A Design Space Exploration of Power-efficient Gaussian Filter Architectures using Logical Optimization and Approximated Adders

Marcio Monteiro1, Ismael Seidel1, José Luis Güntzel1, Mateus Grellert2, Leonardo Soares3, Cristina Meinhardt1

1Universidade Federal de Santa Catarina - PPGCC, Florianópolis, Brazil
2Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil
3Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul, Rio Grande, Brazil

e-mail: marcio.monteiro@posgrad.ufsc.br

Abstract— Gaussian filtering is an important step in many image-processing applications because it reduces image noise. However, this step is also compute-intensive, so power-optimized hardware architectures are necessary to allow its adoption in embedded devices. This work presents a design space exploration of power-efficient Gaussian filter architectures. Differently from related work, this work shows the impacts of the design decisions on two target applications: the Canny Algorithm and an Automatic License Plate Recognition system. To explore the design space, the $3 \times 3$, $5 \times 5$, and $7 \times 7$ kernels were logically refactored using Multiplierless Constant Multiplication and Common Sub-expression Exploration. The adders were approximated using the copy strategy to further reduce power consumption. Systematic experiments show the effects of the adopted strategies on power savings and quality of results compared to exact baselines using the two applications. The approximate strategy reached up to 51 dB of Peak Signal-to-Noise Ratio with power reductions of up to 48% in the best-case scenario of the standalone Gaussian filters. Also, the total power of the Gaussian filter in the Canny Algorithm can be reduced down to 34% while maintaining the precision of results between 57% and 90%. Finally, the proposed strategies reduce up to 64% of the Gaussian filter power consumption when adopted in the plate detection solution with a similar detection rate compared to the exact filter architecture.

Index Terms— Gaussian Filter; Approximate Computing; Low-Power Design; Hardware Optimization; Copy Strategy.

I. INTRODUCTION

Power efficiency is a relevant challenge in Integrated Circuit design for power-constrained applications, e.g., edge computing, green computing, smart cities, or the Internet of Things. Many of these designs are error-tolerant applications, such as image processing. In these cases, it is acceptable to relax the output accuracy to a certain degree to increase power, area, or delay savings. The design techniques trading the accuracy of results for power, delay, or area are referred to as Approximate Computing (AxC) [1].

Image processing applications are largely used in power-constrained devices which require hardware accelerators to meet the application power-quality restrictions. In image processing, noise is one of the most common undesirable artifacts. The Gaussian filter is commonly used to reduce noise [2, 3] as it has a good trade-off between suppressing high frequencies and minimizing spatial spread. However, the Gaussian filter is a compute-intensive arithmetic process demanding energy-efficient implementations. As the adders are the core component of the Gaussian filter, AxC techniques for adders can be explored to achieve power efficiency [4]. Generally, these approaches divide the n-bit adders into an accurate part which uses an exact adder, and an approximate part [5]. The approximate part may use approaches based on modifying the adder’s gate organization [6], changing the calculation of the Least Significant Bits (LSBs) [6], modifying the carry propagation [7], or bit quantization of input data [8]. Also, it is possible to combine different AxC techniques to implement the filter [9] or into the same adder design [10].

In [9] we combined the logical refactoring of $5 \times 5$ kernel Gaussian filter architectures with the copy strategy in the word length LSBs of adders. These strategies led to power-quality trade-offs similar to the ones obtained by [8]. In [11] we introduced a multi-kernel evaluation, extending the ideas from [9] to $3 \times 3$ and $7 \times 7$ kernels. We evaluated the three kernel architectures on the Canny Algorithm. The results show that each kernel size impacts the power-quality trade-off differently.

Those findings enabled opportunities to enlarge the design space exploration of power-quality profiles considering the three kernel sizes and their approximate cases. The design space exploration adopts the preliminary approaches and uses the power-quality results to select the most suitable kernel size and the approximate case to meet the power and quality restrictions for the applications under evaluation.

Thus, in this work, we present a design space exploration of Gaussian filter multiple kernel architectures and evaluate its power-quality trade-off in practical applications. We evaluate the quality and power improvements of the proposed power-efficient Gaussian filter architectures in the preprocessing step of an image edge detection algorithm and on an Automatic License Plate Recognition (ALPR) application. The design space exploration expands the findings of our previous work [9] [11] while presenting a detailed and general explanation of the refactoring strategy combined with the approximate approach. Therefore, the main contributions of this work are:

- A refactoring strategy using Multiplierless Constant Multiplication (MCM) and Common Sub-expression Exploration (CSE) to reduce the power and area of the designs;

- A multi-kernel Gaussian filter design space exploration combining the refactoring and approximate computing strategies to effectively assess the power-quality trade-off of this approach in practical applications;
The evaluation of the proposed Gaussian filter architectures powers-quality trade-off in an edge detection algorithm and ALPR application. To the best of our knowledge, this is the first work to discuss the AxC in the ALPR, considering power-efficient Gaussian filter architectures.

This paper is organized as follows. Section II introduces the Gaussian filter and discusses the related work. Section III presents the proposed approaches to optimize the filter. Section IV presents the power and quality results of our proposed Gaussian filter architectures. Section V shows the results obtained by the proposed filters in applications. Finally, Section V draws conclusions and future work.

