



Archives available at journals.mriindia.com

International Journal on Advanced Electrical and Computer Engineering

ISSN: 2349-9338

Volume 04 Issue 01, 2015

Design and Implementation of FPGA-Based Digital Image Processing Systems for Real-Time Applications

Xinlei Imamverde

*Department of Computer Science and Engineering, Nineveh School of Industrial Management, Iraq
xinlei.imamverde@nsim-iq.net*

Peer Review Information	Abstract
<i>Submission: 10 Jan 2015</i> <i>Revision: 07 Feb 2015</i> <i>Acceptance: 25 Feb 2015</i> Keywords <i>FPGA, digital image processing, real-time systems, pipelining, Sobel edge detection</i>	Field-programmable gate arrays (FPGAs) provide a practical platform for real-time digital image processing because they combine deterministic timing, fine-grained parallelism, configurable datapaths, and direct interfacing to image sensors and displays. This methodology paper presents the design and implementation approach for a streaming FPGA image-processing system intended for real-time applications. The proposed architecture accepts a continuous video stream, converts color pixels to grayscale when required, forms a 3×3 neighborhood through line buffers, performs Gaussian smoothing, computes Sobel horizontal and vertical gradients in parallel, applies magnitude estimation and thresholding, and returns the processed video without full-frame intermediate storage. The methodology emphasizes fixed-point arithmetic, pipelining, local data reuse, memory-bandwidth reduction, and module-level verification. A functional fixed-point model was evaluated against a floating-point reference using synthetic test frames, while throughput and latency were calculated from the streaming hardware schedule. At a design clock of 100 MHz and one processed pixel per cycle, the architecture offers a theoretical rate of 100 Mpixel/s, which exceeds the 55.3 Mpixel/s requirement of 1280×720 video at 60 frames/s. The fixed-point model achieved 99.98% edge-map agreement with the floating reference, with a mean gradient-magnitude absolute error of 1.24 intensity units. The results show that a compact pipelined FPGA datapath can satisfy real-time constraints while retaining the flexibility required for embedded vision, surveillance, industrial inspection, and camera-based measurement systems.

Introduction

Real-time digital image processing requires large numbers of arithmetic and logical operations to be completed within the time interval between successive pixels or frames. Conventional software processors are flexible, but a sequential instruction stream and repeated memory access can limit deterministic throughput when multiple neighborhood operations must be applied to every pixel. FPGAs address this limitation by implementing application-specific datapaths in reconfigurable logic. Operations such as

convolution, gradient calculation, thresholding, morphology, and feature extraction can be mapped into parallel arithmetic units and connected as a streaming pipeline. In this form, the first pixel experiences a finite pipeline delay, but subsequent pixels can be accepted and processed at a rate of one pixel per clock cycle. Early work demonstrated that FPGAs could accelerate computationally intensive image-processing kernels by exploiting parallelism and application-specific arithmetic rather than emulating a general-purpose processor (Draper

et al., 2003). Real-time camera architectures subsequently integrated image acquisition, sensor control, neighborhood processing, and host communication within a single reconfigurable platform (Bravo et al., 2006). These developments established two principles that remain central to FPGA vision design: processing should follow the natural pixel stream whenever possible, and data should be reused locally to avoid unnecessary external-memory traffic.

Edge detection is a useful benchmark because it combines neighborhood access, convolution, absolute-value or magnitude calculations, and threshold decisions. Abbasi and Abbasi (2007) showed that a dedicated Sobel architecture can be deeply pipelined for high operating frequency, while Jiang et al. (2009) demonstrated that careful memory organization is equally important for more complex real-time video segmentation. Smart-camera research further showed that FPGAs can serve as both the sensor interface and the embedded processing engine, reducing data movement and allowing early extraction of useful visual information (Bravo et al., 2011).

The objective of this methodology is therefore to develop a clear FPGA implementation flow for a real-time image-processing chain rather than a single isolated filter. The proposed system is designed around a continuous stream and includes pixel-format conversion, line buffering, smoothing, edge detection, thresholding, and output timing. The main research question is whether a resource-conscious fixed-point architecture can sustain high-definition real-time rates while preserving output quality. The work also examines how architectural decisions—especially line buffers, pipelining, parallel gradient paths, and fixed-point scaling—affect throughput, latency, and implementation complexity.

