ASC-FFT: Area-Efficient Low-Latency FFT Design Based on Asynchronous Stochastic Computing

Patricia Gonzalez-Guerrero, Xinfei Guo, Mircea R. Stan
Department of Electrical and Computer Engineering, University of Virginia, Charlottesville, VA, USA
{lg4er, xg2dt, mrs8n}@virginia.edu

***Abstract*—Asynchronous Stochastic Computing (ASC) is a new paradigm that addresses Synchronous Stochastic Computing (SSC) drawbacks, expensive stochastic number generation (SNG) and long latency, by using continuous time streams (CTS). To go beyond the basic operations of addition and multiplication in ASC we need to incorporate a memory element. Although for SSC the natural memory element is a clocked-flip-flop, using the same approach with no synchronized data leads to unacceptable large error. In this paper, we propose to use a capacitor embedded in a feedback loop as the ASC memory element. Based on this idea, we design a low-error asynchronous adder that stores the carry information in the capacitor. Our adder enables the implementation of more complex computation logic. As an example, we implement an asynchronous stochastic Fast Fourier Transform (ASC-FFT) using a FinFET1X[1] technology. The proposed adder requires 76%-24% less hardware cost compared against conventional and SSC adders respectively. Besides, the ASC-FFT shows 3X less latency when compared with SSC-FFT approaches and significant improvements in latency and area over conventional FFT architectures with no degradation of the computation accuracy measured by the FFT Signal to Noise Ratio (SNR).**

I. Introduction

The idea of processing information via random streams of bits - stochastic computing (SC) - emerged in the 1960s to mitigate area and power constraints for machine learning algorithms [1]. This approach replaces complex processing elements with simple gates. The reduction in circuit complexity is enabled by its particular data representation, which maps inputs to the probability p of having a high (P) or low (N) voltage level in a stream of bits. Thus complex processing elements such as a multiplier or an adder can be implemented as a simple XNOR gate or 2:1-multiplexer respectively (Fig. 1). These characteristics renovate the interest in SC for any cost-constrained system ranging from ultra-low power IoT devices to artificial intelligent applications.

By definition, SC is a synchronous approach, where bits (events) are synchronized with a global clock. Although promising, the SSC approach suffers from two major drawbacks that eclipse any savings in area or power: 1) *Power-area hungry stream generation* and 2) *Energy expensive long latency* [2] [3]. SC on CTS is a promising candidate to address these drawbacks. Recent work showed that synchronization between streams might not be required for SC [4]. Moreover, data can be mapped to the time the signal is in the high level (P) instead of the probability of a discrete event to occur

[1]In modern technologies the node number does not refer to any one feature in the process, and foundries use slightly different conventions; we use 1x to denote the 14/16nm FinFET nodes offered by the foundry.

978-1-7281-0453-9/19/$31.00 ©2019 IEEE

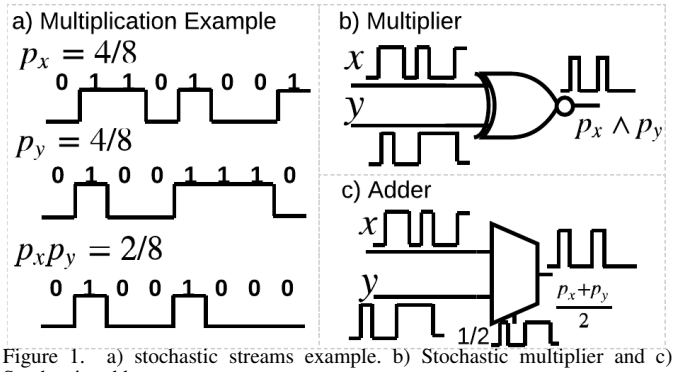

Figure 1. a) stochastic streams example. b) Stochastic multiplier and c) Stochastic adder

[3]. Furthermore, we can eliminate all system clocks by using Asynchronous Sigma Delta Modulators (AΣΔMs) to map data to continuous time streams [5]. The combination of a clockless design and the asynchronous modulators originates a new paradigm: Asynchronous Stochastic Computing (ASC).

