A 1 Million-Point FFT on a Single FPGA

Hans Kanders, Tobias Mellqvist, Mario Garrido Gálvez, Kent Palmkvist and Oscar Gustafsson

The self-archived postprint version of this journal article is available at Linköping University Institutional Repository (DiVA):

<http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-161406>

N.B.: When citing this work, cite the original publication.

Kanders, H., Mellqvist, T., Garrido Gálvez, M., Palmkvist, K., Gustafsson, O., (2019), A 1 Million-Point FFT on a Single FPGA, *IEEE Transactions on Circuits and Systems Part 1*, 66(10), 3863-3873.

<https://doi.org/10.1109/TCSI.2019.2918403>

Original publication available at:

<https://doi.org/10.1109/TCSI.2019.2918403>

Copyright: Institute of Electrical and Electronics Engineers (IEEE)

<http://www.ieee.org/index.html>

©2019 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

A 1 Million-Point FFT on a Single FPGA

Hans Kanders, Tobias Mellqvist, Mario Garrido, *Member, IEEE*,
Kent Palmkvist and Oscar Gustafsson, *Senior Member, IEEE*

Abstract—In this paper, we present the first implementation of a 1 million-point fast Fourier transform (FFT) completely integrated on a single field-programmable gate array (FPGA), without the need for external memory or multiple interconnected FPGAs. The proposed architecture is a pipelined single-delay feedback (SDF) FFT. The architecture includes a specifically designed 1 million-point rotator with high accuracy and a thorough study of the word length at the different FFT stages in order to increase the signal-to-quantization-noise ratio (SQNR) and keep the area low. This also results in low power consumption.

Index Terms—Binary tree, Cooley-Tukey, fast Fourier transform (FFT)

I. INTRODUCTION

NOWADAYS there is an increasing interest for the computation of large fast Fourier transforms (FFT). In radio astronomy, large FFTs allow for wide-band high-resolution spectrum analysis [1]–[3] and lead to precise images of all kinds of astronomical objects [4]. In radar, sonar and echo-graphy, large FFTs for pulse compression increase the range resolution and the signal-to-noise ratio [5]. Likewise, large FFT can be used to detect frequency hopping transmissions [6]. And in medicine, large FFTs provide high-resolution medical images [6].

In this paper, we focus on the design of a 1 million-point FFT, which has always been observed as a great technical challenge. Only a few years ago, researchers from the Massachusetts Institute of Technology claimed [6]: “Today, efficient million-point fast Fourier transform are not practical. Hardware implementations of such large FFTs are prohibitively expensive in terms of high energy consumption and large area requirements”. Indeed, current implementations of 1 million-point FFTs use various interconnected field-programmable gate arrays (FPGAs) [4]. The alternatives are to resort to application-specific integrated circuits [5] or, if the application allows for it, the use of a sparse FFT [6] instead of a conventional FFT.

In fact, the design of a 1 million-point FFT involves multiple technical challenges. The main ones are the selection of the FFT algorithm, the design of the 1 million-point rotator, W_{1M} , and the decision on the word length at the FFT stages. First, the selection of the FFT algorithms determines the rotators in the architecture. Due to their high complexity, it is desirable to include as few large rotators as possible, and the placement of the W_{1M} in the architecture is critical. Second, the W_{1M} rotator must achieve a resolution of 1 millionth part of the circumference. This demands a design that achieves this

resolution with a small rotation error. Third, the word length at the FFT stages has impact on the total data memory of the architecture, as well as on the accuracy of the computations. Thus, a good trade-off between memory size and accuracy needs to be achieved.

In this paper we present the first implementation of a 1 million-point FFT on a single FPGA, without the use of external memory or multiple interconnected FPGAs. The proposed architecture is a pipelined single-delay feedback (SDF) FFT [7], which allows for processing one sample per clock cycle. The FFT algorithm has been carefully selected to reduce the complexity of the rotators, the W_{1M} rotator has been designed to achieve high resolution and the word length throughout the FFT stages has been thoroughly selected to obtain high accuracy in the computations and a relatively small data memory size. Indeed, it is the first time that these challenges are studied in depth for such a large FFT. As a result, the proposed design not only requires small area, but also achieves high accuracy and an affordable power consumption.

