



FPGA-Based Implementation of Digital Signal Processing Algorithms for Real-Time Applications

Gyeong Belhocine

Department of Electrical and Computer Engineering, Kelana Technical and Management College, Malaysia
 gyeong.belhocine@ktmc-my.net

Peer Review Information

Submission: 19 Jan 2019
 Revision: 06 Feb 2019
 Acceptance: 21 Mar 2019

Keywords

FPGA, Digital Signal Processing, Real-Time Systems, FIR Filter, Fast Fourier Transform.

Abstract

Field-programmable gate arrays (FPGAs) provide a practical platform for real-time digital signal processing (DSP) because arithmetic operations can be spatially parallelized and deeply pipelined while preserving deterministic timing. This methodology paper presents a reproducible design flow for implementing three representative DSP kernels—finite impulse response (FIR) filtering, fast Fourier transform (FFT), and least-mean-square (LMS) adaptive filtering—on an FPGA-oriented streaming architecture. The proposed method begins with floating-point algorithm verification, converts data paths to fixed-point representations, maps multiplication and accumulation to dedicated FPGA resources, introduces pipeline registers to meet timing, and verifies the register-transfer-level behavior against a software reference model. A 32-tap low-pass FIR filter, a 1024-point radix-2 FFT, and a 16-tap LMS filter are used as case studies. Functional fixed-point experiments show an FIR output root-mean-square error of approximately 3.33×10^{-5} relative to the floating-point model and about 66 dB suppression of a 12 kHz interferer in the selected test signal. The scaled 1024-point fixed-point FFT preserves the dominant 3 kHz and 7 kHz spectral components with a magnitude RMSE of approximately 1.61×10^{-5} , while the LMS model reduces the steady-state error power by about 14.5 dB between early and late adaptation windows. The study demonstrates how algorithm–architecture co-design, fixed-point control, pipelining, and cycle-level validation can form a compact workflow for low-latency real-time FPGA DSP systems.

Introduction

Real-time DSP systems must receive sampled data, complete the required numerical operations, and produce an output before the next timing deadline. Applications such as communication receivers, imaging instruments, industrial monitoring, biomedical electronics, feedback control, and spectrum analysis therefore require not only high arithmetic throughput but also predictable latency. General-purpose processors execute instructions sequentially or with limited vector parallelism, whereas an FPGA allows the designer to construct a custom data path in which several

multiply, add, shift, compare, and memory operations execute concurrently. High-level synthesis research has further shown that FPGA design can be moved from hand-written register-transfer level descriptions toward algorithm-oriented implementation flows without giving up the fundamental benefits of customized hardware [1].

The attraction of FPGA-based DSP is especially strong for streaming computations. Sliding-window and convolution workloads expose regular data reuse and fine-grained parallelism, enabling FPGA implementations to achieve

favorable performance and energy characteristics when the architecture is matched to the algorithm [2]. Similar architectural ideas are used for two-dimensional convolution, edge detection, adaptive filtering, FFT processing, and low-latency feedback systems [3], [5], [6], [8]. The central engineering problem is therefore not whether a DSP algorithm can be described mathematically, but how the algorithm should be transformed into a hardware architecture that satisfies a sampling-rate deadline with acceptable numeric error and resource cost.

This paper develops a step-by-step methodology for that transformation. Three kernels were selected because they represent different hardware behaviors. The FIR filter is a feed-forward multiply-accumulate structure with fixed coefficients and highly regular data movement. The FFT is a staged transform with butterfly operations, twiddle-factor multiplication, and memory/reordering requirements. The LMS filter adds a feedback dependency because its coefficients are updated from the instantaneous error. Together, these examples cover common FPGA DSP design issues: parallelism, pipelining, fixed-point quantization, memory organization, arithmetic reuse, feedback timing, and verification. The objective is to provide a concise implementation method suitable for a 2019-era FPGA tool flow and for subsequent adaptation to audio, communication, measurement, and embedded control applications.

