Automated Hardware Design Methodology for Digital Filters with High-Level Synthesis

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 28 July 2025, accepted 17 August 2025, date of publication 25 August 2025, date of current version 2 September 2025.

Digital Object Identifier 10.1109/ACCESS.2025.3602232

METHODS

Automated Hardware Design Methodology for Digital Filters With High-Level Synthesis

SHEHRYAR AKBAR[1], LUCIANO LAVAGNO[1], (Senior Member, IEEE), MIHAI T. LAZARESCU[1], (Senior Member, IEEE), AND DAVIDE MARIZ[2]

[1]Department of Electronics and Telecommunications, Politecnico di Torino, 10129 Turin, Italy
[2]Infineon Technologies, 9500 Villach, Austria

Corresponding author: Shehryar Akbar (Shehryar.akbar@studenti.polito.it)

This work was supported in part by Infineon Technologies AG Austria, and in part by Siemens AG.

ABSTRACT The increasing complexity of digital hardware systems and the demand for faster time-to-market for the semiconductor industry require a rapid and flexible design strategy at an increased abstraction level. To this end, we propose a fully automated hardware design methodology for digital signal processing (DSP) applications such as digital filters, using high-level synthesis (HLS) integrated with MATLAB. This approach enables a one-click workflow from concept design to a synthesized and functionally verified netlist, with the requirement of minimal hardware expertise. After comparison with industry standard register transfer level (RTL) designs of multiple filter architectures, the area of HLS-generated implementations exhibited a variance ranging from a 54.4 % reduction to a 35.8 % overhead. This indicates a notable quality of results (QoR), especially when weighed against the significant gains in design time reduction and productivity. Beyond digital filters, the work demonstrates that modern HLS tools, when paired with automation and parameterized code, can deliver rapid industrial grade ASIC results, bridging the gap between algorithm development and hardware deployment in digital signal processing and beyond.

INDEX TERMS Application specific integrated circuits, digital filters, digital signal processing, electronic design automation, high-level synthesis.

I. INTRODUCTION

Digital electronic systems are the backbone of modern technology, which enable the functionality of almost all the devices that we rely on daily. Their significance extends across a wide array of contemporary application domains. However, as the technological landscape continues to evolve rapidly, the design of such systems presents a multitude of challenges that engineers must navigate to ensure reliability, efficiency, and innovation.

The increasing complexities of integrated circuits (ICs) makes it much more difficult to design them in a cost-effective way, not just in terms of finances, but in terms of time and quality of results (QoR) also. Numerous possibilities exist for designing a specific system; nevertheless, choosing the optimal one requires considerable time and profound hardware design expertise. In this evolving field, the race to develop efficient, scalable and better hardware puts pressure on companies to come up with novel ways to speed up the process of digital design.

Considering the current technological landscape and the need for companies to meet deadlines for product launches as well as rapid prototyping, the traditional method of using register transfer level (RTL) code to describe hardware could even become an outdated method in the future [1]. Moreover, there already exists a strong motivation to increase the design productivity of today's VLSI circuits because of their complexities [2], [3]. For this purpose, design abstraction is a highly successful technique for managing complexity and enhancing design efficiency.

These factors have compelled design methodologies and tools to increase the degree of abstraction and introduce automation in the design flow in order to accelerate their time-to-market deadlines. In addition to rapid product launches,

The associate editor coordinating the review of this manuscript and approving it for publication was Ludovico Minati.

micro-architectural exploration is also crucial as it allows developers to investigate different alternatives and then choose the best outcome. At the RTL level, this is a tedious and time-consuming task that demands a high level of competence in and of itself. Also, once a design has been finalized, it is quite difficult to revert back and modify the code to implement a different strategy. There are two possible reasons why this modification can be necessary; either there is a possibility to design a better system with greater QoR, or there can be a requirement from the user to make changes in the system.

This necessitates the use of high-level synthesis (HLS) tools, whose versatility, simplicity in debugging and ease of optimization options are a perfect match for fulfilling the demands of intricate hardware design. HLS is a technology that assists with the transformation of a behavioral description of hardware into an RTL model [4]. It performs a translation of the high-level code into RTL code, starting from the description in a high-level programming language (C++, C, SystemC and others). It is highly useful for raising the abstraction level of hardware design which results in enhanced productivity for both design and verification [5], [6]. A set of constraints and goals must be specified together with the high-level description as they affect the behavior of the final hardware and its implementation. The quality of these final results has been showing progress over the course of time with the newest generation of HLS tools, which suggests an ongoing positive trend in the capabilities of HLS technologies [1], [7].

A recent survey by Lahti and Hamalainen [8] shows that HLS enables a significant reduction in the development effort required for hardware design compared to traditional RTL methods. Here the development effort is quantified by development time and lines of code (LoC). The data indicates that employing HLS decreases the effort by a factor of 2x-4x. This survey covering a total of 93 applications implemented between 2017 and 2024 revealed that, on average, the HLS input code size was 47 % of the corresponding RTL code size. In 88 % of these applications, HLS designs had fewer LoC than their RTL counterparts. Likewise, in 100 % of the cases, the development time with HLS tools was lower than that of manual RTL coding, with the average development time being only about a third of the RTL flow.

One of the earliest mentions of HLS to accelerate time to market is mentioned by McFarland et al. [9]. The paper indicates that a significant portion of the cost in the overall workflow of developing an IC is tied to the design phase, specifically the development effort involved in it. HLS can be used to accelerate the design process which directly contributes to achieving a shorter product development cycle and reduction in costs. This is also demonstrated from the approach adopted by Chips&Media to design their latest intellectual property (IP) for detecting objects in real time using HLS flow [10]. In parallel, they also developed the same IP using the conventional hand-coded Verilog, so that meaningful comparisons could be derived at the end. At the conclusion of the project, it was discovered that the development time was exactly halved in favor of HLS flow, while the performance and area were slightly better for the hand-coded RTL. Similarly, a 32-bit RISC-V microprocessor designed with SystemC and optimized with a commercial HLS tool was reported to contain approximately 2k LoC, while a comparable manually optimized RTL processor (ZERO-RISCY) required more than 6k LoC [3]. In another example, a single graduate student implemented an entire Convolutional Neural Network (CNN) accelerator in less than 6 months using HLS, ahead of the release of Google's TPU, developed by a much larger team. Additionally, this accelerator achieved a 5x speedup and 25x energy reduction compared to a 16-thread CPU implementation [7]. The market consolidation of HLS tools and their increased usage in the design of high-quality hardware has rendered HLS a viable option for fast prototyping and for designs with short time to market [1].

Furthermore, it is possible to leverage the power of HLS tools even further by *automating* the entire process of developing new IP instances starting from parameterized designs that are reusable, flexible, and meet target requirements while achieving the best possible latency and resource sharing. This automation is achieved using scripts and pre-defined source codes which are generic in nature and fully parameterized, enabling the generation of customized hardware architectures without manual intervention.

This paper shows the practical implementation of this novel approach by extending the capabilities of Siemens HLS tool *Catapult* by integrating it seamlessly with MATLAB to deliver a *one-click* hardware design workflow, which facilitates automatic generation of optimized digital filter IPs with minimal user input. Unlike conventional approaches that demand extensive hardware knowledge, this methodology enables software, algorithm and concept engineers to participate directly in hardware development. Our framework supports the creation of multiple filter types through parameterized source codes and reusable templates. It additionally embeds functional verification in the workflow, ensuring that the synthesized designs conform precisely with the original specifications. This allows the user to design seemingly complex IP blocks in an exceedingly short amount of time and with outstanding results.

To validate its practicality and industrial relevance, the proposed methodology is benchmarked against manually designed, industry standard RTL IPs, demonstrating comparable QoR in terms of area and power consumption, while achieving significant reductions in development time. These contributions mark a significant step toward simplifying hardware design in the semiconductor industry.

The paper starts with a discussion of related work in Section II. In Section III, the methodology and the workflow of this work is explained. This is then followed by implementation of this process on four different digital filters in Section IV. Comparisons with identical filters (which are designed manually using a hardware description language such as VHDL or SystemVerilog) are made in Section V

along with the results and finally, the conclusion is drawn in Section VI.

II. RELATED WORK

The work of Ogunfunmi and Desai [11] is closely related to the scope of this work in which one of the earliest versions of Synopsys HLS tool was used for the design and VLSI implementation of a fast finite impulse response (FIR) filter. The comparison of the outcomes with conventional approaches revealed positive findings. Although the tool's results were impressive for their time, they are not up to the level of what can be achieved today with more advanced technologies.

One of the more recent studies on the topic is the analysis of FIR filter implementation carried out by Hanbo et al. in 2015 [12]. This work implements a low pass FIR filter of order 20, using three different tools: Vivado HLS, LabVIEW FPGA, and DSP Builder. Even though Catapult HLS, the software we have used in our study, is not included in the comparison, nevertheless, we can make significant inferences from this work. It provides an elaborate comparison against the conventional approach of IP core design, concentrating on latency, the maximum frequency of devices operating, synthesis time, resource occupation, and optimization possibilities. And the results make it evident that one of the most unique characteristics of HLS tools is their capacity to reduce synthesis time. Also, extended optimization choices are available, and the highest frequency and latency performances are also improved.

Filip et al. [13] developed an automated methodology that is used for the design of FIR filters. It entails an open-source tool that uses automation to create high quality FIR filters where the input from the user consists of high-level frequency-domain (FD) specifications (passband frequency ω_p, stopband frequency ω_s, and bounds δ_p and δ_s on the passband ripple and stopband attenuation) and input/output formats. The proposed tool attempts to perform optimization of the design parameters at minimum cost and ensures a refined hardware architecture of the final filter implementation. The relevant steps include evaluation of minimum filter order N (to match FD parameters), quantization of N to the internal format (to match time-domain parameters) and continuous error analysis until the refined product is obtained. The assessment of this tool is conducted by comparing it with MATLAB's *filterbuilder*, using three distinct filters: high-pass, low-pass, and band-stop. The results are quite promising, with the resource usage and performance being comparable or better to MATLAB results, as well as obtaining a smaller filter order N which is able to satisfy the FD specifications. Additionally, the proposed tool is much faster due to automation, with a typical design time of a filter amounting to less than a minute.