This paper is organized as follows. In Section II, we review the previous FFT technology on which the proposed FFT is based. In Section III, we present the architecture of the proposed FFT. In Sections III-A to III-D, we describe the design of the architecture. Specifically, in Section III-A we discuss the selection of the FFT algorithm. In Section III-B, we describe the design of the W_{1M} rotator, whereas the design of other rotators is explained in Section III-C. And in Section III-D, we provide the study of the word length at the different FFT stages. Experimental results and comparison are discussed in Section IV. Finally, the main conclusions of the paper are provided in Section V.

II. BACKGROUND

A. FFT Algorithms

The N -point discrete Fourier transform (DFT) of an input sequence $x[n]$ is defined as

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{nk}, \quad k = 0, 1, \dots, N-1, \quad (1)$$

where $W_N^{nk} = e^{-j\frac{2\pi}{N}nk}$.

To calculate the DFT, the FFT based on the Cooley-Tukey algorithm [8] is mostly used. The Cooley-Tukey algorithm reduces the number of operations from $O(N^2)$ for the DFT to $O(N \log_2 N)$ for the FFT.

1) *The FFT Flow Graph*: Fig 1 shows the flow graph of a 16-point radix-2 FFT according to the Cooley-Tukey algorithm, decomposed using decimation in frequency (DIF) [9].

All the authors are with the Department of Electrical Engineering, Linköping University, SE-581 83 Linköping, Sweden, contact e-mail: mario.garrido.galvez@liu.se

This work was supported by the the Swedish ELLIIT Program.

Fig. 1. Flow graph of a 16-point radix-2 DIF FFT.

The FFT consists of $n = \log_2 N$ stages. At each stage of the graph, $s \in \{1, \dots, n\}$, butterflies and rotations are calculated. The lower edges of the butterflies are always multiplied by -1 . These -1 are not depicted in order to simplify the graphs.

The numbers at the input represent the index of the input sequence, whereas those at the output are the frequencies, k , of the output signal $X[k]$. Finally, each number, ϕ , in between the stages indicates a rotation by:

$$W_N^\phi = e^{-j\frac{2\pi}{N}\phi}. \quad (2)$$

Whereas W_N^ϕ represents a single rotation value, W_L (with-out exponent) represents a kernel or set of rotations, where

$$W_L = W_L^k, \quad k = 0, \dots, L-1. \quad (3)$$

Rotations by the angles 0° , 90° , 180° and 270° , are called trivial rotations. These rotations correspond to multiplications by 1 , j , -1 and $-j$, respectively. They are called trivial rotations because they can be easily implemented by interchanging the real and imaginary parts and/or changing the sign of the data. Note that the twiddle factor W_4 only includes trivial rotations, whereas larger twiddle factors include other rotations as well.

Different radices only differ in the rotations at the different FFT stages [9], whereas the butterflies are the same. Thus, different algorithms lead to different distributions of rotations. This idea is used later when selecting the FFT algorithm for the 1 million-point FFT.

2) *The Binary Tree Representation:* The binary tree representation [10], [11] is based on dividing an FFT in sub FFTs of smaller size in a recursive manner. Starting with an N -point FFT, it is first split into two FFTs of sizes P and Q so that $N = PQ$. Then, the resulting P -point FFT and the Q -point FFT are split into two factors each. This is repeated iteratively until 2-point FFTs are reached. The result of this process is a binary tree.

Fig. 2. Binary tree representation. (a) Node split in a binary tree. (b) Binary tree for a 32-point radix-2 DIF FFT. (c) Binary tree for a 32-point radix- 2^3 , 2^2 DIF FFT.

Fig. 3. 32-point radix-2 SDF FFT architecture.

