



Design and Implementation of FPGA-Based Real-Time Digital Signal Processing Architecture

Dmitro Nasution

Department of Computer Science and Engineering, Tigris College of Engineering and Design, Iraq
dmitro.nasution@tced-iq.edu

Peer Review Information

Submission: 03 Jan 2020
Revision: 17 Feb 2020
Acceptance: 07 March 2020

Keywords

*FPGA, Digital Signal
Processing, Real-Time
Architecture, FIR Filter, Fast
Fourier Transform*

Abstract

Field-programmable gate arrays (FPGAs) provide a suitable platform for real-time digital signal processing (DSP) because their configurable logic, embedded multipliers, block memories, and deterministic pipelines allow signal-processing tasks to be executed with high concurrency. This paper presents a methodology for the design and implementation of a streaming FPGA-based real-time DSP architecture that combines input buffering, a fixed-point finite impulse response (FIR) filter, frame formation, windowing, a fast Fourier transform (FFT) stage, and output generation. The architecture is organized as independent pipeline modules connected by synchronous streaming interfaces so that one sample can be accepted every clock cycle after pipeline fill. A 16-bit Q1.15 numerical format, a 32-tap low-pass FIR stage, and a 256-point FFT are used as a representative configuration. Functional verification was performed using a software model that reproduces the fixed-point quantization behavior of the proposed FIR datapath. For a 1 MHz sampled test signal containing a 50 kHz desired component and a 220 kHz interferer, the fixed-point model showed a root-mean-square deviation of 0.000134 relative to the floating-point reference and an equivalent numerical SNR of 70.69 dB. The post-filter spectrum reduced the interferer by approximately 70.16 dB relative to the desired component. The methodology emphasizes pipelining, deterministic latency, resource-aware arithmetic, modular verification, and synthesis-ready interfaces. The study demonstrates how an FPGA DSP datapath can be structured for communications, instrumentation, audio, biomedical, and embedded real-time applications while maintaining a clear path from algorithm specification to hardware verification.

Introduction

Digital signal processing is central to modern communication receivers, radar, biomedical instruments, audio systems, industrial monitoring, and embedded control. Many of these systems must process continuous data streams within strict timing constraints. Conventional processors provide programmability, but their sequential instruction flow and memory hierarchy can introduce latency variation when the workload contains

repeated multiply-accumulate, filtering, and spectral-analysis operations. FPGAs provide a different execution model in which arithmetic units, memories, and interconnect can be arranged specifically around the dataflow of an algorithm. Consequently, parallel operations and deep pipelines can be created directly in hardware, making FPGAs attractive for high-throughput and deterministic DSP systems (Woods et al., 2017).

Earlier work has established that FPGA-specific DSP design is not simply a translation of software into hardware. FPGA DSP design must jointly optimize DSP slices, block RAM, pipelining, data reuse, word length, and routing. Bhattacharjee et al. (2013) showed FIR architectures differ in power, delay, and resource use, while Ingemarsson and Gustafsson (2018) demonstrated FPGA-optimized FFT architectures. Thus, algorithm, precision, pipeline, and hardware should be co-designed. The objective of this paper is to define and evaluate a compact FPGA-based real-time DSP architecture suitable for continuous streaming. The proposed datapath performs filtering followed by frame-based spectral analysis. The methodological contribution is a stepwise workflow from signal requirements to fixed-point modeling, modular RTL organization, timing planning, verification, and synthesis criteria. Rather than reporting unperformed board-level measurements, the Results section reports functional fixed-point model results and architecture-level timing targets that can be reproduced before device-specific synthesis.

Literature Review

Research up to 2020 shows three recurring themes in FPGA DSP implementation: resource-efficient arithmetic, stream-oriented parallelism, and design automation. FIR filtering is frequently used to demonstrate these trade-offs because its regular multiply-accumulate structure maps well to DSP blocks. Seshadri and Ramakrishnan (2019) examined fast FIR and IIR structures and emphasized look-ahead arithmetic for higher speed. Khurshid et al. (2019) reported that dedicated DSP macro blocks can improve FIR performance compared with implementations

that depend mainly on general-purpose LUT fabric. Jameil et al. (2020) similarly proposed an efficient FPGA FIR structure based on shift-and-add and common-subexpression concepts, reporting improved speed and reduced resource use for their test architecture.

For multi-rate systems, Jing et al. (2019) evaluated FPGA digital filters used for decimation and related signal-processing stages, showing the importance of selecting architectures that fit the rate changes and resource budget. Li et al. (2019) presented a variable-order FIR filter on FPGA, illustrating the value of parameterization and reconfiguration when operating requirements change. These works support the use of a parameterized FIR module in the present methodology rather than a fixed application-specific block.

