International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. XX, No.X (XXX-XX)

Optimizing State Space Integral Controllers for DC-DC Buck Converters Using FPGA-Based Genetic Algorithms

Mini K. Namboothiripad[1]

[1]Department of Electrical Engineering,
[1]Agnel Charities' Fr. C. Rodrigues Institute of Technology, Vashi
[1] Navi-Mumbai, Maharashtra, India.

Abstract: The state space integral controller approach is highly effective for precise and efficient voltage control of DC-DC buck converters, which are crucial in a wide range of applications. This paper presents strategies for implementing a Genetic Algorithm (GA) to tune the gain parameters of a state feedback controller with integral action using FPGA technology. The GA optimizes controller gains to achieve desired dynamic performance and zero steady-state error under multiple constraints. By leveraging the parallel processing capabilities of FPGAs, the GA's parallel behavior can be effectively utilized to accelerate computations. We implemented the GA on a PYNQ-Z2 SoC FPGA using Vivado high-level synthesis tools, starting with a population of 12 solutions and running for 20 iterations. Our results show that the GA effectively tunes gains to meet various overshoot and settling time requirements while maintaining zero steady-state error. Additionally, we observed a 5.3-fold speed-up in execution with our 100 MHz customized design compared to the 650 MHz ARM processor. By using an FPGA board with more resources, the design clock frequency and thus the throughput and acceleration could be improved further. Integrating GA with FPGA technology significantly reduces the latency for computation making it highly beneficial for any GA based real-time applications.

Keywords: Buck Converter, FPGA, Genetic Algorithm, High Level Synthesis, State Space Integral Controller, Vivado Tools

1. Introduction

Buck converters, also known as step-down DC-DC converters, are crucial in modern power electronics for their ability to efficiently reduce higher input voltages to lower output voltages. Their cost-effectiveness and flexibility make them the preferred choice for a wide range of applications, including consumer electronics, automotive systems, telecommunications, industrial automation, and embedded systems [1]–[4].

Buck converters are essential components for ensuring efficient voltage regulation [5], [6]. They minimize energy loss and heat generation, allowing for compact and space-saving designs [7]. These converters provide stable voltage levels that are vital for the reliable operation of laptops, smartphones, LED drivers etc. [8].

Effective voltage control in buck converters is essential for ensuring stable output and achieving optimal performance. To improve both transient and steady-state behavior, various control strategies are suggested in the literature, such as Proportional-Integral-Derivative (PID), fuzzy, model predictive, artificial neural network (ANN), state feedback control etc. [9]–[12].

PID control is widely favored due to its simplicity and effectiveness in maintaining the desired output voltage by adjusting the duty cycle of the pulse-width modulation (PWM) signal [13]–[15]. However, PID control technique relies on a linearized system model for the controller design. While Fuzzy logic and ANN approaches are suitable for handling nonlinearities, they often lack formal mathematical analysis [12]. Model predictive control offers robust performance but is highly dependent on the system model and can be computationally expensive [16]. In contrast, state-space control effectively handles multi-variable systems and constraints, making it suitable for complex and high-performance applications [17]. It provides a comprehensive approach by modeling the DC-DC converters using first-order differential equations that describe the system in terms of state variables, such as inductor current and capacitor voltage [18]–[21].

Various papers delve into the detailed modeling, gain design and loop implementation aspects of DC-DC converters using state-space representation [22]–[26]. Additionally, the design and implementation of state feedback with integral controllers for DC-DC converters, highlighting their enhanced performance, is explained by many researchers

[16], [27]. By utilizing feedback from internal states and including integral action, these methods enable precise regulation of output voltage and improved dynamic response [28]. However, in this case, tuning involves the solving of a higher order polynomial with multiple optimization constraints which may have to be handled using a heuristic approach.

Genetic Algorithm (GA) is one of the most effective heuristic method, known for its robust global optimization capabilities in complex, non-linear search spaces. By mimicking natural evolutionary processes, GA efficiently explores and exploits the search space to find near-optimal solutions for a wide range of optimization problems. Many papers are available, in which the PID controller tuning is optimized using GA, due to its robust global search capability and effectiveness in handling non-linear optimization problems [29]–[32]. The GA-based approach ensures an optimal set of PID parameters by iteratively improving solutions through selection, crossover, and mutation processes, ultimately enhancing system performance metrics such as overshoot, settling time, and steady-state error.

GAs are inherently parallel and can be easily implemented on parallel computing architectures such as Field-Programmable Gate Arrays (FPGA) [33], [34]. Exploiting this parallelism allows for faster convergence to optimal solutions, making them suitable for real-time applications. This high-performance integration allows GAs to solve complex optimization problems more rapidly and effectively, leveraging the hardware acceleration provided by FPGAs [35], [36]. However, expertise in hardware logic design and programming is necessary for the FPGA based computations. With the development of high level synthesis tools, the algorithm can be coded using high level languages such as C, Python etc. and can be converted to hardware descriptive languages for the implementation on FPGA [37], [38].

In this paper, we employ a state space approach to control the transient and steady-state behavior of the buck converter, a method well-suited for handling nonlinear dynamics. Incorporating a state feedback controller with integral action enables us to attain the desired transient response while eliminating steady-state error. However, the increased system order presents challenges in precisely placing the poles to meet multiple optimization constraints. To overcome this complexity, we advocate for utilizing GA to fine-tune the controller gains, harnessing their heuristic optimization capabilities. Moreover, by implementing the GA on an FPGA, we can capitalize on parallel processing to accelerate convergence towards an optimal solution.

Numerous studies explore voltage control in buck converters, including those employing the state space approach. However, our method introduces a novel approach by utilizing FPGA-based GA for tuning the gain parameters to achieve desired behavior. We implemented the GA algorithm on a PYNQ-Z2 SoC FPGA, initializing a population with 12 solutions and execute for 20 iterations. Our findings confirm the effectiveness of the algorithm in tuning gains for various desired overshoot and settling times, all while maintaining zero steady-state error.

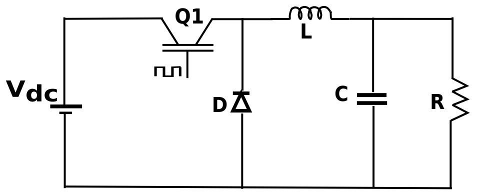


Figure 1. DC-DC Buck Converter Circuit

Remarkably, with our 100 MHz custom IP, we observed a 5.3 times acceleration in execution speed compared to running the algorithm on a 650 MHz ARM processor. The acceleration remained consistent even with a slightly larger population or more iterations, despite the custom IP design operating at a clock frequency of 70 MHz, due to the resource constraints of the PYNQ-Z2 board. A higher speed-up is achievable for larger populations or more iterations with a more resource rich FPGA board. Our approach benefited significantly from the Vivado high-level synthesis tools developed by Xilinx, facilitating the translation of the 'C' coded algorithm into a hardware descriptive language for FPGA implementation.

