On the Implementation of Edge Detection Algorithms with SORN Arithmetic

Moritz Bärthel[1(✉)], Nils Hülsmeier[1], Jochen Rust[2], and Steffen Paul[1]

[1] Institute of Electrodynamics and Microelectronics (ITEM.me), University of Bremen, Bremen, Germany
{baerthel,huelsmeier,steffen.paul}@me.uni-bremen.de
[2] DSI Aerospace Technologie GmbH, Bremen, Germany
jochen.rust@dsi-as.de

Abstract. Sets-Of-Real-Numbers (SORN) Arithmetic derives from the type-II unums and realizes a low-complexity and low-precision digital number format. The interval-based SORNs are especially well-suited for preprocessing large datasets or replacing particular parts of threshold-based algorithms, in order to achieve a significant reduction of runtime, complexity and/or power consumption for the respective circuit.

In this work, the advantages and challenges of SORN arithmetic are evaluated and discussed for a SORN-based edge detection algorithm for image processing. In particular, different SORN implementations of the Sobel Operator for edge filtering are presented, consisting of matrix convolution and a hypot function. The implemented designs are evaluated for different algorithmic and hardware performance measures. Comparisons to a reference Integer implementation show promising results towards a lower error w.r.t. ground truth solutions for the SORN implementation. Syntheses for FPGA and CMOS target platforms show a reduction of area utilization and power consumption of up to 68% and 80%, respectively.

Keywords: SORN · Unum · Computer arithmetic · Image processing

1 Introduction and Related Work

The universal number format unum, proposed by John Gustafson [12], presents a new approach for the computation with real numbers in digital hardware systems. To enhance and overcome traditional number formats, especially the IEEE standard for floating point arithmetic [16], the initial type-I unums were designed to utilize Interval Arithmetic (IA) instead of rounding in order to avoid the propagation of rounding errors. In addition, type-I unums exploit variable mantissa and exponent lengths for a reduced datapath and memory bandwidth. Evaluations and discussions on unum type-I hardware implementations can be found in [5,10] and [2].

Based on the initial unum format, with type-II unums and the corresponding Sets-Of-Real-Numbers (SORN) [11], as well as type-III unums (posits) [13], two

J. Gustafson and V. Dimitrov (Eds.): CoNGA 2022, LNCS 13253, pp. 1–13, 2022.
https://doi.org/10.1007/978-3-031-09779-9_1

further formats were derived. Whereas posits provide a less radical approach with constant bit lengths that can be used as a drop-in replacement for other floating point formats with compatibility to legacy systems, type-II unums and SORNs utilize the implicit IA concept created for type-I unums and radicalize this approach towards a very low precision format enabling low-complexity, -power and -latency implementations of arithmetic operations. A detailed introduction to SORN arithmetic is given in Sect. 2.

Due to the low-precision nature of SORNs, the format is not applicable to any application or algorithm. However, it can be shown that SORNs are especially well suited for preprocessing large systems of equations in order to reduce the amount of solutions for a certain optimization problem, such as in MIMO detection [4] or training of Machine Learning algorithms [14]. Another suitable application for the low-precision SORN arithmetic are threshold-based algorithms were a high accuracy result is not of major interest, as long as a sufficient threshold detection can be provided. In this work such a threshold-based algorithm for image processing is implemented and evaluated for SORN arithmetic. In particular, the Sobel Operator [18] used for edge detection in images is implemented as a full SORN and a hybrid Integer-SORN design and compared to an Integer reference design. Details on the Sobel Operator and the respective SORN implementations are given in Sec. 3. FPGA and CMOS synthesis results, as well as an algorithmic evaluation of the different Sobel implementations based on a reference image data set are provided in Sect. 4.