Another work used for the hardware synthesis of digital filters is discussed in [14] in which an automatic methodology is proposed to design filters which meet the prescribed output accuracy and FD specifications. In addition to resource optimization in terms of internal word length, a system for generating the corresponding RTL codes in VHDL is also developed. These codes are generated using a configuration file and a predefined library of commonly used modules e.g. FIR filters, IIR filters, up-down sampling modules etc. For comparison purposes, a digital intermediate frequency (IF) receiver for software radios is considered with specifications given in [15]. As a result, the numbers of logic elements and LC registers required are evaluated and it is found that they are approximately nine times more than those required for this proposed design method, which illustrates the effectiveness of this methodology.

In addition to digital signal processing (DSP), HLS has numerous other uses [16]. Because it can be described at the algorithmic level [17], it is widely used in algorithm validation, design space exploration [2], migration, hardware acceleration [18], and numerous other domains. HLS has also proven successful across a diverse array of complex applications, including deep learning, video transcoding, graph processing, genome sequencing, wireless communication, and database analytics [3], [5], [7].

For example, a real-time HEVC encoder for high-definition video, developed by NGCodec (later acquired by Xilinx), was entirely implemented using HLS, showcasing its capability for intricate commercial products [7]. It was built using approximately 100k lines of C code, capable of supporting 4K30 resolution. Although the design of the latest generation of video codecs is complicated and lengthy, the development process for this complex system benefited from a hierarchical design flow, which allowed for isolated changes to individual components, thus accelerating tool execution, enabling faster design iteration and optimization, and facilitating independent verification of each part.

However, the performance of HLS tools varies depending on the application and it may not always produce the intended improvement in outcomes. This is demonstrated in [19] where a sparse algorithm matrix multiplication application with dynamic loop bounds showed poor HLS performance because such dynamic structures are unsuitable for the automatic optimizations HLS tools perform. Moreover, HLS tools generally excel at synthesizing computationally intensive datapaths with limited control logic but struggle to produce high-quality RTL for control-dominated branching logic [3]. This suggests that applications with complex control flow might not see the intended performance improvements without significant manual intervention.

The work of Fibich et al. [20] uses four case studies (matrix multiplication, FIR filter, Atmel barcode reader and QR decoding library) to assess the use of two HLS tools (Vivado HLS and PandA Bambu) for offloading computation-intensive algorithms to a field-programmable gate array (FPGA) implementation. These case studies are evaluated both from the software as well as hardware aspects, and the overall conclusion is in favor of execution times of the software implementations as opposed to their hardware counterparts. The above cases demonstrate that while HLS

tools offer productivity benefits, their performance outcomes are highly application dependent.

Another limitation faced by HLS tools is their requirement of additional information to assist the compiler/synthesis suite, often in the form of pragmas [21]. Sohrabizadeh et al. [22] implemented a CNN with only 24 lines of simple HLS code that was 80x slower than a single thread on a processor core. However, it was able achieve a 7000x speedup after inserting 28 pragmas. Optimization strategies play a crucial role in the hardware implementations produced by HLS tools for identical algorithms [23]. In another example, an initial implementation of the computational science pipeline onboard the Advanced Particle-astrophysics Telescope (APT) took over 15 ms to process a single event. With the addition of several pragma directives and HLS-specific optimizations, a 3415 times speedup was achieved, enabling processing of over 200,000 gamma-ray events per second [21]. This illustrates that performance variability can be orders of magnitude and performance improvements are not automatic. Rather it demands a deep understanding and judicious use of HLS-specific techniques, involving manual effort beyond simply writing high-level code.

A. RESEARCH GAP

The persistent debate regarding the selection of HLS tools vs the employment of hand-written VHDL/Verilog code for digital hardware design, particularly in DSP blocks, remains a prominent topic among researchers and industrial engineers alike. The aforementioned literature indicates that a reasonable amount of work has been done to establish the efficacy of using HLS tools or various design methodologies to develop hardware with optimized results. Numerous authors have explored specific use cases or proposed methodologies showing HLS viability for hardware synthesis. However, the scope of each of these works is very limited and often focused on a very specialized implementation. Typically, these studies concentrate on individual DSP blocks such as FIR filters or matrix multipliers, and do not propose scalable, adaptable frameworks that can be generalized across diverse design problems. Moreover, even if there are numerous academic works targeting the effectiveness of such techniques, none of these efforts (to our knowledge) demonstrate experimental evidence of their utilization in the semiconductor industry. While academic results may suggest promising outcomes under controlled environments, there is an evident absence of validation against industrial IP designs or workflows. Existing literature lacks a comprehensive process that integrates the functionality of HLS tools with commercial IP requirements in a streamlined and quick way, as well as furnishing empirical evidence of their effectuality.

Therefore, the proposed approach is novel in four respects.

Firstly, this method merges automation and the functionality of a contemporary HLS tool in a single step for industrial design. Catapult HLS is already an advanced tool which offers accelerated design solutions for outstanding QoR through various optimization options. The integration of automation results in a more resilient and extremely fast approach that addresses various domains, including micro-architectural exploration, adaptability, reusability, and relieves the user from programming. The use of HLS in electronic system-level (ESL) design automation has been recognized as a productivity boost for the semiconductor industry [24], and our approach will address the demand for more efficient design methodologies capable of managing intricate hardware design with reduced time and effort.

Secondly, functional verification of the final filter design is provided in the process which is extremely useful for checking the results against the concept design. This ensures 100% compliance of the design with the intended results.

Thirdly, almost all the other approaches are targeted towards a specific DSP block or digital filter and they only cater to that particular IP in terms of optimization and enhancement of the outcome. There is a gap in existence about a multifaceted portfolio that incorporates various IP types within a singular workflow, and our methodology addresses this necessity.

Lastly, benchmark-based research that includes real-world testing is incorporated in our work by evaluating the results with industry standard IP designs. This type of comparison allows practical testing against actual products which are being used in the market. Table 1 lists the summary of related works along with their scope and demonstrates how our work proposes advancements over existing methodologies in this domain.

B. RESEARCH CONTRIBUTIONS

To address the existing research gap, our main contributions to the state-of-the-art include:

1) Provide a one-click, hardware design workflow which can be used to implement digital filter IPs of four different types of filters: FIR filter, polyphase decimation filter, cascaded integrator-comb (CIC) decimation filter and filter chain.
2) Propose a flow which can be used by software and algorithm engineers, and not just experienced hardware designers.
3) Accelerate hardware design process by integrating the capabilities of a modern HLS tool with MATLAB in a single unified and automatic flow.
4) Evaluate the effectiveness of this method by comparing it with industry standard RTL designs. The benchmarks used for the purpose of evaluation include area and static power consumption.

III. METHODOLOGY

In the proposed design methodology, we use Siemens HLS tool Catapult to develop pre-defined parametric architectures and fuse them with automation to yield a refined process which offers a novel approach to design DSP blocks,

TABLE 1. Summary of key related works vs proposed automated hardware design methodology in this paper.

Ref #	Scope and Contributions	Limitations	Key Differences and Advancements in this Work
[11]	FIR filter design using Synopsys HLS tool	Focused solely on FIR using outdated software and without automation	Uses Catapult HLS with multi-filter support and an automated design flow
[12]	FIR filter comparison using Vivado HLS, LabVIEW FPGA and DSP Builder	Limited to an order 20 FIR filter and without automation	Provides a diverse range of filter architectures and an automated design flow
[13]	Automated FIR filter design from high level FD specifications using open-source tool. Result comparison is carried out manually with MATLAB filter builder	Constrained to algorithm-to-hardware mapping of FIR filter only and without verification flow	Offers an automated workflow using HLS for multiple filter types with MATLAB integration and embedded verification of results
[14]	RTL generation of various filters from FD specifications, configuration files and library of modules	Lack of MATLAB-HLS linkage and no validation of results	Dynamically generates both hardware and golden reference values for verification
[20]	Evaluation of Vivado HLS and PandA Bambu using case studies (e.g. FIR filter)	Limited to FIR filter and without design reuse and automation	Extends Catapult HLS capabilities with generic architectures for multiple filter types that are automated and reusable
[21],[22]	Implementation of designs using minimal HLS code	Highlights the need for manual input of pragma statements for optimized results	Automates optimizations within a reusable source code framework, removing the need for advanced expertise from the designer
[*]	Application of HLS to various fields, highlighting HLS growth and its benefits in increasing design productivity	Lack of experimental evidence of utilization in the semiconductor industry, absence of automated workflows (concept design to netlist level), limited verification strategies and focused on FPGA-centric flows	Demonstrates real-world, benchmarked QoR metrics for multiple filter types by implementing end-to-end automated framework, supporting MATLAB integration, built-in functional verification, ASIC synthesis and design reusability

especially digital filters. The motivation behind this approach was to cater to two primary requirements: Rapid time-to-market demands of the electronics industry and the need to develop inherently complex hardware in a flexible manner.

The proposed methodology includes two main steps:

1) Developing a framework for the purpose of automated hardware design (digital filters).
2) Utilizing this framework to design new filter IPs and carry out meaningful comparative analysis.

To realize the first step of developing a framework for automated hardware design (digital filters), a four-tiered approach has been utilized. This framework is a stand-alone solution which can be used by designers to develop user-customized filter IPs in which the designer needs to only input the required specifications, without concern for the subsequent design process.

Although the general approach to design this framework remains the same for all filters, the specifics of each stage of the distinct filters vary at the lowest level. These individual specifics of the different filters are discussed later in Section IV.

In this section, we only discuss a brief overview of the procedure for the development of this automated hardware design workflow (step 1). A top-level outline of this process is depicted in Fig. 1 which shows a structural framework that allows designers to develop digital filter IPs of their choice in an extremely short amount of time. This framework can be considered as an automated workflow (closed box) in which the user only inputs filter specifications to get an optimized and verified IP at the end.

