



Comparative Analysis of FPGA-Based High-Speed Image Processing Architectures for Real-Time Applications

Kaoru Ongprasert

Department of Electrical and Computer Engineering, Shiraz College of Systems and Management, Iran
kaoru.ongprasert@scsm-ir.org

Peer Review Information

Submission: 28 Dec 2020

Revision: 15 Jan 2021

Acceptance: 07 Feb 2021

Keywords

FPGA, Real-Time Image Processing, Pipelining, High-Level Synthesis, Hardware Acceleration

Abstract

Field-programmable gate arrays (FPGAs) are widely used in real-time image processing because they combine fine-grained parallelism, deterministic latency, reconfigurability, and comparatively low power consumption. However, published FPGA image-processing systems employ different architectural styles, devices, image resolutions, algorithms, and performance metrics, making direct comparison difficult. This methodology paper develops a structured framework for comparing high-speed FPGA architectures used in real-time applications. The study classifies representative designs into systolic/window-based arrays, deeply pipelined streaming architectures, high-level-synthesis (HLS)-optimized datapaths, programmable soft-core/many-core architectures, and heterogeneous FPGA-processor systems. Literature published up to 2021 is evaluated using throughput, latency, logic and memory utilization, power/energy behavior, flexibility, scalability, and development complexity. The proposed method normalizes results at the architectural level rather than claiming device-independent numerical equivalence across different FPGA generations. Evidence from SIFT, Harris, Canny, AKAZE, HOG, and generic window-processing implementations shows that deeply pipelined streaming designs provide the highest deterministic throughput when the algorithm is stable, while HLS and processor-based approaches offer stronger design productivity and reprogrammability. Heterogeneous SoC solutions provide an effective compromise for algorithms combining control-intensive and data-parallel stages. The analysis indicates that architecture selection for real-time imaging should be treated as a multi-objective decision involving speed, resource efficiency, flexibility, and implementation effort rather than throughput alone.

Introduction

Modern machine-vision systems increasingly operate on high-resolution video while being constrained by latency, energy, form factor, and reliability. Applications such as autonomous navigation, robotics, industrial inspection, surveillance, medical imaging, and embedded visual sensing require continuous processing of large pixel streams. General-purpose CPUs provide programming flexibility but may fail to

provide deterministic low latency under strict real-time constraints. GPUs offer substantial throughput but can impose higher power and memory-transfer costs in embedded systems. FPGAs occupy an important middle ground because their programmable logic permits custom datapaths, pipeline concurrency, local buffering, and application-specific arithmetic while retaining post-fabrication reconfigurability. Reviews of image-processing

accelerators have therefore identified FPGAs as particularly attractive where throughput-per-watt, latency, and interface integration are critical (HajiRassouliha et al., 2018).

The main architectural advantage of an FPGA is that image operators can be implemented as spatial hardware rather than as sequential instruction streams. A convolution, gradient calculation, threshold stage, or feature descriptor can be arranged as a pipeline in which a new pixel is accepted every clock cycle after an initial filling delay. Earlier work on configurable systolic arrays demonstrated that data reuse and parallel window processing can substantially reduce memory-access overhead for generic image operations (Torres-Huitzil & Arias-Estrada, 2004). Later research extended these ideas to complete feature extraction pipelines, high-level synthesis, heterogeneous systems, and programmable image-processing processors.

Consequently, a meaningful methodology should not simply rank raw clock frequency or frame rate. It should compare the design principles that generate performance: pipeline depth, pixel-level concurrency, buffering strategy, arithmetic precision, memory organization, reprogrammability, and software/hardware partitioning.

The objective of this paper is to present a repeatable methodology for comparative analysis of FPGA-based high-speed image processing architectures for real-time applications. The study asks three questions: (i) which architecture styles provide the strongest throughput and latency characteristics; (ii) how do these architectures trade hardware resources for flexibility and development productivity; and (iii) which style is most appropriate for different real-time application classes. The comparison is restricted to literature published no later than 2021 and emphasizes peer-reviewed studies with traceable DOI identifiers.