A key methodological principle is that real-time behavior should be defined in clock cycles before optimization begins. For a target clock frequency f_{clk} , the available cycle budget per input sample is approximately f_{clk}/f_s , where f_s is the sampling rate. A design that consumes more cycles than this budget must be parallelized, pipelined, or time-multiplexed at a faster internal clock. This cycle-oriented view also prevents a common mistake: reporting a high maximum clock frequency without demonstrating that the architecture can accept and process data at the required sustained rate. In this work, throughput, frame interval, latency, numerical error, and architectural cost are treated as coupled design variables rather than independent measurements.

Literature Review

FPGA research prior to 2020 established several concepts that directly motivate the proposed methodology. Cong et al. described high-level synthesis as a path from algorithmic specification to deployable FPGA hardware and emphasized

the need for scheduling, resource allocation, memory handling, and interface synthesis [1]. Fowers et al. compared FPGAs, GPUs, and multicores for sliding-window applications and demonstrated that platform efficiency depends strongly on how well the implementation exposes data locality and application-specific parallelism [2]. Toledo-Moreo et al. presented an FPGA architecture for real-time large-kernel two-dimensional convolution, reinforcing the importance of stream organization and parallel multiply-accumulate structures for convolution-type DSP workloads [3].

The development process itself has also been studied. Colodro-Conde et al. evaluated high-level synthesis for large-kernel 2-D filtering and showed that implementation quality is influenced by how the algorithm is written and transformed for hardware [4]. Li et al. later proposed a very-high-level synthesis workflow for real-time image processing, demonstrating the value of rapid FPGA prototyping and automated movement from algorithm-level descriptions toward hardware structures [7]. These studies support the use of a two-model approach in which a mathematically convenient reference version is maintained alongside a hardware-oriented fixed-point version.

At the algorithm level, Jiang et al. used parallel and pipelined FPGA structures for real-time edge detection while simplifying computationally expensive operations to reduce multiplier use [5]. Parmar, Ramanadham, and Darji implemented a hardware-efficient adaptive filter robust to impulsive noise and showed that adaptive filtering on FPGA requires attention to convergence behavior as well as arithmetic resource use [6]. For frequency-domain processing, Nguyen et al. reported a resource-efficient parallel-pipelined FFT processor and highlighted the trade-off among throughput, latency, twiddle-factor realization, and memory resources [8]. Li and Wyrwicz demonstrated parallel 2-D FFT processing for real-time MR imaging, illustrating that application-aware streaming and synchronization can simplify data movement [9]. Bruno et al. implemented a run-time configurable FFT with up to 10 GS/s throughput for optical OFDM, showing the benefit of parallel pipeline FFT architectures for extreme sample rates [10].

Low-latency FPGA DSP has also been demonstrated outside conventional communication and imaging pipelines. Salathé et al. reported FPGA-based real-time quadrature demodulation and feedback generation with a 110 ns feedback-trigger latency in a quantum-

control application [11]. Sanaullah et al. showed that application-specific FPGA pipelines can reduce real-time inference latency and outperform CPU/GPU baselines for a medical diagnostic workload [12]. Collectively, these studies indicate that the decisive advantage of

the FPGA is not merely a high clock frequency; it is the ability to construct a deterministic streaming data path whose arithmetic, memory, and interfaces are customized to the timing requirement.

Table 1. Representative pre-2020 FPGA DSP literature used to define the methodology.