2 Type-II Unums and SORNs

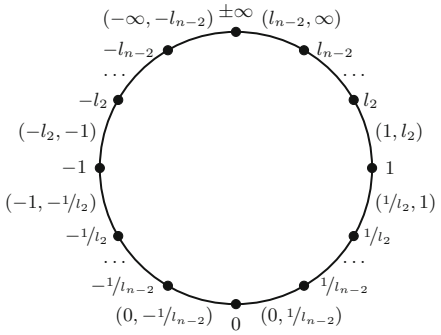
One of the main concepts of type-I unums is implicit IA by means of an extra bit after the mantissa, which indicates the presence of an open interval whenever maximum precision is exceeded [12]. With this approach, rounding errors can be omitted at the expense of a certain imprecision, when an open interval is given as result of a computation instead of a single value. Type-II unums fully utilize this interval concept by reducing the representation of the real numbers to only a small set of exact values and open intervals.

2.1 Original Type-II Unums and SORNs

For the original type-II unum representation proposed in [11], the real numbers are represented by a set of n exact values called *lattice values* l_i, including zero ($l_0 = 0$), one ($l_{(n-1)/2} = 1$) and infinity ($l_{n-1} = \infty$), and the open intervals in between. Every lattice value is included with a positive and negative sign. A basic set with $n = 3$ is given with the lattice values $l_i \in \{0, 1, \infty\}$:

$$\{\pm\infty\ (-\infty,-1)\ -1\ (-1,0)\ 0\ (0,1)\ 1\ (1,\infty)\} \tag{1}$$

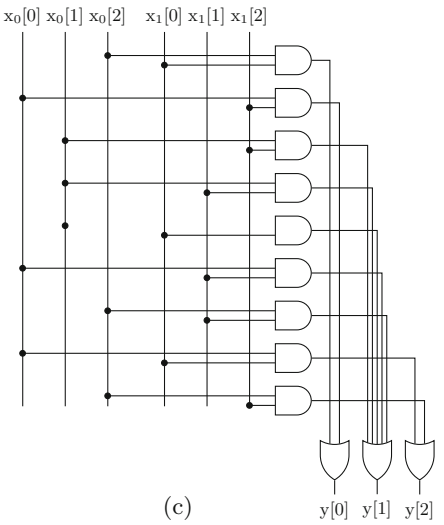
The representation can be extended by introducing further lattice values $l_i > 1$ and their reciprocals to the set. A general representation can be interpreted as depicted in Fig. 1a.



(a)

	×	x0 (−1, 0) 100	x0 0 010	x0 (0, 1) 001
x1	(−1, 0) 100	001	010	100
x1	0 010	010	010	010
x1	(0, 1) 001	100	010	001

(b)



(c)

Fig. 1. (a) Representation of the reals with the original type-II unum format. (b) LUT for the multiplication of a simplified 3 bit SORN datatype. (c) Gate level structure for the 3 bit SORN multiplication LUT.

For the implementation of arithmetic operations, the so-called Sets-Of-Real-Numbers (SORN) binary representation is derived from the unum type-II set. The absence (0) and presence (1) of every lattice value and interval is indicated with a single bit, resulting in a SORN bitwidth $w_{\text{sorn}} = 2^n$. Arithmetic operations with SORNs are carried out using pre-computed lookup tables (LUTs) which contain the result of every possible input combination for a given datatype configuration. Figure 1b shows the LUT for the multiplication of two SORNs using a simplified 3 bit datatype. Some SORN operations may result in union intervals, for example when two open intervals are added. In this case the result is represented by a pattern of consecutive bits:

$$100_{(-1,0)} + 001_{(0,1)} = 111_{(-1,1)} \tag{2}$$

The LUT structures for SORN operations can be implemented for hardware circuits using simple Boolean Logic which enables very fast computing with low-complexity. The corresponding gate level structure for the multiplication LUT in Fig. 1b is depicted in Fig. 1c.

2.2 Adaptions of the SORN Representation

Following the regular unum type-II-based structure for implementing SORNs maintains the unum compatibility and provides an error-free solution for processing arithmetic operations. However, the structure of the LUT-based arithmetics

Table 1. SORN datatype configurations for the hybrid (6 b–11 b) and full (15 b) SORN sobel designs.