Literature Review

Research on FPGA image processing has progressed from generic window-processing engines toward increasingly specialized and heterogeneous architectures. Torres-Huitzil and Arias-Estrada (2004) proposed a compact two-dimensional systolic array for window-based operations. Their prototype reported 3.16 GOPS at 60 MHz and an 8.35 ms processing time for a 7×7 operator on 512×512 grayscale images, illustrating how regular data movement and processing-element reuse can provide a favorable performance-area balance. This architecture established an important principle that remains relevant: in streaming vision,

memory access and neighborhood reuse are as important as arithmetic throughput.

Feature extraction later became a major target for aggressive pipeline design. Vourvoulakis, Kalomiros, and Lygouras (2016) developed a fully pipelined SIFT architecture on a Cyclone IV FPGA. The design processed VGA video at 70 frames/s and produced a feature descriptor in one clock cycle after pipeline filling, demonstrating the performance advantage of eliminating sequential descriptor computation. Li et al. (2018b) proposed another FPGA SIFT design that reduced expensive divisions and used parallel processing for the image pyramid, detector, and descriptor. Kreowsky and Stabernack (2021) subsequently reported a full-featured pipelined SIFT implementation supporting rotation-invariant descriptors and up to 84 frames/s for full-HD input, showing that streaming feature pipelines remained competitive even for more complete algorithmic fidelity.

A parallel research direction has focused on improving programmability. Amiri et al. (2017) presented optimized FPGA-based soft-core processors for image-processing workloads and reported 328 frames/s for a Histogram of Oriented Gradients (HOG) application. Siddiqui et al. (2019) extended the processor concept into a multi-core image-processing architecture on a Zynq-7000 SoC. Using 16 specialized image-processing cores, the study reported speedups of approximately $8\times$ for k-means clustering and $9.6\times$ for a morphology operation relative to ARM software. These studies show that a reusable processor fabric can trade some application-specific efficiency for shorter development cycles and easier functional changes.

High-level synthesis represents another compromise between custom RTL efficiency and software-level productivity. Li et al. (2018a) evaluated HLS code-optimization strategies for real-time image processing and showed that directive ordering, loop transformation, pipelining, and memory organization materially affect generated hardware. HLS is therefore not an automatic replacement for architecture design; it still requires hardware-aware restructuring. Sikka, Asati, and Shekhar (2021) used an HLS-assisted methodology with application-specific bit widths for a Harris corner detector and demonstrated improved speed and area on a Xilinx ZedBoard. The result reinforces the value of bit-width optimization, fixed-point reasoning, and pipeline-aware synthesis for high-speed vision hardware.

Algorithm-specific edge and feature detectors further illustrate the resource-accuracy trade-off. Sangeetha and Deepa (2019) proposed a cost-

effective FPGA Canny edge detector using approximations for gradient magnitude and orientation and pipelining to lower latency and hardware cost. Zhou et al. (2020) explored high-speed parallel Harris corner detection, while Soleimani, Capson, and Li (2021) developed an FPGA architecture for AKAZE nonlinear scale-space generation using image partitioning, parallel channels, multi-scale line buffers, and time-shared memory management. These designs emphasize that algorithm reformulation—rather than direct software translation—is often necessary to exploit FPGA concurrency.

Collectively, the literature suggests five recurring architecture families: systolic/window arrays, deeply pipelined streaming datapaths, HLS-generated optimized pipelines, specialized soft-core/many-core fabrics, and heterogeneous processor-FPGA systems. No single family is universally superior. Dedicated pipelines maximize deterministic throughput, whereas programmable fabrics and heterogeneous systems improve adaptability. The methodology below formalizes a comparison across these dimensions.

Methodology

1. Research Design

The study uses a structured comparative methodology based on peer-reviewed FPGA image-processing literature published through 2021. The unit of analysis is the architecture rather than the individual FPGA device. This distinction is important because raw resource counts and maximum clock rates are strongly dependent on fabrication generation, vendor primitives, synthesis tools, and device speed grade. The comparison therefore combines reported quantitative evidence with architecture-level qualitative assessment.

2. Step-by-Step Comparative Procedure

Step 1 – Literature selection. Studies were retained when they described an FPGA or FPGA-

SoC architecture for image/video processing, reported a real-time or high-throughput objective, and provided enough architectural detail to identify parallelism, buffering, or implementation style. Only publications dated 2021 or earlier were used.