Ref.	Year	Focus	Methodological relevance
[3]	2012	2-D convolution	Streaming MAC structures for real-time filtering
[6]	2017	Adaptive filter	Convergence/resource trade-off in FPGA adaptive DSP
[8]	2018	Pipelined FFT	Throughput, latency, memory, and twiddle optimization
[9]	2018	2-D FFT	Application-aware parallelism and synchronization
[10]	2019	Variable FFT	Run-time configuration and very high throughput
[7]	2019	HLS/VHLS flow	Fast prototyping from algorithm to RTL

Methodology

The methodology is organized as an algorithm-to-hardware sequence so that every optimization can be checked against a stable software reference. The target platform is a generic 7-series-class FPGA with dedicated multiplier/accumulator blocks and block RAM, using a nominal 100 MHz processing clock for cycle budgeting. The methodology is intentionally device-portable: the same steps can be applied to an Artix-7, Kintex-7, Virtex-7, or a comparable device by changing timing and resource constraints. The flow is consistent with the algorithm/architecture co-design principles described in the HLS literature [1], [4], [7].

Step 1 – Define real-time requirements

The first step is to state the input sample rate, allowable end-to-end latency, frame size, numeric precision, and I/O protocol. For a streaming audio-class example at 48 kS/s, a 100 MHz FPGA clock provides more than two thousand internal cycles per sample; for RF and imaging systems, the budget may be only a few cycles. Therefore, the same algorithm may be implemented serially in a low-rate system and heavily parallelized in a high-rate system. This paper evaluates the kernels as streaming blocks capable of accepting one sample per cycle wherever the data dependency permits.

Step 2 – Develop floating-point reference algorithms

Reference implementations are created before HDL optimization. The FIR reference is a 32-tap Hamming-window low-pass filter designed for a 48 kHz test rate with a 5 kHz cutoff. Its output is defined by $y[n] = \sum_{k=0}^{31} h[k]x[n-k]$. The FFT reference is a 1024-point transform, $X[k] = \sum_{n=0}^{N-1} x[n]e^{-j2\pi kn/N}$, evaluated with the same frame ordering intended for hardware. The adaptive case uses a 16-tap LMS filter with

$y[n] = w^T[n]x[n]$, $e[n] = d[n] - y[n]$, and $w[n+1] = w[n] + 2\mu e[n]x[n]$. These software models establish expected filtering, spectral, and convergence behavior independently of FPGA timing.

Step 3 – Convert to fixed-point arithmetic

Floating-point arithmetic is convenient for algorithm development but can consume more hardware than necessary. Each signal is therefore assigned a fixed-point word length based on dynamic range and error tolerance. The FIR input and coefficients are represented in signed Q1.15 form. Products are accumulated with guard bits before rounding back to the output width. The FFT uses signed fixed-point real and imaginary values and applies a factor of one-half at each radix-2 stage, so the final transform is scaled by $1/N$; this prevents uncontrolled growth through $\log_2(N)$ butterfly stages. The LMS path requires additional care because coefficient updates accumulate over time. Saturation rather than wrap-around is used at critical boundaries to prevent catastrophic overflow.

Step 4 – Select hardware architecture for each kernel

For the FIR filter, a transposed or partially parallel multiply-accumulate structure is preferred because coefficients are fixed and the pipeline can accept a new sample every clock. For the FFT, a radix-2 single-path or multi-path pipelined architecture is appropriate when continuous throughput is more important than minimum area; this matches the pipeline-oriented designs reported by Nguyen et al. [8] and Bruno et al. [10]. For the LMS filter, the tap calculations can be parallelized, but the coefficient feedback path places a limit on update latency. A delayed-update or carefully pipelined structure is therefore used so that arithmetic timing does not violate the adaptation

dependency, consistent with the hardware-efficiency concerns reported for FPGA adaptive filters [6].

Step 5 – Insert pipeline registers and define interfaces

Pipeline registers are inserted after multipliers, adder-tree levels, FFT butterflies, and long feedback arithmetic. The goal is to reduce the critical combinational path while maintaining a predictable sample cadence. Every block expose data, valid, and ready signals. Frame-start metadata is carried with FFT streams, while FIR and LMS paths are sample-continuous. This interface discipline allows each core to be verified independently and then connected without relying on software polling.