Label	Configuration
6 b lin	$[0, 50]; (50, 100]; (100, 150]; (150, 200]; (200, 250]; (250, \infty]$
10 b log	$0; (0, 2]; (2, 4]; (4, 8]; (8, 16]; (16, 32]; (32, 64]; (64, 128]; (128, 256]; (256, \infty]$
11 b lin	$[0, 25]; (25, 50]; (50, 75]; (75, 100]; (100, 125]; (125, 150];$ $(150, 175]; (175, 200]; (200, 225]; (225, 250]; (250, \infty]$
15 b lin	$[-\infty, -300); [-300, -250); \ \ldots \ ; [-100, -50); [-50, 0); 0;$ $(0, 50]; (50, 100]; (100, 150]; (150, 200]; (200, 250]; (250, 300]; (300, \infty]$
15 b log	$[-\infty, -512); [-512, -256); \ \ldots \ ; [-32, -16); [-16, 0); 0;$ $(0, 16]; (16, 32]; (32, 64]; (64, 128]; (128, 256]; (256, 512]; (512, \infty]$

with low bitwidths encompasses a major challenge within complex datapaths: computing multiple sequential SORN operations may lead to increasing interval widths at the output, mainly depending on the performed operations. In a worst-case scenario, the result of a SORN computation represents the interval $(-\infty, \infty)$ and does not contain any useful information. This can be counteracted with a higher resolution within the SORN representation. Evaluations in [3] showed that the exact values within a unum-type-II based SORN are barely ever addressed without their adjacent intervals. Consequently, moving away from a strict unum type-II based structure and adapting the SORN representation towards half-open intervals without exact values increases the information-per-bit within a SORN value and reduces the interval growth. Possible SORN representations following this concept are given in Table 1. The corresponding label indicates the number of elements in the Set-Of-Real-Numbers, which is also the number of bits in SORN representation. In addition, the label indicates whether the intervals within the set tile the real number line in a logarithmic or linear manner. In order to find a suitable datatype for a given application, the automatic SORN datapath generation tool from [17] provides an easy and fast way of prototyping SORN arithmetics for hardware circuits.

3 Edge Detection

In this work SORN arithmetic is applied to the Sobel Operator, an algorithm used for edge detection within image processing systems. Edges are regions in a digital image where distinct changes in color or brightness can be detected [1], in order to classify segments of the image, or to detect certain objects. Edge detection is used in various modern applications, such as fingerprint recognition [7], cloud classification via satellite images [8], or autonomous driving [6,20].

3.1 Sobel Operator

The Sobel Operator belongs to the family of first-order convolutional filters that compute the horizontal and vertical gradient of a grayscale image [18]. The Sobel method uses two 3×3 kernels, which are convolved with the grayscale image $\mathbf{A} \in \mathbb{N}^{N_x \times N_y}$ in order to determine the image gradients G_x and G_y in horizontal and vertical direction, respectively [19]:

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} * \mathbf{A}_{3\times 3} \qquad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A}_{3\times 3} \tag{3}$$

After computing the image gradient

$$G = \sqrt{{G_x}^2 + {G_y}^2} \tag{4}$$

a comparison to the pre-defined threshold T determines whether the current pixel is an edge. This process is performed for every single pixel of the image $\mathbf{A}$ and results in a binary image containing all detected edges.

(a) Grayscale Test Image

(b) Reference Sobel Impl. (Integer)

(c) Hybrid SORN Sobel Impl. (11b lin)

(d) Full SORN Sobel Impl. (15b lin)

Fig. 2. (a) Grayscale highway test image [9] with Sobel edge detection results from (b) an Integer reference implementation with threshold $T = 250$, (c) a Hybrid-SORN 11 b implementation with threshold interval $T = (250, \infty]$, and (d) the negated result for a full-SORN 15 b implementation with threshold interval $T = (0, 50]$.