II. Gaussian Filter

The Gaussian filter is a Finite Impulse Response (FIR) filter used to reduce the noise in an image. It can be applied using a vertical and horizontal 1D convolution or a 2D matrix of coefficients, also called kernel. Each 1D coefficient is obtained using Eq. 1, where ksize is the kernel size, $\sigma$ is the variance, $i$ is an index varying from 0 up to ksize–1, and $\alpha$ is a scale factor that makes $\sum_{i=1}^{ksize} G(i)=1$. The 2D kernel is an N×N matrix obtained by multiplying the 1D kernel by its transpose $(2D=1D×1D^T)$.

$$G(i) = \alpha e^{-\frac{(i-(ksize-1)/2)^2}{2\sigma^2}}$$

In this work, we evaluate the 3×3, 5×5, and 7×7 kernels. The floating-point coefficients for each 1D kernel were obtained using Eq. 1 and a variance of 1.36 [8]. Then, each 1D kernel was multiplied by its transpose to obtain the 2D kernel. The 2D coefficients for each kernel size were rounded to integer values, resulting in an initial approximation. The rounding of coefficients reduces hardware usage and implementation complexity compared to floating-point arithmetic. Eq. 2, 3, and 4 present the integer coefficients used in this work for 3×3, 5×5, and 7×7 kernels, respectively.

$$Y_{3×3}=15i_0+19i_1+15i_2+19i_3+24i_4+19i_5+15i_6+19i_7+15i_8$$

The use of equations to process the blocks increases the design space exploration. It also has more room to apply techniques to improve power efficiency or reduce the area of hardware implementation. Different approaches to exploring the Gaussian filter architectures targeting power efficiency are found in the literature.

A. Related Work

Gaussian filters may be implemented in software or hardware. However, the software solution can be 200× slower than the hardware architecture [12]. The hardware implementation may use FPGA [12], ASIC [5, 8, 9, 10, 13], or both platforms [14, 15]. Different approaches have been used to reduce the number of arithmetic operations of the Gaussian filter. Some examples are optimizing memory access [16], exploring the separability of the kernel [17], or combining mirror short pixel approximation and rounding off the coefficients [18]. However, these approaches have a limited impact on power and area. Further optimization can use voltage over scaling, simple instruction multiple data, or AxC approaches [10].

In [13], an imprecision parameter fine-tuning Gaussian filter 3×3 kernel using Lower-OR Part Adder (LOA) is proposed, reducing power over 53% in the best scenario and with impact on the quality by 71% in the worst-case scenario. A floating-point design space exploration to 3×3, 5×5, and 7×7 kernels using precision scaling and memo- 

ization in FPGA is presented in [12]. It considers four filter variations: 1) a 2D kernel using multipliers and adder tree, 2) reordering coefficients, 3) 1D based on the separability property [17], and 4) Look-up Table (LUT)-based. The quality varied from 30 dB to 82 dB for 3×3 kernel, while 5×5 and 7×7 kernels range from 30 dB up to 45 dB. Variations 1, 2, and 4 achieved 8% power reduction for 3×3 kernel and 4% for 7×7 kernel, using variation 3 as a reference. The optimizations in the design space were limited as the architectures use general-purpose floating-point adders and multipliers. Also, floating-point operations are more complex and demand more hardware resources to implement than integer operations.

The 5×5 kernel approximation may use hybrid adders [5, 10], combine different AxC techniques [14, 9, 11], or explore configurability property [8, 19]. In [10], the authors proposed a hybrid adder using copy strategy, Error-Tolerant Adder Type I (ETAI), and truncation that reduces power for the target quality thresholds of 30 dB and 50 dB. The architec- ture synthesis and simulation used the 45 nm Nangate
III. PROPOSED APPROACHES TO OPTIMIZE THE POWER OF GAUSSIAN FILTER ARCHITECTURES

This section presents our design approach to improve the power efficiency of Gaussian filters, which consists in refactoring the kernel equations and using Approximate Computing (AxC). The refactoring combines the MCM and CSE techniques to reduce the number of arithmetic operations, consequently reducing the hardware implementation complexity. The approximated adders were designed by using the copy strategy to reduce power and area. Each Gaussian filter architecture was divided into levels that may use exact or approximated adders to evaluate the impacts of approximation in power and quality.

A. Power Optimization by Refactoring the Gaussian filter Equations

As a first step, we applied logic refactoring to the three kernel functions to reduce the power consumption of the Gaussian filter architecture. The 5×5 kernel architecture is the most frequently found in the related work. Thus, we explain the logic refactoring solution for this kernel size. Fig. 1 presents the 5×5 kernel architecture from [8], which was adopted as baseline architecture in this work. Then, we show the optimized functions for the 3×3 and 7×7 kernels.

The logic refactoring uses MCM to decompose the kernel coefficients into the equivalent sum of power-of-two ($2^n$) components, where $n$ is the number of bits to shift the input value. The decomposition allows the substitution of the general-purpose multipliers by sums and shifts. For example, the MCM of $5\xi_2$ from Eq. 6 is $(4+1)\xi_2$, or $(2^2+2^0)\xi_2$. The CSE reduces the number of arithmetic operations by calculating the common terms only once and reusing the result. For example, the CSE of $8(a+b+c)+7(b+c+d)$ is $8(a+\xi)+7(\xi+d)$ with $\xi=b+c$. Table I presents the integer and power-of-two decomposition of the coefficients (Coef.) for the three kernels. The pure power-of-two coefficients were omitted in this table. Among these decompositions, the constant 15 can be optimized to $2^4-2^0$ instead of $2^4+2^2+2^1+2^0$, reducing it to a single subtraction.

The first step of logic refactoring is to optimize Eq. 6 using MCM. Eq. 8 is obtained by replacing each coefficient on Eq. 6 with its decomposition from Table I. The equation implemented by the baseline architecture [8] shown in Fig. 1 is similar to Eq. 8, with the coefficients reordered.

![Fig. 1: Baseline Gaussian filter architecture of 5×5 kernel from [8]. RB is the register barrier, A’s are adders, S’s are subtractors, Y is the output, and >>, << are shifters. Modified from [8].](image-url)
Table I. Non-power-of-two coefficients (Coef.) decomposition for the three kernels.