This structural framework for automated hardware design has been developed using the following four steps:

1) Creation of reference models of the various filters in MATLAB for obtaining their functionality.
2) Writing the behavioral description of the filters in C++ inside the HLS tool. These descriptions are parametric in nature, which allows them to adapt to user requirements without changing the source code.
3) Usage of optimization techniques inside each of the filters (e.g. resource sharing, loop unrolling, pipelining).
4) Carrying out functional verification of the design, by comparing the output results with the reference model.

All the reference models, HLS source codes and optimization codes are permanent as well as parameterized. This enables automatic modification of the hardware architecture based on input parameters such as filter order, bit width of the various variables, amount of resource sharing etc. The details of these steps are mentioned in the subsequent sub-sections.

A. REFERENCE MODEL

The first step involves designing the reference models of four different types of digital filters. For this, we use MATLAB which is one of the most commonly used tools for this purpose. It is widely used in signal processing domain for the purpose of modelling, simulation and testing development environments [25]. This facilitates the design of various systems along with rapid verification of signal processing algorithms. It provides a DSP System Toolbox which can be used to design and analyze a variety of different types of single-rate and multirate filters. FPGA vendor tools (Xilinx, Altera, Actel etc.) also rely on MATLAB/Simulink for coefficient generation, simulation and for implementing time-domain computations, which are essential for the filtering process [13]. A built-in function library is also provided that can be used to filter any input data using a rational transfer function defined by the numerator and denominator coefficients.

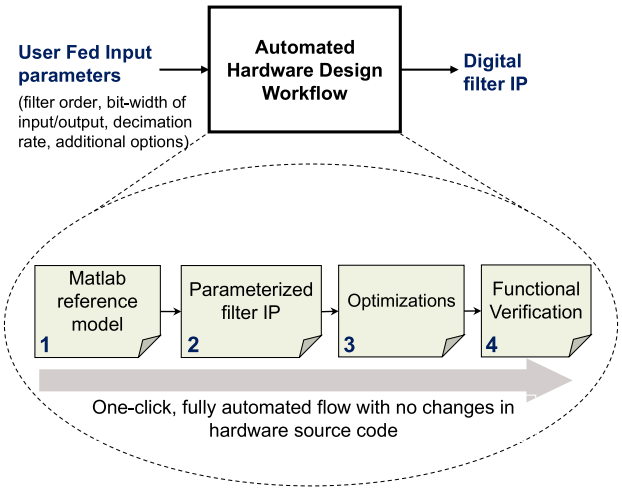

FIGURE 1. Overview of the automated hardware design workflow which acts as a closed box solution for the user. This workflow has been developed using four major steps.

These models act as the golden reference for subsequent functional verification of results as well as providing further data / information about the filters. This data can then be used to devise all the input parameters while carrying out the design in hardware later e.g. the reference model helps to obtain filter coefficients in case of an FIR filter using MATLAB. These coefficients become the necessary input parameter for that specific FIR filter and they can be fed automatically to the input of our hardware design. The user can set the desired parameters (filter order, input/output bit width etc.) which are fed into the reference model to obtain the output data which is stored in a separate file for further use.

B. BEHAVIORAL DESCRIPTION

The next step entails writing the behavioral description of all these filter models in C++, which forms the foundation of HLS coding. These models outline the filter's fundamental operations and core functionality, concentrating on the utilization of resources (multipliers, adders, internal shift registers etc.) in the appropriate sequence.

The goal is to create source codes of these four types of filters that are *completely generic* in nature and which could be used to create the designated kind of filter for any input requirements as mandated by the user. This is achieved with the help of parameterized filter structure algorithms which enable adaptation of the internal hardware architecture automatically without further modification in the source code ever. Although designing hardware for a particular set of predefined requisites usually yields the best results, our work aims to attain the same quality even after provision of a generic framework.

C. OPTIMIZATIONS

This step of the procedure deals with model optimizations to get better throughput, latency or area results. Generally, several fundamental methods are used, such as loop unrolling and pipelining with different initiation intervals (II). Most of the time, area and throughput are a trade-off, but the ultimate decision is based on the filter design and the needs of the user. Resource sharing is possible without a decrease in throughput when multi-rate filters are used; the specifics will be discussed later in this paper. The appropriate application of these optimization strategies relies on both the hierarchical design and how constraints are applied during the design process.

D. TESTBENCH AND FUNCTIONAL VERIFICATION

The verification strategy is composed of the following two steps:

1) Pre-synthesis verification that the required functionality is correctly implemented by the C program.
2) Post-synthesis verification to ensure the RTL code operates as intended.

Using a C++ testbench, this functional verification of the device under test (DUT) is carried out. The synthesized RTL and the C model are both validated using the same testbench, which also guarantees that the RTL's functionality matches that of the C source code.

Initially, the DUT's functional correctness is verified by comparing its C++ code output with the golden reference model that was obtained from MATLAB. This can be accomplished simply by recording the DUT's output values inside the testbench and comparing their numerical accuracy to the corresponding golden value.

To handle floating-point precision errors that may arise in real world applications, our flow uses fixed-point data types in both MATLAB and HLS flows to ensure bit-accurate comparisons. For the MATLAB reference model, all the numbers are first converted to fixed-point notation via the fixed-point designer toolbox. Similarly, all the values in the HLS flow are defined using built-in fixed-point data types. The conversion

from floating-point to fixed-point is automatic in both flows and depends on the required bit width of the design which is prescribed by the user. This ensures identical quantization of the values and offers equal comparison of the results, both in terms of precision and numerical correctness.

Second, it's necessary to examine the RTL code as well. For this purpose, we use the automated verification flow provided inside Catapult HLS which compares the C++ design against the RTL using the same testbench. This flow is used to check that the outputs from the C++ and RTL designs correspond, proving that the C++ and RTL have functional equivalence for the test vectors that the testbench has provided. For the generation of sample test inputs, we use MATLAB to generate a composite sine wave by summing two sine waves of different frequencies, scaling and phase shift, and then normalizing the result. The sampling frequency f_s is adequately chosen to satisfy the Nyquist criterion.

The design approach is not meant for advanced level verification; therefore, a comprehensive testing environment which covers exhaustive test coverage is outside the scope of this paper. We generate numerous random samples, quantize them to the requisite number of bits, and use them as input for basic functional verification.

Due to the nature of this approach, which relies on comparing HLS outputs with reference model results for basic verification, certain limitations emerge. In case of mismatch between the final results, the debugging process can become time-intensive, as it requires a detailed examination of the parameterized and automated source code. This added effort can partially offset the time savings gained through automation, especially in highly configurable or complex design scenarios.

These above-mentioned steps have been applied for the development of an automated workflow, which can now be used by designers as a tool to speed up the process of digital filters creation. Fig. 2 presents a detailed description of this process, wherein the automated workflow receives user inputs consisting of decimation rate, filter order and I/O bit width etc. These parameters dynamically configure the corresponding MATLAB reference model for the selected filter type, enabling the generation of output data for a predefined set of sample inputs. The model applies quantization and rounding operations to ensure accurate bit-level precision, and the resulting outputs serve as the golden reference for functional verification. Simultaneously, the same input parameters are used to update HLS header files, which in turn guide the generation of new hardware implementations from the parameterized source code through appropriate optimizations. After synthesis, the generated hardware design is simulated with the same set of sample inputs and the corresponding outcome is verified against the reference values obtained earlier. In cases where functional verification fails, manual debugging and modification of the parameterized source code is required to resolve discrepancies.

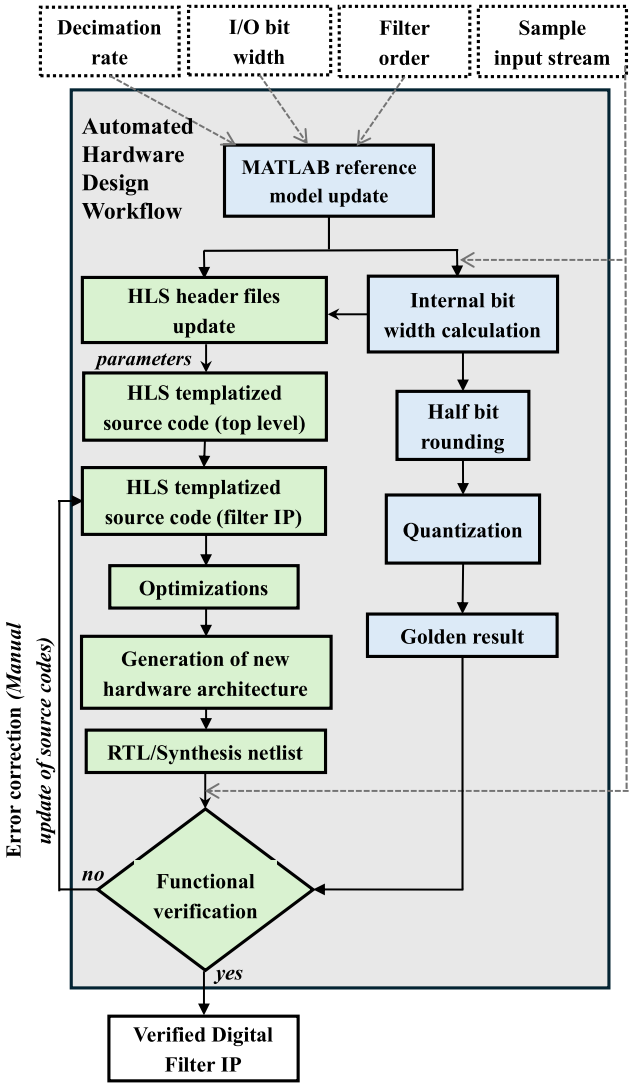

FIGURE 2. Detailed flowchart of the automated hardware design methodology. Steps performed by the HLS tool are shown in green color and steps performed by MATLAB in blue color. The complete flow is a *one-click* solution with no changes required at any stage of the design process.

