



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

Preben Ben-Mizrahi

Department of Computer Science and Engineering, Tonle Sap Institute of Engineering and Commerce, Cambodia

preben.ben.mizrahi@tsiec-kh.org

Peer Review Information

Submission: 05 Jan 2017

Revision: 01 Feb 2017

Acceptance: 26 Feb 2017

Keywords

Field-Programmable Gate Array, Digital Signal Processing, real-time processing, Pipelining, Fixed-Point Arithmetic

Abstract

Field-programmable gate arrays (FPGAs) are widely used for computationally intensive digital signal processing (DSP) because their fine-grained parallelism, dedicated multiplier-accumulator resources, on-chip memories, and deterministic timing can be matched to streaming real-time workloads. This paper presents a methodology for implementing a reusable FPGA-based DSP architecture intended for filtering, spectral analysis, and related real-time applications. The proposed architecture combines fixed-point arithmetic, deep pipelining, parallel multiply-accumulate units, FIFO-based streaming interfaces, block RAM for coefficients and state, and a finite-state control layer. A stepwise implementation flow is defined from algorithm specification and word-length selection through RTL/high-level synthesis, simulation, synthesis, timing closure, and hardware validation. The evaluation framework uses a 64-tap finite impulse response (FIR) kernel as a representative workload and derives analytical throughput and latency at a 125 MHz target clock. A fully parallel pipelined realization can accept one new sample per clock after pipeline filling, corresponding to a theoretical 125 MS/s input rate, whereas a single-MAC sequential realization requires 64 cycles per sample and is limited to about 1.95 MS/s at the same clock. The methodology therefore demonstrates how FPGA spatial parallelism can be traded against DSP-block utilization to meet deterministic real-time constraints. The paper also discusses verification, fixed-point accuracy, resource-performance trade-offs, and limitations of synthesis-oriented results prior to board-level measurement.

Introduction

Modern real-time systems in communications, radar, imaging, instrumentation, motor control, and biomedical electronics frequently require continuous processing of high-rate sampled data. General-purpose processors provide convenient programmability, but strict sample deadlines and large numbers of repeated arithmetic operations can make purely sequential execution inefficient. FPGA devices address this problem by allowing the designer to build a custom data path in which

many arithmetic operations execute concurrently. Contemporary devices also include dedicated DSP slices, block memories, clock-management resources, and high-speed interfaces, making them especially well-suited to streaming DSP pipelines [1].

The central implementation challenge is not simply translating an algorithm into hardware. A useful real-time architecture must satisfy several constraints simultaneously: numerical accuracy, throughput, latency, FPGA resource limits, clock

frequency, memory bandwidth, and implementation effort. High-level synthesis (HLS) has reduced the abstraction gap by converting behavioral descriptions into register-transfer level hardware, and established research has shown that FPGA HLS can improve design productivity while producing competitive implementations [2]. However, timing and resource results remain strongly dependent on architectural decisions such as loop unrolling, operation scheduling, memory partitioning, pipelining, and interface design.

Fixed-point representation is another important design choice. Compared with floating-point arithmetic, fixed-point operators usually consume fewer resources and can support higher parallelism at lower implementation cost. Menard et al. demonstrated that word-length optimization can be coupled with HLS so that implementation cost is reduced while an explicit accuracy constraint is maintained [3]. For a real-time FPGA signal processor, therefore, the numeric format should be treated as an architectural parameter rather than an afterthought.

This paper develops a compact methodology for implementing a modular FPGA DSP engine. The architecture is designed around a streaming path that can host commonly used kernels such as FIR filtering, windowing, fast Fourier transform (FFT) processing, thresholding, and feature extraction. The main contribution is a reproducible step-by-step flow that links algorithm modeling, fixed-point conversion, parallel architecture selection, memory organization, timing constraints, simulation, synthesis, and hardware verification. The evaluation uses analytical performance calculations and literature-grounded design principles rather than claiming unperformed board measurements.

Literature Review

FPGA DSP research prior to and including 2017 established three recurring themes: architectural parallelism, abstraction-aware design, and application-specific streaming. Wang et al. proposed an FPGA architecture explicitly optimized for DSP workloads and reported that embedding DSP-oriented structures can substantially improve the performance of classical DSP cases relative to more traditional FPGA organizations [9]. This work illustrates why the physical organization of arithmetic and routing resources influences the achievable speed of a mapped signal-processing pipeline. At the design-flow level, Cong et al. presented a comprehensive view of HLS for FPGA deployment and described how C-level descriptions can be

transformed through scheduling, allocation, binding, and interface generation [2]. HLS is valuable for algorithm exploration, but its quality depends on architectural constraints supplied by the designer. Menard et al. addressed this issue from the numerical side by linking multiple-word-length optimization with HLS, showing that fixed-point precision and architecture generation should be optimized together [3].