Step 2 – Architecture classification. Each implementation was assigned to one dominant class: systolic/window array, streaming pipeline, HLS-optimized pipeline, soft-core/many-core processor, or heterogeneous FPGA-CPU architecture. A design could exhibit features of more than one class, but the dominant execution model determined its category.

Step 3 – Workload characterization. The target algorithm was recorded as low-level filtering/edge detection, local feature extraction, segmentation/clustering, or mixed vision pipeline. Image resolution and frame-rate evidence were captured when explicitly reported.

Step 4 – Metric extraction. The comparison used seven metrics: throughput, latency, LUT/logic efficiency, on-chip memory demand, DSP/arithmetic demand, power or energy behavior where available, and implementation flexibility/productivity. Reported frame rate and processing time were kept in their original form to avoid invalid cross-device conversion.

Step 5 – Normalization principle. Direct numerical ranking was avoided when workloads were not equivalent. Instead, the study used architecture-level judgments of High, Medium, or Low relative performance. A “High throughput” rating indicates evidence of pixel-level or multi-stage concurrency with real-time performance; “High flexibility” indicates rapid algorithm changes without complete RTL redesign.

Step 6 – Comparative synthesis. Observed strengths and weaknesses were consolidated into an application-selection matrix. The final judgment considers whether the architecture can sustain the input stream, remain within available FPGA resources, and meet development or reconfiguration requirements.

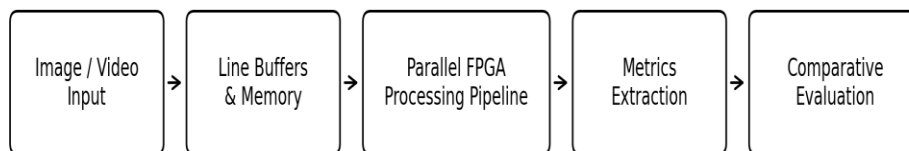


Figure 1. Comparative workflow for evaluating FPGA-based real-time image-processing architectures.

3. Evaluation Measures

For a design processing $W \times H$ pixels at F frames/s, the required pixel rate is $R_{px} = W \times H \times F$ pixels/s. An architecture capable of accepting one pixel per clock after pipeline filling has an ideal stream

capacity close to the operating clock frequency, subject to blanking, memory stalls, and multi-cycle stages. Latency is evaluated separately from throughput because a deep pipeline may have long initial latency yet still deliver one output

every cycle. Resource efficiency is interpreted through the balance of logic, BRAM, and DSP utilization relative to achieved performance. Flexibility captures the effort required to change an algorithm, kernel size, precision, or processing

graph. Development productivity considers abstraction level and verification burden, using the HLS and programmable-processor literature as evidence (Amiri et al., 2017; Li et al., 2018a; Siddiqui et al., 2019).

Table 1. Representative literature used in the comparative analysis.

Study	Architecture / Workload	Published evidence	Main comparison contribution
Torres-Huitzil & Arias-Estrada (2004)	2-D systolic window processor	3.16 GOPS at 60 MHz; 8.35 ms for 512×512, 7×7 operator	Generic parallel window processing and data reuse
Vourvoulakis et al. (2016)	Fully pipelined SIFT	70 fps at VGA; descriptor every clock after pipeline fill	Maximum streaming concurrency for feature extraction
Amiri et al. (2017)	Specialized soft-core processors / HOG	328 fps HOG implementation	Programmability with high FPGA throughput
Li et al. (2018a)	HLS-optimized image pipelines	Optimization-order and directive effects demonstrated	Design productivity and hardware-aware HLS
Li et al. (2018b)	Parallel FPGA SIFT	Divider reduction and parallel image pyramid/detector/descriptor	Resource-efficient algorithm reformulation
Siddiqui et al. (2019)	16-core IPPro on Zynq	≈8× k-means and 9.6× morphology speedup vs ARM	Scalable many-core acceleration
Sikka et al. (2021)	HLS Harris detector	Improved speed/area on ZedBoard through bit-width control	Precision-aware HLS optimization
Kreowsky & Stabernack (2021)	Full-featured pipelined SIFT	84 fps FHD; 21 fps UHD	High-resolution streaming performance