Fig. 2(a) shows how a node is split in the binary tree. The rooted node has the value n , where $N = 2^n$. Its branches have the values p and q , where $P = 2^p$ and $Q = 2^q$. As $N = PQ$, the value of a node in the tree node equals the sum of its sub-nodes, i.e., $n = p + q$.

The values p and q are chosen arbitrarily at each split, as long as they respect $n = p + q$. Thus, different splits lead to different trees. Figs. 2(b) and 2(c) show two examples of binary trees for a 32-point FFT. Note that every node respects the equation $n = p + q$ in both trees.

Each node in a binary tree corresponds to an FFT twiddle factor. On the one hand, the node number indicates the complexity of the twiddle factor. The twiddle factor for a node with value l has $L = 2^l$ angles and represents a W_L rotator. On the other hand, the position of the node determines in which stage of the FFT architecture the corresponding rotator must be placed. This is obtained directly by following the binary tree from left to right, i.e., the leftmost node corresponds to the twiddle factor in the first stage of the architecture, the second node to the second stage, and so on. For instance, the algorithm in Fig. 2(c) yields the order 3-2-5-2. Thus, the twiddle factors in the stages 1 to 4 of the corresponding FFT are W_8 , W_4 , W_{32} and W_4 , respectively.

B. Pipelined FFT Architectures

The high performance requirements of contemporary real-time applications make pipelined FFT architectures [12]–[29] useful. They provide high throughput and reasonably low area and power consumption. The two main types of pipelined FFT architectures are feedback and feedforward. Both types can be implemented with serial or parallel structures. In the case of feedback, these are called single-path delay feedback (SDF) [12]–[15] and multi-path delay feedback (MDF) [16]–[21], respectively. For feedforward architectures, these are known as single-path delay commutator (SDC) [22]–[25] and multi-path delay commutator (MDC) [26]–[29].

1) *Single-Path Delay Feedback (SDF) FFT*: Allowing for high throughput and a low amount of hardware resources makes the SDF architecture one of the most attractive and frequently used architectures. Fig. 3 illustrates the structure of a 32-point radix-2 SDF FFT. It consists of a series of $n = \log_2(32) = 5$ interconnected stages, where each stage is composed of a radix-2 butterfly and a rotator. The outputs of the butterflies are connected to first-in-first-out (FIFO) buffers in a feedback loop, which allows for the single-path input and processing of one sample per clock cycle.

Each stage $s \in \{1, \dots, n\}$ of the SDF FFT calculates all the butterflies and rotations of the corresponding stage in the FFT flow graph. The same way goes for the binary tree representation. The rotator order explained in Section II-A2 describes the stages of the SDF architecture where the different rotators are placed. For instance, the twiddle factors at the FFT stages of Fig. 3 may be 5-4-3-2 as in Fig. 2(b), or 3-2-5-2 as in Fig. 2(c), or another pattern that forms a binary tree.

C. Rotations in Fixed-Point Arithmetic

As explained in [30], a rotation in a digital system is calculated as

$$\begin{bmatrix} X_D \\ Y_D \end{bmatrix} = \begin{bmatrix} C & -S \\ S & C \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}, \quad (4)$$

where $x + jy$ is the complex input, $X_D + jY_D$ is the output, and C and S are the real and imaginary components of the rotation coefficient to rotate by an angle α .

Due to finite word length effects in fixed-point arithmetic, the phase of the complex number $C + jS$ is an approximation of the rotation angle α according to

$$\begin{aligned} C &= R \cdot (\cos \alpha + \epsilon_c), \\ S &= R \cdot (\sin \alpha + \epsilon_s), \end{aligned} \quad (5)$$

where ϵ_c and ϵ_s are the relative quantization errors of the coefficients and R is the scaling factor. The rotation error for the angle α is then calculated as $\epsilon = \sqrt{\epsilon_c^2 + \epsilon_s^2}$, which is the distance between the exact and the quantized rotation. This is depicted in Fig. 4. It should be noted that the scaling factor, R , of a rotation as in (4) can never be exactly a power of two with fixed-point C and S (unless for the trivial cases where $S, C \in 0, \pm 1$), and, hence, there will at least always be a magnitude error [31].

