scalable FPGA-based architecture for real-time implementation of a branch-and-bound algorithm for solving mixed-integer optimization problems has been presented. The need for such a solver is motivated by new modulation method for power electronics applications, oaSVM, which is based on the online solution of a complex mixed-integer optimization problem in the range of tens or hundreds of microseconds. We have shown how oaSVM can be integrated into this FPGA-based architecture, which can serve as an example for future use cases. Most important, the method is general and can be extended or adjusted for application to other problems that require fast mixed-integer optimization.

For validation and performance evaluation, hardware-in-the-loop simulations between FPGA and MATALB have shown that the proposed architecture can successfully run oaSVM for optimization windows of 1-2 in the desired time frame, while maintaining the flexibility of adjusting a number of parameters of the optimization problem, such as the cost function weights or the length of the opimization window, online during run time. Equally important, the abstraction of the proposed architecture lends itself to a number of improvements and extensions, such as the parallelization of the tree search to further speed up the computation time. Such extensions are subject of ongoing and future work and are expected to enable the full exploitation of the method's flexibility for significant application performance improvements.

References

[1] T. Gupta, R. Boudreaux, R. M. Nelms, and J. Y. Hung, "Implementation of a fuzzy controller for dc-dc converters using an inexpensive 8-b microcontroller," *IEEE transactions on Industrial Electronics*, vol. 44, no. 5, pp. 661–669, 1997.

[2] S. K. Sahoo, G. T. R. Das, and V. Subrahmanyam, "Contributions of fpgas to industrial drives: A review," 2007.

[3] E. Monmasson, L. Idkhajine, M. N. Cirstea, I. Bahri, A. Tisan, and M. W. Naouar, "Fpgas in industrial control applications," *IEEE Transactions on Industrial informatics*, vol. 7, no. 2, pp. 224–243, 2011.

[4] G. A. Papafotiou, G. D. Demetriades, and V. G. Agelidis, "Technology readiness assessment of model predictive control in medium-and high-voltage power electronics," *IEEE Transactions on Industrial Electronics*, vol. 63, no. 9, pp. 5807–5815, 2016.

[5] S. Vazquez, J. Rodriguez, M. Rivera, L. G. Franquelo, and M. Norambuena, "Model predictive control for power converters and drives: Advances and trends," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 2, pp. 935–947, 2016.

[6] D. G. Holmes and T. A. Lipo, "Pulse Width Modulation for Power Converters: Principles and Practice," *Wiley-IEEE Press*, 2003.

[7] V. T. T. Lam and G. Papafotiou, "Online adaptive space vector modulation," in *12th International Conference on Power Electronics, Machines and Drives (PEMD 2023)*, vol. 2023. IET, 2023, pp. 426–433.

[8] M. Jeong, S. Fuchs, and J. Biela, "When fpgas meet regionless explicit mpc: An implementation of long-horizon linear mpc for power electronic systems," in *IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society*, 2020, pp. 3085–3092.

[9] S. Baltruweit, E. Liegmann, P. Karamanakos, and R. Kennel, "Fpga-implementation friendly long-horizon finite control set model predictive control for high-power electronic systems," in *2021 IEEE 12th Energy Conversion Congress Exposition - Asia (ECCE-Asia)*, 2021, pp. 1823–1828.

[10] T. Dorfling, H. du Toit Mouton, T. Geyer, and P. Karamanakos, "Long-horizon finite-control-set model predictive control with nonrecursive sphere decoding on an fpga," *IEEE Transactions on Power Electronics*, vol. 35, no. 7, pp. 7520–7531, 2020.

[11] B. Stellato and P. J. Goulart, "Real-time fpga implementation of direct mpc for power electronics," in *2016 IEEE 55th Conference on Decision and Control (CDC)*, 2016, pp. 1471–1476.

[12] F. Simonetti, A. D'Innocenzo, and C. Cecati, "Neural network model-predictive control for chb converters with fpga implementation," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 9, pp. 9691–9702, 2023.

[13] T. Geyer, "Model Predictive Control of High Power Converters and Industrial Drives," *John Wiley & Sons, Ltd*, 2017.

[14] N. Celanovic and D. Boroyevich, "A comprehensive study of neutral-point voltage balancing problem in three-level neutral-point-clamped voltage source pwm inverters," *IEEE Transactions on power electronics*, vol. 15, no. 2, pp. 242–249, 2000.

[15] M.-W. Huang and J. S. Arora, "Optimal design with discrete variables: some numerical experiments," *International Journal for Numerical Methods in Engineering*, vol. 40, no. 1, pp. 165–188, 1997.

[16] J. Lee and S. Leyffer, *Mixed integer nonlinear programming*. Springer Science & Business Media, 2011, vol. 154.

[17] P. Schneider and D. H. Eberly, *Geometric tools for computer graphics*. Morgan Kaufmann Publishers, 2002.

[18] "Zynq-7000 product page." [Online]. Available: https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html

[19] "Axiprotocol." [Online]. Available: https://www.xilinx.com/products/intellectual-property/axi.html