In Fig. 2 edge detection applied to a highway image is shown, which is used for road lane detection in driving assistant systems [6,9]. Figure 2a shows the grayscale test image and Fig. 2b the result of an edge detection using the Sobel method with integer arithmetic.

3.2 SORN Implementation

In this work, the Sobel method described in Eq. (3) and (4) is implemented with SORN arithmetic as a hybrid Integer-SORN and as a full SORN design, both for different SORN datatypes. Additionally, an Integer reference design is implemented in order to compare the SORN designs to a State-of-the-Art (SotA) architecture. The three designs are described in the following.

Integer Reference Design. The grayscale test image **A** contains pixels with values $A_{xy} \in \{0, \ldots, 255\}$ which can be implemented with Integer values of 8 b. The convolution described in Eq. (3) is implemented with conventional Integer additions and subtractions as shown in Fig. 3. For the calculation of the gradient G the square root is omitted and the result G^2 is compared to the squared threshold T^2 instead.

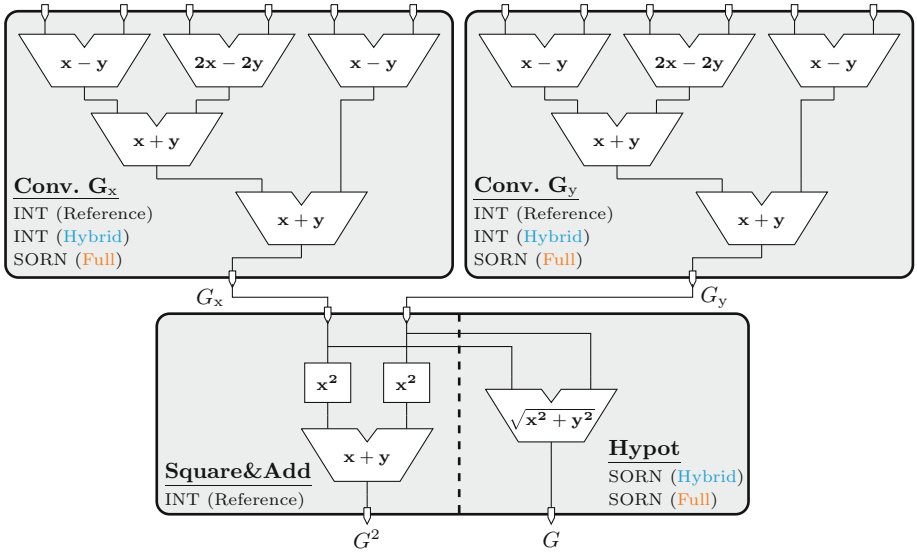


Fig. 3. Block diagram for the three different Sobel implementations: all integer for the reference implementation, integer convolution with SORN hypot for the hybrid approach, and SORN convolution and hypot for the full SORN approach.

Hybrid Integer-SORN Design. For the hybrid design, the convolutions are carried out with Integer operations, similar to the reference implementation. The horizontal and vertical gradients G_x and G_y are then converted to a SORN representation. Since they are squared in the following hypot operation, their absolutes are converted, and SORN datatypes without negative values can be used. The hybrid design is implemented for three different SORN representations with 6 b, 10 b and 11 b, all given in Table 1. The hypot operation is implemented as one single SORN operation, as depicted in Fig. 3 (conversion blocks from Integer to SORN between convolution and hypot are not shown). Since the result G is in SORN representation, the threshold T has to be chosen as one of the SORN intervals from the respective datatype. Figure 2c shows the edge result of the hybrid SORN implementation for the 11 b datatype and the threshold interval $T = (250, \infty]$, which corresponds to the Integer threshold $T = 250$ used for the reference implementation in Fig. 2b.

Full SORN Design. For the full SORN design, the Integer inputs from the test image $\mathbf{A}$ are converted to SORN representation before the convolutions and hypot function are carried out in SORN arithmetic, as shown in Fig. 3 (conversions not shown). Since for the convolution also subtraction is required, the full SORN design is implemented for two different 15 b datatypes with negative values, as shown in Table 1. In order to obtain a comparable edge result, for the full SORN approach thresholds near the zero-bit in SORN representation are selected and the result image is negated afterwards. Figure 2d shows the negated edge result for the full SORN implementation with the 15 b lin datatype and a threshold interval $T = (0, 50]$.