A set of coefficients $C_i + jS_i, i = 1, \dots, M$ is called a kernel set. For a set of rotation angles $\alpha_i, i = 1, \dots, M$, the error for each of the corresponding coefficients $C_i + jS_i$ is expressed as

$$\epsilon_i = \frac{\sqrt{(C_i - R \cos \alpha_i)^2 + (S_i - R \sin \alpha_i)^2}}{R}. \quad (6)$$

The rotation error of the entire set is then defined as the maximum rotation error of the individual angles, i.e.,

$$\epsilon_{max} = \max_i(\epsilon_i). \quad (7)$$

D. Review of previous large FFT architectures

Fig. 5 shows the previous architectures used to calculate a 1 million-point FFT. Fig. 5(a) shows the architectures in [1], [2],

Fig. 4. Calculation of the rotation error, ϵ

Fig. 5. Previous 1 million-point FFT architectures. (a) Pipelined FFT using external memory [1], [2]. (b) Memory-based FFT [5]. (c) Multi-FPGA FFT [4]. (d) Sparse FFT [6].

which are very similar. They make use of two pipelined FFTs of sizes L and M so that $N = LM$ is the total FFT size. In between the two pipelined FFTs there is a rotator to multiply by the twiddle factors stored in memory and a block used for matrix transposition that is in charge of transforming the data order at the output of the rotator into the order required by the M -point FFT. This transposition is implemented by a memory which is external to the FPGA. Fig. 5(b) shows the architecture in [5]. It is a memory-based FFT where the FFT is calculated in 2 epochs. The processing is done by a computation array that consists in 6 processing elements (PE).

Fig. 6. Proposed 1M-point SDF FFT.

Fig. 5(c) shows the approach presented in [4]. This approach makes use of 12 FPGAs, where 4 of them are used for control and 8 for the FFT computations. Finally, Fig. 5(d) shows the architecture for the sparse FFT in [6]. The architecture obtains the largest 500 frequency coefficients from the total 2^{20} frequency coefficients. The idea of this algorithm is to map the 2^{20} frequencies into 4096 buckets. After that, the buckets with larger magnitude are selected and the largest frequencies are located inside the buckets. It is worth noting that the sparse FFT is not a conventional FFT, but a simplification that can be applied only when the input signals are sparse.

III. PROPOSED 1 MILLION-POINT FFT

Fig. 6 shows the architecture of the proposed 1 million-point FFT. It is a radix- 2^4 SDF FFT with $n = \log_2 N = 20$ stages. Each stage consists of a butterfly (R2), a buffer, and a rotator. Trivial rotators are depicted with a diamond shape, W_{16} rotators with squares, complex multipliers with a circle and the W_{1M} rotator with an octagon. The size of the buffers starts at $N/2$ in the first stage and is halved at each consecutive stage.

The challenges faced in the design of the 1 million-point FFT architecture and the proposed solutions are described in Sections III-A to III-D.

A. Selection of the FFT Algorithm

The FFT algorithm has an impact on the rotator placement in an SDF architecture. Since the W_{1M} rotator has the highest complexity of all the rotators, it is preferred to place it as late as possible in the architecture. The reason for this is that the large number of angles in the rotator demands a large data word length in order to achieve high accuracy in the calculations. According to this, it is preferable to have the increase in word length as late as possible in the architecture, so that there are less stages with large word length. This will have an impact in the size of the buffers of the SDF stages. Since the size of the buffers is cut in half for each stage in the FFT, a late placement of the W_{1M} rotator will reduce the required memory.

Fig. 7. Alternative algorithms for the 1 million-point FFT. (a) Radix- 2^5 with split 10-20-10. (b) Radix- 2^4 with split 12-20. (c) Radix- 2^4 with split 16-20-4.

