



e-ISSN: 2278-8875

p-ISSN: 2320-3765

International Journal of Advanced Research

in Electrical, Electronics and Instrumentation Engineering

Volume 15, Issue 10, October 2026



Impact Factor: 9.867

☎ 9940 572 462

☎ 6381 907 438

✉ ijareeie@gmail.com

@ www.ijareeie.com



Design and Implementation of a 3×3 Gaussian Smoothing Filter on Spartan-7 FPGA

Rakshak Sood, Chandana Vanage, Shamil Pawar

Department of Electronics and Computer Science, Vidyalkar Institute of Technology, Mumbai, Maharashtra, India

ABSTRACT: Noise suppression is usually the first stage of any image-analysis chain, and the Gaussian filter remains the most common way to do it. This paper presents a complete hardware implementation of a 3×3 Gaussian smoothing filter on the EDGE Spartan-7 development board (XC7S15). The kernel is the binomial approximation $(1, 2, 1)^T(1, 2, 1)/16$, so every weight is a power of two and the whole convolution collapses into shifts and additions, with no multipliers or DSP slices. A 120×120, 8-bit greyscale picture is held in block RAM and streamed through two line buffers and a nine-register window at one pixel per clock; the result goes into a second block RAM and is shown beside the original on a 640×480, 60 Hz VGA monitor. After place-and-route in Vivado 2025.1 the design occupies 246 LUTs, 236 flip-flops and six of the ten RAMB36 tiles. All timing constraints are met at the 25 MHz pixel clock with 33.3 ns of worst setup slack, and the estimated on-chip power is 0.141 W, most of it consumed by the clocking primitive. One frame is filtered in 14,408 clock cycles, about 576 μ s. A bit-accurate software model of the datapath stays within one grey level of a floating-point reference and cuts the variance of the Laplacian response by roughly a factor of eight, which confirms the intended smoothing.

KEYWORDS: Gaussian filter, FPGA, Spartan-7, line buffer, image smoothing, Verilog HDL, VGA display.

I. INTRODUCTION

Almost every vision pipeline begins by cleaning up the image. Sensor noise, quantisation steps and compression artefacts all show up as small, rapid intensity changes, and later stages such as edge detection or segmentation tend to read those changes as real structure. A low-pass filter is the usual remedy, and of the candidates the Gaussian is the one textbooks introduce first [1]. The preference is not just habit. Marr and Hildreth built their zero-crossing edge detector on a Gaussian-smoothed Laplacian [2]; Canny later showed that the derivative of a Gaussian is a close match to the best possible step-edge detector under his stated criteria [3]; and the scale-space work of Witkin [4] and Lindeberg [5] explains the underlying reason, namely that among linear smoothing kernels the Gaussian is the one that does not invent new structure as the blur grows.

Running such a filter on a general-purpose processor is easy but wasteful for streaming video. Every output pixel costs nine multiply-accumulate operations, and the neighbourhood has to be fetched from memory again and again. An FPGA can do better by consuming one pixel per clock and keeping the last two image rows in on-chip memory, so that each input sample is read from the source exactly once [6], [7]. The idea is close to classic systolic design, where data pulses through a regular array of simple cells and each operand is reused as it passes [8], and modern reconfigurable fabric makes it practical for small teams because the whole pipeline, memories included, fits on an inexpensive part [9].

This paper targets the EDGE Spartan-7 board, and doubles as a self-contained demonstrator: the original and the smoothed picture appear side by side on an ordinary VGA monitor, so the effect can be judged by eye without a PC or a capture card. Our contributions are fourfold. First, we use the 1-2-1 binomial kernel so that the convolution needs no multipliers. Second, we describe the line-buffer and window architecture together with an exact cycle-level timing analysis. Third, we explain the display path that turns two 120×120 images into a 640×480 video signal. Fourth, we report post-implementation resource, timing and power figures along with image-quality measurements obtained from a bit-accurate model of the datapath.

II. THEORETICAL BACKGROUND

A. The Gaussian kernel

In two dimensions a zero-mean, isotropic Gaussian with standard deviation σ is given by

$$G(x, y) = \frac{1}{2\pi\sigma^2} \exp\left[-\frac{x^2+y^2}{2\sigma^2}\right] \quad (1)$$



where the leading factor normalises the area to one. Because the exponent splits into an x part and a y part, the kernel is separable:

$$G(x, y) = g(x)g(y), \quad g(t) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{t^2}{2\sigma^2}\right] \quad (2)$$

Separability means a $K \times K$ mask can be applied as two K -tap passes, which lowers the work per pixel from K^2 to $2K$ operations. For a 3×3 window the gain is marginal, and the direct single-pass form used here avoids the extra transposition buffer that a two-pass scheme would need.