2. Sate Space Modelling of Buck Converter

A buck converter, shown in Fig. 1, is a DC-DC power converter that reduces DC voltage from a higher level to a lower level. Due to the presence of two energy storage elements in a buck converter, it becomes necessary to define two state variables.

These variables can be selected as the current flowing through the inductor and the voltage across the capacitor. The state space representation of the buck converter can be derived using averaging techniques by incorporating both the ON and OFF states of the switch.

During ON condition, the state space representation can be derived as follows: Since the diode is OFF, by applying Kirchhoff's Voltage Law, the relation becomes,

$$V_L = V_i - V_c \tag{1}$$

where V_i is the input voltage and V_L and V_c are the voltage across the inductor and the capacitor respectively. With I_L as the current through the inductor with inductance L, the equation can be written as,

$$\frac{dI_L}{dt} = \frac{V_i}{L} - \frac{V_c}{L} \tag{2}$$

Considering I_c as the current through the capacitor and R as the resistance, the Kirchhoff's Current Law equation can

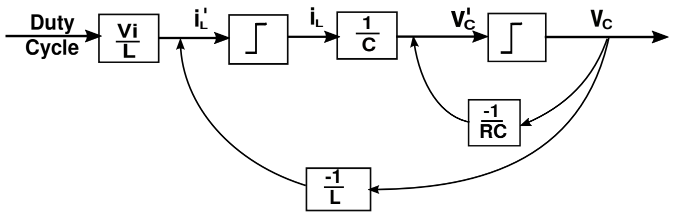


Figure 2. State Space representation of Buck Converter

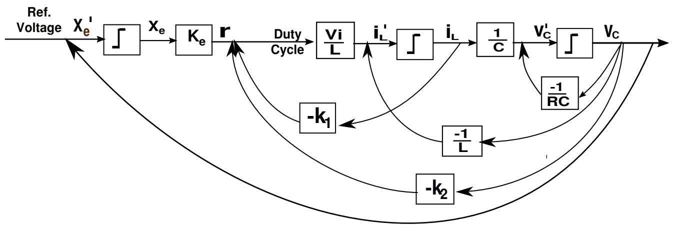


Figure 3. State Space representation of Buck Converter with State Feedback and Integral Controller

be expressed as,

$$I_c = I_L - \frac{V_c}{R} \tag{3}$$

which can be re-written as,

$$\frac{dV_c}{dt} = \frac{I_L}{C} - \frac{V_c}{RC} \tag{4}$$

Equation 2 and 4 can be expressed in matrix form as,

$$\begin{bmatrix} \frac{dI_L}{dt} \\ \frac{dV_c}{dt} \end{bmatrix} = \begin{bmatrix} 0 & \frac{-1}{L} \\ \frac{1}{C} & -\frac{1}{RC} \end{bmatrix} \begin{bmatrix} I_L \\ V_c \end{bmatrix} + \begin{bmatrix} \frac{1}{L} \\ 0 \end{bmatrix} V_i \tag{5}$$

$$V_o = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} I_L \\ V_c \end{bmatrix} \tag{6}$$

Similarly, during the OFF condition of the switch, with the diode ON, state space representation can be derived as,

$$\begin{bmatrix} \frac{dI_L}{dt} \\ \frac{dV_c}{dt} \end{bmatrix} = \begin{bmatrix} 0 & \frac{-1}{L} \\ \frac{1}{C} & -\frac{1}{RC} \end{bmatrix} \begin{bmatrix} I_L \\ V_c \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} V_i \tag{7}$$

$$V_o = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} I_L \\ V_c \end{bmatrix} \tag{8}$$

However, the control input to the buck converter is the ON/OFF signal to the switch with the duty cycle D. Thus state equation with input D can be derived using the averaging technique, ie. Equation 5 * D + Equation 7*(1-D) and can be simplified as,

$$\begin{bmatrix} \frac{dI_L}{dt} \\ \frac{dV_c}{dt} \end{bmatrix} = \begin{bmatrix} 0 & \frac{-1}{L} \\ \frac{1}{C} & -\frac{1}{RC} \end{bmatrix} \begin{bmatrix} I_L \\ V_c \end{bmatrix} + \begin{bmatrix} \frac{V_i}{L} \\ 0 \end{bmatrix} D \tag{9}$$

and the output equation is,

$$V_o = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} I_L \\ V_c \end{bmatrix} \tag{10}$$

The above state space model can be represented using the signal flow graph as shown in Fig. 2. The desired transient behaviour can be achieved by adding the state feedback gains k_1, k_2 from the state variables I_L and V_c respectively to the input. However, this arrangement does not make the output steady state value to the required value.

The error in the output is determined by taking the output feedback and comparing it with the reference value. The steady state error can be reduced to zero by adding an additional integral controller with gain K_e in the forward path. This entire arrangement is shown in Fig. 3. It is clear from the figure that, it adds one more integrator and the order of the system increases by one. Thus the state and the output equation need to be determined again. Considering the control input as Duty Cycle D, which is equal to $-k_1 I_L + -k_2 V_c + r$,

$$\frac{dI_L}{dt} = -\frac{V_c}{L} + \frac{V_i}{L}(-k_1 I_L + -k_2 V_c + r) \tag{11}$$

in which $r = K_e X_e$ where X_e is the newly added state variable due to the integral controller. However, it is clear from Fig. 3 that the equation for $\frac{dV_c}{dt}$ remains same as that of 4. The new state equation corresponds to the derivative of X_e is,

$$\frac{dX_e}{dt} = V_{ref} - V_o = V_{ref} - V_c \tag{12}$$

Thus the new state equation can be written in matrix form as,

$$\begin{bmatrix} \frac{dI_L}{dt} \\ \frac{dV_c}{dt} \\ \frac{dX_e}{dt} \end{bmatrix} = \begin{bmatrix} -\frac{V_i k_1}{L} & \frac{-1}{L} - \frac{V_i k_2}{L} & \frac{V_i K_e}{L} \\ \frac{1}{C} & -\frac{1}{RC} & 0 \\ 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} I_L \\ V_c \\ X_e \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} V_{ref} \tag{13}$$

and the output equation becomes,

$$V_o = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} I_L \\ V_c \\ X_e \end{bmatrix} \tag{14}$$

By comparing the characteristic equation of 13 with the desired characteristic equation, the controller gains k_1, k_2 and K_e can be determined. However, finding the characteristic equation of 13, which involves the determinant of $sI - A$ where A is the system matrix in 13, becomes complex due to the presence of s, especially for larger system matrices. Also, an optimum gains need to be determined to satisfy all the given transient and steady state parameters. Therefore, it is proposed to use heuristic optimization methods such as GA to determine the optimum value as the controller gains.

3. Genetic Algorithm to Tune the Controller Gains

Genetic Algorithm (GA) is a computational technique used to solve search problems by mimicking the natural selection process of evolution where the fittest individuals are more likely to survive and produce offspring. This algorithm operates on a population, a set of diverse solutions. Within this population, each solution is called as a chromosome, with its individual components termed as genes. GA iteratively refines candidate solutions over generations, by favoring the survival and reproduction of the most successful solutions.