Results

The comparative evidence indicates that architectural regularity is the primary driver of deterministic high throughput. Systolic and deeply pipelined streaming architectures exploit data locality and are especially effective for neighborhood operations and feature pipelines. The 2004 systolic processor and the later SIFT pipelines show the same underlying pattern: buffer pixels locally, reuse neighborhoods, and maintain continuous data flow through concurrently active stages (Torres-Huitzil & Arias-Estrada, 2004; Vourvoulakis et al., 2016). The principal disadvantage is that major algorithmic changes can require substantial datapath redesign.

HLS-optimized architectures provide a second high-performance path. The results reported by Li et al. (2018a) and Sikka et al. (2021) indicate that HLS can approach the structure of manual RTL when designers explicitly manage pipelining, memory access, loop structure, and arithmetic widths. HLS therefore improves design productivity but does not eliminate architectural reasoning. Poorly organized C/C++ or model-based code can synthesize into a slow or resource-heavy circuit, whereas hardware-aware code can expose initiation intervals close to one cycle for streamable stages.

Programmable soft-core and many-core designs achieve the highest flexibility in the comparison. Amiri et al. (2017) and Siddiqui et al. (2019) show that specialized image-processing processors can retain substantial parallel performance while allowing algorithm changes through software or dataflow mapping. Their cost is lower area efficiency than a fully dedicated datapath because instruction storage, control logic, registers, interconnect, and FIFOs consume resources that an application-specific pipeline can devote directly to arithmetic.

Heterogeneous SoC architectures are most attractive when an image-processing application contains both regular pixel computation and irregular control. The CPU can manage configuration, memory orchestration, or non-streaming stages while the FPGA fabric accelerates convolution, gradients, feature extraction, or other data-parallel kernels. Siddiqui et al. (2019) illustrate this partitioning on Zynq, while Soleimani et al. (2021) demonstrate that memory management and image partitioning must be designed carefully to keep parallel processing units supplied with data.

Table 2. Architecture-level comparative result synthesized from the reviewed evidence.

Architecture class	Throughput	Resource efficiency	Flexibility	Development productivity	Best-fit use
Systolic / window array	High	High	Medium	Low–Medium	Convolution, morphology, fixed neighborhood operators
Deep streaming pipeline	Very high	High	Low–Medium	Low	Stable real-time feature/vision pipelines
HLS-optimized pipeline	High	Medium–High	Medium	High	Rapid development with strict performance targets
Soft-core / many-core	Medium–High	Medium	Very high	High	Frequently changing or multi-algorithm workloads
Heterogeneous FPGA–CPU SoC	High	Medium–High	High	Medium–High	Mixed control + data-parallel embedded vision

The highest real-time performance is therefore expected when the full algorithm can be converted into a regular pixel stream with bounded local memory. This is visible in the fully pipelined SIFT results, including 70 fps VGA in Vourvoulakis et al. (2016) and 84 fps full-HD in Kreowsky and Stabernack (2021). When algorithm fidelity requires irregular memory access, iteration, or frequent updates, a programmable or heterogeneous approach becomes more attractive even if its peak throughput-per-resource is lower. The Canny, Harris, and AKAZE studies also show that mathematical approximation, fixed-point precision, partitioning, and parallel memory organization can materially alter the hardware cost without abandoning the core algorithm (Sangeetha & Deepa, 2019; Zhou et al., 2020; Sikka et al., 2021; Soleimani et al., 2021).

1. Detailed Interpretation of Critical Design Factors

Design productivity and verification effort. RTL remains the most direct route to cycle-accurate control, but it requires detailed handling of data validity, pipeline alignment, memories, interfaces, and timing. HLS raises the abstraction level and can shorten development time, yet Li et al. (2018a) demonstrate that optimization directives and source-code transformations must be applied strategically. This means productivity should be measured by the time required to reach a verified timing-closed implementation, not merely by the number of source-code lines. For projects with changing requirements, the additional abstraction of HLS or programmable processors can outweigh a modest loss in peak

efficiency; for fixed high-volume pipelines, custom RTL may still provide the best performance-per-resource.