IV. FRAMEWORK DESIGN OF INDIVIDUAL FILTERS

A. FIR FILTER

FIR filters are feedback-free digital filters with a finite impulse response, whose output values y(n) are calculated from a stream of input signals with value x(n) [26]. In the time domain, the implementation takes the form of sum of products.

$$y(n) = \sum_{k=0}^{N} b_k x(n-k) \tag{1}$$

where $y(n)$ is the output, $x(n)$ is the input sample, b_k is the coefficient value and N is the filter order. The general implementation of an FIR filter is depicted in Fig. 3.

The impulse response of such a filter is defined by the filter coefficients, which are the values that determine how the filter processes the signal. The filter coefficients are typically determined by a design algorithm that takes into account the desired frequency response and other design constraints, such as filter order and passband ripple. The number of filter coefficients, or the number of taps, determines the filter's length as well as complexity, and is responsible for determining how it affects different frequencies of the input signal. Bandpass, band-reject, high-pass, and low-pass frequency responses are only a few of the possible configurations for FIR filters. In our work, we have decided to use low-pass filters.

Prior to designing a parameterized filter in HLS, the general functionality of this filter was defined and optimized for the most effective implementation. For this purpose, MATLAB was used to create the necessary reference model of an FIR filter. Depending on the specifications, the coefficients are obtained using the inbuilt *fir1()* function, which are then quantized to the required bit width using the MATLAB fixed point conversion tool.

The MATLAB built-in *filter* function, which receives the quantized inputs and quantized coefficients as its parameters, is used to calculate the output values. Again, the result is quantized to the desired number of bits set by the user and stored in a different file that will serve as the golden reference model.

1) ALGORITHMIC DESCRIPTION AND OPTIMIZATIONS

The operations required by FIR filters can be categorized into two broad categories: Shift operation and multiply-accumulate (MAC) operation. With each new incoming input, all samples are shifted; one new sample enters the filter and the last sample is discarded. Additionally, the MAC operation is carried out after each shifting, multiplying each input sample by its corresponding coefficient and adding the results to generate the output for the current instance.

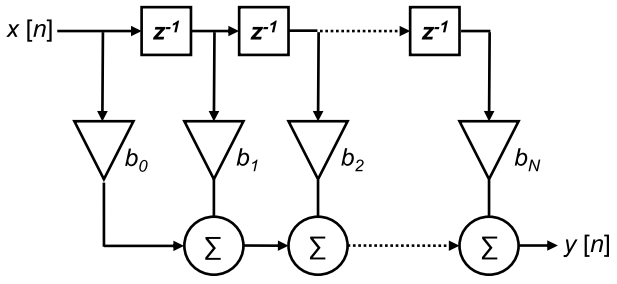

FIGURE 3. General implementation of a direct-form FIR filter with order N. It consists of an N-stage delay line at the top with N+1 taps and an equal number of coefficients.

In HLS, such repetitive tasks are performed with the help of *loops* which serve as the primary mechanism for applying synthesis constraints as well as moving data/IO into or out of an algorithm. The design consists of two basic loops:

1) **Shift Loop:** It is merely a shift register which is used to shift all the values in the array to the next location in the array, while the first location takes the input sample. It is iterated for a number of times equal to the tap count of the filter (order + 1).
2) **MAC Loop:** It is used for the multiplication of all the corresponding shift register values with their coefficients and add them together. It also has the same iteration value as the shift loop.

Although functionally correct, this implementation does not yield practical results because inherently, Catapult HLS utilizes rolled loops which implies that each iteration of the loop takes at least one clock cycle to execute in hardware. Consequently, this results in only one MAC unit being used here and the result of the complete operation takes multiple clock cycles to execute, which is not the desired output rate of a practical FIR filter.

In order to yield the best QoR, the following optimizations are carried out in the design:

1) **Loop Unrolling:** It is the primary mechanism to add parallelism into a design by automatically adding multiple loop iterations in parallel. Loop unrolling can theoretically execute all loop iterations within a single clock cycle if there are no dependencies between successive iterations and enough multiply and add resources.
2) **Pipelining:** It refers to how often to start the next iteration of the loop. This is controlled by the Initiation Interval (II) which determines how many clock cycles are required before starting the next loop iteration. In our design, we keep II=1 which means that a new loop iteration is started every clock cycle. The throughput reaches *one* as a result of these optimizations, indicating that the design is consistently producing a result at each clock cycle. However, the area grows significantly because of the additional MAC units.
3) **Coefficient Storage Inside Memory:** Instead of feeding the coefficient values from outside, our design stores these values internally inside registers. These coefficients, when synthesized to RTL, offer great dividends in terms of area savings, when the filter implementation is fully unrolled as discussed previously. During the synthesis process, Catapult performs an automatic optimization such that no multipliers are utilized in the design and all the computations are performed using additions and shift operations.
4) **Coefficient Folding:** It is possible to reduce the number of multipliers if the filter coefficients are symmetric around the center coefficient. This allows for addition of two input samples prior to their multiplication with the same coefficient value, which results in a single multiplier instead of two. This can be easily understood by

$$(x[1]\times coeff) + (x[2]\times coeff) = (x[1]+x[2])\times coeff \tag{2}$$

This is a parametrizable option which the user can set while giving initial constraints of the design.

Subsequent to implementing these optimizations, there is a significant enhancement in the overall QoR. The throughput and latency reach *one*, which is the intended output of a practical FIR filter. However, both parameters are adjustable within the Catapult HLS environment. Although we did not exploit it in this work due to the final requirements, adjustment of these two parameters produce results with different resource usages and performance. The area score demonstrates a significant reduction by over *50 %* on average as compared to the unoptimized design, although the exact figures vary with the order of the filter. This substantial decrease arises from the transformation of all multipliers into adders and shifters.

These optimizations constitute a permanent aspect of the automated design methodology, thereby guaranteeing that all designs generated through this process yield optimal outcomes.

B. POLYPHASE FILTER

Polyphase filters fall under the category of multi-rate filters which use different input and output frequencies. In our work, we have used a decimation filter which implies that down-sampling has been achieved at the output, resulting in a lower output frequency with respect to the input frequency.

An M-fold decimator takes an input x(n) and produces an output sequence [27].

$$y_D(n) = x(Mn) \tag{3}$$

where M is an integer, also known as the decimation rate.

This difference in the sample rate can be used to achieve more area efficient designs by reusing hardware such as adders or multipliers at the lower frequency side.

Usually, the process of decimation involves periodically discarding output samples to match the desired reduction in rate e.g. a decimation rate of 'M' would imply that every M^{th} sample is kept and the rest are discarded. However, this approach wastes a lot of resources. One way to fix this issue is to use a structure that is only "enabled" when the output has to be generated. This multiplexes the filter and divides it into M stages, or sub-filters, which sort the incoming data and pair it with the corresponding coefficient. This results in a greater number of clock cycles available per operation and resource sharing can be carried out [28].

Consider Fig. 4, where we have a filter of the form

$$H(z) = \sum_{n=-\infty}^{\infty} h(n)z^{-n} \tag{4}$$

By separating the even and odd number coefficients, we get [29]

$$H(z) = \sum_{n=-\infty}^{\infty} h(2n)z^{-2n} + \sum_{n=-\infty}^{\infty} h(2n+1)z^{-2n} \tag{5}$$

And if we define

$$E_0(z) = \sum_{n=-\infty}^{\infty} h(2n)z^{-n}, E_1(z) = \sum_{n=-\infty}^{\infty} h(2n+1)z^{-n} \tag{6}$$

then H(z) can be written as

$$H(z) = E_0(z^2) + z^{-1}E_1(z^2) \tag{7}$$

Equation (7) is represented as the structure in Fig. 4(a), which can be redrawn as shown in Fig. 4(b). As evident, the second implementation is more efficient because both the sub-filters are now operating at a lower rate which allows the possibility of resource sharing between them.

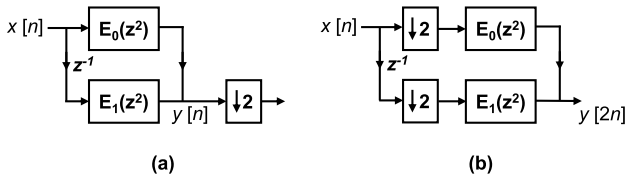

FIGURE 4. Polyphase implementation of decimation filter. (a) Two polyphase components. (b) Equivalent configuration in which down sampling is performed before the filtering process [29].

1) RESOURCE SHARING

There are two approaches to build this kind of filter in Catapult: either using a *low-level* manual hardware sharing approach or a *high-level* algorithmic one [30]. While the first option requires a thorough knowledge of how data flows through the tap shift register and how the output data is produced by the filter, the second option raises the abstraction level and spares the hardware designer from delving too deeply into the underlying architecture. Although this approach is less time consuming, it may not always yield the most optimal outcomes. On the other hand, the first method gives the designer more control over the underlying architecture and the process which is used to generate the final RTL. This low-level approach has been adopted for the purpose of this work in order to achieve better results.

The code is structured to compute only the pertinent outputs and given that there are 'M' clock cycles between each output, hardware resources can be utilized efficiently. Firstly, the number of MAC units needed inside the architecture are calculated, which is the ratio between filter tap count (order + 1) and decimation rate. Then, the design is broken down into multiple phases, equivalent to the number of MAC units previously computed. Each phase starts at a new clock cycle and it is the hardware design's equivalent of a distinct function call, similar to multiple sub-filters inside a polyphase architecture.

By linking the coefficient index and register index as a relation of input position in the shift register, decimation factor and order of the filter, it is possible to derive a generic technique which gives the most efficient architectural implementation of the polyphase decimation filter in terms of maximum resource sharing of MAC units.

2) ALGORITHMIC DESCRIPTION AND OPTIMIZATIONS

The operation of our polyphase filter can also be categorized into shift and MAC loops. The shift loop has the same functionality as a shift register, as previously elaborated. However, the execution of the MAC loop is exclusively determined by the manual hardware sharing approach, which allocates the input samples across the MAC units in a seamless manner to achieve optimal resource sharing.

The optimizations for the polyphase filter include loop unrolling, pipelining and coefficient storage inside memory, similar to that of FIR filters.