Early design-space prediction is also important because the complete translate, place, route, and timing-analysis loop is time consuming. Wang, Stitt, Lam, and George proposed core-level modeling for FPGA DSP applications and reported an average clock-frequency prediction error of 3.6% across their evaluated cases, illustrating the value of parameter estimation before full implementation [4]. Such modeling supports rapid comparison of alternative levels of parallelism, pipeline depth, and component selection.

Modern FPGA DSP slices are more capable than simple multipliers. Bajaj and Fahmy showed that arithmetic graphs must be mapped with awareness of the internal pipeline structure of DSP blocks if maximum throughput is required [5]. Their work motivates the use of explicit pipeline registers and multiply-accumulate structures in the proposed architecture. For two-dimensional signal processing, Toledo-Moreo et al. demonstrated a modular FPGA convolution architecture capable of real-time operation with large kernels by exploiting parallelism and distributed arithmetic [6]. This confirms that spatial decomposition is effective when a regular dataflow pattern dominates the workload.

Application studies provide further evidence of FPGA suitability for time-critical signal processing. Quan et al. implemented real-time compressive sensing reconstruction on a high-density Xilinx FPGA, partitioning the computation into hardware blocks to balance speed, resource use, and accuracy [7]. Amiri et al. proposed FPGA-based soft-core processors for image processing and used a streaming multi-core approach to improve programmability while maintaining high throughput; their reported HOG case study reached 328 frames/s [8]. Woods et al. synthesized these ideas into a broad FPGA-DSP design perspective, emphasizing parallelism, dedicated arithmetic resources, memory organization, and architecture derivation for complex signal-processing systems [1].

The literature therefore supports a methodology in which the designer first identifies independent arithmetic operations and streaming dependencies, selects a fixed-point format, maps multiply-accumulate intensive work to dedicated DSP resources, inserts pipeline registers to

shorten critical paths, and uses local memories/FIFOs to avoid external-memory bottlenecks. These principles form the basis of the implementation flow presented next.

Methodology

The proposed methodology treats the FPGA implementation as a sequence of verifiable transformations from a reference DSP algorithm to a timed streaming hardware system. A

representative processing chain is shown in Figure 1. The chain accepts samples from an ADC or host interface, buffers the stream, applies FIR/window processing, performs an FFT or other block-oriented transform, formats the output, and transfers the result to a DAC or host. Dedicated DSP blocks execute multiply-accumulate operations, while block RAM stores coefficients, state, and intermediate samples.

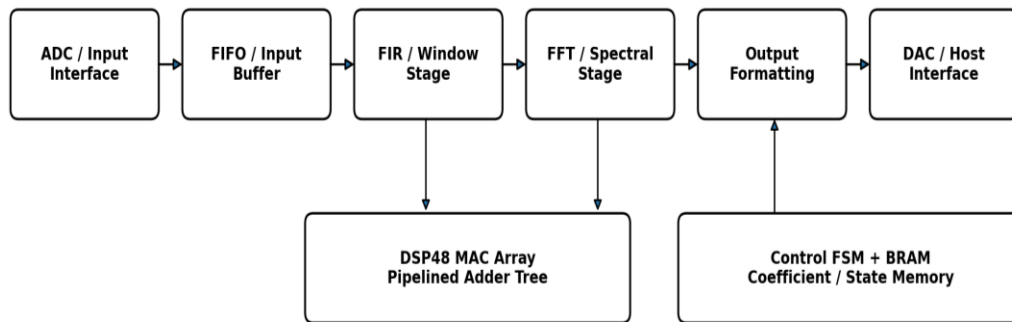


Figure 1. Proposed streaming FPGA-based DSP architecture for real-time processing.

Step 1: Define real-time requirements. Specify input sampling rate, data width, acceptable end-to-end latency, required DSP functions, interface protocol, and FPGA resource/power constraints. The minimum throughput is determined by the input sample rate; the design must sustain this rate continuously without FIFO overflow.

Step 2: Develop a floating-point reference model. Implement the algorithm in MATLAB, Python, or equivalent software and validate its functional behavior using deterministic test vectors. This model becomes the golden reference for later bit-accurate comparisons.

Step 3: Select fixed-point formats. Determine integer and fractional word lengths for input samples, coefficients, accumulator paths, and outputs. Quantization noise, overflow, rounding, and saturation are evaluated against the floating-point model. Following the principle in [3], precision is reduced only while the application-specific error limit remains satisfied.

Step 4: Decompose the algorithm into streaming kernels. Represent the algorithm as FIR/window, FFT, magnitude, threshold, or other independent processing blocks. Data dependencies are inspected to identify operations that can run concurrently. The dominant repeated multiply-accumulate operations are mapped to DSP48-class resources, consistent with throughput-aware DSP-block mapping [5].