4 Evaluation

Figure 2 shows a test image and the edge detection results of the three different Sobel implementations described in Sect. 3.2. By visual comparison they seem to be quite similar, even though a few differences can be found, for example when comparing the detection of the cars on the road. For a comprehensive evaluation, however, a visual comparison of different edge results is not sufficient. Unfortunately, measuring the performance and comparing different edge detection methods or implementations is an open problem. In [15] various error and performances metrics are evaluated and compared, and the authors conclude that no convincing general-purpose solution exists. Since in this work no different methods, but only different implementations are to be compared, the most intuitive approach is a numerical comparison of the different edge results. Therefore the normalized absolute error *nae* between the SORN results and the Integer reference implementation can be defined as

$$nae = \frac{\sum_{x=1}^{N_x} \sum_{y=1}^{N_y} (E_{int}(x, y) \neq E_{sorn}(x, y))}{N_x N_y} \tag{5}$$

with the respective edge detection results E_{int} and E_{sorn} and the test image dimensions N_{x} and N_{y}. This metric basically counts the number of different pixels between the Integer and SORN edge images and normalizes the result by the total number of pixels. Applied to the edge images from Fig. 2, the errors read as follows:

$$nae|_{\text{hybridSORN,11b}} = 0.0181 \tag{6}$$

$$nae|_{\text{fullSORN,15b}} = 0.0287 \tag{7}$$

This metric can not determine whether the SORN implementation performs better or worse than the Integer reference, but it can show that the difference between both results is below 2% and 3%, respectively, which is in line with the visual evaluation. In order to further evaluate the different Sobel implementations, in the following section a larger number of test images is considered.

4.1 Algorithmic Evaluation with BSDS500

The Berkeley Segmentation Data Set 500 (BSDS500) [1] is a set of images for the performance evaluation of contour detection and image segmentation algorithms, consisting of images of humans, animals, objects and landscapes. For a comprehensive evaluation, the 200 test images from the data set are processed with the different Sobel implementations for all presented SORN datatypes. Additionally, two different thresholds per configuration are analyzed. For the hybrid designs, the two rightmost SORN intervals with indices $w_{\text{sorn}} - 1$ and w_{sorn} are used as thresholds. The results are compared to the corresponding Integer threshold for the reference design. For the 6 b datatype for example, the interval thresholds are $T = (200, 250]$ and $T = (250, \infty]$, the corresponding Integer thresholds are $T = 200$ and $T = 250$. For the full SORN implementation thresholds near the zero-bit are utilized and the resulting edge images are negated, in order to achieve the best performance. Therefore the equivalent threshold T_e is given, which corresponds to the compared Integer threshold.

In Table 2 the results for the mean normalized absolute error between the SORN and reference edge results are given. The utilized metric represents the mean of the *nae* from Eq. (5) over all test images. For both the hybrid and full SORN versions the designs utilizing a linear distributed SORN datatype perform better than the log-based versions. Furthermore, the rightmost SORN

Table 2. Mean normalized absolute error between SORN and reference integer implementation for 200 test images from BSDS500 [1].

SORN Datatype		hybrid SORN				full SORN	
		6 b lin	10 b log	11 b lin		15 b log	15 b lin
mnae	$T = w_{\text{sorn}}$	0.0659	0.1200	0.0598	$T_e = w_{\text{sorn}}$	0.1396	0.0667
	$T = w_{\text{sorn}} - 1$	0.1167	0.2323	0.0852	$T_e = w_{\text{sorn}} - 1$	–	0.0673

interval thresholds lead to the best results by means of lowest difference to the Integer reference. Compared to the results for the image in Fig. 2, given in Eq. (6)–(7), the errors are slightly higher, but still below 7%. It is mentioned again, that this metric can only measure the difference between Integer and SORN implementation. For a rating of the different designs, a third, independent reference is required.