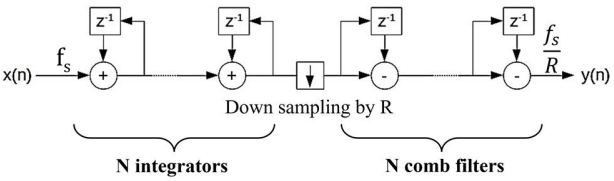

FIGURE 5. An integrator section with N integrator stages that processes the input data samples at a sampling rate fs, and a comb section that operates at the lower sampling rate fs / R.

Moreover, there is one additional optimization for this filter when the tap count of the filter is not an integer multiple of the decimation rate ($N \neq M$). In this scenario, a few sub-filters in the design are not completely populated with coefficients, which alters the phase response. *Coefficient padding* is used to overcome this issue in which the vacant slots of the sub-filters are filled with zero coefficients. In this manner, the general polyphase implementation formula remains valid without influencing the filter's response.

Due to the storage of coefficients internally in the memory, the extra zero coefficients cannot be added from the outside. Rather, a generic method for initializing the correct number of total coefficients (actual number + additional zeroes) inside the memory is required to ensure that calculations made with varying specifications always produce the right results. In order to accomplish this, we utilize C++ preprocessor directives to *internally* determine the total number of coefficients needed. Next, a new array (internal registers) is initialized with a value of zero in all the registers. Lastly, the values of the N filter coefficients—where N is the initial tap count—are overwritten in this array. The remaining registers will retain the value of zero, which is equivalent to coefficient padding implementation.

C. CIC FILTER

CIC filters are also categorized as multi-rate filters and are commonly used when dealing with different input and output sample rates. They belong to the class of narrowband low-pass filters and are highly computationally effective [31].

Reference [32] gives the structure of the CIC filters that chains N number of integrator and comb sections to form a decimator or interpolator. This work, however, deals with CIC decimation filters only. Fig. 5 illustrates the block diagram of such a filter where the input signal passes through N integrator stages, followed by a decimator, and then finally through N comb stages. The system response of such a filter is given by

$$H(z) = [\sum_{k=0}^{RM-1} z^{-k}]^N$$

or

$$H(z) = \frac{(1-z^{-RM})^N}{(1-z^{-1})^N} \tag{8}$$

where N is the filter order (number of comb-integrator pairs or number of stages), M is the differential delay (usually one) and R is the decimation factor.

The absence of arithmetic multipliers in the design of CIC filters is one of their main benefits. They also save space in terms of less data storage because there are no multiplication operations required as they do not require any coefficients. There are only two arithmetic operations: addition and subtraction [32].

When the output of the filter given by (8) is decimated (down sampled) by a factor R, the new response of the filter at M=1 can be expressed in (9) as

$$H(z) = \frac{(1-z^{-R})^N}{(1-z^{-1})^N} \tag{9}$$

This response is at the lower, down-sampled rate and it can be viewed as cascade of N integrators and N comb filters.

$$H(z) = \frac{1}{(1-z^{-1})^N} \times (1-z^{-R})^N \tag{10}$$

One crucial consideration that must be made in order to guarantee complete correctness of the results is the register bit widths. The integrator (higher frequency) part of the CIC filter can be visualized as an accumulator. With a continuous input stream, arithmetic overflow eventually occurs since the integrator stage accumulates results at every stage. However, if the subsequent two requirements are satisfied, this overflow does not affect the outcomes.

1) Each stage uses two's complement arithmetic (non-saturating) [33].
2) The maximum value expected at output of a stage is less than or equal to the range of stage's number system.

Equation (11) has been used in this work to calculate the minimum bit width of the registers used inside the CIC filter in order to avoid overflow.

$$register\_width = x\,(n) + ceil\left[N \times \log_2\,(M \times R)\right] \tag{11}$$

where $x(n)$ is the input sample bit width, M is the differential delay, N is the filter order and R is the decimation rate. This implies that the input samples are first sign extended to the higher number of bits before entering the integrator section.

1) ALGORITHMIC DESCRIPTION AND OPTIMIZATIONS

Our CIC filter's algorithmic description, expressed in HLS, is partitioned into two components: *integrator* and *comb*. The high-frequency portion of the filter, known as the integrator section, is made up of a series of adders and feed-back pathways. It receives input data samples every clock cycle. It sends its output only when the counter reaches the value of decimation factor.

The filter's low frequency comb section, which consists of a series of subtractors and feed-forward circuits, receives input data samples from the integrator at a slower rate (equivalent to the decimated rate).

Both these sections are parameterized through templates and they are instantiated as classes by a top-level module, which is used for passing the template parameters such as input and output bit widths along with decimation rate. Similar to the FIR and polyphase filters, the intermediate bit width of the accumulator inside the CIC filter is calculated mathematically inside the source files, as a pre-processor directive. Equation (11) can be rewritten inside the HLS top-level module as

```
register_width
  = ac: :log 2 _ceil<power<M, N>:
  : value *_power<R, N >
  :: value >:: val + x(n)
```

For optimization purposes, loop unrolling and pipelining have already been incorporated in the design. Additionally, *resource sharing in the comb section* has been incorporated to further optimize our filter. As the comb section operates at a lower frequency as compared to the integrator section, there are extra number of clock cycles available for computation before it accepts the next input. By setting the initiation interval of this section equal to the decimation rate, it is possible to utilize resource sharing capability provided by Catapult HLS. For example, the comb section of a CIC filter with order =8 typically needs eight adders, or subtractors. However, just two adders are utilized at a decimation rate of four because they are both operating throughout each clock cycle to generate the eight calculations that are originally required.

D. FILTER CHAIN

The architecture, algorithmic description, and optimizations pertaining to three different filter types—FIR, Polyphase, and CIC—have been covered in the last three sections. However, in the field of signal processing, it often preferred to implement a chain of filters where the output of one filter is directly coupled to the input of the next. Better overall results can be obtained with such an implementation, which would otherwise not be possible with a single filter. Additionally, the designer can impose fewer constraints on each filter when using a filter chain, ultimately leading to simpler individual designs.

This section pertains to the design of filter chains using the *parameterized filter structures of FIR, polyphase and CIC filters already available* (as discussed in the previous sections). The method to successfully combine all of the filters into a single top-level design so that it functions as a single block, comprising of several independent sub-blocks, is also discussed in this section.

The general architecture of such a filter chain is depicted in Fig. 6, in which multiple filter blocks are seamlessly coupled to one another. Prior to synthesis, the user feeds all the parameters into the design from the outside. By doing this, Catapult is able to produce each filter's associated design architecture in the most efficient manner. The incoming sample x is an n-bit digital stream that is fed into the first block of the filter, which might be a polyphase, FIR, or CIC filter. The first filter executes the filtering process and outputs the result on a user-specified number of bits, which are subsequently fed into the second filter. Until the entire chain is depleted, this filtering and feeding operation for the subsequent filters keeps on repeating till the last result.

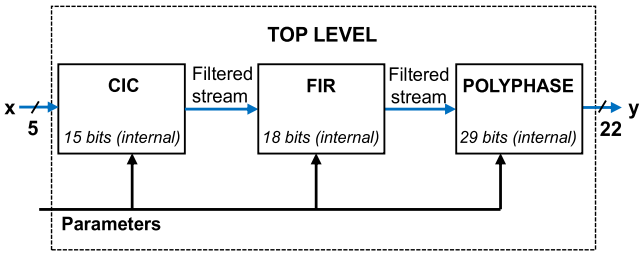

FIGURE 6. Hierarchical design of a filter chain. Output y is generated after the input x has passed through all the individual filters in the filter chain.

1) ALGORITHMIC DESCRIPTION AND OPTIMIZATIONS

The filter chain in our design is created using hierarchical C++ blocks made up of classes, which are in fact the filters already designed previously. Adding hierarchy allows the different blocks to run in parallel and also manage the synchronization of different data rates present at each input and output of the individual blocks, without compromising the throughput of the design.

The top-level module class contains the necessary code to instantiate all the individual filters and the parameters of each of them are defined by the user at the start. These parameters determine the characteristics as well as architecture of each filter. Furthermore, this top-level module is also used to define all the interconnecting channels which connect the different sub blocks of filters together.

All the distinct optimizations inherent in the various filters are also incorporated inside this filter chain. This leads to an extremely well-optimized system design, which would have been exceedingly challenging and time-consuming to attain otherwise. This is particularly useful when the cumulative growth of the decimation rates of the individual filters results in an overall increase in the decimation rate progressively. As a result, while proceeding down the chain, the sampling frequency is further diminished, increasing the potential for sharing hardware resources.

E. COMPARITIVE ANALYSIS

Building on the previously introduced digital filter types, this section deals with the comparative analysis of these four different filters and highlights their distinctive strengths, limitations, and optimal use cases. A thorough comparison reveals distinct trade-offs in performance, complexity and application suitability, which can be used as a baseline for designers to select the appropriate type of filter during the design process. FIR filters are well-known for their inherent stability and exact linear phase characteristics, making them ideal for applications where waveform integrity is critical, such as audio processing and communication systems [34]. However, achieving sharp transition bands or high attenuation with FIR filters often requires a large number of taps, leading to increased computational complexity and hardware cost [35]. A major drawback of FIR filter is the large number of arithmetic operations involved during the implementation which limits its speed and demands more power [23]. In contrast, polyphase decimation filters offer considerable efficiency in multi-rate signal processing by partitioning the filtering operation across several lower-rate sub-filters, thereby improving throughput and reducing latency [27]. This decomposition enables significant reductions in computational load and is highly suited for high-speed digital down converters [36]. However, the design complexity of polyphase filters increases with the number of phases, requiring careful coefficient partitioning and control logic management. Polyphase filters find their way into a wide range of applications such as radio frequency receivers [37], wideband signal monitoring [38], and radar signal processing [39] etc.