Furthermore, it is desired to use as many trivial rotators as possible in the architecture, and also reduce the complexity of the rest of rotators. For this purpose, radix- 2^4 and radix- 2^5 are a good alternatives.

By combining the use of radix- 2^4 or radix- 2^5 and the late placement of the W_{1M} , there exist three alternative algorithms, which are shown in Fig. 7. The algorithms are denoted 10-20-10, 12-20-8 and 16-20-4, which comes from the split in the root node. The rotators for the three alternative algorithms are shown in Table I. The first column shows the stages of the FFT and the second through fourth columns show the rotators of the different algorithms. In the table, trivial rotators are highlighted in light gray and the W_{1M} rotator in dark gray.

Firstly, it is obvious that 10-20-10 contains more non-trivial rotators than 12-20-8 and 16-20-4 which is not desirable. In 10-20-10, the W_{1M} rotator also appears the earliest in

TABLE I
ROTATORS FOR THE FFT ALGORITHMS IN FIG. 7.

Stage	Algorithm		
	10-20-10	12-20-8	16-20-4
1	W_4	W_4	W_4
2	W_8	W_{16}	W_{16}
3	W_{32}	W_4	W_4
4	W_4	W_{256}	W_{256}
5	W_{1024}	W_4	W_4
6	W_4	W_{16}	W_{16}
7	W_8	W_4	W_4
8	W_{32}	W_{4096}	W_{65k}
9	W_4	W_4	W_4
10	W_{1M}	W_{16}	W_{16}
11	W_4	W_4	W_4
12	W_8	W_{1M}	W_{256}
13	W_{32}	W_4	W_4
14	W_4	W_{16}	W_{16}
15	W_{1024}	W_4	W_4
16	W_4	W_{256}	W_{1M}
17	W_8	W_4	W_4
18	W_{32}	W_{16}	W_{16}
19	W_4	W_4	W_4
Trivial rotators	8	10	10

Fig. 8. Decomposition of the W_{1M} into three stages: W_4 , W_{512} and a nano-rotator.

the pipeline. This algorithm is therefore discarded, because it would use more memory. Between 12-20-8 and 16-20-4, both have the same number of trivial rotators. In 16-20-4 the W_{1M} rotator appears the latest in the pipeline, but another very large rotator, W_{65k} , is required. Hence, this algorithm is also discarded because it would require the implementation of another large rotator. This leaves us with 12-20-8. This is the chosen algorithm since it has the least number of non-trivial rotators, places W_{1M} later than 10-20-10 and does not introduce another large rotator as the 16-20-4 architecture does.

B. Design of the W_{1M} Rotator

A good strategy for large rotators is to use a series of rotation stages to carry out the rotation. The proposed W_{1M} rotator uses this approach and calculates the rotation by three rotators connected in series: A trivial rotator, W_4 , a W_{512} rotator and a nano-rotator. This is shown in Fig. 8. The design of the nano-rotator and the W_{512} rotator is explained in Sections III-B1 and III-B2, respectively.

1) *The Nano-Rotator*: The nano-rotator is used to rotate by small angles, $\alpha_{in} < 1^\circ$ [32] and is needed in the W_{1M} in order to reach the high resolution of the rotator.

Consider the coefficient set $P_k = C + jk$ where C is a constant, $k = 0, \dots, M$ and $M \ll C$. The angle set is determined by

$$\alpha_k = \tan^{-1} \left(\frac{k}{C} \right). \quad (8)$$

Fig. 9. Error of the nano-rotator. (a) By using only positive angles. (b) By using positive and negative angles.