B. Binomial approximation

A sampled Gaussian has real-valued weights and therefore needs multipliers. The pyramid coding work of Burt and Adelson [10] relies on short, symmetric kernels whose weights come close to a Gaussian, and the shortest useful member of that family is the three-tap binomial $b = [1 \ 2 \ 1]/4$, which is the second row of Pascal's triangle. Its variance is

$$\sigma_b^2 = \sum_{k=-1}^1 k^2 b(k) = \frac{1}{2} \quad (3)$$

so the filter behaves like a Gaussian with $\sigma \approx 0.71$ pixel. Taking the outer product of b with itself gives the 3×3 mask

$$H = b^T b = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \quad (4)$$

Let $w(i, j) = 16 H(i, j)$ denote the integer weights. The filtered value at position (x, y) of an input image I is then

$$O(x, y) = \left\lfloor \frac{1}{16} \sum_{i=-1}^1 \sum_{j=-1}^1 w(i, j) I(x + j, y + i) \right\rfloor \quad (5)$$

The floor operation is realised in hardware by simply dropping the four least-significant bits of the sum, which costs nothing.

C. Frequency response and noise reduction

The Fourier transform of b is $(2 + 2 \cos \omega)/4 = \cos^2(\omega/2)$, so the two-dimensional response is

$$H(\omega_x, \omega_y) = \cos^2\left(\frac{\omega_x}{2}\right) \cos^2\left(\frac{\omega_y}{2}\right) \quad (6)$$

The gain is 1 at DC, drops to 0.5 at $\omega = \pi/2$ and reaches zero at the Nyquist frequency in each axis. Detail and noise at the highest spatial frequencies are therefore removed completely, while slow brightness changes pass through unharmed. For zero-mean white noise of variance σ_n^2 , the output variance is

$$\sigma_{\text{out}}^2 = \sigma_n^2 \sum_{i,j} h^2(i, j) = \frac{9}{64} \sigma_n^2 \quad (7)$$

which is a reduction in noise power by a factor of $64/9 \approx 7.1$, or 8.5 dB. The price, as always with linear smoothing, is that genuine edges are softened as well: the filter cannot tell noise from a sharp but legitimate transition.

D. Fixed-point considerations

With 8-bit pixels the largest possible weighted sum is

$$S_{\text{max}} = 255 \sum_{i,j} w(i, j) = 255 \times 16 = 4080 < 2^{12} \quad (8)$$

A 12-bit accumulator is therefore sufficient, and because $4080/16 = 255$ the quotient fits back into eight bits. No saturation logic is required. Truncation rather than rounding introduces a small negative bias of about half a grey level on average, which we measure in Section VII.

III. SYSTEM ARCHITECTURE

Fig. 1 shows how the design is organised. A clocking block produces the pixel clock and a synchronised reset; a read-only image memory feeds the filter engine; the filter writes into a frame memory; and a VGA timing generator together with an address generator reads both memories to draw the picture. All blocks after the clock generator run from the same 25 MHz clock, so there are no clock-domain crossings to manage.

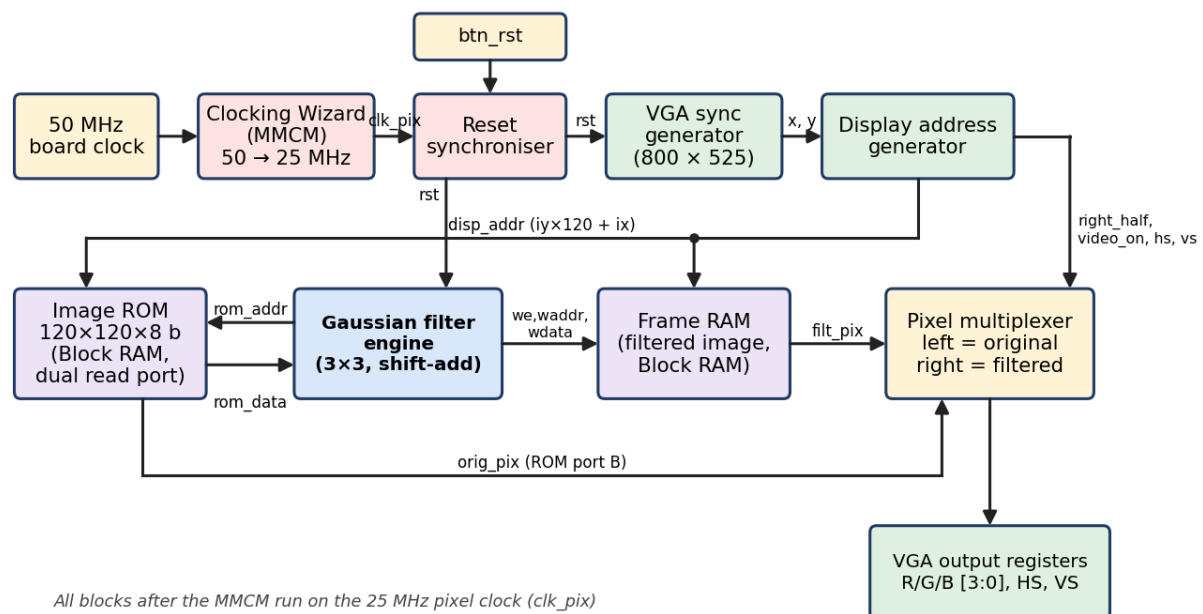


Fig. 1 Top-level block diagram of the Gaussian smoothing system.