Scalability and reconfiguration. Dedicated pipelines scale efficiently when the same operation is replicated across lanes or stages, but scaling can rapidly consume BRAM, routing resources, and DSP blocks. Soft-core and many-core architectures take a different approach by providing reusable processing elements and programmable interconnect. Amiri et al. (2017) show that a specialized soft-core can reach high image-processing rates, and Siddiqui et al. (2019) demonstrate performance growth through multiple IPPro cores. The benefit is that a new algorithm may be mapped onto existing processors without redesigning every arithmetic stage. The drawback is that each programmable core carries control and storage overhead, so the maximum useful core count is eventually constrained by FPGA area and communication bandwidth.

Arithmetic precision and approximation. FPGA designers can deliberately select fixed-point word lengths and hardware-friendly approximations to reduce logic and DSP usage. The Canny implementation of Sangeetha and Deepa (2019) replaces computationally expensive gradient operations with approximation techniques, while Sikka et al. (2021) use application-specific bit widths in an HLS design flow. Such optimizations can increase clock frequency and lower area, but they introduce a verification requirement: the hardware result must remain acceptably close to the reference algorithm. Therefore, the methodology treats numerical precision as a

design variable rather than assuming that all implementations are functionally identical.

Pipeline initiation interval and latency. A major advantage of streaming FPGA designs is the ability to separate latency from throughput. A long chain of filters can require many clock cycles before the first valid output appears, yet after the pipeline is filled it can potentially accept a new pixel every clock. For continuous video this steady-state initiation interval is often more important than the first-output latency. The fully pipelined SIFT architecture of Vourvoulakis et al. (2016) is an illustrative case because feature detection and descriptor generation are overlapped so that descriptor production does not become a sequential bottleneck. Kreowsky and Stabernack (2021) similarly demonstrate how pipelining enables high-resolution SIFT processing while preserving a more complete algorithm.

Memory bandwidth and data reuse. In high-speed vision systems, arithmetic units are useful only when pixels and neighborhood samples arrive at the required rate. Window-based filters therefore benefit from line buffers, shift registers, and local BRAM structures that reuse previously received pixels rather than repeatedly reading complete neighborhoods from external memory. The compact systolic design of Torres-Huitzil and Arias-Estrada (2004) and the later multi-scale buffering strategy of Soleimani et al. (2021) show two forms of the same principle. When a design is limited by external-memory transactions, increasing the number of multipliers or processing elements may not increase frame rate. Thus, an architecture should be evaluated not only by computational parallelism but also by whether its memory organization can continuously supply that parallelism.

Discussion

The central implication is that FPGA architecture selection should be treated as a multi-objective optimization problem. Raw frame rate is insufficient because a system with excellent throughput may be difficult to modify, may consume most of the device BRAM, or may only support a simplified form of the algorithm. Conversely, a programmable architecture may deliver lower peak efficiency but reduce verification time and support multiple tasks on a single deployed platform. This is consistent with the broader accelerator comparison of HajiRassouliha et al. (2018), which emphasizes that hardware choice must account for architecture, tools, power, and application constraints rather than processing speed alone. For fixed industrial inspection or camera-front-end functions, a deeply pipelined or systolic

architecture is generally the preferred option. For research platforms and products whose algorithms change frequently, HLS or specialized soft-core architectures provide a more practical balance. For autonomous and robotic systems that mix video preprocessing, feature extraction, decision logic, and communication, heterogeneous FPGA-CPU SoCs are particularly suitable. The methodological limitation of the present comparison is that the reviewed studies do not use identical FPGA technologies, image resolutions, datasets, or power-measurement procedures. Consequently, the qualitative rankings should be interpreted as architecture-level guidance rather than a device-normalized benchmark.