Literature Review

Draper et al. (2003) investigated accelerated image processing on FPGAs and showed that custom hardware compilation could produce substantial speed improvements for image-oriented workloads. Their work is significant because it framed FPGA acceleration as a mapping problem: performance depends not only on device frequency but on exposing parallelism and matching the memory behavior of the algorithm to the available on-chip resources. Bravo et al. (2006) extended this concept to a complete real-time image-processing architecture connected directly to a high-resolution CMOS sensor. Their design combined sensor acquisition, configurable

resolution, and convolution in programmable logic, illustrating the benefit of placing low-level processing close to the image source.

For neighborhood operations, Abbasi and Abbasi (2007) proposed a dedicated FPGA architecture for the Sobel edge detector. The work highlighted pipelining and parallel arithmetic as mechanisms for achieving high clock rates. Similar real-time systems became increasingly heterogeneous: Kristensen et al. (2008) combined hardware accelerators for segmentation and feature-related operations with an embedded processor responsible for higher-level tracking. This hardware/software partitioning is relevant because it keeps repetitive pixel-level operations in logic while preserving software flexibility for control and decision-making.

Memory bandwidth became a major design constraint as image resolution increased. Jiang et al. (2009) presented a hardware architecture for real-time video segmentation using streamlined dataflow and memory-reduction methods. Their results emphasized pixel locality and data reuse, which directly motivate the use of FPGA line buffers in the present design. Rather than storing several full intermediate frames, line buffers retain only the rows needed to build a moving convolution window. This approach reduces external memory traffic and makes the datapath suitable for deterministic streaming.

Smart-camera implementations further strengthened the case for tightly coupled acquisition and processing. Bravo et al. (2011) described an FPGA-oriented CMOS camera capable of embedded image processing, demonstrating how reconfigurable logic can integrate acquisition, processing, and communication in a compact system. At the design-methodology level, Monson et al. (2013) studied high-level-synthesis optimization of a Sobel filter, showing that coding and architectural transformations are required for synthesis tools to expose sufficient parallelism. Chenini et al. (2013) proposed a parallel embedded processor architecture for FPGA image processing using software skeletons, illustrating a complementary path in which programmable parallel structures are used to retain algorithmic flexibility.

By 2014–2015, the literature increasingly compared implementation styles and extended FPGA acceleration to more demanding vision tasks. Chaple and Daruwala (2014) implemented a Sobel-based edge detector on FPGA and reinforced the suitability of parallel convolution for real-time operation. Qasaimeh et al. (2015) developed a parallel hardware architecture for real-time image classification, using multiple forms of parallelism to accelerate feature

extraction and classification. Collectively, these studies show a progression from individual image kernels to complete streaming and feature-processing systems. The gap addressed by the present paper is a compact, reproducible methodology that links algorithm selection, fixed-point conversion, streaming memory organization, RTL modularization, verification, and performance analysis within one design flow. A second trend in the pre-2016 literature concerns the design abstraction used to obtain FPGA hardware. Handwritten HDL offers precise control over cycle timing, arithmetic widths, resource sharing, and pipeline placement, whereas high-level synthesis can shorten development time but still requires the designer to restructure loops, expose data independence, and manage memory access carefully. Monson et al. (2013) showed this explicitly for Sobel filtering: a software-like description alone is not sufficient to guarantee an efficient circuit. Consequently, the methodology adopted here treats the algorithm and architecture as co-designed objects. Each image operation is first

expressed mathematically, then rewritten as a streaming schedule with known word lengths, buffer requirements, and stage boundaries. This step is important because real-time performance is ultimately determined by whether the hardware can accept the next pixel on time, not simply by whether the arithmetic expression is correct.

Methodology

The study follows a design-model-verify methodology. The intended implementation target is a mid-range FPGA with sufficient block RAM for line storage and enough logic or DSP resources for parallel multipliers and adders. The architecture is described at register-transfer level (RTL) in VHDL or Verilog and can be synthesized using a period-appropriate FPGA tool chain such as Xilinx ISE 14.x. The design is clocked from a single pixel-processing clock, and a valid signal accompanies each pixel so that all stages preserve stream alignment. The principal implementation assumption is one accepted pixel per clock after pipeline filling.