Step 5: Choose parallelism and pipeline depth. For each kernel, select the number of processing elements and insert registers between arithmetic stages. A fully parallel N-tap FIR uses N multipliers and can achieve an initiation interval

of one sample per clock; a resource-shared implementation uses fewer multipliers but requires several cycles per sample. The selected architecture must satisfy the real-time throughput with timing margin.

Step 6: Design the memory and interface structure. Use shift registers or BRAM-based circular buffers for sample history, BRAM/ROM for coefficients, and FIFO interfaces between kernels. Streaming ready/valid handshakes isolate pipeline stages and absorb short-term scheduling differences. Block-oriented kernels such as FFTs use ping-pong buffers where required.

Step 7: Implement RTL or HLS modules. Describe the architecture in VHDL/Verilog or synthesizable C/C++. For HLS, apply pipeline, unroll, array-partition, and interface directives deliberately rather than relying only on automatic inference [2]. Clock/reset logic, parameter registers, and the control FSM are integrated around the data path.

Step 8: Perform functional and timing verification. Run unit-level and system-level simulation with impulse, step, sinusoidal, multitone, and random vectors. Compare fixed-point output against the reference model, then synthesize and implement the design. Check setup/hold timing, critical paths, resource utilization, and estimated power. Frequency and latency predictions can be used to guide iteration before full place-and-route [4].

Step 9: Hardware-in-the-loop validation. Program the FPGA board, inject known sample streams, capture output using an integrated logic

analyzer or host interface, and compare results with the golden model. Stress tests should include maximum sustained input rate, coefficient changes, reset during traffic, FIFO boundary conditions, and long-duration runs. Only these board-level tests should be labeled as measured hardware results.

For analytical evaluation, this paper uses a 64-tap FIR stage as a representative high-throughput kernel. The clock target is 125 MHz and the input/coefficient word length is 16 bits. In the sequential baseline, one DSP multiplier-accumulator is reused for all 64 taps, requiring approximately 64 arithmetic cycles per output sample. In the proposed fully unrolled architecture, 64 parallel multipliers feed a balanced pipelined adder tree. Once the pipeline is filled, one result is accepted and produced every clock cycle. These calculations establish a

throughput bound; actual post-route clock frequency and resource use must still be measured for a selected device.

Results

Table 1 summarizes the analytical performance of the representative FIR stage at the 125 MHz target. Because the two architectures perform the same 64 multiply-accumulate operations per output, the comparison exposes the central FPGA trade-off: spatial parallelism increases resource consumption but sharply improves throughput and deterministic latency. The sequential design processes 125 million clock cycles per second divided by 64 cycles per sample, or approximately 1.95 MS/s. The parallel pipelined design has an initiation interval of one and therefore a theoretical sustained input rate of 125 MS/s after pipeline filling.

Table 1. Analytical throughput comparison for a 64-tap FIR kernel. Values are design-study calculations, not *physical-board measurements*.

Metric	Sequential single-MAC FIR	Proposed parallel/pipelined FIR
FIR length	64 taps	64 taps
DSP multipliers used	1	64
Initiation interval	64 cycles/sample	1 cycle/sample
Theoretical sample rate @ 125 MHz	1.95 MS/s	125 MS/s
Time between accepted samples	512 ns	8 ns
Approx. arithmetic latency	≥ 512 ns	$\sim 64\text{--}80$ ns (pipeline dependent)
Primary advantage	Minimum DSP usage	Deterministic high throughput
Primary limitation	Cannot serve high-rate streams	Higher DSP/register usage

The calculated 64-fold throughput increase is the direct consequence of converting temporal reuse into spatial parallelism. This scaling is conceptually consistent with prior FPGA DSP work showing the benefit of parallel arithmetic structures [1], DSP-aware mapping [5], and streaming application decomposition [6]–[8]. It should not be interpreted as a universal speedup: a real implementation can be limited by routing congestion, DSP placement, BRAM port conflicts, interface bandwidth, clock-domain crossings, or an adjacent kernel with a larger initiation interval.

The proposed architecture also improves scheduling determinism. In a pipelined stream, a new sample advances every clock without software scheduling jitter. This is particularly important in closed-loop control, radar, and communication systems where a missed sample deadline can corrupt subsequent processing. On the other hand, the parallel design uses significantly more DSP slices and registers, so a

partially unrolled design may be preferable when the required sample rate is lower. For example, an eight-MAC realization would require about eight cycles per 64-tap output and can be selected when approximately 15.6 MS/s is sufficient at 125 MHz.

The literature suggests that practical implementations should also include margin between the required sample rate and the theoretical maximum. Core-level prediction methods can reduce design iterations [4], while fixed-point optimization can reduce area without unnecessarily sacrificing numerical quality [3]. Board-level verification remains essential because synthesis reports do not capture all system effects, including converter timing, I/O protocol behavior, clock quality, and sustained external data movement.