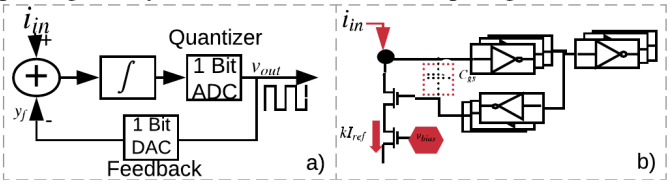

Figure 2. AΣΔM block diagram (a) and Architecture (b) [5].

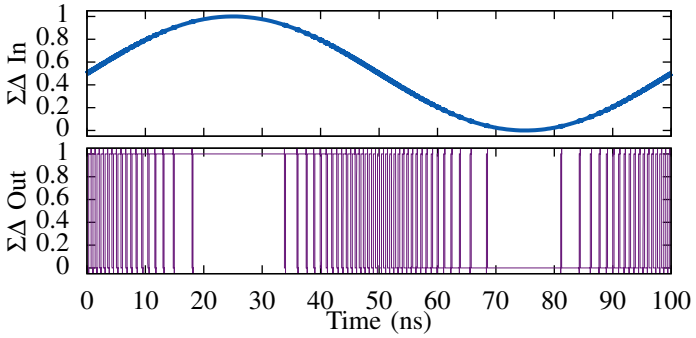

Figure 3. ΣΔ streams example for a sine wave. The maximum frequency occurs when the input is 0.5, on the other hand the frequency is zero when the input is at its minimum or maximum [5].

An AΣΔM maps a DC input (i_{in}) to the frequency and duty cycle of a periodic square signal. Fig. 2 shows the block diagram and the architecture for the modulator and Fig. 3 shows a ΣΔ stream for a sinusoidal input. This approach, reduces the power and area associated with streams generation by up to 70% and 50% by using alternative sources for stream de-correlation [5]. Besides, with an asynchronous approach, we eliminate the need of clock distribution networks (CDNs) which contributes a significant amount of design effort and power consumption [6].

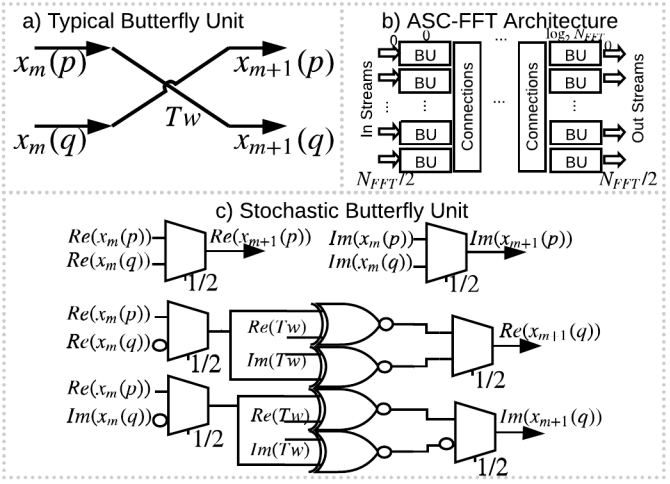

Figure 4. a) Buttterfly unit as described in equation 2 b) ASC-FFT architecture c) Stochastic Implementation of the BU

Computation on CTS has been limited to single-stage combinational logic, because more complex algorithms require the incorporation of a memory element. In SSC the memory elements are just flip-flops, which unfortunately require synchronized streams. Yet, algorithms like the FFT, one of the most used and expensive operations in digital signal processing, could benefit from the ASC savings in area and latency. Therefore, we need to find an effective, yet inexpensive way to incorporate a memory element in asynchronous computing units.