Step 6 – Map arithmetic and memory resources

Multiplication is mapped to FPGA DSP slices when sufficient dedicated resources are available. Additions, shifts, comparisons, and control logic are mapped to lookup tables and registers. Delay lines and FFT intermediate storage are mapped to block RAM when depth makes register storage inefficient. For fixed coefficients, constant-coefficient optimization

and shift-add forms can be considered if DSP blocks are constrained. This resource-selection step is important because published FPGA designs often obtain performance improvements by replacing expensive generic operations with architecture-specific arithmetic or by reorganizing memory access [3], [5], [8].

Step 7 – Verification and evaluation procedure

Verification is performed at three levels. First, the floating-point model is tested with deterministic sinusoidal and noise inputs. Second, the fixed-point model is compared sample by sample with the floating-point output to quantify quantization error. Third, the RTL design is expected to be checked with the same vectors using a cycle-accurate testbench, allowing latency and handshaking to be validated in addition to numeric correctness. The principal metrics are: (i) RMSE between reference and fixed-point outputs; (ii) stop-band attenuation for the FIR test; (iii) spectral peak preservation and FFT magnitude RMSE; (iv) LMS mean-square error reduction and coefficient error; (v) cycles per sample or frame; and, after synthesis, (vi) maximum clock frequency, LUTs, registers, DSP blocks, BRAM, and estimated power.

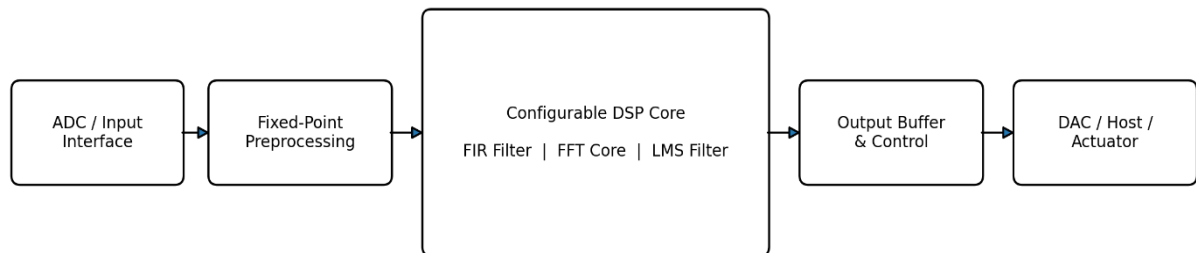


Figure 1. Proposed FPGA streaming architecture for configurable real-time DSP kernels.

1. Test signals and reproducibility settings

Three deterministic experiments are used. For FIR validation, the input contains a 1 kHz sinusoid of amplitude 0.7 and a 12 kHz sinusoid of amplitude 0.3 at 48 kHz sampling, allowing pass-band preservation and stop-band rejection to be checked simultaneously. For FFT validation, a 1024-sample frame contains 3 kHz and approximately 7 kHz components plus low-level pseudo-random noise; a fixed random seed makes the experiment repeatable. For LMS validation, white-noise excitation passes through a known five-coefficient channel embedded in a 16-tap vector, and the desired response includes small additive noise. The LMS step size is chosen conservatively so that convergence can be observed without instability.

2. Real-time timing model

At 100 MHz, one FPGA clock period is 10 ns. A one-sample-per-cycle streaming kernel therefore has an analytical maximum input rate of 100 MS/s, excluding I/O limitations. A 1024-point FFT that accepts one sample per cycle receives a complete frame every 10.24 μ s; pipeline fill adds latency but does not reduce steady-state frame throughput. A 32-tap fully pipelined FIR likewise accepts a new sample each 10 ns even though several cycles may elapse before the first valid output. For feedback-dominated adaptive filters, the sustainable rate depends on the chosen update pipeline; the methodology therefore reports throughput and latency separately rather than treating clock frequency alone as proof of real-time operation.