Refining the solutions to generate new population involves various steps such as fitness evaluation, cross over, mutation etc. Initially, fitness of each individual is evaluated by using a user defined fitness function. Subsequently, highly fit individuals are forming pairs known as parents, and are chosen for reproduction. Reproduction involves crossover and mutation to facilitate the creation of potentially improved offspring.

Crossover is the main operation to explore the search space and it is performed with a probability called crossover probability P_C. This operation involves the mixing of chromosomes of parents to produce two offspring, facilitating the offspring to inherit a diverse mixture of desirable characteristics from both parents. By producing random changes in the offspring, called mutation, new characteristics can be introduced with the offspring. It enhances the exploration of the search space and thus the quality of the potential solution. The mutation operation is also performed with a probability called mutation probability, P_M.

In GA, the initial generation may either be specified by the user or generated randomly. Following the crossover and mutation, the new generation replaces the previous one, evolving iteratively until a predefined stopping criterion is reached. However, crossover and mutation introduce a risk of losing the optimal solution. To address this concern, a strategy is adopted such that the best individual from a population is safeguarded and included in the new population.

In our approach to determine the controller gains, an initial population of 12 feasible solutions, or chromosomes, is considered and the iterations are performed for 20 generations. The controller gains k_1, k_2 and K_e are considered as the genes in each chromosome. The fitness of each chromosome is assessed using an error function, which serves as the basis for selecting chromosomes for the next generation.

The genes of each chromosome, representing the controller gains k_1, k_2 and K_e, are used to determine the damping ratio ζ, natural frequency ω_n and settling time T_s. The error function, Err, is defined by comparing these values with their user specified desired values as given in equation 15.

$$Err = \alpha(\zeta_{cal} - \zeta_{des}) + \beta(T_{sdes} - T_{scal}) + \gamma(\omega_{ncal} - \omega_{ndes}) \quad (15)$$

Here, ζ_{cal} is the calculated damping ratio (from the chromosome) and ζ_{des} is the desired (user specified) damping ratio. Similarly, T_{sdes} and ω_{ndes} are the desired values of settling time and natural frequency respectively, and T_{scal} and ω_{ncal} are the values calculated from each chromosome. The fitness of each chromosome is determined by taking the inverse of the respective error value.

The values α, β and γ are weight factors for proper tuning of controller gains. In our design, these weight factors are made larger in case of $\zeta_{cal} < \zeta_{des}$ and $T_{scal} > T_{sdes}$ to ensure the percentage overshoot and the settling time do not exceed the desired values.

After evaluating fitness, the solutions are sorted based on their fitness values, and the top-performing solutions are selected for crossover operations. In our design, the top four solutions are chosen for crossover to generate offspring. These top solutions are paired as parents, and by exchanging some genes within each pair, new offspring are created. To extend the search space, neighbors of the fittest solutions are generated by averaging the genes of each pair with the top four solutions.

The new population consists of the top four solutions, four offspring from the crossover, and four neighbors of the fittest solutions. Mutation is then performed by slightly modifying a gene in any random chromosome, except for the top solution, to further explore the search space.

Now the process is continued for fixed number of iterations or generations to make the algorithm converge to a better solution. Overall algorithm can be summarized as follows:

Algorithm 1 :Genetic Algorithm to tune the controller gains

```
1: Initialize the number of populations, $Npop$
2: Initialize chromosomes in the initial population.
3: Initialize the number of generations, $NGen$
4: for ($Iter = 0; Iter < NGen; Iter++$) do
5:     for ($Sol = 0; Sol < Npop; Sol++$) do
6:         $\zeta_{cal}$, $T_{scal}$ and $\omega_{cal} \leftarrow$ Genes
7:         Error $\leftarrow$ Equation 15
8:         Fitness $\leftarrow \frac{1}{error}$
9:     end for
10:    Sort the population with the fitness values
11:    Parents $\leftarrow$ Top-performing chromosomes
12:    Offspring $\leftarrow$ Crossover operation with parents
13:    Neighbourhoods $\leftarrow$ Average(Top-performing ones)
14:    New population $\leftarrow$ Offspring, Neighbourhood & Top-performing ones.
15:    Mutated Population $\leftarrow$ Change any gene of any chromosome except the top-performer.
16: end for
17: Controller gains $\leftarrow$ Top-performer of last generation
```

Although this algorithm appears sequential, several parts, such as the fitness evaluation of each chromosome, the crossover operation, and the calculation of neighboring chromosomes for the top performers, can be executed in parallel. This parallelization can significantly reduce overall computational latency. Therefore, we propose using an

FPGA for the GA computation.

4. Genetic Algorithm Implementation on FPGA

An FPGA consists of components such as Look-Up Tables (LUTs), DSP blocks, flip-flops, I/O pads, Block RAM, and PLLs, which can be interconnected to perform desired computations. Although FPGA-based designs are reprogrammable, the development process involves several stages, including programming with Hardware Description Languages (HDLs), compilation, synthesis, fitting, and timing analysis. Consequently, the overall development process can be quite complex and time-consuming.

However, High-Level Synthesis (HLS) tools like Vivado HLS enable programming using C/C++, which is then converted to HDL. This approach makes it easier to accelerate algorithms with intensive computations, even for those with limited hardware coding knowledge. Additionally, Vivado HLx tools facilitate hardware-software codesign. This process involves implementing a subset of the desired system, which needs acceleration, on the programmable logic (PL) part of the System on Chip (SoC) FPGA board, while the remaining portion is implemented on the processing system (PS) available on the SoC. This approach allows for efficient utilization of both hardware and software resources.

The GA algorithm for tuning a state space based integral controller for a buck converter is implemented using Vivado tools on Xilinx's PYNQ-Z2 FPGA board, which belongs to the Zynq 7000 SoC family [39]. This board features 650 MHz ARM Cortex-A9 dual core PS with 512MB DDR-Dynamic RAM (DDR-DRAM) and PL with 13,300 logic slices (each with four 6-input look up tables (LUTs) and 8 flip-flops (FF)), 630 KB block RAM (BRAM) and 220 DSP slices.

It is decided to implement the entire GA on PL by taking the circuit parameters, desired damping ratio, natural frequency and settling time as input as shown in Fig. 4. The initial population is stored on the BRAM to reduce the communication latency between the PL and the PS. The outputs are the controller gains k_1, k_2 and K_e.

Our SoC-based FPGA implementation encompasses several key stages: First, we utilize Vivado HLS to convert C/C++ code into HDL code, which is then transformed into intellectual property (IP). Next, we employ Vivado HLx tools to integrate this customized IP with the ZYNQ processing system and generate the bitstream. Finally, we use Vivado Software Development Kit (SDK) to write code for the PS to control and communicate with the PL, and then build and download these to the FPGA.

Using the Vivado HLS tool, first, the GA algorithm functionality in C/C++ is verified by performing a C-simulation with a C-test bench. Next, the code is C-synthesized to generate RTL logic in VHDL/Verilog. During the synthesis, arguments in the top-level function are mapped into interfaces and input/output ports, sub functions are mapped into modules, and arrays into memory or registers depending upon their size and settings.