In this paper, we propose to use a capacitor embedded in simple feedback loop as the asynchronous memory element. Based on this idea, we design a low-error asynchronous adder that is able to keep track of the addition carry. Then, we implement an ASC-FFT architecture based on this adder using a FinFET1X technology. We compare its accuracy, area, and latency, against conventional FFT architectures and find that the ASC-FFT shows better area-delay performance than pipelined, and minimum hardware FFT architectures for $N_{FFT} \geq 32$ and $N_{FFT} \geq 128$ respectively, where N_{FFT} is the FFT size.

II. Asynchronous Stochastic Computing FFT Architecture

The radix-2 FFT is a multistage algorithm based on butterfly units (BUs) described by:

$$x_{m+1}(p) = x_m(p) + x_m(q) \tag{1}$$

$$x_{m+1}(q) = [x_m(p) - x_m(q)]\, e^{-j\frac{2\pi}{N}nk} \tag{2}$$

and shown in Fig. 4a, where $m = 1, \ldots, \log_2(N_{FFT})$ labels the stage, $x(p)$ and $x(q)$ label the inputs for a particular BU, N_{FFT} is the FFT size, $x_{m+1}(p)$ and $x_{m+1}(q)$ are the BU complex outputs. Finally, $e^{-j\frac{2\pi}{N_{FFT}}nk}$ are the twiddle (Tw) coefficients with $k, n = 0, \ldots, N_{FFT} - 1$. Directly mapping conventional multipliers and adders to the typical stochastic versions (Fig.4c) yields poor SNR due to the adder ouput being divided by two which we refer as the adder scaling effect (Fig. 1c) [7]. The authors in [8] present a SSC-FFT, which achieves better SNR response due to a non-scaling adder whose output result is $p_x + p_y$ instead of $(p_x + p_y)/2$. This adder, relies on a digital counter (flip-flops) that keeps track of carry bits. Unfortunately, this SSC-FFT implementation still suffers from the drawbacks described before, expensive SNG and long latency. To address these challenges, we propose an efficient non-scaling asynchronous adder for CTS that exploits the simplicity of a current based design approach.

A. Asynchronous non-scaling adder

Fig. 5a shows an example of SSC streams addition. A(t)+B(t) at each clock cycle can be: ① N+N, ② N+P=0 or ③ P+P. In case ③(①) the output stream can only take one P(N). The other P(N) is accumulated in a clocked counter as a carry. A carry can be released when case ② occurs and the counter decreases (increments) by one [8] [9] [10]. For ASC streams the time the inputs are in a given state {N, P} is not quantized, thus counting carry bits would require an unfeasible fast clock. Instead, we propose to use an integrator that accumulates "time" when ③(①) and decrements when the carry "time" is sent to the output.

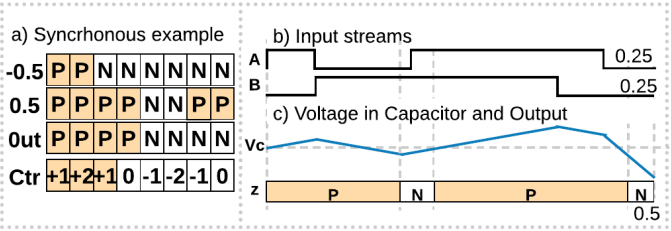

Figure 5. a) Example of non-scaling synchronous addition with digital integrator. b),c) Example of asynchronous adder, input streams (b) and evolution of the voltage in the capacitor and output stream (c).

This adder can be implemented with the block diagram in Fig. 6a. The integrator output can be expressed as:

$$y(t) = y(t_o) + \frac{1}{k}\int_{t_0}^{t} \Big(A(\tau) + B(\tau) - y_f(\tau)\Big)\, d\tau \tag{3}$$

where $y(t_0)$ is the initial condition on the integrator and k is the integrator constant. The integrator output is converted to an asynchronous stream by a quantizer. If $y(t) > Thr$, $z(t) = \text{P}$, otherwise $z(t) = \text{N}$. $y_f(t)$ is the output of the system converted back to the input units.