<table>
<thead>
<tr>
<th>Coef.</th>
<th>Integer values</th>
<th>Power-of-two values</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>4 + 1</td>
<td>$2^2 + 2^0$</td>
</tr>
<tr>
<td>9</td>
<td>8 + 1</td>
<td>$2^3 + 2^0$</td>
</tr>
<tr>
<td>10</td>
<td>8 + 2 or 2(4 + 1)</td>
<td>$2^3 + 2^0$ or $2^2(2^2 + 2^0)$</td>
</tr>
<tr>
<td>12</td>
<td>8 + 4 or 4(2 + 1)</td>
<td>$2^3 + 2^0$ or $2^2(2^3 + 2^0)$</td>
</tr>
<tr>
<td>13</td>
<td>8 + 4 + 1</td>
<td>$2^3 + 2^2 + 2^0$</td>
</tr>
<tr>
<td>15</td>
<td>8 + 4 + 2 or 16 + 1</td>
<td>$2^3 + 2^2 + 2^0$ or $2^2 - 2^0$</td>
</tr>
<tr>
<td>19</td>
<td>16 + 2 + 1</td>
<td>$2^4 + 2^3 + 2^2$</td>
</tr>
<tr>
<td>24</td>
<td>16 + 8 or 8(2 + 1)</td>
<td>$2^4 + 2^3$ or $2^2(2^3 - 2^0)$</td>
</tr>
</tbody>
</table>

The second step reduces the number of arithmetic operations on Eq. 8 using the CSE. First, the coefficients with more than one power-of-two value are multiplied by $i_k$ to obtain an equation with all terms in the form $\Omega i_k$, where $\Omega$ is a power-of-two constant, e.g., $(4 + 1) i_2$ from Eq. 8 is replaced by $4 i_2 + i_2$. Then, the power-of-two terms are put in evidence, as shown in Eq. 9, to identify the redundant terms. To find the redundant terms, we used the CSE function from the Symbolic Mathematics in Python (SymPy) library [21]. The output of SymPy was the equation with the redundant terms represented by variables calculated only once. For example, using Eq. 9 as input to SymPy leads to Eq. 10 with $\psi = i_2 + i_{10} + i_{14} + i_{22}$, $\delta = i_6 + i_8 + i_{16} + i_{18}$, and

$$\tau = i_7 + i_{11} + i_{13} + i_{17}.$$
the different hardware implementation complexity. It shows that logic refactoring impacts the hardware implementation of the kernel under evaluation without quality degradation.

The proposed refactored equations reduced the number of sums from 36 to 27 operations (25%), from 20 to 11 (45%), and from 54 to 47 (13%) for 5×5, 3×3, and 7×7 kernels, respectively. Furthermore, the refactoring also modifies the word length of the adders. Table II presents the number of adders per bit-width used by the 3×3 kernel, the two 5×5 kernels (baseline [8] and optimized), and 7×7 kernel architectures. The optimized architecture needs fewer bits than the baseline and exhibits a different bit-width distribution. The baseline architecture [8] has most of its adders between 9-bit and 13-bit, while the optimized architecture has most of the adders with 8-bit to 10-bit. Another contribution from the change in the bit-width distribution of the 5×5 kernel relies on moving the left shifts from just after the input register barrier in [8] to the closest possible to the output (Y) in our proposed architecture. Also, the change in the bit-width reduced 34% the number of full adder cells while maintaining the precision of results. Comparing the architectures for 3×3, 5×5 optimized, and 7×7 kernels, the larger number of adders’ bit-width is between 8-bit and 10-bit. These adders are used in the first levels of the adder tree, which contains the largest number of operators.

<table>
<thead>
<tr>
<th>Bit-width</th>
<th>3×3</th>
<th>5×5</th>
<th>7×7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline [8]</td>
<td>1 1 3 13 10 12 14 15 16</td>
<td>420</td>
<td></td>
</tr>
<tr>
<td>Optimized</td>
<td>1 1 3 13 10 6 2 2 1</td>
<td>275</td>
<td></td>
</tr>
<tr>
<td>3×3</td>
<td>1 1 4 3 2 - - - -</td>
<td>114</td>
<td></td>
</tr>
<tr>
<td>5×5</td>
<td>1 1 7 4 2 - - - -</td>
<td>114</td>
<td></td>
</tr>
<tr>
<td>7×7</td>
<td>2 1 23 12 6 4 1 1 1 -</td>
<td>454</td>
<td></td>
</tr>
</tbody>
</table>

Table II: Adder bit-width distribution for 3×3, 5×5, and 7×7 kernel architectures.

B. Exploring Levels of Approximation on Gaussian Architectures

We introduced approximate adders in a five-level division of the previously optimized architectures to further reduce power and area. The first step to optimizing power with approximate adders consists in selecting an approximate strategy and then, in the second step, determining the number of bits approximated in the adders.

B.1 Approximate Adder Strategy: The first decision is to select the AxC approach to approximate the adders. The copy strategy significantly reduces the power and area by substituting the full adder cell with buffers to connect the inputs to the outputs despite the 50% error probability [6]. Thus, we selected the copy strategy because it has a good compromise between the quality of results and power/area reduction.

The n-bit adder is divided into the precise part using exact adders and the approximate part using the copy strategy, as shown in Fig. 3. We let the synthesis tool select the adder topology to implement the precise part of the approximated adder. In the approximate part, input A was connected to the output S, and carry-in (Cin) to the precise part is the B_{(n/2)−1} input. Also, copy strategy allows the removal of the carry propagation on the approximate part, reducing the delay and increasing the power and area savings.

\[ ENMED = \frac{1}{2^{(bit-width−#approx+2)}} \]  

Fig. 3: N-bit copy strategy adder. Modified from [11].