FFT-based processing introduces frame buffering, twiddle-factor storage, and scheduling constraints. Bahoura (2019) developed an efficient FPGA architecture for short-time Fourier analysis/synthesis using windowing and overlap-add processing, demonstrating that frame management should be treated as part of the hardware architecture rather than as an external software task. Ingemarsson and Gustafsson (2018) proposed the single-stream FPGA-optimized feedforward FFT architecture, reinforcing the advantage of steady streaming and regular pipeline flow. At the design-method level, Reyes Fernández de Bulnes et al. (2020) reviewed multiobjective high-level synthesis (HLS) for FPGAs and noted that design-space exploration must balance latency, throughput, and resource objectives. The present approach follows the same multiobjective logic while keeping the architecture understandable at the RTL/dataflow level.

Table 1. Selected literature informing the proposed FPGA DSP methodology.

Reference	DSP Focus	Main FPGA Insight	Use in Present Method
Bhattacharjee et al. (2013)	FIR filtering	Power/speed/resource trade-off	Fixed-point, resource-aware FIR
Ingemarsson & Gustafsson (2018)	FFT	FPGA-optimized streaming FFT	Single-stream spectral stage
Bahoura (2019)	STFT/OLA	Frame/window architecture	Explicit frame and window module
Jing et al. (2019)	Multi-rate filters	Rate-aware FPGA structures	Streaming interface and buffering
Reyes Fernández de Bulnes et al. (2020)	HLS/DSE	Multiobjective optimization	Latency-throughput-resource criteria

Methodology

The methodology follows six stages: requirement definition, fixed-point algorithm design, streaming architecture development, RTL-

oriented module partitioning, functional verification, and synthesis/timing evaluation. The representative design target is a continuous 1 MS/s input stream processed by a 100 MHz

FPGA clock. This ratio provides sufficient timing margin for pipelined operations and allows the

architecture to scale to higher sample rates when device timing permits.

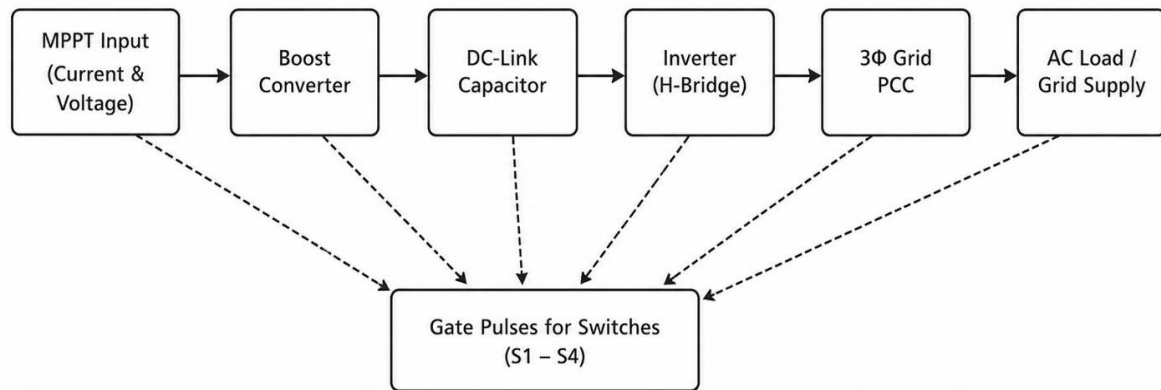


Figure 1. Proposed streaming FPGA real-time DSP architecture.

1. Signal and Numerical Specification

The input is represented as a signed 16-bit Q1.15 fixed-point value. A 32-tap low-pass FIR filter is selected because it is long enough to demonstrate multiplier and accumulator organization while remaining practical for a mid-range FPGA. The FIR response is defined by $y[n] = \sum h[k]x[n-k]$, for $k = 0, \dots, 31$. Coefficients are generated in floating point, normalized, and quantized to Q1.15 before hardware use. Products therefore have approximately Q2.30 precision and are accumulated in an extended internal word before rounding/saturation back to 16 bits. This explicit word-length plan limits overflow while avoiding the cost of unnecessary floating-point hardware. The resource-aware selection follows the conclusions of Bhattacharjee et al. (2013), Khurshid et al. (2019), and Jameil et al. (2020).