Similarly, CIC filters are also widely adopted in multi-rate signal processing due to their multiplier-free implementation and low resource usage [40], which makes them especially advantageous for low-power embedded systems. Due to the absence of multipliers, CIC filters offer unmatched hardware efficiency for decimation, particularly in delta-sigma analog-to-digital converters [33]. They are particularly useful in filtering out of band quantization noise and preventing excess aliasing introduced during sample rate decrease [31], hence they are extensively used as anti-aliasing filters prior to decimation. CIC filters are also employed in high-speed communication and DSP pipelines during front-end processing stages to handle large bandwidth signals with minimal computational overhead [41]. While computationally efficient, CIC filters suffer from notable passband droop and poor stopband attenuation [42], [43], necessitating the use of additional compensation filters, often FIR-based, to meet spectral requirements [44]. Moreover, when higher flexibility is needed, combining multiple filters into a filter chain becomes advantageous. Filter chains allow system designers to leverage the efficiency of CIC filters in early decimation stages followed by FIR or polyphase filters for refined spectral shaping [45]. Filter chains also allow designers to address many limitations of the individual filter types through modular design stages, by optimizing different aspects of performance such as frequency response and computational load [46]. However, the cumulative latency and complexity of filter chains poses considerable challenges for the designer [26], both in terms of design and resource utilization. Moreover, the interaction between the different filter stages, such as phase distortion, delay alignment and dynamic range matching requires additional validation effort [47].

In practice, the selection of appropriate filter architecture is guided by application-specific requirements such as throughput, latency and resource constraints. FIR filters are particularly well suited for scenarios where high precision and linear phase response are essential, whereas polyphase filters are advantageous in multi-rate and real-time systems requiring computational efficiency. CIC filters are preferred where hardware simplicity and efficiency are paramount, though they often require post processing to meet stringent spectral demands. Filter chains offer a flexible and scalable solution for complex systems such as software-defined radios, where a balance of performance, complexity and resource constraints is required. Ultimately, understanding these trade-offs is crucial in the selection and design of filter architectures for high performance DSP systems.

V. EXPERIMENTAL RESULTS AND COMPARISONS

A structural framework for automated hardware design is obtained by successful completion of all the steps discussed in Sections III and IV. It consists of the complete hierarchy, templates, source codes and constraints. This framework allows the user to input the required parameters and obtain the optimized hardware architecture of DSP blocks (digital filters in our case) in an extremely short amount of time.

To evaluate the quality of our generated designs using this structural framework, we have compared them against existing IPs which were developed using hand-written RTL. The specifications of these IPs were imported from the existing designs and used as input parameters in our automated workflow. Comparisons were then made between the design generated by the automated HLS workflow and the reference RTL designs (both of them having the same parameters and synthesized by the same tool).

For the purpose of this work, *area* and *static power consumption* have been used as benchmarks, and our work specifically targets application specific integrated circuits (ASIC) implementations.

The RTL netlists have been synthesized using Cadence Genus Synthesis. The frequency of all designs is kept at 100 MHz, thus ensuring equal performance between RTL and HLS designs, and the cell library used for the technological mapping is 130 nm, provided by Infineon Technologies, AG Austria. To ensure a fair comparison of the synthesized hardware for a given performance target, all the designs (HLS and RTL) have been constrained to the same latency and throughput.

The presented results will illustrate that the QoR achieved through our automated design methodology remains at

par with traditional RTL, across different design contexts and filter specifications. In some cases, the performance of HLS-generated designs shows significant improvements, while in certain scenarios, it falls short in terms of area and power consumption. This variation in the final outcomes arises primarily from differences in the micro-architectures generated by our methodology compared to those manually coded in RTL, even when both implementations perform the same functional tasks.

The observed variation in these micro-architectures can be primarily attributed to two key factors. First, HLS tools inherently apply architectural transformations based on pragma-driven optimization strategies, which may sometimes lead to insertion of additional components. These decisions, while often beneficial, can diverge slightly from the architecture that a human designer might create manually. Second, the final hardware realization is highly dependent on the expertise and design approach of the individual engineers involved in both the HLS and traditional RTL flows. As such, both tool behavior and designer proficiency play critical roles in shaping the micro-architectural outcomes.

A. POLYPHASE FILTER

The first comparison is carried out with a polyphase filter of *order* = *28*, *decimation* = *8*, *input bit width* = *2* and *output bit width* = *13*; an IP which had been developed using traditional HDL coding.

Fig. 7 shows the performance results of the HLS automated flow for area and static power consumption compared to the existing HDL-based implementation. The automated flow can be seen to provide a filter implementation with a smaller area (*20.2 %* area reduction) and having a slightly lower static power consumption.

The smaller area of the design can be attributed to resource sharing of multipliers. Normally, a filter of order *n* will utilize *n+1* multipliers to instantaneously calculate the products of all input samples with their respective coefficients. However, in our design the minimum number of multipliers which are needed to perform the filtering process by keeping in view the required decimation rate is automatically calculated. Instead of using 29 multipliers, the design utilizes only 4, which are re-used over multiple clock cycles according to the decimation rate. This technique results in a significant reduction in the overall area.

Moreover, the overall design time of this filter can be approximated to ∼3 minutes. This time includes the initial setting of user parameters in the MATLAB file, all the way up to the functional verification after creation of the new design. This result highlights the utility of the proposed methodology in the rapid generation of efficient micro-architecture, characterized by a high degree of resource sharing and architectural optimization.

The next comparison is implemented on a polyphase filter having an *order* = *123*, *input bit width* = *18* and *output bit width* = *37*. Two distinct *decimation rates (2 and 62)* are selected to allow a thorough examination of the impact of various rates on the area comparisons. The provided RTL design has been developed with the exact same specifications, only for the purpose of cross evaluation.

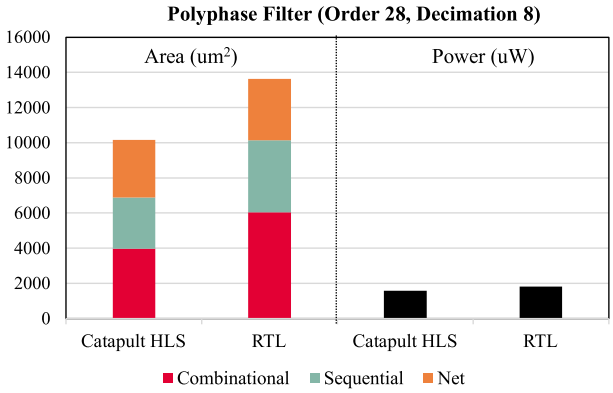

FIGURE 7. Performance results of Catapult HLS vs RTL design on an order 28 polyphase filter. The HLS generated design exhibits a 20.2 % area reduction and 12.7 % static power reduction.

Fig. 8 shows the comparisons of the two polyphase filters having different decimation rates; for the HLS design flow and the RTL design. Note that the two designs offer vastly contrasting results for separate decimation rates. For a large decimation rate = 62, the HLS flow achieves an area reduction of *54.4 %* with only 2 multipliers and 5 multiplexers in the design. This can be explained by the fact that the HLS filter achieves more effective resource sharing and effective hardware reuse when *the architecture is more complicated, i.e. with a large decimation factor.* To achieve a similar level of reuse, the HDL designer will need to spend a considerable amount of time and effort.

However, for a small decimation rate = 2, the HLS automated flow delivers an area overhead of *26.3 %*, compared to the RTL design. This can be attributed to the presence of large number of multiplexers in the design which are automatically inserted by the HLS flow.

The result obtained for higher decimation rates is very promising for HLS automated flow, implying that it is a viable alternative to polyphase filter design in industrial flows. However, a notable limitation of this approach is observed at lower decimation rates, where the synthesized designs tend to exhibit suboptimal quality of results, highlighting a scenario in which the methodology is less effective.

B. CIC FILTER

The first CIC filter to be designed using the automated HLS flow has *order* = *3*, *decimation* = *512*, *input bit width* = *2* and *output bit width* = *16*. After input from the user, these four parameters are automatically updated in the relevant header and source files, providing the hardware design of this filter via Catapult HLS. The results are compared with an existing CIC filter IP of the same specifications being used in an industrial design.

The bill of materials (BOM) of the HLS synthesized design conforms to the theoretical estimates with a total of eight adders in all. Six are used in the integrator and comb portions (three each), one adder serves as a counter and the last adder is employed for half-bit rounding of the result. The internal accumulator bit width is automatically adjusted to 29 bits, which allows for full precision and zero loss of accuracy.

The comparative area and static power results are shown in Fig. 9. The HLS design can be seen to provide better results by achieving *16.1 %* area reduction and *27.4 %* static power reduction. Upon closer analysis of the synthesized netlist, it is observed that the area occupied by sequential elements of the HLS design is larger as compared to its traditional counterpart due to higher number of FFs. This is due to the insertion of two extra registers i.e. the output register of 29 bits at the integrator stage and another output register of 16 bits at the comb stage (which is also the final output). However, the HLS design's combinational area exhibits a significant reduction and this can be attributed to a difference in architectural implementation.

Within the integrator stage, our HLS design integrates the delay registers in a straight, feedforward path to match the outcomes with the MATLAB algorithm that employs an identical implementation. As a result, slower and smaller adders can be employed because the timing limitations are lessened. As opposed to this, the manually designed architecture includes these registers in the feedback path, leaving one straight combinational path that connects the input to the decimation block. Due to this, faster adders are used to achieve the timing constraints.

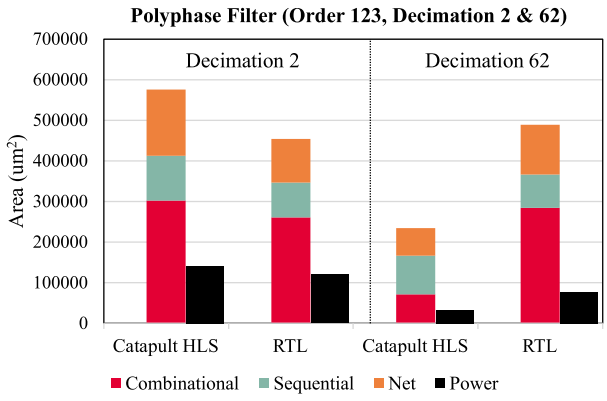

FIGURE 8. Performance results of Catapult HLS vs RTL design on an order 123 polyphase filter with varying decimation rates. The HLS generated design exhibits a 54.4 % area reduction for decimation 62; however, it shows a 26.3 % overhead for decimation 2.