The AXI interface is the most commonly used for input/output ports, and the communication protocol between the PL and the PS is specified by the directive/pragma called *interface* [40]. The three AXI protocols are AXI4-Lite, AXI4, and AXI4-Stream, with corresponding directives *s_axilite*, *m_axi*, and *axis*. AXI4-Stream supports data bursts but is not mapped to memory, whereas AXI4-Lite and AXI4 transfer data with specified memory addresses. AXI4-Lite transfers a single data item per address, making it resource-efficient and simple to design. Given the minimal number of arguments in our design, we chose to implement the AXI4-Lite interface as shown below by considering *GA_buck_IP* as the top level function with *RLCVi*[4], *ZetaWnTs*[3] and *Output*[3] as the input output arguments.

```
void GA_buck_IP(float RLCVi[4], float ZetaWnTs[3],
    float Output[3])
  {
    #pragma HLS INTERFACE s_axilite port=RLCVi
    #pragma HLS INTERFACE s_axilite port=ZetaWnTs
    #pragma HLS INTERFACE s_axilite port=Output
    #pragma HLS INTERFACE s_axilite port=return
```

Here, the argument *RLCVi*[4] is for inputting the circuit parameters, resistance, capacitance, inductance and input voltage, and the argument *ZetaWnTs*[3] is for inputting the desired damping ratio, natural frequency and settling time. The Gains, k_1, k_2 and K_e, are the elements in the output argument *Output*[3].

After C-synthesis, the estimated resource utilization and latency of the design can be analyzed. If the design constraints are not met, adjustments to the clock frequency and optimization directives/pragmas can be made to improve the design. Optimization can be achieved using HLS directives/pragmas such as *array_partition*, *pipeline*, and *unroll* [37], [38], [40].

During C-synthesis, a larger array is typically mapped into Block RAM with a maximum of two ports by default. To enable parallel computations, memory elements need to be accessed concurrently. The *array_partition* pragma facilitates this by partitioning the array into multiple RAMs or registers, allowing parallel access to memory elements [40].

For loops, default synthesis generates logic for a single iteration and then sequentially reuses this logic for subsequent iterations. The *unroll* pragma can be used to partially or fully unroll the loop, creating multiple copies of the loop body in the RTL logic design. This enables concurrent execution of all or a specified number of loop iterations, reducing the overall latency [40]. However, this approach increases resource utilization, which may lead to higher area and power consumption.

Operations within a loop are executed sequentially by

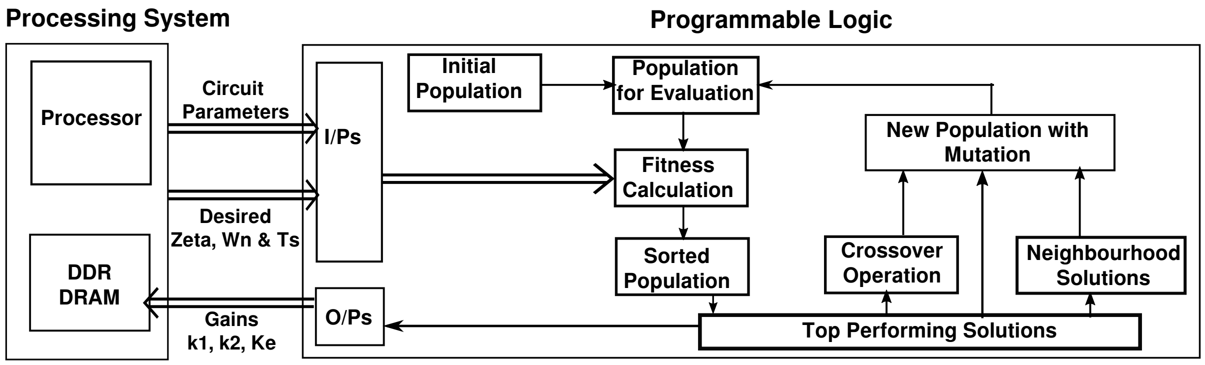


Figure 4. Block Representation of Computations in SoC FPGA

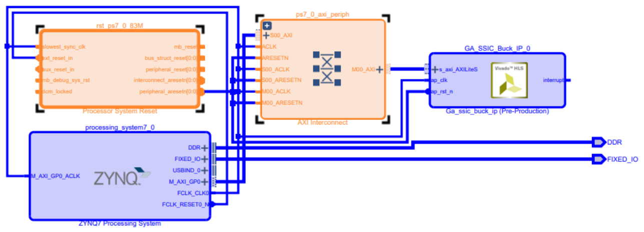


Figure 5. GA Computing System Block in Vivado HLx

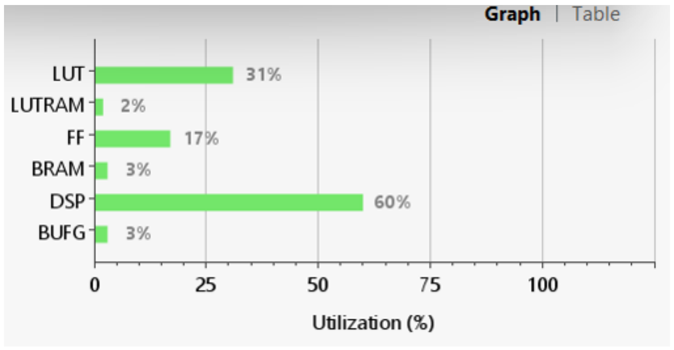


Figure 6. Resource Utilization Report from HLx

default. Using the *pipeline* pragma, these operations can be performed in parallel. This pragma also allows new iterations to begin processing before the previous ones have completed, enhancing the parallelism and efficiency of the design [40].

We start with an initial design derived from verified C/C++ code and incrementally improve it by integrating the mentioned optimization directives one by one. This iterative process continues until the design requirements are fully met. Once the design constraints are satisfied and optimized, the design can be packaged as customized intellectual property (IP) and exported to Vivado HLx. There, it's integrated with the processing system for implementation on an SoC-based FPGA.

The system block, created through Vivado HLx, showcases the integration of our customized IP with the Zynq processing system (PS) via the AXI fabric, as depicted in Fig. 5. Within this configuration, as illustrated in the figure, the customized IP, $GA\_SSIC\_Buck\_IP\_0$, is seamlessly integrated with the ZYNQ ($processing\_system7\_0$) using the AXI Interconnect ($ps7\_0\_axi\_periph$). The customized IP and the ZYNQ PS are highlighted for easy reference.