Ground Truth Reference Comparison. For this purpose, the BSDS500 contains so-called ground truth edge results. These are human made edge detections from different human subjects [1]. For evaluating the edge detections of the different SORN implementations in comparison to the Integer reference, Fig. 4 shows the mean normalized absolute error between 6 different ground truth solutions GT and the respective edge detection results E, with the image dimensions N_x and N_y and the number of test images N_i:

$$\mathrm{mnae} = \frac{\sum_{i=1}^{N_i} \left(\frac{\sum_{\mathrm{x}=1}^{N_\mathrm{x}} \sum_{\mathrm{y}=1}^{N_\mathrm{y}} (GT_i(\mathrm{x},\mathrm{y}) \neq E_i(\mathrm{x},\mathrm{y}))}{N_\mathrm{x} N_\mathrm{y}} \right)}{N_i} \tag{8}$$

For the hybrid and full SORN implementations, for each datatype the threshold configuration with the best results is shown, as well as the corresponding Integer configurations. Similar to the previous evaluation, those SORN implementations utilizing linear distributed datatypes perform better than the log-based versions. For this evaluation, the linear-based SORN implementations outperform even the corresponding Integer references. As mentioned above and discussed in [15], this does not necessarily indicate that the SORN-based edge detection is better than the Integer-based for any application. Nevertheless, this evaluation on BSDS500, as well as the example in Fig. 2 show that the hybrid and full

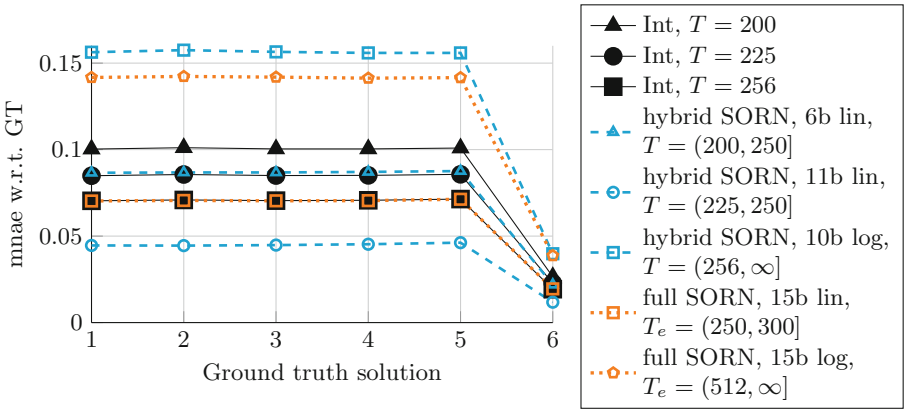


Fig. 4. Mean normalized absolute error w.r.t. 6 ground truth solutions for the different Integer and SORN Sobel implementations over 200 test images from BSDS500.

SORN-based edge detection implementations of the Sobel operator provide, at least, a similar result quality as the Integer implementation and can serve as a replacement for the SotA implementation.

4.2 Hardware Performance

In addition to the algorithmic evaluation, also the hardware performance in terms of latency, complexity and power consumption for the respective hybrid and full SORN designs, as well as for the Integer reference design is evaluated. In the following, the results of FPGA and CMOS syntheses of all designs described in Sect. 3.2 are presented.

FPGA Results. In Table 3 the synthesis results for an Artix-7 AC701 FPGA from Xilinx are given for all presented designs for a target frequency of 100 MHz. All designs are evaluated without internal pipeline registers and no DSPs are used. The worst negative slack (WNS) shows that solely the 6 b and 10 b hybrid SORN implementations are able to run at a target frequency of 100 MHz, yet all SORN designs achieve a higher maximum frequency than the Integer reference design. Concerning the required LUTs and the power consumption, the hybrid SORN approach significantly outperforms the reference design with a complexity reduction of up to 68%, whereas for the full SORN approach only the 15 b log configuration achieves a lower power consumption, all other measures can not compete with the reference design.