B.2 Number of Exact / Approximate Bits: A second decision regarding the approximation is to define the number of bits to approximate. To determine it, we evaluated different numbers of bits approximated using the copy strategy in various bit-widths. To do it, we used an adder model in C++ and approximating 25%, 50%, 75%, and 100% of LSBs word length, considering different adder bit-widths.

A traditional metric for evaluating error is the Normalized Mean Error Distance (NMED) [22], where the average error is divided by the representation of the maximum value of a given bit-width \((2^{bit-width−1})\) and the error is defined as the absolute difference between the exact result and the approximate result considering the same inputs. However, obtaining the adder’s NMED considering all combinations of inputs could be a time-consuming task. To avoid this, we proposed the Estimated Normalized Mean Error Distance (ENMED) metric to estimate the value of NMED as presented in Eq. 13. This new metric provides exactly the same values obtained by the traditional NMED in a few seconds using only the adder’s bit-width and the number of approximate bits (#approx). The approximate case of each adder was evaluated using the ENMED metric considering different word lengths and number of bits approximated, resulting in the data presented in Fig. 4. Considering the results, approximating all bits achieves a constant ENMED of 25% independently of the word length while partially approximating the adder’s bit-width reduces the ENMED between 3.125% and 12.5%, as presented in Fig. 4. Among the four evaluated word length percentages, approximating 50% of bits achieves a good trade-off between the precision loss and the number of bits approximated. Based on this fact, we decided to approximate the half LSBs of the adders’ bit-width used in the Gaussian filter architectures.

Fig. 4: Estimated Normalized Mean Error Distance (ENMED) of adders with bit-width from 1 to 8 bits approximating 25%, 50%, 75%, and 100% of word length.
B.3 Approximate Levels: The adder tree from each architecture is divided into five levels to assess the impacts of approximation in the three kernel architectures. These levels are highlighted by different colors in Fig. 2b, Fig. 2a, and Fig. 2c for 5×5, 3×3, and 7×7 kernels, respectively. The levels were identified based on the depth of the adder in the adder tree. The first level (L1) comprises the adders close to the register barrier, while level 5 (L5) contains the adders closest to the output Y. Each level could use precise or approximated adders, totaling 32 possible Gaussian filter cases for each kernel. The case that uses only precise adders is named exact, and C_number identifies the approximated cases. For the sake of compactness, cases with very similar low-quality outputs are omitted and represented by the case that uses approximate adders only, which is the most aggressive. Hence, from all 32 possible cases, in this work, we evaluated, in terms of area and power, only cases C1 to C17. Table V presents the mapping of each case combination and its levels approximated (AxC Level), and the results are discussed in the next section.

IV. Power and Quality Evaluation of the Proposed Optimizations

The power and quality evaluation of the proposed optimizations is performed, firstly considering the impact of refactoring the exact Gaussian architectures, then evaluating the approximate architectures considering power and accuracy outcomes.

The evaluation of the Gaussian filter architectures adopts the following configurations: The architectures were described in Verilog HDL and synthesized using Synopsys DC© [23] in Topographical mode. The input and output delays were set to 60% of the clock period. The maximum primary input and output capacitances were set to 10× and 30× 2-input AND gate, respectively. The syntheses used the TSMC 45 nm technology at a nominal voltage of 0.9 V. The frequency was 249 MHz to allow processing images at 4k@60fps [8]. The architectures were simulated using Synopsys VCS© to perform functional verification and obtain realistic switching activity. All architectures process one block of pixels per clock cycle. We adopted ten images from Berkeley Segmentation Dataset and Benchmark (BSD) [24] to evaluate the Gaussian filter architectures.

The rounding of coefficients from floating-point to integer values is a first approximation that impacts the quality of results, area, and power. We evaluated its impact using the BSD images and the Peak Signal-to-Noise Ratio (PSNR) metric [25]. We obtained 50 dB, 35 dB, and 46 dB to 3×3, 5×5, and 7×7 kernels with respect to the floating-point OpenCV Gaussian filter implementations.

A. Refactoring Impact

The refactoring of the kernels is explored to improve power efficiency. Reductions in the proposed architectures’ power and area are expected due to the reduction in the number of adders and bit-width. In this step, we evaluate the exact architectures proposed in the refactoring approach to observe the impact on power and delay. Firstly, we compare the optimized 5×5 kernel with the baseline from [8]. After that, we evaluate the area and power results for the exact 3×3 and 7×7 kernels compared with the 5×5 kernel optimized. This evaluation allows us to identify the energy profile of the different kernel sizes.

Table III presents the area, leakage (Leak.), and total power (Total) results of the 5×5 baseline [8] and optimized architectures. Our refactored solution architecture presents an area reduction of 12%, while the total power reduction is 20%. It shows that only logically refactoring the architecture reduces the power and area, maintaining the precision of the results. Also, the average leakage represents 6% of the total power on both architectures. Based on these findings, we extended the logic refactoring to include other kernel sizes.

Table IV presents the normalized area and power results of the exact case for 3×3 and 7×7 kernels with respect to 5×5 kernel. The 3×3 kernel reduces 55% of total power and 57% of the area while the 7×7 kernel increases 1.2× the total power with 53% of area increment. It shows that a 3×3 kernel is more suitable for power-constrained applications as it has the lowest power consumption. In contrast, the 5×5 kernel is most appropriate for relaxed power-constrained applications. Finally, the 7×7 kernel has the largest power and area results, making it more useful in unconstrained power applications.