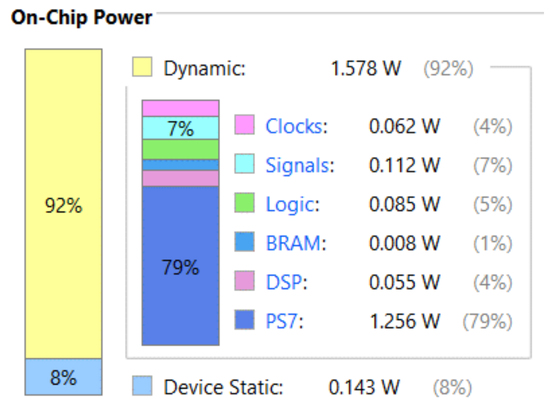
After the creation of an HDL wrapper for the design, it progresses through simulation, synthesis, implementation (including fitting and routing), and the generation of a bit-stream tailored for the PYNQ-Z2 FPGA board, facilitated either through graphical user interface or Tool Command Language (Tcl) codes.

Upon completion, the implemented design undergoes rigorous verification to ensure timing constraints are met and to assess resource utilization. The power analysis is also conducted. The verification reveals that all specified timing constraints are satisfied, and the resource utilization is well within control, as depicted in Fig. 6, which shows total resource utilization of 33% LUT, 17% FF, 3% BRAM, and 60% DSP.

The power analysis report, as shown in Fig. 7, indicates a total on-chip power of 1.721 W, comprising 1.578 W dynamic and 0.143 W static utilization. Within the dynamic utilization, the PS7 contributes 1.256 W, with the remaining attributed to DSP, BRAM, Logic, clock, etc.

The bit-stream file generated is exported to the Xilinx SDK, where an application program is developed in C/C++ to initiate processes and facilitate communication between the PS and the PL of the SoC FPGA.

For the GA based controller gain calculation, the parameters of the buck converter circuit, along with the desired damping ratio, natural frequency, and settling time, are transmitted to the PL using memory-mapped instructions. The instruction $Xil\_Out32(u32Addr, u32Value)$ is utilized to write a specified 32-bit value into a designated address. Similarly, the $Xil\_In32(u32Addr)$ instruction is employed to read 32-bit output gains from the specified address.


On-Chip Power

🟨	Dynamic:	1.578 W	(92%)
🟪	Clocks:	0.062 W	(4%)
🟦	Signals:	0.112 W	(7%)
🟩	Logic:	0.085 W	(5%)
🟦	BRAM:	0.008 W	(1%)
🟪	DSP:	0.055 W	(4%)
🟦	PS7:	1.256 W	(79%)
🟦	Device Static:	0.143 W	(8%)

Figure 7. Power Analysis Report from HLx

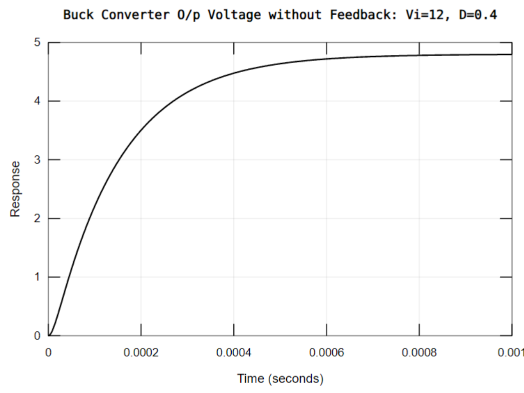


Figure 8. Output Voltage of the Buck Converter where V_i=12 and duty cycle=0.4 without any Feedback

*u*32*Addr* incorporates the base address, and the offset address which varies depending on the specific memory requirement. Integer pointers are necessary for the input-output communication with floating-point values.

Below are excerpts from our application program code demonstrating the writing to and reading from the implemented hardware along with the *start* and *ap_done* control signals.

```
Xil_Out32(BaseAddr+0x10, *((int *)&RLCVi[0]));
Xil_Out32(BaseAddr+0x14, *((int *)&RLCVi[1]));
Xil_Out32(BaseAddr+0x18, *((int *)&RLCVi[2]));
Xil_Out32(BaseAddr+0x1C, *((int *)&RLCVi[3]));
Xil_Out32(BaseAddr+0x20, *((int *)&ZetaTsWn[0]));
Xil_Out32(BaseAddr+0x24, *((int *)&ZetaTsWn[1]));
Xil_Out32(BaseAddr+0x28, *((int *)&ZetaTsWn[2]));
Xil_Out32(BaseAddr+0x00, 1);
while(0==(2 & Xil_In32(BaseAddr+0x00)));
OutputGains[0]=Xil_In32(BaseAddr+0x30);
OutputGains[1]=Xil_In32(BaseAddr+0x34);
OutputGains[2]=Xil_In32(BaseAddr+0x38);
```

Here, *BaseAddr* denotes the base address of the IP ports generated in our design. The offset addresses for each port to access the input are $0x10$, $0x14$, up to $0x28$, with the offset address $0x00$ designated for the control signal. The base address of the peripherals are defined at the file *xparameters.h* in the BSP folder. The $offset$ address of the ports and the details about the control signal are available at *ip_hw.h* file, in the same folder. These files are generated during RTL creation in Vivado HLS and transferred to SDK upon exporting the design from Vivado HLx.

The control signal 1 indicates the *start* signal, which needs to be provided to initiate computation in the PL after transmitting the input. Polling is then performed to wait for the control signal to change to *ap_done*. Once the *ap_done* signal is received, the output values can be read from the respective offset addresses using memory-mapped instructions, as shown in the code snippet above.

Building the files in the project directory generates an executable file (elf). This file, along with the bit-stream file from Vivado HLx, can be loaded onto the FPGA board using the Xilinx Software Command-Line Tool (XSCT). XSCT is an interactive command-line tool based on the Tcl that interfaces with Xilinx SDK. The output gains are stored in the PS's DDR DRAM for further analysis.

5. Results and Discussions

The buck converter depicted in Fig. 1 is modeled using state space, as described by equations 9 and 10, coded and simulated in MATLAB with a duty cycle of 0.4 as the control input and the voltage across the resistor R as the output. The circuit parameters for the implementation are as follows: resistance $R = 1\Omega$, inductance $L = 155\mu$H, capacitance $C = 10\mu$F, input voltage $V_i = 12$ volt.

The output voltage obtained with the open loop system is shown in Fig. 8, which clearly demonstrates that it settles to the desired value within 0.8 milliseconds. If the transient behaviour need to be controlled, such as reducing the settling time, the state variable feedback method described in section 2 can be employed.

State variable feedback gains, k_1 and k_2 are determined by comparing the desired characteristics equation with the characteristic equation of the system (Buck converter with state variable feedback). The corresponding output voltage is shown in Fig. 9, as dotted line. The design requirements are specified as 10% overshoot and 0.2 millisec settling time. The calculated gains are k_1=-0.7750 and k_2=0.8395. As seen in the figure, the transient performance objectives (percentage overshoot and settling time) are met by applying these state feedback gains. However, the steady-state voltage is 2.65V instead of the desired 4.8V. This steady-state error can be eliminated by incorporating an integral controller, as detailed in section 2.