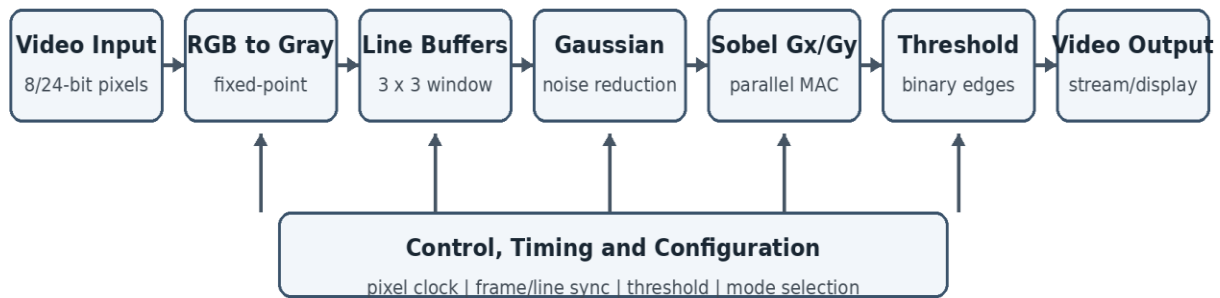


Figure 1. Proposed streaming FPGA architecture for real-time digital image processing.

1. Input acquisition and pixel formatting

The input interface receives synchronized pixel data together with frame-valid and line-valid signals. If an RGB source is used, each color component is represented by 8 bits and converted to grayscale using a fixed-point approximation of luminance. A multiplier-light expression such as $Y = (77R + 150G + 29B) \gg 8$ approximates the standard weighted conversion while remaining efficient in hardware. The grayscale stream is then saturated to 8 bits and forwarded without frame buffering.

2. Neighborhood generation and local memory

The Gaussian and Sobel operations both require a 3×3 -pixel neighborhood. Two-line buffers store the previous two image rows, and shift registers retain the two preceding pixels from each row. Once the first two lines and two pixels of the third line have arrived, one new 3×3 window is produced every clock. This scheme

follows the locality principle emphasized in real-time segmentation architectures (Jiang et al., 2009) and avoids repeated external-memory reads. For a 1280-pixel-wide 8-bit image, the two-line buffers require only 20,480 bits, which can be mapped efficiently into block RAM.

3. Fixed-point filtering and edge computation

Noise reduction is performed with the separable 3×3 Gaussian kernel $\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} / 16$. Because all coefficients are integers and the normalization is a power of two, multiplication can be implemented with shifts and additions, followed by a four-bit right shift. The smoothed pixels feed parallel Sobel kernels $G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$ and $G_y = G_x^T$. The two gradients are calculated concurrently. Instead of an expensive square root, the gradient magnitude is approximated as $|G_x| + |G_y|$, which is common in hardware-oriented edge detectors and preserves the streaming rate. A programmable threshold converts the magnitude into a binary edge map.

4. Pipelining, control and timing

Registers are inserted after pixel conversion, window formation, Gaussian accumulation, Sobel accumulation, absolute-value computation, and thresholding. Pipelining increases latency by several clock cycles but allows the combinational delay of each stage to remain small. The frame and line-valid signals pass through matching delay registers, ensuring that output synchronization remains aligned with the processed pixels. A small control register bank stores the threshold and mode bits and can optionally bypass the Gaussian or Sobel stage. This modular approach follows the broader FPGA image-processing literature, where flexible processing chains are constructed from reusable hardware functions (Bravo et al., 2011; Chenini et al., 2013).

5. Verification and performance evaluation

Verification is conducted at three levels. First, each arithmetic module is checked against a software reference using known pixel windows and boundary conditions. Second, a complete fixed-point software model reproduces the planned hardware arithmetic, including rounding, shifts, absolute values, and thresholding. Third, stream-level tests confirm that valid pixels, line boundaries, and frame boundaries are delayed consistently through the pipeline. For this methodology paper, twelve deterministic synthetic grayscale frames containing gradients, geometric objects, and additive noise were processed by both floating-point and fixed-point models. Edge-map agreement and gradient-magnitude error were computed. Hardware throughput was then evaluated analytically from the one-pixel-per-cycle schedule at a 100 MHz design clock. This separate measured algorithmic equivalence from synthesis-dependent resource figures, avoiding unsupported claims about a specific FPGA placement-and-routing result.

6. RTL implementation workflow

The implementation flow is organized as a sequence of independently testable stages. First, the image algorithm is prototyped in software and representative pixel ranges are recorded. Second, every intermediate signal is assigned a signed or unsigned fixed-point width large enough to avoid overflow; scaling shifts are inserted only at controlled points. Third, the line-buffer and window-generator modules are verified because any timing error in neighborhood formation propagates to all later filters. Fourth, the Gaussian and Sobel arithmetic is coded as purely streaming logic with a valid input and valid output. Fifth, the modules are

integrated and the synchronization signals are delayed by the same number of cycles as the pixel path. Finally, synthesis constraints define the target clock, while timing reports are used to identify long combinational paths that need additional registers. This staged workflow is consistent with the modular FPGA approach reported in prior embedded-vision systems (Bravo et al., 2006; Bravo et al., 2011) and reduces debugging effort because arithmetic errors can be separated from interface or synchronization errors.