Table V. Area and power of exact 5×5 kernel architectures.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Power (W)</th>
<th>Area (mm²)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline [8]</td>
<td>33</td>
<td>628</td>
</tr>
<tr>
<td>Optimized</td>
<td>28</td>
<td>503</td>
</tr>
<tr>
<td>Reduction (%)</td>
<td>15%</td>
<td>20%</td>
</tr>
</tbody>
</table>

Table IV. Absolute and normalized area and power results for exact 3×3 and 7×7 kernels regarding 5×5 kernel.

Table V. Normalized to 5×5

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Power (µW)</th>
<th>Area (µm²)</th>
<th>Power</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>5×5</td>
<td>28</td>
<td>503</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3×3</td>
<td>11</td>
<td>226</td>
<td>0.39</td>
<td>0.45</td>
</tr>
<tr>
<td>7×7</td>
<td>44</td>
<td>1097</td>
<td>1.57</td>
<td>2.18</td>
</tr>
</tbody>
</table>

The power and area reductions presented in Table IV are the upper bounds achieved by the logic refactoring for the baseline architecture [8]. One alternative to further reduce the power and area of the optimized architecture is combining logic refactoring with other techniques. Approximate Computing (AxC) approaches are suitable candidates to reduce the power and area such as applications that use Gaussian filters are error-tolerant.

B. Approximate Architectures Evaluation

We developed a C++ framework presented in Fig. 5 to evaluate the quality of results for the three kernels with the approximate strategy proposed. The Gaussian filter step in the framework uses the proposed architectures, while the noise addition, block division, and PSNR steps were implemented using OpenCV functions. The noise addition emulates a real-world image capture adding a Gaussian additive noise with a mean of 0 and variance of 0.1 [26]. Then, the block division splits the image into blocks the same size as the kernel size of the Gaussian filters evaluated in the third
step. For example, to evaluate the 3×3 kernel, the block division splits the image into 3×3 blocks. The final step evaluates the PSNR of the output image using the exact case of our proposed architectures as the reference.

![Workflow](image)

Fig. 5: Workflow used to evaluate the proposed Gaussian filter approximated cases with respect to the exact case.

The results of the three kernels are presented in Table V, which shows each case’s name, approximate levels (AxC Level), the number of bits approximated per case (# AxC bits), and PSNR results for the Gaussian filter architectures. The PSNR results were the selection criteria to choose the cases to evaluate the area and power. It reduced the space exploration to 18 cases per kernel size: the exact case, the 15 cases with the best PSNR, and the corner cases C5 and C17.

Evaluating the area reductions of the three kernels, for a single level approximated (C1 to C5), L1 achieves the highest area reductions, as it has the larger number of bits approximated. On the other hand, the approximated level with the lower area reduction depends on the kernel size. The most notable case is C5 in 7×7 kernel which has no impact in the area while reducing 10% and 18% for 3×3 and 5×5 kernels, respectively. Considering the cases with multiple levels approximated (C6 to C17), the cases with the larger area reductions approximate L1. In conclusion, L1 is the most crucial level to approximate if the main goal is to reduce area.

Considering the quality of results, the cases combining the approximation on Levels 1 to 4 (L1 to L4) obtained PSNR above 30 dB, which according to [13] is considered an acceptable result for many applications. On the other hand, all cases with L5 approximated obtained around 16 dB of PSNR, meaning a significant quality degradation. The cause of the quality reductions could be that the intermediate L5 results were not right-shifted, maintaining the errors generated by the copy strategy. In contrast, the intermediate results for cases L1 to L4 approximated were right-shifted in L5 discarding bits, which reduces the approximation errors. Fig. 6 presents the images obtained using the 5×5 kernel exact case (Fig. 6b), C4, which achieves the best quality (Fig. 6c) and all levels approximated (Fig. 6d).

![Table V](image)

Table V: Power and area reductions, quality of results in Peak Signal-to-Noise Ratio (PSNR), and the number of bits approximated (# AxC bits) to all three kernel cases. The PSNR and power and area reduction reference is the exact case.

In an overall evaluation, the proposed approaches optimized the power consumption of Gaussian filter architectures by exploring logic refactoring and the copy strategy. Only refactoring the 5×5 kernel function reduces 20% of the power and 12% of the area without affecting the quality of results. However, the different combinations of approximated levels obtained a variety of power and quality scenarios. Also, the results showed that approximating L5 is the primary culprit of the quality degradation of results and thus, its approximation should be avoided.

C. Comparison with Related Work

Table VI summarizes the main features observed in the related work, including the AxC technique, the kernel size (ksize), power reduction (Power Red.), image quality (PSNR), technology (tech), and frequency (freq). Our exact architecture using only logic refactoring reduced power by 12% and area by 20% without quality degradation when compared to [8]. Also, most related work presents only the interval of power reduction and quality for each kernel. Differently, our proposal offers a set of results for each kernel size to select the most suitable power-quality trade-off subject to the application's restrictions.

Considering the kernel sizes, the 3×3 kernel was also evaluated by [13, 14, 12], whereas the remaining related work targets the 5×5 kernel. Solely [12] includes the 7×7 kernel in its evaluations. Furthermore, the level division evaluation considering multi-kernel is an approach restricted to our previous work. Moreover, we can observe different platforms explored, making the power comparison unappropriate. The work in [12] adopted a floating-point FPGA.
implementation, increasing the results’ quality with a high hardware cost. The 5×5 kernel used by [5, 10, 8] is the same as ours, allowing the comparison considering the results we obtained using our environment for the exact architecture from [8] as the baseline. Table VII presents the total power (Tot. Power) estimations, the average energy per sample (E/Sam.), and the power reduction targeting three thresholds defined in [10].

Table VI: Related work summary.