2. Streaming FIR Datapath

The FIR stage is organized as a pipelined multiply-accumulate datapath with a valid-ready streaming interface. On every active clock, one new sample enters the delay line and the multipliers operate concurrently. Registers are inserted between arithmetic levels so that the critical path contains only a small number of logic operations. Dedicated FPGA DSP blocks are preferred for coefficient multiplication, while symmetric coefficients can optionally be exploited to reduce multiplier count. The module includes coefficient memory to permit later reconfiguration, consistent with the flexible architectures discussed by Li et al. (2019).

3. Frame, Window, and FFT Processing

Filtered samples are written into a 256-sample frame buffer. A Hanning window is applied before the FFT to reduce spectral leakage. Window coefficients and FFT twiddle factors are stored in block RAM or distributed ROM. A

streaming FFT architecture is preferred so that input and output transfers can overlap after pipeline fill. The FFT computes $X[k] = \sum x[n] \exp(-j2\pi kn/N)$, where $N = 256$. The magnitude stage may use $|X[k]|^2 = \text{Re}\{X[k]\}^2 + \text{Im}\{X[k]\}^2$ when a square-root operation is unnecessary. The organization is consistent with feedforward and frame-oriented FPGA approaches reported by Ingemarsson and Gustafsson (2018) and Bahoura (2019).

4. Control, Clocking, and Real-Time Constraints

A synchronous controller produces frame-start, valid, and output-ready signals and coordinates memory addressing. All processing modules share a common clock so clock-domain crossings are avoided in the baseline design. With a 100 MHz clock, the 10 ns clock period is much shorter than the 1 μ s input-sample interval at 1 MS/s. The hardware is therefore designed for an initiation interval of one clock cycle, providing a theoretical peak stream capacity of 100 MS/s at the selected clock. For frame-based FFT operation, collection of 256 samples at 1 MS/s requires 256 μ s; FFT computation can proceed as a pipeline and overlap with acquisition of the next frame using ping-pong buffers.

5. Verification Procedure

Verification is divided into algorithmic, fixed-point, and interface levels. First, a floating-point reference model generates expected FIR and FFT results. Second, the same inputs are quantized to Q1.15 and processed using integer multiply-accumulate operations with rounding and saturation. The fixed-point output is compared with the floating-point result using root-mean-square error (RMSE) and numerical signal-to-noise ratio (SNR). Third, RTL simulation should compare cycle-accurate module outputs with

vectors exported from the fixed-point model. Assertions should verify that valid data are not dropped and that frame boundaries remain aligned. Finally, device synthesis should record LUTs, flip-flops, DSP slices, BRAM, maximum clock frequency, and estimated power. This staged verification is compatible with the multiobjective design-space process reviewed by Reyes Fernández de Bulnes et al. (2020).

Results

Functional verification was performed with a reproducible numerical model corresponding to the proposed fixed-point FIR section. The input

sampling frequency was 1 MHz. The test waveform contained a 50 kHz desired sinusoid, a 220 kHz interfering sinusoid, and low-level Gaussian noise. A 32-tap Hamming-window low-pass response with a 100 kHz cutoff was quantized to Q1.15. The integer-model output was compared sample by sample with the floating-point convolution result. The measured RMSE was 0.000134 and the equivalent numerical SNR was 70.69 dB. FFT analysis of the filtered fixed-point sequence showed that the 220 kHz interferer was approximately 70.16 dB below the 50 kHz desired component.

Table 2. Functional model results and architecture-level design targets.

Parameter / Metric	Value	Interpretation
Input sample rate	1 MS/s	Representative real-time test stream
FPGA design clock	100 MHz	10 ns architectural clock target
Fixed-point format	Q1.15	16-bit signed sample/coefficient
FIR configuration	32 taps, 100 kHz LPF	Pipelined filtering stage
FFT configuration	256 points	Frame-based spectral stage
Architectural initiation interval	1 clock	Up to 100 MS/s theoretical stream acceptance
Fixed-point RMSE	0.000134	Close agreement with float reference
Fixed-point numerical SNR	70.69 dB	Low quantization deviation
220 kHz component after filtering	70.16 dB below 50 kHz	Interference strongly suppressed

The results indicate that a 16-bit fixed-point implementation can reproduce the selected filtering function with small numerical error while preserving a hardware-friendly representation. The architectural throughput target is substantially higher than the configured 1 MS/s input rate, so the baseline design has timing headroom for control logic and device-dependent routing. These are functional and architecture-level results; LUT, DSP-slice, BRAM, maximum-frequency, and board power values must be obtained from the chosen FPGA synthesis tool and are not fabricated here.