Table 3. FPGA synthesis results without DSPs for an Artix-7 AC701 FPGA (xc7a200tfbg676-2).

Module		Int	hybrid SORN			full SORN	
			6 b lin	10 b log	11 b lin	15 b log	15 b lin
Target Freq.	[MHz]	100	100	100	100	100	100
WNS	[ns]	−1.487	**0.554**	0.492	−0.173	−0.466	−1.042
Max Freq.	[MHz]	87.055	**105.865**	105.175	98.299	95.548	90.563
LUTs		457	**148**	207	219	597	712
Total power	[W]	0.145	**0.136**	0.137	0.138	0.140	0.147

CMOS Results. Table 4 shows the synthesis results for the proposed designs without pipeline registers for a 28 nm SOI CMOS technology from STM. Each configuration is synthesized for a target frequency of 1 GHz and for the respective maximum frequency. For the 1 GHz comparison, all SORN-based designs achieve a lower area and power consumption than the reference design, with reductions of up to 45% for area and 44% for power, respectively. Targeting maximum

Table 4. CMOS STM 28nm SOI technology synthesis results.

Module		Int	hybrid SORN			full SORN	
			6 b lin	10 b log	11 b lin	15 b log	15 b lin
Target Freq.	[MHz]	1000	1000	1000	1000	1000	1000
Runtime	[ns]	0.962	**0.958**	0.962	0.962	0.961	0.962
Area	[μm^2]	1153.987	**638.765**	693.110	733.421	989.808	1132.282
Power	[µW]	550.337	329.210	349.387	349.964	**309.294**	324.075
Max. Freq.	[MHz]	1263	1681	1603	1605	1661	**1715**
Runtime	[ns]	0.792	0.595	0.624	0.623	0.602	**0.583**
Area	[μm^2]	2087.165	**1100.294**	1157.251	1245.706	1661.213	2017.642
Power	[µW]	1979.710	757.566	774.962	838.914	**403.631**	413.465

frequency, all SORN-based designs achieve at minimum a 27% higher frequency than the reference design while still requiring less area and power. For this maximum frequency comparison, the hybrid SORN designs show a significantly lower area requirement than the reference design (up to 47% reduction), whereas the full SORN designs require significantly less power (up to 80% reduction).

5 Conclusion

The Sobel Operator for edge detection can be implemented as a hybrid SORN design with Integer convolution and SORN hypot function, or as a full SORN approach. Depending on the utilized SORN datatype and the chosen thresholds, both versions provide a similar algorithmic performance than the Integer reference implementation. For the presented evaluation on BSDS500 with the corresponding ground truth reference comparison, some of the SORN configurations even show a lower difference to ground truth than the Integer reference. Regarding hardware performance, the presented evaluations show that the SORN approach achieves higher frequencies and significantly lower complexity and power consumption than the Integer reference for both FPGA and CMOS.

For future work the SORN-based edge detection can be integrated into a more complex image processing system in order to provide further evaluations on the quality of the edge detection results in the context of an actual (real-time) application, for example lane detection in autonomous driving. In addition, other edge detection methods such as the Canny detector or the Marr Hildreth Operator can be implemented and evaluated for SORNs.

References

1. Arbelaez, P., Maire, M., Fowlkes, C., Malik, J.: Contour detection and hierarchical image segmentation. IEEE Trans. Pattern Anal. Mach. Intell. **33**(5), 898–916 (2011). https://doi.org/10.1109/TPAMI.2010.161