A. Clocking and reset

The board's 50 MHz oscillator drives a Clocking Wizard instance [13] built around one MMCM. The generated configuration multiplies the input by 18.25 to reach a 912.5 MHz VCO and then divides by 36.5:

$$f_{\text{out}} = \frac{f_{\text{in}} M}{D O} = \frac{50 \times 18.25}{1 \times 36.5} = 25 \text{ MHz} \quad (9)$$

This is 0.7 % below the nominal 25.175 MHz of the standard 640×480 mode, a difference small enough for monitors to lock without complaint. The MMCM's locked output and the push-button are combined and passed through a three-stage flip-flop chain; the resulting synchronised, active-high reset holds the VGA counters and the filter in their initial state. Pressing the button therefore restarts the filtering pass, which is handy during demonstrations.

B. Memories

The source picture lives in a block-RAM image ROM initialised from a hexadecimal file of 14,400 bytes. The ROM has two independent read ports: port A belongs to the filter and port B to the display logic. Block RAM provides this true dual-port behaviour natively [12]. The filtered result is written into a frame RAM of the same size and read back by the display. One design choice here is easy to overlook but useful: the frame RAM is preloaded with the original image. A 3×3 window cannot produce an output for the one-pixel frame around the picture, and with this preload those pixels simply keep their original values, so no special border logic is needed.

IV. FILTER DATAPATH AND TIMING

The filter engine is drawn in Fig. 2. Pixels are read from the ROM in raster order, one per clock, through a counter-driven address. The synchronous ROM adds one cycle of latency, and the data are registered once more in x0. From x0 the stream follows three routes at the same time: directly into the bottom row of the window, into line buffer 1, and, after passing through that buffer, into line buffer 2. In this way the three window rows always hold pixels from image rows n, n-1 and n-2 that share the same column.

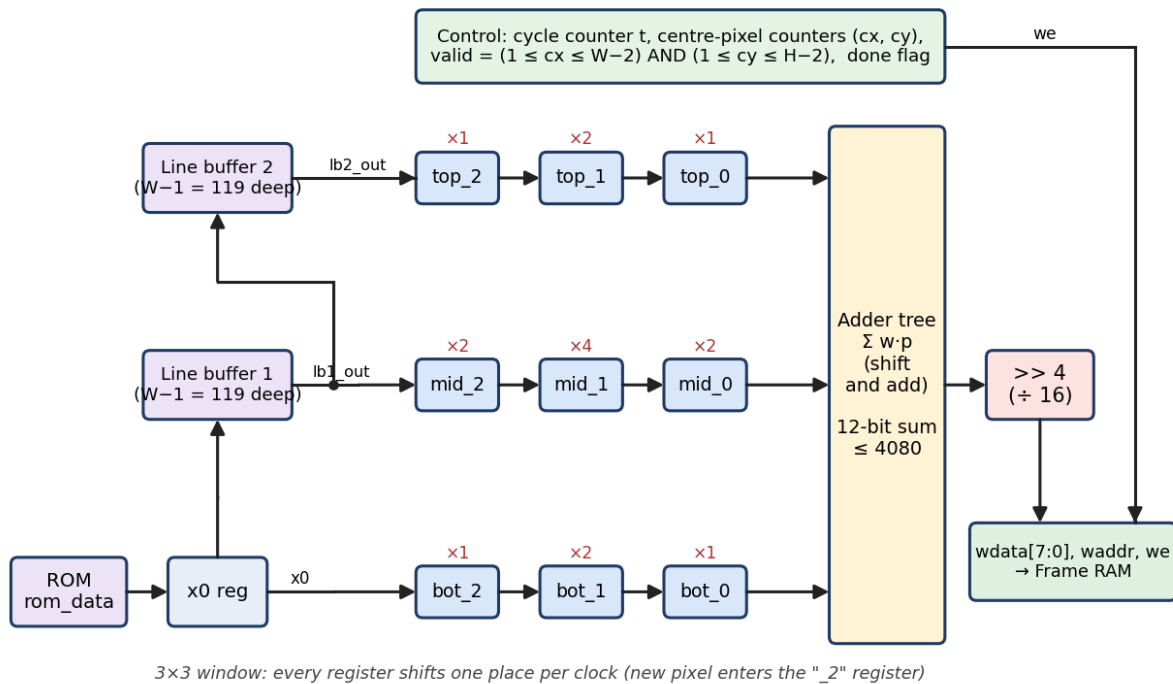


Fig. 2 Block diagram of the Gaussian filter engine: line buffers, 3x3 window, adder tree and control.