By adding the integral controller, as illustrated in Fig. 3, the size of the system matrix increases by one, making the determination of the optimum controller gains more complex. This complexity necessitates the use of a heuristic approach. In our method, we use GA, which begins with an

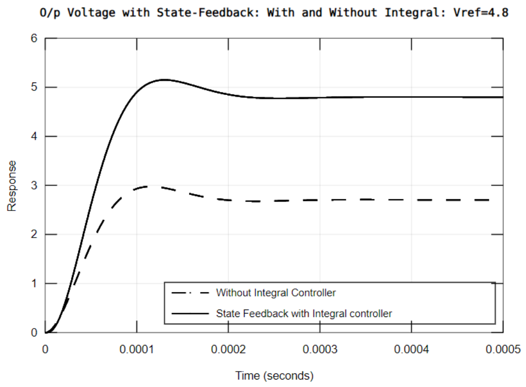


Figure 9. Output Voltage of the Buck Converter with and without integral controller: V_i=12, Vref=4.8

initial population and iterates until it converges to the best-performing solution based on a customized fitness function.

GA is coded in 'C', and starts with an initial population of 12 solutions, as described in section 3. The algorithm executes 20 iterations to determine the best performing solution using the fitness function defined in equation 15. The inputs to the functions are ζ, ω_n, T_s and circuit parameters while the outputs are the controller gains.

However, it is observed that the algorithm is parallelizable and the overall execution latency can be reduced by utilizing its parallel behaviour. Consequently, a custom design IP, $GA\_SSIC\_Buck\_IP$, is implemented on a PYNQ-Z2 SoC FPGA using Vivado tools to tune the state space integral controller gains using GA, as discussed in section 4. The IP design uses a 100 MHz clock signal, resulting in a 10 nsec clock period. In Vivado HLS, the observed latency for the entire GA computation is 8546 clock cycles, equivalent to 85.46 microseconds.

During on-chip execution, the observed latency is 88.44 microseconds for the entire gain computation using the GA on the PL, including the time required for PL-PS communication. This shows that communication latency was reduced to 3 microseconds by storing the initial population in the BRAM.

To analyze the achieved acceleration, the same gain calculation using GA was performed on the 650 MHz ARM processor available on the SoC, resulting in a latency of 463.46 microseconds. This demonstrates an acceleration factor of 5.3 when the entire computation is performed on the PL with a 100 MHz clock, compared to the ARM processor with a 650 MHz clock. These observations are detailed in Table I. The computation in the PL includes both the latency for computation within the PL and the latency for communication (input-output data transfer) between the PL and PS. The last column shows the acceleration achieved by using the PL for the computation.

Further analyzed the computational latency with a larger population and also considering more number of iterations for the convergence. The frequency of design, latency and acceleration corresponds to these cases are also included in Table I. The frequency of customized IP has to be reduced to 83 MHz for the case with 40 iterations and to 70 MHz for the design with 24 solutions in the population. This is mainly due to the resource constraints with PYNQ SoC FPGA. However, the acceleration is found to be consistent even with a lower clock frequency. This shows that the FPGA computational strategy for GA is effective to achieve considerably lower latency and can be extended for any GA application for its real time implementation.

TABLE I. Latency Report in HLS and also During Execution

Design	Latency with HLS (μs)	Latency for GA computation in μs		
		Comp. in PL with freq. in MHz	Comp. in PS with 650 MHz	Accel. with PL
Soln=12, Iter=20	85.46	88.44μs, 100MHz	463.46μs	5.3
Soln=12, Iter=40	173.6	176.9μs, 83MHz	915.5μs	5.2
Soln=24, Iter=20	418.3	421.1μs, 70MHz	2105μs	5.0

The resource utilization and power analysis report using Vivado HLx is summarized in Table II. With the use of the $array\_partition$ pragma for arrays like population and top-performing solution (to enhance parallel accessibility), BRAM utilization is minimized to 3%. Due to the numerous multiplication operations in the fitness calculation, DSP utilization reaches 60%. LUT utilization stands at 33%, primarily due to the comparison operations involved in the sorting process.

TABLE II. Resource Utilization and Power Report Using Vivado HLx

Design	BRAM %	DSP %	FF %	LUT %	On-chip Power (W)
Soln=12, Iter=20	3	60	17	33	1.721
Soln=12, Iter=40	3.2	63.2	14.4	30.2	1.733
Soln=24, Iter=20	4	25	7	16	1.533

The calculated values for various desired transient conditions, such as percentage overshoot and settling time, are presented in Table III. The percentage overshoot ranges from 8% to 14% and the settling time varies from 0.2 milliseconds to 0.5 milliseconds. For each set of desired ζ, ω_n, the GA calculated gains, the respective calculated ζ, ω_n along with the corresponding percentage overshoot (%OS) and settling time (T_s), are also shown in the table. In all the cases, the steady state error is found out to be zero because of the inclusion of the integral term. The results clearly demonstrate that the GA-calculated gains can achieve the desired percentage overshoot and settling time, with zero steady state error, and thus an effective method for tuning.

TABLE III. Desired and Actual Transient Parameters for GA based State-space Controller Tuning for Buck Converter

Desired Values		Values Obtained from FPGA		
%OS, T_s	ζ, ω_n	Gains	ζ, ω_n	%OS, T_s
%OS=10, T_s=$2e^{-4}$	ζ=0.59, ω_n=$3.4e^4$	k_1=2, k_2=-0.8, K_e=29250	ζ=0.64, ω_n=$3.3e^4$	%OS=7.2 T_s=$1.9e^{-4}$
%OS=10 T_s=$3e^{-4}$	ζ=0.59 ω_n=$2.3e^4$	k_1=1.081 k_2=-0.42 K_e=13500	ζ=0.61 ω_n=$2.6e^4$	%OS=8.8 T_s=$2.5e^{-4}$
%OS=10 T_s=$4e^{-4}$	ζ=0.59 ω_n=$1.7e^4$	k_1=0.2725 k_2=0.097 K_e=4976.5	ζ=0.64 ω_n=$1.9e^4$	%OS=7.3 T_s=$3.2e^{-4}$
%OS=10 T_s=$5e^{-4}$	ζ=0.59 ω_n=$1.3e^4$	k_1=0.0635 k_2=0.174 K_e=1906.2	ζ=0.59 ω_n=$1.3e^4$	%OS=9.7 T_s=$4.9e^{-4}$
%OS=8 T_s=$5e^{-4}$	ζ=0.62 ω_n=$1.3e^4$	k_1=0.08 k_2=0.191 K_e=1687.5	ζ=0.63 ω_n=$1.3e^4$	%OS=7.7 T_s=$4.9e^{-4}$
%OS=12 T_s=$5e^{-4}$	ζ=0.56 ω_n=$1.4e^4$	k_1=0.045 k_2=0.156 K_e=2148.4	ζ=0.56 ω_n=$1.4e^4$	OS=11.9% T_s=$4.9e^{-4}$
%OS=12 T_s=$4e^{-4}$	ζ=0.56 ω_n=$1.8e^4$	k_1=0.37 k_2=-0.08 K_e=5000	ζ=0.58 ω_n=$1.9e^4$	OS=10.2% T_s=$3.6e^{-4}$
%OS=8 T_s=$4e^{-4}$	ζ=0.63 ω_n=$1.6e^4$	k_1=0.232 k_2=-0.01 K_e=3312.5	ζ=0.63 ω_n=$1.6e^4$	OS=7.7% T_s=$3.9e^{-4}$
%OS=8 T_s=$3e^{-4}$	ζ=0.63 ω_n=$2.1e^4$	k_1=1.362 k_2=-0.45 K_e=17000	ζ=0.67 ω_n=$2.7e^4$	OS=5.9% T_s=$2.2e^{-4}$
%OS=14 T_s=$3e^{-4}$	ζ=0.53 ω_n=$2.5e^4$	k_1=1.4 k_2=-0.61 K_e=20000	ζ=0.57 ω_n=$2.9e^4$	OS=11% T_s=$2.3e^{-4}$