These findings can also serve as an indication of the fact that an automated design methodology allows for faster investigations of better functional architectures for the same set of specifications.

For the next evaluation, the specifications of the CIC filter are $order = 4$, $decimation = 8$, $input\ bit\ width = 2$ and $output\ bit\ width = 14$. Fig. 10 shows the results for area and static power consumption.

The HLS design exhibits an area overhead of *35.8 %* and this additional area arises from some different architectural details inside the two filters. Due to its integration within another IP, the HDL design excludes both an output register at the integrator stage and a register alongside the final subtractor in the comb section. This disparity in the sequential elements *(153 FFs vs 108 FFs)* due to extra registers as well as additional *net area* accounts for the overall area increase. This also shows the limit of using an automated approach which increases the area when merging with systems that are not designed with HLS. The HLS flow is seen to add extra FFs at the input and output of the design which can be avoided when implementing manual RTL coding.

C. FILTER CHAIN

The final evaluation is done on a filter chain containing a total of *seven* filters of different types. The individual specifications of each of the filters are specified in Table 1.

TABLE 2. Filter chain specifications.

	CIC	CIC	CIC	CIC	CIC	FIR	Poly phase
Order	4	4	5	8	14	8	122
Decimation	2	2	2	2	2	1	2
Input width	2	6	10	15	23	18	18
Output Width	6	10	15	23	18	18	16

At a clock frequency of 100 MHz, the final area of the HLS automated flow is *658996 um^2* while the area of the RTL flow is *640717 um^2*, resulting in a mere *2.9 %* overhead. The area and static power comparisons are depicted in Fig. 11.

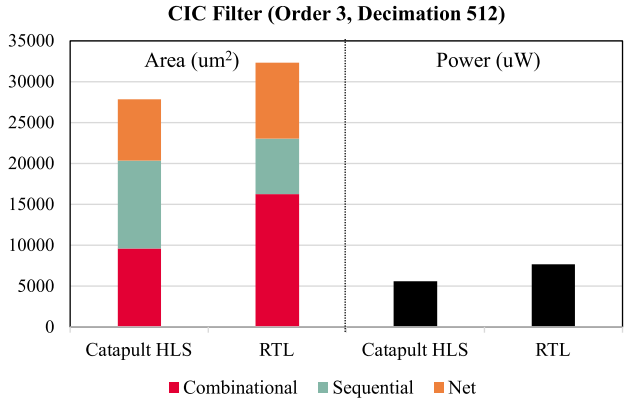

FIGURE 9. Performance results of Catapult HLS vs RTL design on an order 3 CIC filter. The HLS generated design achieves a 16.1 % area reduction and 27.4 % static power reduction.

The graph illustrates how the two design flows produce roughly equivalent area outputs. Nevertheless, additional scrutiny reveals that the size of the area scores varies for each separate filter inside the chain. The preceding sections have covered the causes of these variations, which can be summed up as follows:

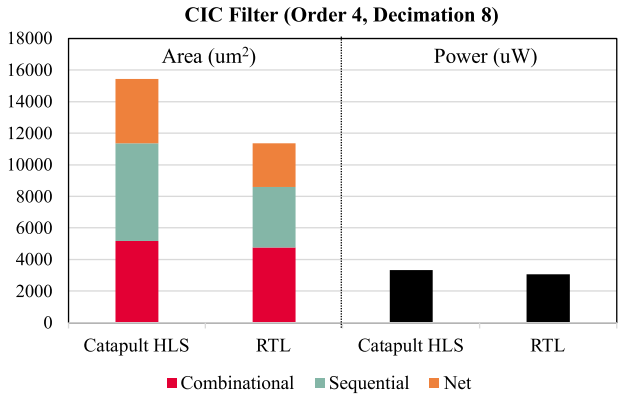

FIGURE 10. Performance results of Catapult HLS vs RTL design on an order 4 CIC filter. The HLS generated design exhibits a 36.8 % area overhead and 8.1 % static power overhead.

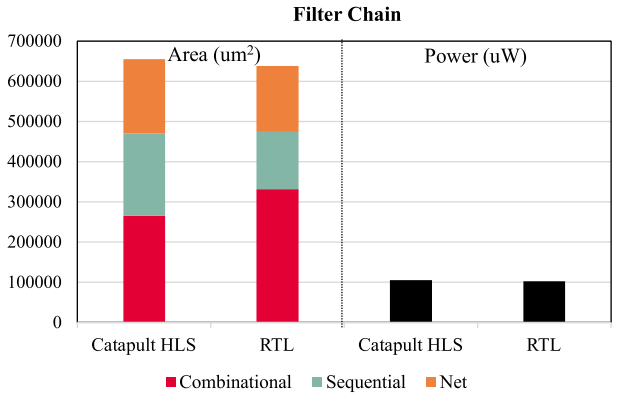

FIGURE 11. Performance results of Catapult HLS vs RTL design on a filter chain. Both the designs achieve almost similar QoR with a difference of 2.9 % in favor of RTL design.

1) The FIR and CIC filters from the HLS implementation are smaller in size due to optimizations such as resource sharing and usage of constant adders / shifters instead of multipliers.
2) Due to the decimation rate of 2, the RTL implementation's polyphase filter has a smaller size.
   This demonstrates that the discrete filters inside the chain play a significant role in determining the overall output in terms of area and power consumption.

VI. CONCLUSION

In this paper, a novel methodology of digital hardware design (specifically digital filter IPs) has been proposed that aims to reduce design cycle time and offer customizability while retaining a high QoR. The complete design procedure integrates the HLS tool Catapult with MATLAB to produce a fully automated, one-click process that starts from concept design and ends at a synthesized netlist, complete with functional verification. By integrating a modern HLS tool with an algorithm-level modeling environment, this approach enables software and algorithm engineers, who may lack deep expertise in hardware description languages, to participate directly in hardware development. The effectiveness of the proposed methodology is evaluated by the realization of various types of digital filters and then comparing the outcomes with industry standard RTL-coded filter IPs.

The results show that the design time is significantly reduced, while the area and static power consumption of the IPs are at par with classical RTL designs. While the methodology demonstrates clear benefits, it is not without limitations. Scenarios like low decimation rates in polyphase filters and code debugging in case of mismatched outputs require detailed scrutiny of the complete automated workflow, introducing overhead and reduced efficiency. However, the speed and accuracy of this approach along with a high QoR makes it a viable option for designing custom, high-quality solutions with little effort from the designer. This accelerated and streamlined methodology is particularly advantageous in application domains where rapid prototyping and customization are essential to keep pace with evolving customer requirements. By simplifying rapid iterations and ensuring hardware efficiency, it facilitates extensive adoption in commercial product development related to DSP systems. This increased accessibility of hardware design also aligns with broader industry trends toward raising the abstraction level of design methodologies and increasing productivity [5].

The implications of this research extend beyond digital filters design. It demonstrates that modern HLS tools, when combined with automation scripts and behavioral modeling environments, can deliver industrial grade results suitable for deployment in ASIC systems. This represents a significant step forward in bridging the gap between algorithmic development and hardware implementation, especially in industries where time-to-market pressures are intense and design teams are increasingly cross-disciplinary.

By building upon this work, there is potential for the development of a virtual toolbox capable of designing different types of DSP IPs with high QoR. This toolbox can be used as a foundation for architectural exploration of DSP blocks, aiming to achieve high-quality results by combining the power of automation with the capabilities of modern HLS tools.

The results of this study can have a significant impact for both academia and industry: they show that a one-click, customizable workflow, especially for datapaths with limited control logic, can bridge the gap between high-level algorithm development and optimized hardware implementation. With the results obtained through the adoption of this automated methodology along with the progress achieved in the newest generation of HLS tools, we can conclude that this framework is ready for adoption by the semiconductor industry in prototyping and fast product development.

REFERENCES

[1] S. Lahti, P. Sjövall, J. Vanne, and T. D. Hämäläinen, "Are we there yet? A study on the state of high-level synthesis," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 5, pp. 898–911, May 2019.