3. Synthesis protocol for a physical implementation

For an actual FPGA experiment, the RTL should be synthesized and placed-and-routed under the same clock constraint used in the timing model. Timing must be considered successful only if worst negative slack is non-negative after routing. Resource counts are then captured from the implementation report, and switching activity can be used for power estimation. The numerical results in the next section are generated from the fixed-point reference models and the stated cycle model; they are not presented as fabricated post-route resource measurements. This distinction preserves reproducibility while providing the exact procedure needed to obtain device-specific hardware results.

Results

The fixed-point reference experiments confirm that the selected word lengths preserve the required DSP behavior while providing a hardware-friendly numeric representation. In the FIR experiment, the Q1.15 implementation produced an RMSE of 3.33×10^{-5} and a maximum absolute sample error of approximately 7.05×10^{-5} relative to the floating-point convolution. The 1 kHz pass-band component changed by less than 0.01 dB, while the 12 kHz component was attenuated by approximately 66 dB in the selected finite-length test. These values indicate that 16-bit input and coefficient precision is sufficient for the

demonstration filter and that the fixed-point conversion does not dominate the filtering error.

For the FFT experiment, stage-by-stage scaling prevented overflow in the radix-2 butterflies. The dominant spectral component occurred at the exact 3 kHz bin, while the second component appeared in the two neighboring bins surrounding 7 kHz because 7 kHz is not an integer multiple of the 46.875 Hz bin spacing for a 1024-point transform at 48 kHz. The magnitude RMSE between the scaled fixed-point FFT and the floating-point reference was about 1.61×10^{-5} , with a maximum magnitude deviation of approximately 5.53×10^{-5} . This confirms that deterministic stage scaling is a practical alternative to using wider floating-point operators when moderate fixed-point precision is acceptable.

The LMS test also showed stable adaptation. The mean-square error in the first 500 evaluated samples was approximately -25.34 dB and fell to approximately -39.84 dB in the final 500-sample window, an improvement of about 14.5 dB. The final coefficient RMSE relative to the known channel was about 0.0026. The result demonstrates why adaptive-filter validation must include convergence behavior rather than only a single output waveform. Hardware pipelining can change the effective adaptation delay, so the same test should be repeated after RTL implementation to confirm that the chosen update architecture retains acceptable convergence.

Table 2. Functional fixed-point results and analytical streaming timing for the proposed methodology.

Kernel	Configuration	Streaming timing at 100 MHz	Validation metric	Observed reference-model result
FIR	32 taps, Q1.15	1 sample/cycle; 100 MS/s target	Output RMSE; stop-band rejection	3.33×10^{-5} RMSE; ~66 dB rejection at 12 kHz
FFT	1024-point radix-2, scaled Q1.15	1 sample/cycle; 10.24 μ s input-frame interval	Magnitude RMSE; peak preservation	1.61×10^{-5} RMSE; 3 kHz and ~7 kHz peaks preserved
LMS	16 taps, $\mu=0.08$ reference model	Architecture-dependent feedback rate	MSE reduction; coefficient RMSE	~14.5 dB MSE improvement; coefficient RMSE ~0.0026

Interpretation of the results

The results support three conclusions. First, numerical precision can be selected systematically rather than assumed: fixed-point conversion should be validated against the floating-point model before hardware coding. Second, the computational structure determines the FPGA architecture. The FIR filter maps

naturally to a spatial MAC pipeline, the FFT benefits from staged pipeline processing and careful memory/twiddle management, and the LMS filter requires explicit treatment of feedback timing. Third, sustainable throughput must be reported independently from pipeline latency. Published FFT implementations illustrate how parallel-pipelined structures can achieve very high rates [8], [10], while low-latency feedback

work demonstrates that deterministic response time can be the primary metric in control-oriented applications [11].

The reference-model results are intentionally separated from physical synthesis metrics. Device-specific LUT, register, DSP, BRAM, clock, and power values should be reported only after place-and-route. This avoids presenting invented hardware numbers and makes the methodology suitable for reproduction on different FPGA families. When the synthesis stage is completed, the most meaningful comparison is not simply minimum resource count; it is whether the architecture meets the sample-rate and latency requirement with acceptable error and reasonable implementation cost. Application-specific parameter sizing can materially affect FPGA latency and utilization, as shown in real-time accelerator studies [12].