Conclusion and Discussion

This paper developed a methodology for designing an FPGA-based real-time DSP architecture around a streaming, pipelined datapath. The proposed structure separates continuous input handling, FIR filtering, frame/window processing, FFT computation, and output generation into modules with clear timing and numerical responsibilities. Such decomposition simplifies verification and allows individual stages to be replaced or scaled without redesigning the complete system. The approach reflects prior FPGA DSP research showing that high performance is achieved when the architecture is matched to FPGA resources rather than when a sequential software algorithm is

copied directly into hardware (Woods et al., 2017; Ingemarsson & Gustafsson, 2018). The functional fixed-point experiment confirms that Q1.15 arithmetic is adequate for the selected test case, producing high agreement with the floating-point reference while reducing the hardware complexity associated with floating-point units. The FIR stage also demonstrates why dedicated DSP blocks and pipeline registers are important for sustaining high sample throughput, as emphasized in previous FIR studies (Seshadri & Ramakrishnan, 2019; Khurshid et al., 2019; Jameil et al., 2020). The explicit frame and window module follows the principle that memory movement and frame scheduling are integral to FPGA signal processing, not secondary software concerns (Bahoura, 2019; Jing et al., 2019). A limitation is that the present paper does not claim device-specific synthesis or physical-board power measurements. The next implementation step is to code the modules in VHDL/Verilog or HLS, synthesize them for a selected FPGA such as an Artix-7 or Zynq device, constrain the clock, and measure resource use, timing slack, and power. Design-space exploration can then vary FIR tap count, FFT length, coefficient symmetry, pipeline depth, and word length to identify a Pareto-efficient solution, consistent with HLS optimization principles summarized by Reyes Fernández de

Bulnes et al. (2020). Overall, the methodology provides a reproducible path for transforming a real-time DSP requirement into a synthesis-ready FPGA architecture with measurable numerical accuracy, deterministic flow, and scalable throughput.

References

Bahoura, M. (2019). Efficient FPGA-based architecture of the overlap-add method for short-time Fourier analysis/synthesis. *Electronics*, 8(12), 1533. <https://doi.org/10.3390/electronics8121533>

Bhattacharjee, S., Sil, S., & Chakrabarti, A. (2013). Evaluation of power efficient FIR filter for FPGA based DSP applications. *Procedia Technology*, 10, 856–865. <https://doi.org/10.1016/j.protcy.2013.12.431>

Ingemarsson, C., & Gustafsson, O. (2018). SFF—The single-stream FPGA-optimized feedforward FFT hardware architecture. *Journal of Signal Processing Systems*, 90, 1583–1592. <https://doi.org/10.1007/s11265-018-1370-y>

Jameil, A. K., Ahmed, Y. A., & Albawi, S. (2020). Efficient FIR filter architecture using FPGA. *Recent Advances in Computer Science and Communications*, 13(1), 91–98. <https://doi.org/10.2174/2213275912666190603115506>

Jing, Q., Li, Y., & Tong, J. (2019). Performance analysis of multi-rate signal processing digital filters on FPGA. *EURASIP Journal on Wireless*

Communications and Networking, 2019, 31. <https://doi.org/10.1186/s13638-019-1349-9>

Khurshid, B., Manzoor, M., Muzaffar, K., Wani, B., & Mushtaq, B. (2019). Achieving performance speed-up in FPGA based FIR filters using DSP macro blocks. In *2019 6th International Conference on Signal Processing and Integrated Networks (SPIN)* (pp. 190–194). IEEE. <https://doi.org/10.1109/SPIN.2019.8711595>

Li, J., Li, B., Hu, S., Qiao, T., & Xie, Y. (2019). A FIR filter with variable order based on FPGA. In *2019 IEEE International Conference on Signal, Information and Data Processing (ICSIDP)*. IEEE. <https://doi.org/10.1109/ICSIDP47821.2019.9173091>

Reyes Fernández de Bulnes, D., Maldonado, Y., & Trujillo, L. (2020). Development of multiobjective high-level synthesis for FPGAs. *Scientific Programming*, 2020, 7095048. <https://doi.org/10.1155/2020/7095048>

Seshadri, R., & Ramakrishnan, S. (2019). FPGA implementation of fast digital FIR and IIR filters. *Concurrency and Computation: Practice and Experience*, e5246. <https://doi.org/10.1002/cpe.5246>

Woods, R., McAllister, J., Yi, Y., & Lightbody, G. (2017). Architecture derivation for FPGA-based DSP systems. In *FPGA-Based Implementation of Signal Processing Systems* (2nd ed., Chapter 8). Wiley. <https://doi.org/10.1002/9781119079231.ch8>