A. Line buffers

Each line buffer is a circular buffer built from distributed (LUT) RAM. On every clock it first reads the oldest entry and then overwrites that location with the new sample, so its output is its input delayed by a fixed number of cycles equal to its depth. Since the x0 register already contributes one cycle, each buffer is made $W-1 = 119$ entries deep and the total delay between vertically adjacent pixels is exactly one image line:

$$\tau_{row} = 1 + (W - 1) = W \text{ cycles} \quad (10)$$

Synthesis rounds each buffer up to 128×8 bits and maps it to RAM16, RAM32 and RAM64 primitives. Together the two buffers use 64 LUTs configured as memory and no block RAM, which leaves the RAMB36 tiles for the two frames.

B. Window and arithmetic

Nine 8-bit registers form the window. At each clock the three columns shift one place and a new column enters from the line-buffer taps. Numbering the registers p_0 to p_8 in row-major order ($p_0 = \text{top}_0$, ..., $p_8 = \text{bot}_2$), the weighted sum and the output are

$$S = (p_0 + p_2 + p_6 + p_8) + 2(p_1 + p_3 + p_5 + p_7) + 4p_4, \quad O = \left\lfloor \frac{S}{16} \right\rfloor \quad (11)$$

In the Verilog source the factors 2 and 4 are written as left shifts by one and two places, so the synthesis tool sees only additions. The sum is then truncated to its upper eight bits, which is the division by 16. Vivado turns the nine-operand addition into a short chain of carry primitives; the post-route report lists 23 CARRY4 cells for the whole design and no DSP block.

C. Control and cycle-accurate timing

A 16-bit counter t counts clock cycles after reset. While t is below $N = W \cdot H$ the ROM address equals t ; afterwards it parks on the last pixel. Counting the ROM cycle, the x0 register and the window shift, the window centre coincides with image pixel c at time t , where

$$c = t - W - 4, \quad (c_x, c_y) = \left(c \bmod W, \left\lfloor \frac{c}{W} \right\rfloor \right) \quad (12)$$

Two small counters, c_x and c_y , track this centre pixel. A write is enabled only when $1 \leq c_x \leq W-2$ and $1 \leq c_y \leq H-2$, which skips the border. The write enable, address and data are all registered so that they stay aligned. The first centre



appears at $t = W + 4 = 124$, i.e. $4.96 \mu\text{s}$ after reset is released, and a done flag is raised at $t = N + 8$. The time needed to filter a whole frame is therefore

$$T_f = \frac{W \cdot H + 8}{f_{\text{pix}}} = \frac{14408}{25 \text{ MHz}} \approx 576.3 \mu\text{s} \quad (13)$$

Throughput is one pixel per clock, i.e. 25 million pixels per second, regardless of the kernel because the arithmetic is fully pipelined. A useful consequence of the reset arrangement is that the filter and the VGA counters start together: the first picture line is drawn at line 120, roughly 3.84 ms after reset, whereas the filter finishes after 0.58 ms . The filtered frame is therefore complete about 3.3 ms before the monitor first reads it, and the display never shows a half-finished result.

V. IMPLEMENTATION AND EVALUATION SET-UP

The design consists of six small Verilog modules (top level, filter engine, line buffer, image ROM, frame RAM and VGA sync), about 310 lines including comments. It was built with Vivado 2025.1 for the XC7S15-1FTGB196C device on the EDGE Spartan-7 board. The constraints file assigns the 50 MHz oscillator to pin H11 with a 20 ns clock definition, uses the centre push-button as reset, and maps the twelve colour bits and the two sync lines to the board's VGA connector. Synthesis, placement, routing and bitstream generation all completed, and the reports discussed below come from the routed design.

Because the arithmetic is purely integer and fully specified by Eq. (11), the output of the hardware can be reproduced exactly in software. We therefore wrote a short Python model that applies the same weighted sum, the same truncation and the same untouched one-pixel border to the same hexadecimal image file. All image figures and quality numbers in Section VII were produced with this model rather than by photographing the monitor. We also compared it with a floating-point implementation of the mask in Eq. (4) to quantify the effect of truncation. The model reproduces the datapath equations; it is not a substitute for a gate-level simulation of the RTL.