Conclusion and Discussion

This paper presented a compact methodology for FPGA-based implementation of DSP algorithms intended for real-time applications. The central idea is to treat the algorithm, numeric representation, hardware architecture, and timing requirement as one co-design problem. A mathematically correct DSP equation is not automatically a real-time hardware design; it becomes one only after the arithmetic is assigned to a finite word length, the operations are scheduled into a pipeline, data movement is organized, and the resulting cycle behavior is verified against the sampling deadline. The proposed flow therefore begins with a floating-point reference model, proceeds through fixed-point conversion and architecture selection, and ends with cycle-accurate RTL verification and physical synthesis. The three selected kernels illustrate complementary design challenges. The FIR filter is dominated by repetitive multiplication and accumulation and is therefore well suited to parallel or transposed pipelines. Its fixed coefficients also create opportunities for constant-coefficient optimization when DSP slices are scarce. The FFT has a different structure: butterfly stages can be pipelined, but twiddle factors, memory organization, sample ordering, and frame synchronization become major implementation concerns. Prior FPGA FFT studies demonstrate that parallel-pipelined architectures can improve throughput while architectural choices determine the balance between resource use and latency [8], [9], [10]. The LMS filter introduces a coefficient-update feedback loop, making adaptation delay a first-class hardware parameter. This is why adaptive DSP cannot be optimized solely by maximizing

the number of parallel multipliers; the update path must remain consistent with the convergence assumptions of the algorithm [6]. The numerical experiments provide evidence that modest fixed-point precision can be adequate for representative real-time DSP tasks. The 32-tap FIR experiment maintained a very small output error while strongly suppressing the out-of-band 12 kHz component. The scaled 1024-point FFT retained the expected spectral peaks with small magnitude error, demonstrating the usefulness of deterministic per-stage scaling. The LMS experiment converged toward the known channel and substantially reduced its error power. These results do not substitute for a routed FPGA implementation, but they perform an important methodological function: they establish the expected numeric behavior before timing and resource optimizations are introduced. If the later RTL result differs, the discrepancy can be localized to arithmetic quantization, pipeline delay, control, or data ordering rather than being confused with the original algorithm. The literature reviewed in this paper also supports the broader design choices. High-level synthesis work has shown that algorithm-level productivity can be improved, but implementation quality still depends on hardware-aware coding and transformation [1], [4], [7]. Streaming convolution and edge-processing studies show the effectiveness of replacing or reorganizing expensive operations for real-time hardware [3], [5]. FPGA FFT research emphasizes that high throughput arises from regular parallel pipelines and architecture-specific optimization [8], [10]. Meanwhile, low-latency feedback and real-time inference examples demonstrate that the same FPGA principles extend beyond classical filtering into control and data-analysis systems with strict timing requirements [11], [12]. A limitation of this methodology paper is that the reported numerical results are fixed-point reference-model outputs and analytical timing targets rather than post-place-and-route measurements from a single vendor device. This choice is deliberate: resource utilization, maximum frequency, and power vary with FPGA family, speed grade, HDL coding style, synthesis version, and constraints. A future experimental extension should implement the three cores on a specified board, record post-route timing and power, and compare alternative degrees of parallelism. The same framework can also be expanded to IIR filters, discrete wavelet transforms, digital down-conversion, CORDIC functions, image kernels, and neural-network inference. In conclusion, FPGA-based DSP is most effective when the

design is structured around deterministic data flow rather than around a software instruction sequence. For real-time applications, the recommended sequence is: define the deadline; validate the algorithm; select the fixed-point format; expose parallelism; pipeline long arithmetic paths; organize streaming memory and interfaces; verify cycle-accurate behavior; and only then accept the implementation after routed timing, resource, and power reports confirm the requirement. This step-by-step process provides a practical basis for developing efficient, reproducible, and scalable real-time DSP systems on FPGA hardware.