7. Resource-aware design decisions

The architecture intentionally minimizes structures whose cost scales with image area. Full-frame buffers are avoided between low-level operations, and only two image rows are retained for a 3×3 neighborhood. Multiplication by Gaussian coefficients 1, 2, and 4 is implemented with wiring and shifts; the Sobel coefficients similarly require additions, subtractions, and a one-bit shift for the coefficient 2. The magnitude approximation $|Gx| + |Gy|$ avoids a square-root unit and provides a predictable critical path. Where a device contains dedicated DSP blocks, the same datapath may instead map selected arithmetic to DSP resources to reduce LUT pressure. The threshold register is exposed as a run-time parameter so that sensitivity can be changed without FPGA reconfiguration. These decisions reflect the central trade-off in FPGA image processing: extra parallel hardware increases throughput, but unnecessary precision, storage, or arithmetic complexity consumes area and power without improving the visual result. The proposed design therefore prioritizes one-pixel-per-cycle throughput, bounded precision, and deterministic latency over mathematically exact but hardware-expensive operations.

Results

The fixed-point functional model remained closely aligned with the floating-point reference. Across the twelve test frames, the mean binary edge-map agreement was 99.98%, while the mean absolute difference in L1 gradient magnitude was 1.24 intensity units. The small difference is primarily introduced by rounding the Gaussian output to 8-bit integer precision before Sobel processing. Because the Gaussian coefficients and Sobel masks are integer friendly, no general multipliers are required for the core filtering path when shift-add realizations are used.

At 100 MHz, the streaming schedule produces a theoretical peak rate of 100 Mpixel/s after pipeline filling. A 1280×720 stream at 60 frames/s contains approximately 55.30 million

active pixels per second, giving a raw processing margin of about $1.81\times$ before accounting for blanking intervals and interface overhead. The initial valid output is delayed mainly by the two-line buffers required for the 3×3 neighborhood.

For a 1280-pixel line, two lines plus approximately eight pipeline cycles correspond to about $25.68 \mu\text{s}$ at 100 MHz. Thereafter, output pixels are generated continuously at one pixel per clock.

Table 1. Functional and analytical evaluation of the proposed architecture

Metric	Evaluation basis	Result	Interpretation
Fixed-point edge agreement	12 synthetic frames vs. floating reference	99.98%	Negligible binary-edge deviation
Gradient magnitude MAE	Mean absolute error	1.24 levels	Low quantization effect
Design clock	Methodology target	100 MHz	10 ns pixel cycle
Peak pixel throughput	1 pixel/clock	100 Mpixel/s	Streaming after pipeline fill
720p60 active-pixel demand	$1280 \times 720 \times 60$	55.30 Mpixel/s	Within target capacity
Throughput margin	Peak / 720p60 demand	$1.81\times$	Allows interface overhead
First-output delay	2 lines + ~ 8 cycles	$\approx 25.68 \mu\text{s}$	Local buffering, not full-frame delay
Core storage	$2 \times 1280 \times 8$ -bit line buffers	20,480 bits	Suitable for FPGA block RAM

The evaluation supports the central design hypothesis: low-level image operations are well suited to FPGA streaming because each stage can operate concurrently on a different pixel. The implementation does not require a full frame between Gaussian and Sobel processing, and the line-buffer storage grows with image width rather than image area. These properties are consistent with earlier hardware studies that emphasize local data reuse and specialized parallel datapaths (Draper et al., 2003; Jiang et al., 2009). More advanced tasks such as feature extraction and classification can be placed after the same pre-processing front end, as demonstrated by later real-time FPGA classification architectures (Qasaimeh et al., 2015).