Conclusion and Discussion

This paper presented a step-by-step methodology for implementing an FPGA-based

DSP architecture for real-time applications. The design is organized as a streaming pipeline with input buffering, fixed-point processing kernels, dedicated DSP multiply-accumulate resources, local coefficient/state memory, and a control layer. The implementation flow begins with explicit real-time requirements and a validated floating-point model, then proceeds through word-length selection, architecture decomposition, parallelism selection, memory design, RTL/HLS implementation, verification, synthesis, timing closure, and hardware-in-the-loop testing. The principal architectural decision is the amount of parallelism. A fully unrolled 64-tap FIR example at a 125 MHz target can theoretically accept 125 MS/s after pipeline filling, whereas a single-MAC sequential realization is limited to approximately 1.95 MS/s. This result illustrates why FPGAs are effective for regular DSP workloads: computation can be distributed across many dedicated arithmetic blocks and separated by pipeline registers so that several data items are processed simultaneously at different stages. The cost is increased use of DSP slices, registers, routing, and potentially on-chip memory. The methodology is consistent with research up to 2017. HLS provides a practical route from algorithm descriptions to FPGA hardware [2], but quality still depends on informed architecture constraints. Fixed-point/HLS co-optimization can reduce implementation cost while preserving an accuracy requirement [3]. Core-level prediction can accelerate design-space exploration [4], and DSP-aware mapping is important for achieving maximum arithmetic throughput [5]. Real-time convolution, compressive sensing, and image-processing studies further demonstrate the value of modular parallel FPGA architectures [6]–[8]. The broader FPGA-DSP design framework described by Woods et al. reinforces the importance of matching algorithmic parallelism, arithmetic structure, memory, and device technology [1]. The main limitation of the present paper is that the numerical results are analytical design-study values rather than measurements from a fabricated or development-board prototype. Therefore, no claim is made about exact LUT, flip-flop, BRAM, DSP, power, or post-route frequency for a specific FPGA. A complete experimental study should select a target device, implement at least sequential, partially unrolled, and fully pipelined variants, and report utilization, worst negative slack, maximum frequency, dynamic power, fixed-point error, and sustained end-to-end throughput. It should also compare RTL and HLS productivity when both implementations are available. Future work can extend the architecture with run-time coefficient

updates, partial reconfiguration, dynamic precision, AXI-stream interfaces, DMA-based host transfer, and application-specific kernels such as adaptive filtering, beamforming, channelization, or image convolution. The same methodology remains applicable: quantify the deadline, expose parallelism, localize data movement, pipeline the critical path, verify numerical accuracy, and validate the final design under sustained real-time traffic.

References

- R. Woods, J. McAllister, Y. Yi, and G. Lightbody, *FPGA-Based Implementation of Signal Processing Systems*, 2nd ed. Wiley, 2017. doi: 10.1002/9781119079231.
- 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.
- D. Menard, N. Herve, O. Sentieys, and H.-N. Nguyen, "High-Level Synthesis under Fixed-Point Accuracy Constraint," *Journal of Electrical and Computer Engineering*, vol. 2012, Art. 906350, 2012. doi: 10.1155/2012/906350.
- G. Wang, G. Stitt, H. Lam, and A. George, "Core-Level Modeling and Frequency Prediction for DSP Applications on FPGAs," *International Journal of Reconfigurable Computing*, vol. 2015, Art. 784672, 2015. doi: 10.1155/2015/784672.
- R. Bajaj and S. A. Fahmy, "Mapping for Maximum Performance on FPGA DSP Blocks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 4, pp. 573–585, 2016. doi: 10.1109/TCAD.2015.2474363.
- F. J. Toledo-Moreo, J. J. Martínez-Alvarez, 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.
- Y. Quan, Y. Li, X. Gao, and M. Xing, "FPGA Implementation of Real-Time Compressive Sensing with Partial Fourier Dictionary," *International Journal of Antennas and Propagation*, vol. 2016, Art. 1671687, 2016. doi: 10.1155/2016/1671687.

M. Amiri, F. M. Siddiqui, C. Kelly, R. Woods, K. Rafferty, and B. Bardak, "FPGA-Based Soft-Core Processors for Image Processing Applications," *Journal of Signal Processing Systems*, vol. 87, pp. 139–156, 2017. doi: 10.1007/s11265-016-1185-7.

L. Wang, J. Lai, J. Tong, P. Tang, X. Chen, X. Duan, L. Chen, J. Wang, and Y. Wang, "A New FPGA Architecture Suitable for DSP Applications," *Journal of Semiconductors*, vol. 32, no. 5, Art. 055012, 2011. doi: 10.1088/1674-4926/32/5/055012.