VI. RESULTS AND DISCUSSION

A. Resource utilisation

Table II summarises the post-placement utilisation, which is also shown in the Vivado report in Fig. 3(a). The design is tiny in terms of logic: 246 LUTs (3.1 % of the device), 236 flip-flops (1.5 %) and 100 occupied slices (5 %). Block RAM is the only resource with noticeable use, at six of ten RAMB36 tiles, and this is because two full frames of 14,400 bytes have to be stored; the synthesis report shows the frame RAM alone taking four tiles. No DSP slices are used. Kuon and Rose [15] point out that arithmetic built from FPGA logic is comparatively costly next to an ASIC implementation, so avoiding multipliers entirely is a sensible choice for a soft design like this one.

TABLE II. POST-PLACEMENT RESOURCE UTILISATION (XC7S15)

Resource	Used	Available	Utilisation (%)
Slice LUTs	246	8,000	3.08
LUT as logic	181	8,000	2.26
LUT as distributed RAM	64	2,400	2.67
Slice registers (FF)	236	16,000	1.48
Occupied slices	100	2,000	5.00
Block RAM (RAMB36)	6	10	60.00
DSP slices	0	20	0.00
Bonded IOB	16	100	16.00
MMCM	1	2	50.00

B. Timing and power

Table III collects the timing and power figures, and the corresponding Vivado timing summary is shown in Fig. 3(b). With a 40 ns period for the 25 MHz pixel clock, the worst setup slack is 33.3 ns and the worst hold slack is 0.043 ns, with no failing endpoint among the 972 checked. The critical path starts at a window register and ends at the output



data register, which is the adder tree; it has a delay of 6.673 ns spread over eight logic levels (four CARRY4, two LUT3 and two LUT4), with 47 % of the time in logic and 53 % in routing. If we take the slack at face value, the same logic could in principle run at about $1/(40 - 33.3)$ ns $\approx$ 149 MHz. This is a rough, constraint-based estimate that ignores block-RAM and clock-network limits, so we treat it only as an indication of the headroom.

The power report (Fig. 3(c)) puts the total on-chip power at 0.141 W, of which 0.020 W is device static power. Of the 0.121 W dynamic part, the MMCM alone takes 0.115 W, whereas the block RAMs need about 3 mW and the filter engine roughly 1 mW. In other words, the smoothing itself is almost free; the cost of the demonstrator is dominated by generating the clock. The report's confidence level is "medium" because it uses default switching activity rather than simulated activity, so the absolute numbers should be read with that in mind.