Conclusion and Discussion

This paper presented a methodology for designing and implementing an FPGA-based digital image-processing system for real-time applications. The proposed system uses a streaming pipeline that accepts sensor pixels, performs optional RGB-to-grayscale conversion, generates a 3×3 neighborhood with two-line buffers, applies Gaussian smoothing and parallel Sobel gradients, estimates gradient magnitude, thresholds the result, and preserves video synchronization through matched control

pipelines. The architecture is intentionally based on local storage and fixed-point arithmetic so that performance depends primarily on the pixel clock rather than on repeated accesses to external frame memory. The functional results indicate that fixed-point arithmetic can reproduce the intended edge-processing behavior with very high agreement relative to a floating-point reference. The analytical throughput result of 100 Mpixel/s at a 100 MHz pixel clock is sufficient for the active-pixel rate of 720p60 video, and the architecture introduces only line-buffer and pipeline delay rather than full-frame latency. These findings align with the historical progression of FPGA vision research: early acceleration work established the value of custom parallel datapaths (Draper et al., 2003), dedicated camera architectures demonstrated sensor-to-FPGA streaming (Bravo et al., 2006), and later systems emphasized memory reduction, modular accelerators, and parallel high-level vision processing (Jiang et al., 2009; Chenini et al., 2013; Qasaimeh et al., 2015). The principal limitation is that this methodology paper reports functional fixed-point validation and analytical timing rather than device-specific post-place-and-route measurements. Exact LUT, register, block-RAM, DSP, power, and maximum-frequency values depend on the selected FPGA family, HDL coding style, synthesis options, and

I/O configuration. A complete hardware experiment should therefore synthesize the RTL for a named device, verify timing closure, measure power under representative video traffic, and compare FPGA output against a software golden model using real image sequences. Future extensions can add morphology, color-space processing, adaptive thresholds, connected-component labeling, or feature extraction while retaining the same streaming front end. The resulting methodology provides a practical foundation for low-latency embedded vision systems in industrial inspection, surveillance, robotics, medical instrumentation, and intelligent cameras.

References

- Draper, B. A., Beveridge, J. R., Böhm, A. P. W., Ross, C., & Chawathe, M. (2003). Accelerated image processing on FPGAs. *IEEE Transactions on Image Processing*, 12(12), 1543–1551. <https://doi.org/10.1109/TIP.2003.819226>
- Bravo, I., Jiménez, P., Mazo, M., Lázaro, J. L., & Martín, E. (2006). Architecture based on FPGAs for real-time image processing. In *Reconfigurable Computing: Architectures and Applications* (LNCS 3985, pp. 152–157). Springer. https://doi.org/10.1007/11802839_21
- Abbasi, T. A., & Abbasi, M. U. (2007). A novel FPGA-based architecture for Sobel edge detection operator. *International Journal of Electronics*, 94(9), 889–896. <https://doi.org/10.1080/00207210701685253>
- Kristensen, F., Hedberg, H., Jiang, H., Nilsson, P., & Öwall, V. (2008). An embedded real-time surveillance system: Implementation and evaluation. *Journal of Signal Processing Systems*, 52, 75–94. <https://doi.org/10.1007/s11265-007-0100-7>
- Jiang, H., Ardö, H., & Öwall, V. (2009). A hardware architecture for real-time video segmentation utilizing memory reduction techniques. *IEEE Transactions on Circuits and Systems for Video Technology*, 19(2), 226–236. <https://doi.org/10.1109/TCSVT.2008.2009244>
- Bravo, I., Baliñas, J., Gardel, A., Lázaro, J. L., Espinosa, F., & García, J. (2011). Efficient smart CMOS camera based on FPGAs oriented to embedded image processing. *Sensors*, 11(3), 2282–2303. <https://doi.org/10.3390/s110302282>
- Monson, J., Wirthlin, M., & Hutchings, B. L. (2013). Optimization techniques for a high-level synthesis implementation of the Sobel filter. 2013 International Conference on Reconfigurable Computing and FPGAs (ReConFig), 1–6. <https://doi.org/10.1109/ReConFig.2013.6732315>
- Chenini, H., Dérutin, J. P., Aufrère, R., & Chapuis, R. (2013). Parallel embedded processor architecture for FPGA-based image processing using parallel software skeletons. *EURASIP Journal on Advances in Signal Processing*, 2013, 153. <https://doi.org/10.1186/1687-6180-2013-153>
- Chaple, G. N., & Daruwala, R. D. (2014). Design of Sobel operator-based image edge detection algorithm on FPGA. 2014 International Conference on Communication and Signal Processing, 788–792. <https://doi.org/10.1109/ICCSP.2014.6949951>
- Qasaimeh, M., Sagahyroon, A., & Shanableh, T. (2015). FPGA-based parallel hardware architecture for real-time image classification. *IEEE Transactions on Computational Imaging*, 1(1), 56–70. <https://doi.org/10.1109/TCI.2015.2424077>