The main SC advantage is the computing units low hardware cost (Fig. 1). To keep this advantage, we adopt a current based methodology, eliminating the need of switched capacitors and high gain active components like operational amplifiers. The adder's architecture is depicted in Fig. 6b. To implement the addition of the inputs and the feedback path, we adopt a current based adder. By Kirchhoff law, this adder is just connecting the currents to the same node. Fig. 6c, d shows one current mirror per input stream $A(t)$, $B(t)$ and a third current mirror for $y_f(t)$. Fig. 6c shows the current flow when the inputs are PN (NP). In this case the net current is zero and the input for the integrator depends entirely on $y_f(t)$. On the other hand, Fig. 6d shows the current flow when the input streams are either PP or NN. In these cases, the current from $A(t) + B(t)$ is $\pm 2I_b$ and the input to the integrator can be $\pm I_b$ or $\pm 3I_b$ depending on $z(t)$. The current integrator is implemented as a

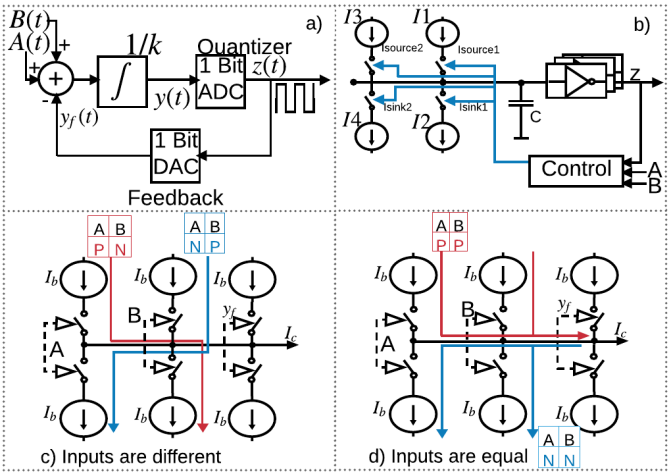

Figure 6. Proposed asynchronous adder. $A(t)$ and $B(t)$ are the continuous time input streams and $Z(t)$ is the continuous time output stream. a) Block diagram. b) Architecture. c), d) Current adder theory. c) Current flux when inputs are PN or NP. d) Current flux when inputs are PP or NN.

capacitor given its current-voltage relation $V_c = \frac{1}{c}\int I_c dt$. The current-comparator can be implemented as an inverter [11].

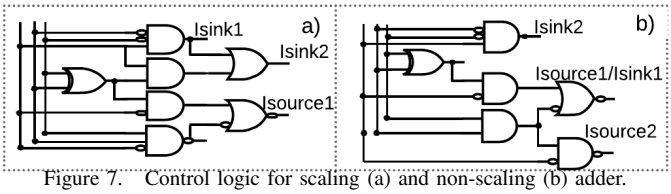

Figure 7. Control logic for scaling (a) and non-scaling (b) adder.

Fig. 5b shows the adder operation. In this example $A(t)$ and $B(t)$ are streams with associated probability of $p_A = 0.25$ and $p_B = 0.25$. Fig. 5c shows the evolution of the voltage in the capacitor V_c (integrator) and the output of the comparator $z(t)$. The slope for V_c is given by the integrator input current and can be either ± 1 or ± 3. Table I shows the slope for all the different combinations of $A(t)$, $B(t)$ and the output of the modulator $z(t)$. Based on this we can reduce the number of current supplies from 6 to 4 as shown in Fig. 6b. When the slope is +3 (-3) current sources I1, I3 (I2, I4) are active at the same time. The control logic to activate the current sources is shown in Fig. 7a which is implemented based on Table I.

Table I
TRUTH TABLE FOR CONTROL SIGNALS FOR THE ASYNCHRONOUS ADDER.