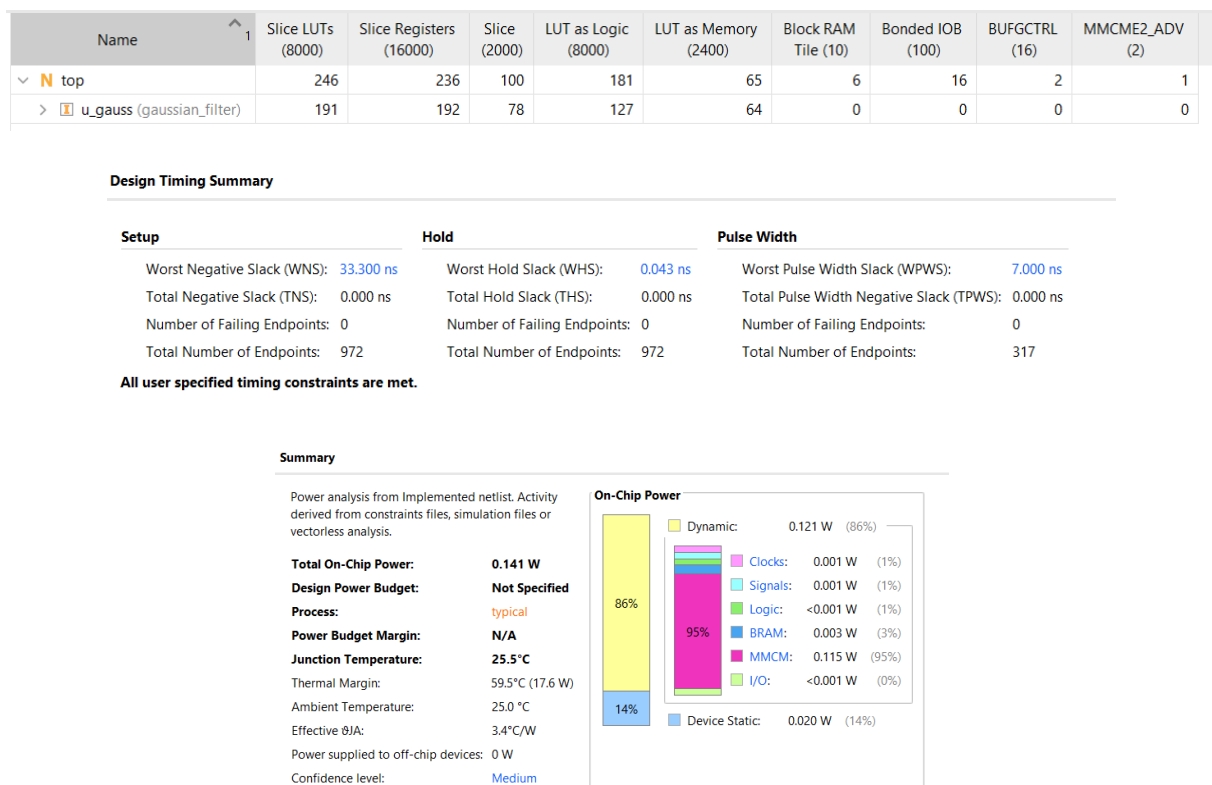


Fig. 3 Vivado 2025.1 post-implementation reports: (a) resource utilisation, (b) design timing summary, (c) on-chip power summary.

TABLE III. TIMING, POWER AND THROUGHPUT SUMMARY

Metric	Value
Pixel clock	25 MHz (40 ns period)
Worst setup slack (WNS)	33.300 ns
Worst hold slack (WHS)	0.043 ns
Critical-path delay	6.673 ns (8 logic levels)
Total on-chip power	0.141 W
Dynamic / static power	0.121 W / 0.020 W
MMCM / block-RAM power	0.115 W / 0.003 W
Junction temperature	25.5 °C
Frame filtering time	14,408 cycles $\approx$ 576.3 μ s
VGA refresh rate	$\approx$ 59.52 Hz

C. Image-quality results

Fig. 4 shows the test image, the output of the filter model and the absolute difference between them. The picture is a 120×120 greyscale version of a familiar test photograph. The smoothed result looks softer, with fine texture in the grass and the coat flattened, while large flat areas such as the sky are practically unchanged. The difference map confirms this: it is dark almost everywhere and lights up along edges and fine detail. This is what a low-pass filter should do, and it also shows that the difference between the original and smoothed images is a simple high-pass signal that could feed an edge-detection or sharpening stage.



Fig. 4 (a) Original image, (b) output of the 3×3 Gaussian filter, (c) absolute difference.

The intensity profile along one image row (Fig. 5a) makes the behaviour clearer. The large step at the edge of the coat is preserved but its transition is slightly broadened, and the narrow spikes around columns 60–70 are cut down noticeably. The histogram in Fig. 5b shows the same thing statistically: the isolated extreme values near 0 and above 200 vanish after filtering, and the tall peaks near 15 and 165 become lower and wider.

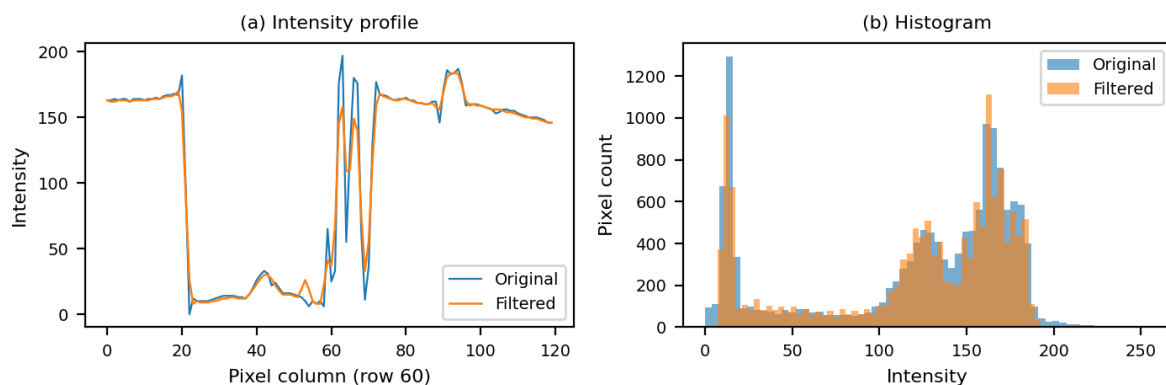


Fig. 5 (a) Intensity profile along row 60. (b) Histograms of the original and filtered images.

For a quantitative view we use the mean squared error and the peak signal-to-noise ratio between the original and filtered images,

$$\text{MSE} = \frac{1}{W \cdot H} \sum_{x,y} (I(x,y) - O(x,y))^2, \quad \text{PSNR} = 10 \log_{10} \frac{255^2}{\text{MSE}} \quad (16)$$

together with the structural similarity index of Wang et al. [14] and two sharpness indicators: the mean gradient magnitude and the variance of the 4-neighbour Laplacian,

$$L(x,y) = I(x+1,y) + I(x-1,y) + I(x,y+1) + I(x,y-1) - 4I(x,y) \quad (17)$$



Note that MSE, PSNR and SSIM here measure how far the output has moved from the input, so lower values mean stronger smoothing, not worse quality. The numbers are collected in Table IV.