The output voltage from the buck converter, by adding an integral controller with the gains, k_1=2, k_2=-0.8, K_e=29250, obtained from the GA computation using FPGA (data correspond to %OS=10, $T_s = 2e^{-4}$, the first row data in Table III), is also shown in Fig. 9. It is clear that by adding the integral controller along with state feedback, the output voltage of the buck converter becomes the reference voltage 4.8V.

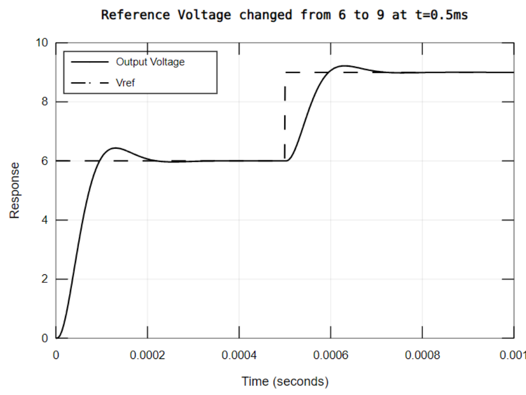


Figure 10. Output Voltage of the Buck Converter with integral controller: Vref changed from 6V to 9V at 0.5 msec

Also, analysed the response by changing the reference value from 6V to 9V at 0.5 msec and the corresponding plot is shown in Fig. 10. It clearly indicates that the output voltage could track the reference voltage with the desired transient parameters (%OS=10,T_s = $2e^{-4}$) at each input step variation. Reference input is also shown in the figure as dotted line.

These analysis demonstrates that using GA, we can determine state space integral controller gains that achieve the desired transient parameters with zero steady-state error, specifically for the voltage control of a buck converter. Moreover, this GA method can be extended to any application requiring heuristic parameter optimization, not just DC-DC power electronic converters. Additionally, implementing the GA on an FPGA significantly accelerates computations, making it highly suitable for real-time applications.

6. Conclusion

In this paper, we control the transient and steady-state behavior of buck converters using a state-space approach, which is particularly effective for managing nonlinear dynamics. This topic is crucial due to the widespread use of buck converters in power electronics, where precise control is essential for optimal performance and efficiency. We propose using a Genetic Algorithm (GA) to tune the gain parameters of a state feedback controller with integral action. The inherent parallelism in GA is effectively leveraged using an FPGA, specifically implemented on a PYNQ-Z2 SoC FPGA. We used an initial population of 12 and ran the algorithm for 20 iterations. The output gains were verified for various desired overshoot and settling times while maintaining zero steady-state error. Our results show that the algorithm optimally tunes the gain parameters to meet the desired specifications. Additionally, our 100 MHz

GA implementation on FPGA achieved a 5.3 times speed-up compared to a 650 MHz ARM processor. For larger populations or more iterations required for convergence, using an FPGA with more resources, GA implementation could achieve even greater speed-up compared to performing the entire computation on the processor.

References

[1] N. H. Baharudin, T. M. N. T. Mansur, F. A. Hamid, R. Ali, and M. I. Misrun, "Performance analysis of dc-dc buck converter for renewable energy application," *Journal of Physics: Conference Series*, vol. 1019, no. 1, p. 012020, jun 2018. [Online]. Available: https://dx.doi.org/10.1088/1742-6596/1019/1/012020

[2] D. M. Bellur and M. K. Kazimierczuk, "Dc-dc converters for electric vehicle applications," in *2007 Electrical Insulation Conference and Electrical Manufacturing Expo*, 2007, pp. 286–293.

[3] S. Masri, N. Mohamad, and M. H. M. Hariri, "Design and development of dc-dc buck converter for photovoltaic application," in *2012 International Conference on Power Engineering and Renewable Energy (ICPERE)*, 2012, pp. 1–5.

[4] S. Kumar, S. Subbaraj, and S. Kingsly, "Design of buck dc-dc converter portable charging application for electric vehicles," 11 2023.

[5] S. A. Tali, F. Ahmad, and I. H. Wani, "Design and analysis of feedback control for dc-dc buck converter," in *New Frontiers in Communication and Intelligent Systems*, R. Srivastava and A. K. S. Pundir, Eds. SCRS, India, 2022, pp. 319–328.

[6] G. Abbas, C. Condemine, and N. Abouchi, "Digitally-controlled high-frequency dc-dc buck converter," in *6th Conference on Ph.D. Research in Microelectronics Electronics*, 2010, pp. 1–4.

[7] H. Xue, B. Jinying, W. Chunsheng, and X. Honghua, "Design and implementation of a pv dc/dc converter with high efficiency at low output power," 10 2010.

[8] V. C. Valchev, D. J. Mareva, and D. D. Yudov, "Analysis and improvements of a buck converter based led driver," in *2017 XXVI International Scientific Conference Electronics (ET)*, 2017, pp. 1–4.

[9] N. Bajoria, P. Sahu, R. Nema, and S. Nema, "Overview of different control schemes used for controlling of dc-dc converters," in *2016 International Conference on Electrical Power and Energy Systems (ICEPES)*. IEEE, 2016, pp. 75–82.

[10] N. D. Bhat, D. B. Kanse, S. D. Patil, and S. D. Pawar, "Dc/dc buck converter using fuzzy logic controller," in *2020 5th International Conference on Communication and Electronics Systems (ICCES)*, 2020, pp. 182–187.

[11] W. Dong, S. Li, X. Fu, Z. Li, M. Fairbank, and Y. Gao, "Control of a buck dc/dc converter using approximate dynamic programming and artificial neural networks," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. PP, pp. 1–9, 01 2021.

[12] S. Saadatmand, P. Shamsi, and M. Ferdowsi, "The voltage regulation of a buck converter using a neural network predictive controller," 02 2020, pp. 1–6.

[13] A. Samosir, T. Sutikno, and L. Mardiyah, "Simple formula for designing the pid controller of a dc-dc buck converter," *International Journal of Power Electronics and Drive Systems (IJPEDS)*, vol. 14, p. 327, 03 2023.