Z	Inputs		Slope		Active Source	
	A	B	Scaling	Non-Scaling	Scaling	Non-scaling
N (P)	N (P)	N (P)	0	-1 (1)	None	I2 (I1)
N (P)	P (N)	P (N)	2 (-2)	3 (-3)	I1,I3 (I2,I4)	I1,I3 (I2,I4)
N (P)	x	x	1	1 (-1)	I1 (I2)	I1 (I2)

B. Asynchronous scaling-adder

We can implement a low-error scaling adder $(\frac{A(t)+B(t)}{2})$ following the same reasoning used to design the non-scaling version. Based on the implementation in Fig. 6b, we modify the capacitor charging slopes to ± 2 and ± 1 depending on $A(t)$, $B(t)$ and $z(t)$. To adapt for the new slopes I1 and I2 become I_b instead of $2I_b$.

Fig. 8 shows the scaling adder operation. In this example $A(t)$ and $B(t)$ are streams with associated probability

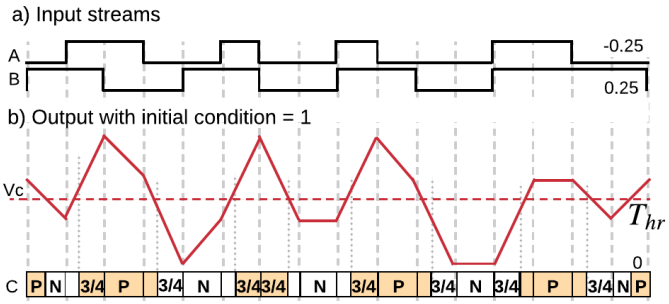

Figure 8. Example for average adder. a) Input streams $A(t)$ and $B(t)$. b) Voltage in the capacitor and output stream for initial condition $V_c > Thr$

$p_A = 0.25$ and $p_B = -0.25$. Fig. 8c shows the evolution of the voltage in the capacitor V_c (integrator) and the output of the comparator. For different initial conditions, the evolution of V_c is different but the expected result is correct. Notice that if the input streams are PP(NN) and the output is P(N) there is not accumulation in the integrator. On the contrary, if the output stream is N(P) the integrator accumulates the input current until the comparator switches to P(N).

C. Asynchronous FFT Architecture

We replace the multiplexer based stochastic adder with our asynchronous adders and implement the FFT architecture shown in Fig. 4b. We scaled the outputs at each FFT stage by two to avoid overflow errors, and both inputs and twiddle coefficients are CTS generated with AΣΔM. In the following section, we compare the area and delay performance for the proposed ASC-FFT with SSC-FFT and typical FFT pipelined (optimized for throughput) [12] and one-BU (optimized for area) [13] architectures .

III. RESULTS

FFT SNR Perfomance: We implement the ASC-FFT architecture (Fig. 4b) using a FinFET1X technology. We run SPICE level simulations and calculate the average and the FFT SNR for the resulting streams. Fig. 9a shows the SNR versus N_{FFT} for different SC-FFT implementations (SSC [8], un-even, and Direct [7]). The direct BU SSC implementation is not practical given the quick performance degradation as N_{FFT} increments [7]. The SNR performance for the ASC-FFT is better than the un-even SSC-FFT architecture for $N_{FFT} \geq 8$ when the scaling degrading effect on the SNR is more notoriuous for the un-even architecture. Fig. 9b shows the ASC-FFT SNR versus delay for $N_{FFT} = 8, 16, 32, 64$. Increasing the computation time has a positive effect on the overall computation accuracy. However, to obtain an accuracy comparable with SSC-FFT architectures ($SNR \approx 20dB$) the ASC-FFT requires 3X less computing time. Table II summarizes the performance results for the ASC adder and FFT.