Fig. 10. Structure of the nano-rotator

Since α_k is small, the relation $\alpha_k \approx \tan(\alpha_k)$ holds and can be simplified to $\alpha_k \approx \frac{k}{C}$. This simplified rotation can be described as

$$\begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} C & -k \\ k & C \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}. \quad (9)$$

The constant C is calculated by considering the smallest rotation that the nano-rotator can carry out, α_{min} . For this angle, it is fulfilled that $k = 1$. Therefore, C can be obtained from equation (8) as

$$C = \frac{k}{\tan(\alpha_k)} = \frac{1}{\tan(\alpha_{min})}. \quad (10)$$

In the 1 million-point FFT the smallest rotation is obtained by dividing the circumference in one million parts, i.e., $\alpha_{min} = 2\pi/2^{20}$. This leads to

$$C = \frac{1}{\tan\left(\frac{2\pi}{2^{20}}\right)} = 166886. \quad (11)$$

As $\alpha_{in} < 1^\circ$ for the nano-rotator, the other rotators must reduce the remaining rotation angle to a value smaller than 1° . A W_L rotator makes the remaining angle be $2\pi/L$, i.e., the L -th fraction of the circumference. As L is a power of 2, in order to make this angle smaller than 1° , L must be at least 512. According to this, the maximum input angle to the nano-rotator is $\alpha_{in} \approx 0.70^\circ$. This also explains why the W_{512} rotator is used in the 1 million-point FFT.

Fig. 10 shows the structure of the nano-rotator. Since C is constant, Cx and Cy are calculated by using shift-and-add techniques instead of using expensive general multipliers. By

Fig. 11. General complex multiplier with a coefficient memory.

using the algorithm in [33], each multiplication by C requires 8 adders.

The products $-ky$ and kx require general multipliers but no rotation coefficients needs to be stored in memory, as k is directly the value ϕ of the angle according to [9]. Thus, no coefficient memory is needed for the nano-rotator. Due to the large number of rotation angles in this stage, this represents significant memory savings.

2) *The W_{512} Rotator:* The W_{512} rotator works as a bridge between the trivial rotator and the nano-rotator. Although it is the second stage of the W_{1M} , it depends on the nano-rotator and its design is done later.

As the W_{512} is preceded by a trivial rotator, the W_{512} rotator only needs to rotate in a single quadrant of the complex plane. Therefore, the rotator is designed by using the 129 angles in the angle set

$$\alpha_i = \left\{ 0, \frac{2\pi}{512}, \frac{2 \cdot 2\pi}{512}, \dots, \frac{126 \cdot 2\pi}{512}, \frac{127 \cdot 2\pi}{512}, \frac{\pi}{2} \right\}. \quad (12)$$

Since the W_{512} rotator is a medium-sized rotator, it is a good choice to implement it as a complex multiplier rotator where the rotation coefficients are stored in memory. This is shown in Fig. 11.

The scaling of the W_{1M} rotator is calculated as the product of the scalings of the rotators that it involves, i.e.,

$$R_{1M} = R_4 \cdot R_{512} \cdot R_{nano}. \quad (13)$$

Unity scaling [14] for the W_{1M} rotator is achieved when $R_{1M} = 2^h$, $h \in \mathbb{N}$. As the scaling of the trivial rotator is $R_4 = 1$, then

$$R_{512} = \frac{2^h}{R_{nano}}. \quad (14)$$

The value h is selected by considering the total rotation error of the W_{1M} rotator. Fig. 12 shows this error as a function of the number of bits h that the W_{1M} rotator scales the input. It can be observed that the error decreases and then it almost stays constant. Therefore, we choose $h = 37$ as the number of bits scaled by the W_{1M} rotator. This results in a rotation error $\epsilon = 1.02 \cdot 10^{-5}$, which corresponds to an effective word length [34] of $WL_E = 18.08$, which means that the 18 MSBs after the rotation do not have error.

By considering $h = 37$, the scaling of the W_{512} results in

$$R_{512} = \frac{2^{37}}{R_{nano}} \approx 823542. \quad (15)$$

which corresponds to coefficients with word length $WL_C = \lceil \log_2 R_{512} \rceil = 20$ bits, where $\lceil \cdot \rceil$ denotes a ceiling operation.