<table>
<thead>
<tr>
<th>Work</th>
<th>AsC Approach</th>
<th>ksize</th>
<th>Power Red. (%)</th>
<th>PSNR (dB)</th>
<th>tech (nm)</th>
<th>freq (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[13]</td>
<td>LOA</td>
<td>3</td>
<td>53</td>
<td>17-57</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>[14]</td>
<td>copy strategy</td>
<td>3</td>
<td>44</td>
<td>38</td>
<td>31</td>
<td>100 to 1000</td>
</tr>
<tr>
<td>[5]</td>
<td>copy strategy &amp; ETAI</td>
<td>5</td>
<td>26</td>
<td>27-37</td>
<td>45</td>
<td>100</td>
</tr>
<tr>
<td>[10]</td>
<td>copy strategy &amp; ETAI &amp; truncation</td>
<td>5</td>
<td>8-73</td>
<td>24-57</td>
<td>45</td>
<td>63</td>
</tr>
<tr>
<td>[12]</td>
<td>Precision Scaling &amp; Memorization</td>
<td>3</td>
<td>8</td>
<td>30-82</td>
<td>FPGA</td>
<td>N/A</td>
</tr>
<tr>
<td>[8]</td>
<td>closest truncation</td>
<td>3</td>
<td>13-68</td>
<td>28-34</td>
<td>65</td>
<td>65 &amp; 249</td>
</tr>
<tr>
<td>This work</td>
<td>copy strategy</td>
<td>5</td>
<td>1-49</td>
<td>15-51</td>
<td>45</td>
<td>249</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Work</th>
<th>Power Red. (%) to 30 dB</th>
<th>Power Red. (%) to 40 dB</th>
<th>Power Red. (%) to 50 dB</th>
</tr>
</thead>
<tbody>
<tr>
<td>[5]</td>
<td>25</td>
<td>471</td>
<td>1.9</td>
</tr>
<tr>
<td>[8]</td>
<td>13</td>
<td>546</td>
<td>2.2</td>
</tr>
<tr>
<td>[10]</td>
<td>8</td>
<td>578</td>
<td>2.3</td>
</tr>
<tr>
<td>This work</td>
<td>59</td>
<td>258</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Table VII: Related work estimation of total power (Tot. Power), energy per sample (E/Sam.), and power reductions (Power Red.) for thresholds 30 dB, 40 dB, and 50 dB [10].

Evaluating the results, it can notice that our proposed architecture achieves higher power reductions as it now includes the impact of logic refactoring. Comparing [8, 10] and our proposed architecture, they achieve similar energy intervals per sample, although the power reductions differ due to the different approaches used in each work. Also, our proposal achieved similar results to [5] despite the different divisions of the adder tree to approximate the adders.

Regarding the threshold quality, the 30 dB threshold is the most aggressive as it allows the approximation of more bits, resulting in high power reductions than the other thresholds. On the other hand, the 50 dB threshold is the most restrictive threshold to the number of bits approximated, diminishing the power reductions. Considering the three thresholds, our proposal achieved higher power reductions than the counterparts. Unfortunately, [5] does not provide the power reductions for the thresholds, while [10] does not provide results to 40 dB. On the other hand, [8] achieves the maximum PSNR of 42 dB missing the 50 dB threshold.

V. Evaluating the Proposed Gaussian Filters in Practical Applications

The literature review shows us that there is still a lack of works evaluating approximate Gaussian filters in real-world applications to understand the impacts on the quality of results. In this work, we selected two image applications to evaluate the impact on the output quality using our proposed Gaussian filter architectures: the Canny Algorithm for edge detection and an ALPR application for plate detection. Both applications use the Gaussian filter as a preprocessing step to enhance the input image and improve the results’ quality. All the experiments follow the same experimental setup adopted to evaluate the Gaussian filter architectures in Section IV. It is important to highlight that the ALPR uses as a benchmark the BR subset of the openALPR dataset [27].

A. Canny Algorithm

The Canny Algorithm is one of the most used edge detectors in image processing applications, including medical image processing and object-tracking applications. It is considered a standard due to the results’ reliability with noisy

![Example of 5×5 kernel output images. Peak Signal-to-Noise Ratio (PSNR) reference is the exact case.](image_url)
images [28]. The Canny Algorithm is implemented in multiple steps [29], as shown in Alg. 1. The first step enhances the image using a Gaussian filter, then the edge gradient and pixel directions are determined using the Gradient step. The Non-Maximal Suppression (NMS) step removes the pixels that do not contribute to the edges of the image. Finally, the thresholding selects the edges added to the output image.

Algorithm 1: Canny Algorithm

**Input:** Gray-scale image
**Output:** Edge image
1. `filteredImage = GaussianFilter(inputImage)`
2. `gradientImage = Gradient(filteredImage)`
3. `nmsImage = NMS(gradientImage)`
4. `edgeImage = thresholding(nmsImage)`

To evaluate the Gaussian filter in the Canny Algorithm, we extended the framework from Fig. 5 to include the Canny function from OpenCV. The extended workflow is shown in Fig. 8, where the first three steps are the same as in Fig. 5, and the fourth is the OpenCV Canny function. The final step evaluates the quality of the solutions. Since the edges are generally represented as white pixels (i.e., 255) and the non-edge information as black pixels (i.e., 0), the differences between precise and approximate images may indicate four possible responses: i) true positives (tp), where the approximate version detects an edge pixel also detected by the precise one; ii) false negatives (fn), where the approximate version does not detect an edge pixel which is detected by the precise one; iii) false positives (fp), where the approximate version detects an edge pixel which is not detected by the precise one and iv) true negatives (tn), where the approximate and exact versions do not detect an edge pixel. Based on that, two indicators are used to evaluate the performance of a given edge detector: recall (Eq. 14) and precision (Eq. 15). The recall can be defined as the ratio between the number of edge pixels correctly detected by the approximate version and the ones that should be correctly detected by the precise Canny Algorithm. The precision could be defined as the ratio between the number of pixels correctly detected as edges and the total number of pixels detected as an edge by the approximate version. The Performance Conformance (PfCf) metric shown in Eq. 16 relates these two metrics, and it is mostly used to evaluate the edge detection [30].