[14] N. F. Nanyan, M. A. Ahmad, and B. Hekimoğlu, "Optimal pid controller for the dc-dc buck converter using the improved sine cosine algorithm," *Results in Control and Optimization*, vol. 14, p. 100352, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666720723001546

[15] V.-D. Andries, L. Goras, E. David, A. Buzo, and G. Pelz, "On the pole-placement technique for the design of a dc-dc buck converter discrete pid control," in *2020 23rd International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, 2020, pp. 1–4.

[16] H. Al-Baidhani, A. Sahib, and M. Kazimierczuk, "State feedback with integral control circuit design of dc-dc buck-boost converter," *Mathematics*, vol. 11, p. 2139, 05 2023.

[17] G. F. Franklin, J. D. Powell, and M. L. Workman, *Digital Control of Dynamic Systems*. Addison-Wesley, 1998.

[18] G. Keller, D. Lascu, and J. Myrzik, "State-space control structures for buck converters with/without input filter," in *2005 European Conference on Power Electronics and Applications*, 2005, pp. 10 pp.–P.10.

[19] G. Gkizas, C. Yfoulis, C. Amanatidis, F. Stergiopoulos, D. Giaouris, C. Ziogou, S. Voutetakis, and S. Papadopoulou, "Digital state-feedback control of an interleaved dc–dc boost converter with bifurcation analysis," *Control Engineering Practice*, vol. 73, pp. 100–111, 2018.

[20] D. Daftary and C. H. Raval, "Controller design for buck-boost converter using state-space analysis," in *Renewable Energy and Climate Change*, D. Deb, A. Dixit, and L. Chandra, Eds. Singapore: Springer Singapore, 2020, pp. 129–140.

[21] A. M. Dissanayake and N. C. Ekneligoda, "Discrete time adaptive state feedback control of dc-dc power electronic converters," in *2019 IEEE Applied Power Electronics Conference and Exposition (APEC)*, 2019, pp. 2962–2967.

[22] M. Mazlan, N. Haqkimi, C. Charin, N. Fairuz, N. Izni, and M. Annuar, "State feedback controller using pole placement method for linear buck converter to improve overshoot and settling time," *Applied Mechanics and Materials*, vol. 793, pp. 211–215, September 2015. [Online]. Available: https://doi.org/10.4028/www.scientific.net/amm.793.211

[23] A. Oliva, H. Chiacchiarini, and G. Bortolotto, "Development of a state feedback controller for the synchronous buck converter," *Latin American applied research*, vol. 35, 04 2005.

[24] D. Shuai, Y. Xie, and X. Wang, "Optimal control of buck converter by state feedback linearization," vol. 28, 07 2008, pp. 2265 – 2270.

[25] Y. Fetene, E. Ayenew, and S. Feleke, "Full state observer-based pole placement controller for pulse width modulation switched mode voltage-controlled buck converter," *Heliyon*, vol. 10, no. 9, p. e30662, 2024. [Online]. Available: https://doi.org/10.1016/j.heliyon.2024.e30662

[26] T. Tarczewski, J. Niewiara, M. Skiwski, and L. M. Grzesiak, "Gain-scheduled constrained state feedback control of dc–dc buck power converter," *IET Power Electronics*, vol. 11, no. 4, pp.

735–743, 2018. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-pel.2017.0370

[27] T. Djamel, T. Amieur, B. Mohcene, S. Kahla, and S. Moussa, *State Feedback Control of DC-DC Converter Using LQR Integral Controller and Kalman Filter Observer*, 06 2021, pp. 1699–1709.

[28] R. Seeber and M. Tranninger, "Integral state-feedback control of linear time-varying systems: A performance preserving approach," *Automatica*, vol. 136, p. 110000, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0005109821005264

[29] Z. Ortatepe, "Genetic algorithm based pid tuning software design and implementation for a dc motor control system," *Gazi University Journal of Science Part A: Engineering and Innovation*, vol. 10, pp. 286–300, 09 2023.

[30] M. Sergio, J. Carlos, M. Irma, and J. Ramón, "Pi controller tuning using genetic algorithm," in *2021 International Conference on Mechatronics, Electronics and Automotive Engineering (ICMEAE)*, 2021, pp. 15–19.

[31] M. B. P and S. Bhat, "Steady state and transient state analysis of buck-boost converter with genetic algorithm optimized pid controller," in *2022 International Conference on Smart Generation Computing, Communication and Networking (SMART GENCON)*, 2022, pp. 1–6.

[32] D. C. Meena and A. Devanshu, "Genetic algorithm tuned pid controller for process control," in *2017 International Conference on Inventive Systems and Control (ICISC)*, 2017, pp. 1–6.

[33] W. Tang and L. Yip, "Hardware implementation of genetic algorithms using fpga," in *The 2004 47th Midwest Symposium on Circuits and Systems, 2004. MWSCAS '04.*, vol. 1, 2004, pp. I–549.

[34] M. S. Ben Ameur, A. Sakly, and A. Mtibaa, "Implementation of real coded genetic algorithms using fpga technology," in *10th International Multi-Conferences on Systems, Signals Devices 2013 (SSD13)*, 2013, pp. 1–6.

[35] M. Torquato and M. Fernandes, "High-performance parallel implementation of genetic algorithm on fpga," *Circuits, Systems, and Signal Processing*, vol. 38, pp. 4014–4039, 2019. [Online]. Available: https://doi.org/10.1007/s00034-019-01037-w

[36] M. Vavouras, K. Papadimitriou, and I. Papaefstathiou, "High-speed fpga-based implementations of a genetic algorithm," in *2009 International Symposium on Systems, Architectures, Modeling, and Simulation*, 2009, pp. 9–16.

[37] M. Namboothiripad, M. Datar, M. Chandorkar, and S. Patkar, "Fpga accelerator for real-time emulation of power electronic systems using multiport decomposition," 12 2019, pp. 1–6.

[38] K. Namboothiripad and Mini, "Analyzing the impact of discretization techniques on real time simulation of dc servo motor using fpga," *International Journal of Computing and Digital Systems*, vol. 15, no. 1, p. 103, March 15 2024.

[39] Xilinx, *PYNQ-Z2 Reference Manual v1.0*, May 17 2018, https://pynq.readthedocs.io/en/latest/getting_started.html.

[40] AMD, *Introduction to FPGA Design with Vivado High-Level Synthesis*, v1.0 ed., July 2 2013, available: https://docs.amd.com/v/u/en-US/ug998-vivado-intro-fpga-design-hls.

Mini K. Namboothiripad received the B.Tech. degree in electrical and electronics engineering from Govt. Engineering College Thrissur, University of Calicut, Kerala, India in 1995, and the M.Tech. degree in 2011, and the Ph.D. degree in 2021, both in Electrical Engineering from the Indian Institute of Technology Bombay, Mumbai, India. She has been working as an Assistant Professor with the Department of Electrical Engineering, Fr. C. Rodrigues Institute of Technology, Navi-Mumbai, India, since 2001. Her research interests include FPGA-based Fast Computing, Real-Time Simulation, Mathematical Modelling, and Control of Electrical Systems.