Fig. 12. Rotation error ϵ_{1M} as a function of the number of bits, h , that the W_{1M} rotator scales its input. Fig. 13. Coefficient symmetry for a W_L rotator.

This is the unsigned representation. However, since the multiplications are signed, the coefficients will use 21 bits in 2's complement representation.

The coefficients C and S for the W_{512} rotator are then obtained as

$$\begin{aligned} C &= \lfloor R_{512} \cdot \cos \alpha_i \rfloor, \\ S &= \lfloor R_{512} \cdot \sin \alpha_i \rfloor, \end{aligned} \quad (16)$$

where $\lfloor \cdot \rfloor$ denotes the rounding operation.

Due to symmetry in the complex plane, not all coefficients need to be stored. Fig. 13 clarifies this. Here, $C' + jS'$ represents the coefficients that are fed into the multipliers, and $C + jS$ are the values stored in memory. The only unique coefficient values are found in the first half of a quadrant, i.e., $[0, \dots, \frac{\pi}{4}]$. This means that out of 129 different angles, only 65 angle coefficients need to be stored in memory. The rest of the angles are obtained by swapping the values of C and S .

3) *Control Unit:* The rotation angle $\alpha_{W_{1M}}$ of the W_{1M} rotator has to be translated to the rotation angles $\alpha_{W_4}, \alpha_{W_{512}}, \alpha_{nano}$ for each rotator stage to fulfill

$$\alpha_{W_{1M}} = \alpha_{W_4} + \alpha_{W_{512}} + \alpha_{nano}, \quad (17)$$

where

$$\begin{aligned}
\alpha_{W_4} &= 90^\circ \cdot i, & i &\in \{0, 1, 2, 3\}, \\
\alpha_{W_{512}} &= 0.703^\circ \cdot j, & j &\in \{0, 1, 2, \dots, 128\}, \\
\alpha_{nano} &= \frac{360^\circ}{2^{20}} \cdot k, & k &\in \{-1023, -1022, \dots, 1024\},
\end{aligned} \tag{18}$$

and i, j, k are the control signals to the rotator stages. Considering the fact that the angles of the nano-rotator are positive and negative, as discussed in Section III-B1, the rotation angles for the stages of the W_{1M} rotator are calculated as

$$\begin{aligned}
\alpha_{W_4} &= \left\lfloor \frac{\alpha_{W_{1M}}}{90^\circ} \right\rfloor \\
\alpha_{W_{512}} &= \left\lceil \frac{\alpha_{W_{1M}}}{360^\circ/512} \right\rceil \\
\alpha_{nano} &= \alpha_{W_{1M}} - \left\lceil \frac{\alpha_{W_{1M}}}{360^\circ/512} \right\rceil \cdot \frac{360^\circ}{512},
\end{aligned} \tag{19}$$

where $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ denote floor and rounding operations, respectively.

C. Design of the Other Rotators in the FFT

1) *Design of the W_{16} Rotators:* The W_{16} rotators in stages 2, 6, 10, and 14 of the proposed architecture are implemented as shift-and-add. This is more hardware-efficient than using a complex multiplier due to the small number of angles in these rotators.

Accurate rotators with few adders for W_{16} are provided in Table VII in [34]. From this table, we have selected the rotator by

$$\begin{aligned}
C_0 + j \cdot S_0 &= 21059, & \text{for } \alpha_0 = 0^\circ, \\
C_1 + j \cdot S_1 &= 19456 + j \cdot 8059, & \text{for } \alpha_1 = 22.5^\circ, \\
C_2 + j \cdot S_2 &= 14891 + j \cdot 14891, & \text{for } \alpha_2 = 45^\circ,
\end{aligned} \tag{20}$$

which requires 10 adders and has a rotation error $\epsilon = 2.67 \cdot 10^{-5}$, which corresponds to an effective word length of $WL_E = 16.69$.