\[
recall = \frac{tp}{tp + fn} \quad (14) \quad precision = \frac{tp}{tp + fp} \quad (15)
\]

\[
PfCf(\%) = \min(\text{recall}, \text{precision}) \times 100 \quad (16)
\]

![Fig. 8: Extended framework used to evaluate the Gaussian filter in Canny Algorithm.](image)

We extend the power-efficient approaches in this experiment, considering turning off the Gaussian filter. It is the most aggressive approach to reduce power consumption due to avoiding the use of the Gaussian filter in the Canny Algorithm. The results show that the Gaussian filter step is responsible for 13%, 19%, and 11% of the total execution time in the Canny Algorithm for the $3 \times 3$, $5 \times 5$, and $7 \times 7$ kernels, respectively. The processing time indicates the relevance of the design of power-efficient hardware for the Gaussian filter, and it can impact the total power in the Canny Algorithm solution. However, removing the Gaussian filter reduces the PfCf to 20%, 7%, and 7% for $3 \times 3$, $5 \times 5$, and $7 \times 7$ kernels, respectively. The PfCf for Gaussian filter with integer coefficients is 86% for $3 \times 3$ kernel, 41% for $5 \times 5$ kernel, and 78% for $7 \times 7$ kernel.

Fig. 9 presents the power reduction and the PfCf obtained for the 16 cases evaluated using the Canny Algorithm. The findings from [11] show that approximating L5 (C5 and C17) has the largest impact on quality, reducing the PfCf by nearly 50% in the best case. On the other hand, approximating L1 to L4 has a negligible effect on quality, with up to 44% power reduction. It shows that the Canny Algorithm achieved a good trade-off between power and quality using the approximate cases. Fig. 10 presents examples of Canny Algorithm output using the $5 \times 5$ kernel from OpenCV (Fig. 10a), the best power-quality trade-off (Fig. 10b), and the lowest quality (Fig. 10c) using our approximated $5 \times 5$ kernel, and disabling the Gaussian filter (Fig. 10d). A visual analysis of Fig. 10d shows the removal of the Gaussian filter makes the edge detection unfeasible, while using L5 approximated makes the edge detection costly and reduces the quality. These results show the relevance of the Gaussian filter in the Canny Algorithm to improve the quality of results.

**B. Automatic License Plate Recognition (ALPR) System**

The second practical application selected was an Automatic License Plate Recognition (ALPR) system. ALPR recognizes the license plate characters from an image or video using object detection, image processing, or pattern recognition [31]. It is a relevant application in daily activities like traffic management, vehicle monitoring, and automatic toll payment [32]. However, the ALPR system must be resilient to environmental interference as it is exposed to different plate configurations and weather and image conditions [31].

A common approach to implementing ALPR is to divide it into four steps [32], as shown in Alg. 2. The first step (line 1), `PreProc`, is responsible for enhancing the image using the Gaussian filter. The second step (line 2), `LocatePlate`, finds the plate location in the image and determines the plate orientation and size to compensate for the plate’s skew. The third step (line 3), `CharSegmentation`, uses segmentation techniques to find the alphanumeric plate’s characters. Finally, `CharRecognition` (line 4) identifies each character and performs the syntactical and geometrical analysis to determine the plate’s characters.

We are considering an ALPR based on computer vision algorithms that adopt the Gaussian filter as the pre-processing step. We adopted the License Plate EXtractor (LPEX) [33] to evaluate our proposed filter architectures in the ALPR. The LPEX restricts its implementation to the `PreProc` and the `LocatePlate` steps from Alg. 2 and uses the floating-point arithmetic Gaussian filter from OpenCV. In this work, we only...
modify the first step (PreProc) of LPEX to use our proposed Gaussian filter architectures. We named the LPEX implementation using the floating-point Gaussian filter in the PreProc step as default LPEX, while the one using our proposed Gaussian filter architectures as modified LPEX.

Firstly we determined the impact of the Gaussian filter on the total execution time and the precision using the default LPEX and the Detection Plate Success (DPS) metric [34]. DPS is defined by Eq. 17 as the rate of the plates correctly detected and the number of images in the dataset.

\[
DPS = \frac{\text{Plates\ detected}}{\text{Total images}} \times 100 \tag{17}
\]

The default LPEX achieves 72% of DPS while suppressing the Gaussian filter decreases the DPS to 38% at a reduction of only 0.25% of the total execution time. It shows that using the Gaussian filter as a pre-processing step is relevant to achieve high-precision results in LPEX, albeit its lower share in the total execution time. Our LPEX evaluation assumes that its steps are implemented in the same device. Then the Gaussian filter’s short total execution time could result from using complex and time-consuming morphology and thresholding functions to implement the LocatePlate step. However, using LPEX in edge computing or IoT scenarios, the pre-processing step may be implemented in a power-restricted sensor or camera and the other ALPR steps in a device on edge. Also, other applications may use the same camera or sensor, changing the Gaussian filter total execution timeshare and power consumption.

The impacts on the precision of our proposed Gaussian filter architectures were evaluated by comparing the DPS of the default LPEX and the modified LPEX. The default LPEX achieved 72% of DPS, while the modified LPEX using our exact proposed architectures achieved 70% of DPS for the 5×5 and 7×7 kernels and 60% for the 3×3 kernel. The difference between the default and the exact version is due to the adoption of integer arithmetic, while the default LPEX uses floating-point arithmetic Gaussian filter. The 3×3 kernel reduces DPS due to the reduced number of coefficients than the other kernels, making the error of rounding coefficient more relevant.