TABLE IV. IMAGE METRICS FOR THE 120×120 TEST IMAGE

Metric	Original	Filtered
MSE (original vs filtered)	—	119.30
PSNR (dB)	—	27.36
SSIM	—	0.913
Mean gradient magnitude	24.24	17.18
Laplacian variance	2645.1	327.1
Std. deviation (interior)	61.43	58.45
Mean grey level (interior)	118.11	117.65
Pixels changed	—	10,763 of 14,400

Edge strength, measured by the mean gradient magnitude, falls by about 29 %, and the Laplacian variance falls from 2645 to 327, a factor of 8.1. Both figures say that high-frequency content has been strongly reduced. The overall contrast is hardly affected (standard deviation 61.4 to 58.4), and the average brightness moves by only -0.46 grey levels. That small drop is the signature of truncation: dropping the four low bits of the sum always rounds down, which on average costs half a grey level. Against the floating-point reference of Eq. (4), the largest difference over the interior is 0.94 grey levels and the mean difference is 0.46, so the fixed-point hardware loses practically nothing compared with an ideal implementation. If the half-level bias mattered for a particular application, adding 8 to the sum before the shift would turn truncation into rounding at the cost of a single extra adder input.

D. Discussion

Three observations stand out. First, the design is limited by memory, not by logic; on a part with only ten RAMB36 tiles, a larger image or a second frame buffer would run out of block RAM long before LUTs became scarce. Second, the timing margin is generous at VGA speeds, which suggests the same engine could process several pixels per cycle or feed a faster video format with little change. Third, using a 3×3 binomial kernel keeps the hardware trivial but gives only a mild blur ($\sigma \approx 0.71$). Larger smoothing can be obtained by cascading the filter, since the convolution of two binomial kernels is again binomial, or by widening the window to 5×5 , which would need two more line buffers and a wider adder tree but still no multipliers.

VII. CONCLUSION AND FUTURE WORK

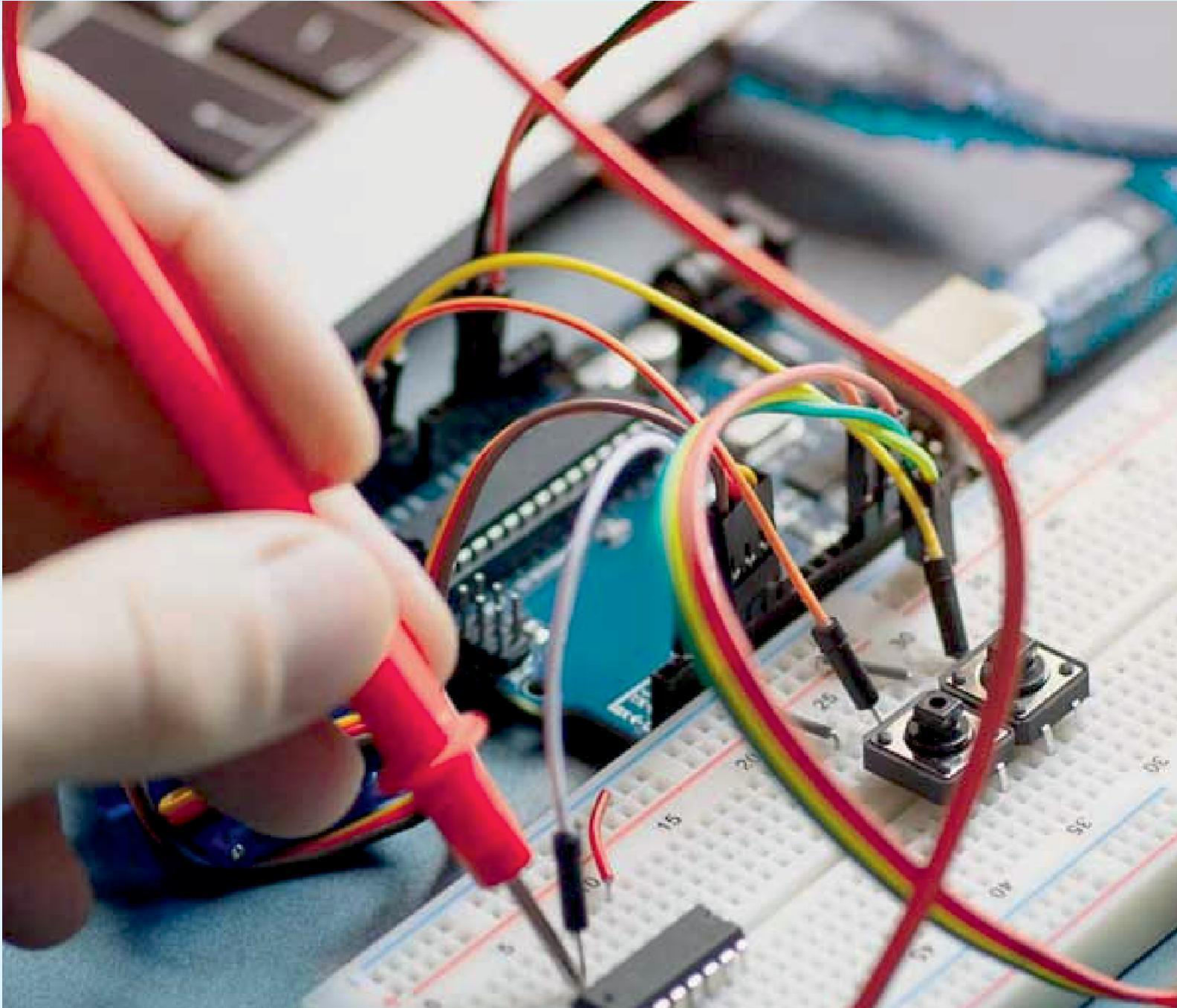
We have described and evaluated a streaming 3×3 Gaussian smoothing filter on a low-cost Spartan-7 FPGA. By choosing the 1-2-1 binomial kernel, the convolution reduces to shifts and additions, and with two line buffers and a nine-register window the engine produces one filtered pixel per clock after a start-up delay of 124 cycles. The complete system, including dual-port image memory, frame memory and a VGA demonstrator, needs only 246 LUTs and 236 flip-flops, meets timing at 25 MHz with 33.3 ns of slack, and draws an estimated 0.141 W, most of it in the clock generator. The image measurements show a clear reduction in high-frequency content with an error of under one grey level relative to an ideal implementation.