2. Bärthel, M., Rust, J., Paul, S.: Hardware implementation of basic arithmetics and elementary functions for unum computing. In: 2018 52nd Asilomar Conference on Signals, Systems, and Computers, pp. 125–129, October 2018. https://doi.org/10.1109/ACSSC.2018.8645453
3. Bärthel, M., Rust, J., Paul, S.: Application-specific analysis of different SORN datatypes for unum type-2-based arithmetic. In: 2020 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1–5 (2020). https://doi.org/10.1109/ISCAS45731.2020.9181182
4. Bärthel, M., Knobbe, S., Rust, J., Paul, S.: Hardware implementation of a latency-reduced sphere decoder With SORN preprocessing. IEEE Access **9**, 91387–91401 (2021). https://doi.org/10.1109/ACCESS.2021.3091778
5. Bocco, A., Durand, Y., De Dinechin, F.: SMURF: scalar multiple-precision unum Risc-V floating-point accelerator for scientific computing. In: Proceedings of the Conference for Next Generation Arithmetic 2019, pp. 1–8 (2019)
6. Bounini, F., Gingras, D., Lapointe, V., Pollart, H.: Autonomous vehicle and real time road lanes detection and tracking. In: 2015 IEEE Vehicle Power and Propulsion Conference (VPPC), pp. 1–6 (2015). https://doi.org/10.1109/VPPC.2015.7352903
7. Cui, W., Wu, G., Hua, R., Yang, H.: The research of edge detection algorithm for Fingerprint images. In: 2008 World Automation Congress, pp. 1–5. IEEE (2008)
8. Dim, J.R., Takamura, T.: Alternative approach for satellite cloud classification: edge gradient application. Adv. Meteorol. **2013** (2013)
9. Gatopoulos, I.: Line detection: make an autonomous car see road lines. Towards Data Sci. (2019). https://towardsdatascience.com/line-detection-make-an-autonomous-car-see-road-lines-e3ed984952c
10. Glaser, F., Mach, S., Rahimi, A., Gurkaynak, F.K., Huang, Q., Benini, L.: An 826 MOPS, 210uW/MHz Unum ALU in 65 nm. In: 2018 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1–5. IEEE (2018). https://doi.org/10.1109/ISCAS.2018.8351546
11. Gustafson, J.L.: A Radical Approach to Computation with Real Numbers. Supercomput. Front. Innov. **3**(2) (2016). https://doi.org/10.14529/jsfi160203
12. Gustafson, J.L.: The end of error: Unum computing. CRC Press, Boca Raton, Chapman & Hall/CRC Computational Science Series (2015)
13. Gustafson, J.L., Yonemoto, I.T.: Beating floating point at its own game: posit arithmetic. Supercomput. Front. Innov. **4**(2) (2017). https://doi.org/10.14529/jsfi170206
14. Hülsmeier, N., Bärthel, M., Rust, J., Paul, S.: SORN-based cascade support vector machine. In: 2020 28th European Signal Processing Conference (EUSIPCO), pp. 1507–1511. IEEE (2021)
15. Lopez-Molina, C., De Baets, B., Bustince, H.: Quantitative error measures for edge detection. Pattern Recogn. **46**(4), 1125–1139 (2013)
16. Microprocessor Standards Committee of the IEEE Computer Society: IEEE Standard for Floating-Point Arithmetic. IEEE Std. **754–2008**, 1–70 (2008). https://doi.org/10.1109/IEEESTD.2008.4610935
17. Rust, J., Bärthel, M., Seidel, P., Paul, S.: A hardware generator for SORN arithmetic. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **39**(12), 4842–4853 (2020). https://doi.org/10.1109/TCAD.2020.2983709
18. Sobel, I.: An Isotropic 3×3 Image Gradient Operator. Presentation at Stanford A.I. Project 1968 (2014)
19. Solomon, C., Breckon, T.: Fundamentals of Digital Image Processing: A practical approach with examples in Matlab. Wiley, Hoboken (2011)

20. Yang, X., Yang, T.A., Wu, L.: An edge detection IP of low-cost system on chip for autonomous vehicles. In: Arabnia, H.R., Ferens, K., de la Fuente, D., Kozerenko, E.B., Olivas Varela, J.A., Tinetti, F.G. (eds.) Advances in Artificial Intelligence and Applied Cognitive Computing. TCSCI, pp. 775–786. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-70296-0_56