To evaluate the power-quality trade-off of ALPR considering Approximate Computing, we used the approximated 5×5 kernel cases in the PreProc step of the modified LPEX. The 5×5 kernel was selected because it achieved lower power and area with similar DPS compared to the 7×7

Algorithm 2: Automatic License Plate Recognition

**Input:** Image

**Output:** Plate characters

1. filteredImg = PreProc(inputImage)
2. plateLocImg = LocatePlate(filteredImg)
3. locatedCharsImg = CharSegmentation(plateLocImg)
4. plateVal = CharRecognition(locatedCharsImg)

execute

Table VIII. Detection Plate Success (DPS) of License Plate EXtractor (LPEX) using our approximated 5×5 kernel architecture and power reductions of the approximate cases subject to the exact case. The bold DPSs are the best results.

<table>
<thead>
<tr>
<th>Case</th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
<th>C5</th>
<th>C6</th>
<th>C7</th>
<th>C8</th>
<th>C9</th>
<th>C10</th>
<th>C11</th>
<th>C12</th>
<th>C13</th>
<th>C14</th>
<th>C15</th>
<th>C16</th>
<th>C17</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPS (%)</td>
<td>65</td>
<td>73</td>
<td>70</td>
<td>69</td>
<td>43</td>
<td>68</td>
<td>69</td>
<td>70</td>
<td>64</td>
<td>70</td>
<td>71</td>
<td>66</td>
<td>64</td>
<td>70</td>
<td>61</td>
<td>70</td>
<td>41</td>
</tr>
<tr>
<td>Power Red. (%)</td>
<td>33</td>
<td>30</td>
<td>35</td>
<td>40</td>
<td>39</td>
<td>48</td>
<td>45</td>
<td>47</td>
<td>36</td>
<td>41</td>
<td>42</td>
<td>55</td>
<td>58</td>
<td>53</td>
<td>44</td>
<td>64</td>
<td>68</td>
</tr>
</tbody>
</table>
Table VIII presents the LPEX precision and power results to approximated 5×5 kernel cases. The first row contains the cases, the second and third rows the DPS, and the power reduction subject to the exact case. The seven cases with equal or better DPS than the exact case are highlighted in bold in Table VIII. Our Gaussian filter exact case obtained total power of 447.6 nW. Evaluating the DPS and the power reduction, cases with L5 precise the DPS varied from 61% to 73%, while power reduction ranged from 30% to 64%. The cases C5 and C17 reached DPS around 43% with power reductions of 39% and 68%, respectively. Analyzing the results for a single level approximated (C1 to C5), cases C2 and C3 achieved DPS equal to or better than the exact case with 30% and 35% power reductions, respectively. Comparing the results for multi-level (C6 to C17), cases C8, C14, and C16 have the best power-quality trade-off as they achieved the same quality as the exact case while reducing power up to 64%. These three cases have in common the levels L1 and L4 approximated. Also, more than half of the approximate cases achieved DPS between 68% and 73%, which is close to the default LPEX implementation. It shows that using AxC in LPEX could maintain the quality of results compared to the exact case while reducing power.

Fig. 11 presents the total power and DPS for cases with L5 exact to illustrate better the power and power trade-off. Evaluating the seven approximated cases with equal or better DPS than the exact case, except for cases C2 and C3, the cases approximated Level 4, pointing out its importance in reducing power while maintaining precision. Among these seven cases, C2 has the highest DPS (73%) among all cases evaluated, with a lower power reduction (30%). On the other hand, C16 has a more considerable power reduction (64%) with the same DPS than the exact case. Regarding the cases with lower DPS than the exact case, there is an equivalent case with better DPS and similar power reduction. It shows that those cases should be avoided to implement LPEX in a power-constrained device.

In an overall evaluation of the LPEX results, using the Gaussian filter as a pre-processing step significantly improves the quality of results. However, the Gaussian filter has a short total execution time in LPEX as the LocatePlate step uses time-expensive functions. Considering the results of the approximate case, there are cases with significant power reduction while maintaining the quality of results similar to the default LPEX. Also, it shows an opportunity to explore other approximation techniques in ALPR steps to assess the impacts on power and the quality of results. Finally, our experiments show that ALPR is resilient to errors and maybe a relevant real-world application to evaluate approximate adders using approaches other than the copy strategy.

VI. Conclusion

In this work, we present a design space exploration considering three Gaussian filter kernels to evaluate two practical applications: Canny Algorithm and an Automatic License Plate Recognition (ALPR) system. The approximated filter cases evaluated in the Canny Algorithm reduced 48% of power in the best-case scenario. The results also show that the 3×3 kernel is more appropriate for power-constrained applications, the 5×5 kernel is more suitable for quality-constrained applications, and the 7×7 kernel is more convenient for unrestricted power applications. The Gaussian filter can improve 32% the quality of results of the ALPR showing its importance to achieving high-quality results. Comparing the ALPR using our proposed architectures and the default implementation, the 3×3 kernel reduced the quality of results, while 5×5 and 7×7 kernels achieved similar results. Thus, the 5×5 kernel approximate cases were evaluated in the ALPR, reducing the total power up to 64% in the best-case scenario while maintaining similar precision subject to the exact case. The application results show that the approximated Gaussian filter cases have different impacts on the power-quality trade-off and should be carefully evaluated to find the best match subject to the application constraints. Our design space exploration has the potential to develop a configurable architecture able to select the kernel size subject to the application’s restrictions. Also, it is possible to use configurable approximate adders to add fine-grain configurability to the architecture.

Acknowledgement

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001 and by Brazilian Council for Scientific and Technological Development (CNPq).

References