Conclusion

This paper presented a comparative methodology for evaluating FPGA-based high-speed image-processing architectures for real-time applications using literature published up to 2021. The framework classifies designs by execution model and compares them using throughput, latency, resource efficiency, memory behavior, flexibility, and development productivity. The review shows that deeply pipelined streaming and systolic architectures achieve the strongest deterministic throughput when an algorithm can be expressed as regular local computations. HLS-optimized designs can preserve much of this performance while reducing development effort, provided that designers explicitly optimize loops, memory, and numeric precision. Specialized soft-core and many-core architectures offer superior reprogrammability at the cost of additional control and interconnect resources. Heterogeneous SoCs provide the best general-purpose compromise for mixed workloads. Future comparative studies should benchmark multiple architecture styles on the same FPGA device, image datasets, precision, and power-measurement method so that throughput-per-watt and performance-per-resource can be compared quantitatively.

References

- Amiri, M., Siddiqui, F. M., Kelly, C., Woods, R., Rafferty, K., & Bardak, B. (2017). FPGA-based soft-core processors for image processing applications. *Journal of Signal Processing Systems*, 87, 139–156. <https://doi.org/10.1007/s11265-016-1185-7>
- HajiRassouliha, A., Taberner, A. J., Nash, M. P., & Nielsen, P. M. F. (2018). Suitability of recent hardware accelerators (DSPs, FPGAs, and GPUs) for computer vision and image processing

algorithms. *Signal Processing: Image Communication*, 68, 101–119. <https://doi.org/10.1016/j.image.2018.07.007>

Kreowsky, P., & Stabernack, B. (2021). A full-featured FPGA-based pipelined architecture for SIFT extraction. *IEEE Access*, 9, 128564–128573. <https://doi.org/10.1109/ACCESS.2021.3104387>

Li, C., Bi, Y., Benezeth, Y., Ginhac, D., & Yang, F. (2018a). High-level synthesis for FPGAs: Code optimization strategies for real-time image processing. *Journal of Real-Time Image Processing*, 14, 701–712. <https://doi.org/10.1007/s11554-017-0722-3>

Li, S.-A., Wang, W.-Y., Pan, W.-Z., Hsu, C.-C. J., & Lu, C.-K. (2018b). FPGA-based hardware design for scale-invariant feature transform. *IEEE Access*, 6, 43850–43864. <https://doi.org/10.1109/ACCESS.2018.2863019>

Sangeetha, D., & Deepa, P. (2019). FPGA implementation of cost-effective robust Canny edge detection algorithm. *Journal of Real-Time Image Processing*, 16, 957–970. <https://doi.org/10.1007/s11554-016-0582-2>

Siddiqui, F., Amiri, S., Minhas, U. I., Deng, T., Woods, R., Rafferty, K., & Crookes, D. (2019). FPGA-based processor acceleration for image processing applications. *Journal of Imaging*, 5(1), 16. <https://doi.org/10.3390/jimaging5010016>

Sikka, P., Asati, A. R., & Shekhar, C. (2021). Real time FPGA implementation of a high speed and area optimized Harris corner detection algorithm. *Microprocessors and Microsystems*, 80, 103514. <https://doi.org/10.1016/j.micpro.2020.103514>

Soleimani, P., Capson, D. W., & Li, K. F. (2021). Real-time FPGA-based implementation of the AKAZE algorithm with nonlinear scale space generation using image partitioning. *Journal of Real-Time Image Processing*, 18, 2123–2134. <https://doi.org/10.1007/s11554-021-01089-9>

Torres-Huitzil, C., & Arias-Estrada, M. (2004). Real-time image processing with a compact FPGA-based systolic architecture. *Real-Time Imaging*, 10(3), 177–187. <https://doi.org/10.1016/j.rti.2004.06.001>

Vourvoulakis, J. V., Kalomiros, J. A., & Lygouras, J. N. (2016). Fully pipelined FPGA-based architecture for real-time SIFT extraction. *Microprocessors and Microsystems*, 40, 53–73. <https://doi.org/10.1016/j.micpro.2015.11.013>

Zhou, H., Dou, R., Yu, S., Qi, N., Liu, J., Wu, N., & Liu, L. (2020). A high-speed parallel FPGA implementation of Harris corner detection. In *2020 IEEE International Conference on Integrated Circuits, Technologies and Applications (ICTA)* (pp. 71–72). <https://doi.org/10.1109/ICTA50426.2020.9332111>