Hardware Cost: The ASC adder has 76% and 24% less hardware cost than a conventional binary adder and other stream-adder approaches respectively (see transistor count (TC) in Table II). The hardware cost for the conventional FFT architectures depends on the number of adders, multipliers and registers between stages. We compare the TC neglecting interconnection and routing logic for different FFT architectures in Fig. 9c. The ASC-FFT hardware cost is placed between the

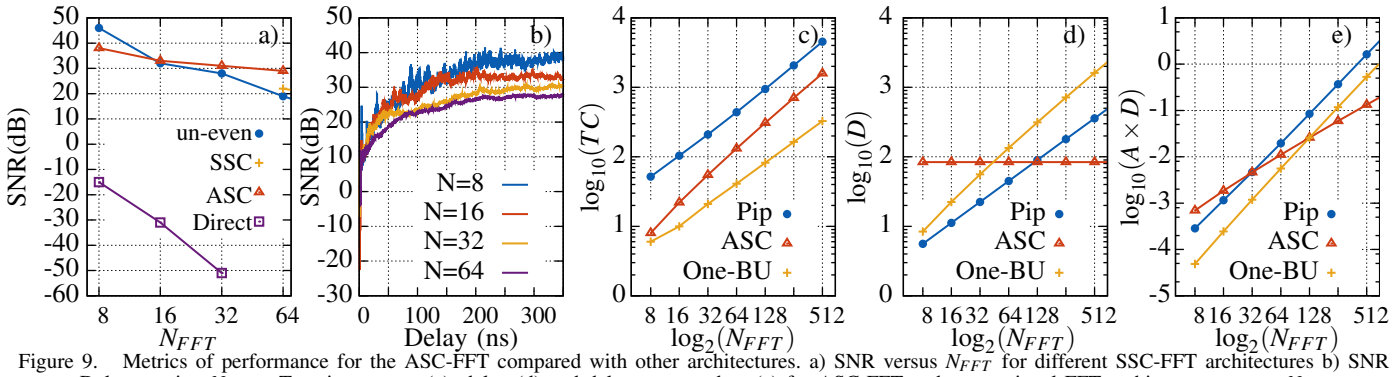

Figure 9. Metrics of performance for the ASC-FFT compared with other architectures. a) SNR versus N_{FFT} for different SSC-FFT architectures b) SNR versus Delay varying N_{FFT}s. Transistor count (c), delay (d) and delay-area product (e) for ASC-FFT and conventional FFT architectures versus N_{FFT}.

Table II
HARDWARE PERFORMANCE COMPARISON. *VALID FOR L=4 BITS. **AT MAXIMUM FREQUENCY OF OPERATION AND N_{FFT}=64

FFT/Adder Implementation	Adder transistor count	Multiplier transistor count	# Clock cycles	Latency for FFT (ns)**
Binary-One-BU [13]	444	3379	192	135
Binary-Pipelined [14]	444	3379	64	45
SSC D.Adder [9]*	146	-	-	-
SSC DigitalΣΔ [10]	240	10	1024	328
SSC Dual-Line [7]	156	10	1024	256
ASC (This Work)	108	10	-	84

pipelined and the one-BU architecture. Even though in the ASC and pipelined FFT architectures, the number of adders and multipliers is growing with N_{FFT}, the registers in the pipelined architecture grows faster (2*N/3-2) [12] while in the ASC-FFT there is no memory involved in the computation. For the one-BU architecture, on the other hand, the number of computing elements is constant and only the registers grow with N_{FFT}.

Delay: For conventional and SSC FFT architectures, we estimate the delay as $cc \times T_{min}$ where cc is the number of clock cycles to finish the FFT computation, which grows with N_{FFT} and T_{min} is determined by the worst propagation delay in the logic data-path. To keep the comparison fair, for the ASC-FFT architecture we choose a computation time that results in the same SNR performance as the SSC-FFT architecture (latency column in table II). Fig. 9d shows delay (D) versus N_{FFT}. The computation delay for the ASC-FFT architecture is better for $N_{FFT} \geq 32$ and $N_{FFT} > 128$ than the One-BU and the pipelined architecture respectively.

Fig. 9e shows that the AxD for the ASC-FFT is better for $N_{FFT} > 16$ and $N_{FFT} > 32$ than the pipelined and the one-BU architectures respectively.