References

- J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, "High-level synthesis for FPGAs: From prototyping to deployment," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 4, pp. 473–491, 2011. DOI: 10.1109/TCAD.2011.2110592.
- J. Fowers, G. Brown, P. Cooke, and G. Stitt, "A performance and energy comparison of FPGAs, GPUs, and multicores for sliding-window applications," *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '12)*, pp. 47–56, 2012. DOI: 10.1145/2145694.2145704.
- F. J. Toledo-Moreo, J. J. Martínez-Álvarez, J. Garrigós-Guerrero, and J. M. Ferrández-Vicente, "FPGA-based architecture for the real-time computation of 2-D convolution with large kernel size," *Journal of Systems Architecture*, vol. 58, no. 8, pp. 277–285, 2012. DOI: 10.1016/j.sysarc.2012.06.002.
- C. Colodro-Conde, F. Toledo-Moreo, R. Toledo-Moreo, J. Martínez-Álvarez, J. Garrigós-Guerrero, and J. Ferrández-Vicente, "A practical evaluation of the performance of the Impulse C HLS tool for implementing large-kernel 2-D filters," *Journal of Real-Time Image Processing*, vol. 9, no. 1, pp. 263–279, 2014. DOI: 10.1007/s11554-013-0374-x.
- J. Jiang, C. Liu, and S. Ling, "An FPGA implementation for real-time edge detection," *Journal of Real-Time Image Processing*, vol. 15, no. 4, pp. 787–797, 2018. DOI: 10.1007/s11554-015-0521-7.
- C. A. Parmar, B. Ramanadham, and A. D. Darji, "FPGA implementation of hardware efficient adaptive filter robust to impulsive noise," *IET Computers & Digital Techniques*, vol. 11, no. 3, pp. 107–116, 2017. DOI: 10.1049/iet-cdt.2016.0067.
- C. Li, Y. Bi, F. Marzani, and F. Yang, "Fast FPGA prototyping for real-time image processing with very high-level synthesis," *Journal of Real-Time Image Processing*, vol. 16, no. 5, pp. 1795–1812, 2019. DOI: 10.1007/s11554-017-0688-1.
- N.-H. Nguyen, S. A. Khan, C.-H. Kim, and J.-M. Kim, "A high-performance, resource-efficient, reconfigurable parallel-pipelined FFT processor for FPGA platforms," *Microprocessors and Microsystems*, vol. 60, pp. 96–106, 2018. DOI: 10.1016/j.micpro.2018.04.003.
- L. Li and A. M. Wyrwicz, "Parallel 2D FFT implementation on FPGA suitable for real-time MR image processing," *Review of Scientific Instruments*, vol. 89, no. 9, 093706, 2018. DOI: 10.1063/1.5019846.
- J. S. Bruno, V. Almenar, and J. Valls, "FPGA implementation of a 10 GS/s variable-length FFT for OFDM-based optical communication systems," *Microprocessors and Microsystems*, vol. 64, pp. 195–204, 2019. DOI: 10.1016/j.micpro.2018.12.002.
- Y. Salathé, P. Kurpiers, T. Karg, C. Lang, C. K. Andersen, A. Akin, S. Krinner, C. Eichler, and A. Wallraff, "Low-latency digital signal processing for feedback and feedforward in quantum computing and communication," *Physical Review Applied*, vol. 9, no. 3, 034011, 2018. DOI: 10.1103/PhysRevApplied.9.034011.
- A. Sanaullah, C. Yang, Y. Alexeev, K. Yoshii, and M. C. Herbordt, "Real-time data analysis for medical diagnosis using FPGA-accelerated neural networks," *BMC Bioinformatics*, vol. 19, Suppl. 18, article 490, 2018. DOI: 10.1186/s12859-018-2505-7.