[2] S. A. Butt and L. Lavagno, "Design space exploration and synthesis for digital signal processing algorithms from simulink models," in *Proc. 8th IEEE Design Test Symp.*, Dec. 2013, pp. 1–6.
[3] P. Mantovani, R. Margelli, D. Giri, and L. P. Carloni, "HL5: A 32-bit RISC-V processor designed with high-level synthesis," in *Proc. IEEE Custom Integr. Circuits Conf. (CICC)*, Mar. 2020, pp. 1–8.
[4] P. Coussy, D. D. Gajski, M. Meredith, and A. Takach, "An introduction to high-level synthesis," *IEEE Design Test Comput.*, vol. 26, no. 4, pp. 8–17, Jul. 2009.
[5] A. Takach, "High-level synthesis: Status, trends, and future directions," *IEEE Des. Test*, vol. 33, no. 3, pp. 116–124, Jun. 2016.
[6] H. Ren, "A brief introduction on contemporary high-level synthesis," in *Proc. IEEE Int. Conf. IC Design Technol.*, May 2014, pp. 1–4.
[7] J. Cong, J. Lau, G. Liu, S. Neuendorffer, P. Pan, K. Vissers, and Z. Zhang, "FPGA HLS today: Successes, challenges, and opportunities," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 15, no. 4, pp. 1–42, Dec. 2022.
[8] S. Lahti and T. D. Hämäläinen, "High-level synthesis for FPGAs—A hardware engineer's perspective," *IEEE Access*, vol. 13, pp. 28574–28593, 2025.
[9] M. C. McFarland, A. C. Parker, and R. Camposano, "The high-level synthesis of digital systems," *Proc. IEEE*, vol. 78, no. 2, pp. 301–318, 1990.
[10] Chips&Media. *Chips&Media: Design and Verification of Deep Learning Object Detection IP*. Accessed: Jan. 20, 2025. [Online]. Available: https://resources.sw.siemens.com/en-U.S./white-paper-chips-and-media-design-and-verification-of-deep-learning-object-detection-ip/
[11] T. Ogunfunmi and S. Desai, "Fast FIR filter implementation using high-level synthesis tools," in *Proc. 37th Midwest Symp. Circuits Syst.*, vol. 1, Aug. 1994, pp. 58–61.
[12] L. Hanbo, W. Shaojun, and Z. Yigang, "Design of FIR filter with high level synthesis," in *Proc. 12th IEEE Int. Conf. Electron. Meas. Instrum. (ICEMI)*, vol. 2, Jul. 2015, pp. 1067–1071.
[13] S. Filip, M. Istoan, F. Dinechin, and N. Brisebarre, "Automatic generation of hardware FIR filters from a frequency domain specification," HAL, France, May 2017. [Online]. Available: https://about.hal.science/en/
[14] S. C. Chan, K. M. Tsui, and S. H. Zhao, "A methodology for automatic hardware synthesis of multiplier-less digital filters with prescribed output accuracy," in *Proc. APCCAS IEEE Asia–Pacific Conf. Circuits Syst.*, Dec. 2006, pp. 61–64.
[15] K. S. Yeung and S. C. Chan, "The design and multiplier-less realization of software radio receivers with reduced system delay," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 51, no. 12, pp. 2444–2459, Dec. 2004.
[16] K. Wakabayashi, T. Takenaka, and H. Inoue, "Mapping complex algorithm into FPGA with high level synthesis reconfigurable chips with high level synthesis compared with CPU, GPGPU," in *Proc. 19th Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2014, pp. 282–284.
[17] G. Inggs, S. Fleming, D. Thomas, and W. Luk, "Is high level synthesis ready for business? A computational finance case study," in *Proc. Int. Conf. Field-Program. Technol. (FPT)*, Dec. 2014, pp. 12–19.
[18] M. A. Mansoori and M. R. Casu, "Hardware acceleration of biomedical microwave techniques using high level synthesis," in *Proc. 16th Eur. Conf. Antennas Propag. (EuCAP)*, Mar. 2022, pp. 1–5.
[19] S. Skalicky, C. Wood, M. Lukowiak, and M. Ryan, "High level synthesis: Where are we? A case study on matrix multiplication," in *Proc. Int. Conf. Reconfigurable Comput. FPGAs (ReConFig)*, Dec. 2013, pp. 1–7.
[20] C. Fibich, S. Tauner, P. Rossler, M. Horauer, H. Taucher, and M. Matschnig, "Preliminary evaluation of high-level synthesis tools–Xilinx Vivado and PandA Bambu," in *Proc. IEEE 13th Int. Symp. Ind. Embedded Syst. (SIES)*, Jun. 2018, pp. 1–4.
[21] M. Sudvarg, C. Zhao, Y. Htet, M. Konst, T. Lang, N. Song, R. D. Chamberlain, J. Buhler, and J. H. Buckley, "HLS taking flight: Toward using high-level synthesis techniques in a space-borne instrument," in *Proc. 21st ACM Int. Conf. Comput. Frontiers*, May 2024, pp. 115–125.
[22] A. Sohrabizadeh, C. H. Yu, M. Gao, and J. Cong, "AutoDSE: Enabling software programmers to design efficient FPGA accelerators," *ACM Trans. Design Autom. Electron. Syst.*, vol. 27, no. 4, pp. 1–27, Jul. 2022.
[23] L. Tan and J. Jiang, *Digital Signal Processing: Fundamentals and Applications*. New York, NY, USA: Academic, 2018.
[24] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, "High-level synthesis for FPGAs: From prototyping to deployment," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 30, no. 4, pp. 473–491, Apr. 2011.
[25] V. Katsikis, *MATLAB: A Fundamental Tool for Scientific Computing and Engineering Applications*, vol. 3. Rijeka, Croatia: InTech, 2012.
[26] S. K. Mitra, *Digital Signal Processing: A Computer-Based Approach*. New York, NY, USA: McGraw-Hill, 2001.
[27] P. Vaidyanathan, *Multirate Systems and Filter Banks*. London, U.K.: Pearson, 2006.
[28] M. Bellanger, G. Bonnerot, and M. Coudreuse, "Digital filtering by polyphase network: Application to sample-rate alteration and filter banks," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. ASSP-24, no. 2, pp. 109–114, Apr. 1976.
[29] P. P. Vaidyanathan, "Multirate digital filters, filter banks, polyphase networks, and applications: A tutorial," *Proc. IEEE*, vol. 78, no. 1, pp. 56–93, 1990.
[30] M. Fingeroff, *High-Level Synthesis: Blue Book*. Bloomington, IN, USA: Xlibris Corporation, 2010.
[31] R. Teymourzadeh and M. Othman, "VLSI implementation of cascaded integrator comb filters for DSP applications," 2018, *arXiv:1808.09369*.
[32] E. Hogenauer, "An economical class of digital filters for decimation and interpolation," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. ASSP-29, no. 2, pp. 155–162, Apr. 1981.
[33] R. G. Lyons, *Understanding Digital Signal Processing, 3/E*. London, U.K.: Pearson Education India, 1997.
[34] A. V. Oppenheim, *Discrete-Time Signal Processing*. London, U.K.: Pearson Education India, 1999.
[35] T. N. Davidson, "Enriching the art of FIR filter design via convex optimization," *IEEE Signal Process. Mag.*, vol. 27, no. 3, pp. 89–101, May 2010.
[36] D. Datta and H. S. Dutta, "High efficient polyphase digital down converter on FPGA," *Circuits, Syst., Signal Process.*, vol. 40, no. 11, pp. 5787–5798, Nov. 2021.
[37] F. Haddad, O. Frioui, W. Rahajandraibe, L. Zaid, and R. Bouchakour, "Design of radio frequency polyphase filter and its application in low-IF receivers for large image rejection," in *Proc. Int. Conf. Design Technol. Integr. Syst. Nanosc. Era*, 2007, pp. 142–147.
[38] D. I. Kaplun, D. M. Klionskiy, A. S. Voznesenskiy, and V. V. Gulvanskiy, "Application of polyphase filter banks to wideband monitoring tasks," in *Proc. IEEE NW Russia Young Researchers Electr. Electron. Eng. Conf.*, Feb. 2014, pp. 95–98.
[39] B.-B. Cheng, J. Xu, and H. Zhang, "The application of poly-phase filter in rader signal processing," in *Proc. Int. Conf. Comput. Intell. Softw. Eng.*, Sep. 2010, pp. 1–4.
[40] F. J. Harris, *Multirate Signal Processing for Communication Systems*. Copenhagen, Denmark: River Publishers, 2022.
[41] S. Aggarwal and P. K. Meher, "Enhanced sharpening of CIC decimation filters, implementation and applications," *Circuits, Syst., Signal Process.*, vol. 41, no. 8, pp. 4581–4603, Aug. 2022.
[42] B. P. Stošić and V. D. Pavlović, "Design of new selective CIC filter functions with passband-droop compensation," *Electron. Lett.*, vol. 52, no. 2, pp. 115–117, Jan. 2016.
[43] D. E. T. Romero and M. G. C. Jimenez, "Efficient wide-band droop compensation for CIC filters: Ad hoc and reconfigurable FIR architectures," *Electron. Lett.*, vol. 53, no. 4, pp. 228–229, Feb. 2017.
[44] G. Stephen and R. W. Stewart, "High-speed sharpening of decimating CIC filter," *Electron. Lett.*, vol. 40, no. 21, pp. 1383–1384, Oct. 2004.
[45] P. M. Okiljevic, I. P. Pokrajac, and D. Jelusic, "Optimizing CIC and FIR filter's coefficients in GC4016 DDC chain," in *Proc. 20th Telecommun. Forum (TELFOR)*, Nov. 2012, pp. 756–759.
[46] Y. Jang, D.-H. Kim, and W.-S. Lee, "Efficient digital frequency down converter structure using CIC filters and interpolated fourth-order polynomials," in *Proc. Int. Conf. Secur. Cryptogr.*, vol. 2. SCITEPRESS, 2007, pp. 161–164.
[47] G. Jovanovic-Dolecek, "Efficient technique for improving the frequency response of CIC decimation filter," in *Proc. IEEE Int. Symp. Commun. Inf. Technol. (ISCIT)*, vol. 1, Oct. 2005, pp. 291–294.

SHEHRYAR AKBAR received the M.S. degree in electronic engineering from the Politecnico di Torino, Italy, in 2024, with a focus on embedded systems.

He carried out his thesis research work with the Digital Design Group, Infineon Technologies, Austria, where he worked on ASIC technologies. His research interests include high-level synthesis, HW/SW co-design, and low-power edge data processing.

LUCIANO LAVAGNO (Senior Member, IEEE) received the Ph.D. degree in electrical engineering and computer science from UC Berkeley, in 1992.

He was an Architect with the POLIS HW/SW co-design tool. From 2003 to 2014, he was an Architect with the Cadence CtoSilicon high-level synthesis tool. Since 1993, he has been a Professor with the Politecnico di Torino, Italy. He has co-authored four books and over 200 scientific articles. His research interests include synthesis of asynchronous circuits, HW/SW co-designs, high-level synthesis, and design tools for wireless sensor networks.

MIHAI T. LAZARESCU (Senior Member, IEEE) received the Ph.D. degree in electronics and communications from the Politecnico di Torino, Italy, in 1998.

He was a Senior Engineer at Cadence Design Systems and founded several startups. He is currently an Assistant Professor with the Politecnico di Torino. He has co-authored over 60 scientific publications, four books, and international patents. His research interests include design tools for WSN/IoT platforms, ubiquitous environmental sensing, efficient neural networks, indoor human localization, edge and leaf IoT data processing, and high-level HW/SW co-design and synthesis.

DAVIDE MARIZ received the B.S. degree in electronics and telecommunication engineering from the University of Trento, Italy, in 2015, and the M.S. degree in electronics engineering from the University of Padova, Italy, in 2018.

Since 2017, he has been at Infineon Technologies, first as a master's thesis Student and afterward as a Digital Design Engineer. His research interest includes the optimization of digital design processes in the arena of mixed-signal IPs.