IV. CONCLUSION AND FUTURE WORK

Recent work demonstrated that using CTS reduces the latency, power and energy consumption for SC, opening the door for a new paradigm: ASC. However, this asynchronous approach has been limited to simple logic which constrains its application. In this paper, we first present a low-error asynchronous adder for CTS that incorporates a capacitor embedded in a feedback loop as a memory element to keep carry "time". This adder, enables more complex functions such as the ASC-FFT presented here. Our adder requires 70%-30% less transistors when compared to conventional and SC adders respectively. Besides, the ASC-FFT shows 3X less latency than SSC-FFTs. Furthermore, there is a significant advantage in hardware cost when compared with pipelined architectures. Overall, simulation results show that ASC-FFT outperforms both pipelined and one-BU FFT architectures for $N_{FFT} > 32$ and $N_{FFT} \geq 128$ in terms of area-delay efficiency. As future work we will evaluate the impact of the output interface on power, area and SNR performance. Due to improved area and latency performance, ASC-FFT architecture can be embedded as a promising accelerator for energy and area constrained systems such as sensors and IoT devices.

REFERENCES

[1] B. R. Gaines, "Stochastic Computing," in *Proceedings of the April 18-20, 1967, spring joint computer conference on - AFIPS '67*, p. 149, ACM Press, 1967.

[2] W. Qian *et al.*, "An Architecture for Fault-Tolerant Computation with Stochastic Logic," *IEEE Transactions on Computers*, vol. 60, pp. 93–105, 1 2011.

[3] M. H. Najafi *et al.*, "Time-Encoded Values for Highly Efficient Stochastic Circuits," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, pp. 1644–1657, 5 2017.

[4] M. H. Najafi *et al.*, "Polysynchronous stochastic circuits," in *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 492–498, IEEE, 1 2016.

[5] P. Gonzalez-Guerrero *et al.*, "SC-SD: Towards Low Power Stochastic Computing using Sigma Delta Streams," in *2018 IEEE International Conference on Rebooting Computing (ICRC)*, IEEE, 2018.

[6] P. Gonzalez-Guerrero *et al.*, "Ultra-low-power dual-phase latch based digital accelerator for continuous monitoring of wheezing episodes," in *IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S)*, pp. 1–2, IEEE, 10 2017.

[7] B. Yuan *et al.*, "Area-Efficient Error-Resilient Discrete Fourier Transformation Design using Stochastic Computing," in *Proceedings of the 26th edition on Great Lakes Symposium on VLSI*, pp. 33–38, ACM, 2016.

[8] B. Yuan *et al.*, "Area-Efficient Scaling-Free DFT/FFT Design Using Stochastic Computing," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 63, pp. 1131–1135, 12 2016.

[9] E. Vahapoglu *et al.*, "Accurate Synthesis of Arithmetic Operations with Stochastic Logic," in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 415–420, IEEE, 7 2016.

[10] Y. Liu *et al.*, "Hardware-Efficient Delta Sigma-Based Digital Signal Processing Circuits for the Internet-of-Things," *Journal of Low Power Electronics and Applications*, vol. 5, pp. 234–256, 11 2015.

[11] P. Crawley *et al.*, "Switched-current sigma-delta modulation for A/D conversion," in *[Proceedings] 1992 IEEE International Symposium on Circuits and Systems*, vol. 3, pp. 1320–1323, IEEE.

[12] J. Johnston, "Parallel pipeline fast fourier transformer," *IEE Proceedings F Communications, Radar and Signal Processing*, vol. 130, no. 6, p. 564, 1983.

[13] X. Xiao *et al.*, "An Efficient FFT Engine With Reduced Addressing Logic," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 55, pp. 1149–1153, 11 2008.

[14] L. Yang *et al.*, "An efficient locally pipelined FFT processor," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 53, pp. 585–589, 7 2006.