REFERENCES

- [1] R. C. Gonzalez and R. E. Woods, Digital Image Processing, 4th ed. New York, NY, USA: Pearson, 2018.
- [2] D. Marr and E. Hildreth, "Theory of edge detection," Proc. R. Soc. Lond. B, Biol. Sci., vol. 207, no. 1167, pp. 187–217, Feb. 1980.
- [3] J. Canny, "A computational approach to edge detection," IEEE Trans. Pattern Anal. Mach. Intell., vol. PAMI-8, no. 6, pp. 679–698, Nov. 1986.
- [4] A. P. Witkin, "Scale-space filtering," in Proc. 8th Int. Joint Conf. Artificial Intelligence (IJCAI), Karlsruhe, Germany, Aug. 1983, pp. 1019–1022.
- [5] T. Lindeberg, "Scale-space theory: A basic tool for analysing structures at different scales," J. Appl. Statist., vol. 21, nos. 1–2, pp. 225–270, 1994.



- [6] D. G. Bailey, Design for Embedded Image Processing on FPGAs. Singapore: Wiley–IEEE Press, 2011.
- [7] C. T. Johnston, K. T. Gribbon, and D. G. Bailey, “Implementing image processing algorithms on FPGAs,” in Proc. 11th Electronics New Zealand Conf. (ENZCon), Palmerston North, New Zealand, Nov. 2004, pp. 118–123.
- [8] H. T. Kung, “Why systolic architectures?,” Computer, vol. 15, no. 1, pp. 37–46, Jan. 1982.
- [9] S. Hauck and A. DeHon, Eds., Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation. Burlington, MA, USA: Morgan Kaufmann, 2008.
- [10] P. J. Burt and E. H. Adelson, “The Laplacian pyramid as a compact image code,” IEEE Trans. Commun., vol. COM-31, no. 4, pp. 532–540, Apr. 1983.
- [11] P. P. Chu, FPGA Prototyping by Verilog Examples: Xilinx Spartan-3 Version. Hoboken, NJ, USA: Wiley, 2008.
- [12] Xilinx, Inc., “7 Series FPGAs memory resources: User guide,” UG473. [Online]. Available: <https://docs.amd.com>
- [13] AMD (Xilinx), “Clocking Wizard LogiCORE IP product guide,” PG065. [Online]. Available: <https://docs.amd.com>
- [14] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, “Image quality assessment: From error visibility to structural similarity,” IEEE Trans. Image Process., vol. 13, no. 4, pp. 600–612, Apr. 2004.
- [15] I. Kuon and J. Rose, “Measuring the gap between FPGAs and ASICs,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 26, no. 2, pp. 203–215, Feb. 2007.



INNO  SPACE
SJIF Scientific Journal Impact Factor



ISSN INTERNATIONAL
STANDARD
SERIAL
NUMBER
INDIA



International Journal of Advanced Research

in Electrical, Electronics and Instrumentation Engineering

 9940 572 462  6381 907 438  ijareeie@gmail.com



www.ijareeie.com

Scan to save the contact details