2) *Design of the W_{256} and W_{4096} Complex Multiplier Rotators:* The W_{256} and W_{4096} rotators have intermediate size. For this size it is not advantageous to use shift-and-add, so we have implemented these rotators by using a memory and a complex multiplier as in Fig. 11. This has the advantage that the scaling of the rotator can be selected arbitrarily by simply scaling the values of the coefficients stored in memory. In Section III-D we take advantage of this fact in order to select the scaling of the FFT stages.

Due to the symmetries shown in Fig. 13, only $L/8 + 1$ coefficients are required for a W_L rotator. Thus, the W_{256} and W_{4096} rotators only need to store 33 and 513 coefficients, respectively. These are small memories compared to the large data memory required by the proposed FFT.

D. Scaling, Memory Usage and SQNR

Scaling throughout the FFT stages, data memory usage and signal-to-quantization-noise ratio (SQNR) are intimately related. Scaling involves all the effects that modify the amplitude of the signal throughout the FFT stages. This includes the increase of one bit in the butterflies, the scaling of the rotators and the truncation at the end of each stage to reduce the word length and keep it at a reasonable size. Likewise, the word

Fig. 14. Signal scaling to avoid overflow.

length at each stage has impact in the data memory of that stage and in the SQNR of the FFT.

Table II shows the scaling throughout the stages of the 1 million-point FFT. In the table, s indicates the stage, WL_I is the input word length at the corresponding stage, WL_B is the word length at the output of the butterfly of the stage, $Rotator$ identifies the rotator in that stage, R is the scaling of the rotator, WL_R is the word length at the output of the rotator, T_{MSB} and T_{LSB} are the number of most and least significant bits that are truncated after the rotator, respectively, WL_O is the output word length at the corresponding stage, S is the scaling of the stage, S_{NORM} is the normalized scaling and S_{ACC} is the accumulated scaling.

As $WL_B = WL_I + 1$, at each stage of the FFT the butterflies increase the word length by one bit. This is required in order to represent the value at the output of the butterfly without losing accuracy. The rotators increase the word length from WL_B to WL_R and, then, T_{MSB} and T_{LSB} are applied to reduce the word length to WL_O . As all stages are interconnected, it is fulfilled that $WL_I(s) = WL_O(s-1)$.

The scaling of the stage is calculated from the scaling of the rotator and T_{LSB} as

$$S = \frac{R}{2^{T_{LSB}}}. \tag{21}$$

The normalized scaling is calculated as

$$S_{NORM} = \frac{S}{2^{WL_O - WL_I - 1}}. \tag{22}$$

In practice, both S and S_{NORM} are the same when the word length increases by one throughout the stage and $S_{NORM} = 2 \cdot S$ when the word length is kept constant throughout the stage.

The accumulated scaling is then obtained as

$$S_{ACC}(s) = \prod_{i=1}^s S_{NORM}(i). \tag{23}$$

The selection of the truncation at the different stages is done in such a way that S_{ACC} is in the range $[1/(2\sqrt{2}), 1/\sqrt{2}] \approx [0.35, 0.70]$. The reason for this is illustrated in Fig. 14. When we normalize a complex number to its word length, the number will be inside the large square with side length 1, and its magnitude can be up to $\sqrt{2}$. However, if this number is rotated, it may exceed the limits of the square, causing overflow. The way to solve this issue is to scale the numbers by $1/\sqrt{2} \approx 0.70$. This sets the maximum value of the real and imaginary

TABLE II
SCALING IN THE 1 MILLION-POINT FFT

s	WL_I	WL_B	Rotator	R	WL_R	T_{MSB}	T_{LSB}	WL_O	S	S_{NORM}	S_{ACC}
1	16	17	W_4	1	17	0	0	17	1.00	1.00	1.00
2	17	18	W_{16}	21059	33	0	15	18	0.64	0.64	0.64
3	18	19	W_4	1	19	0	0	19	1.00	1.00	0.64
4	19	20	W_{256}	212000	38	0	18	20	0.81	0.81	0.52
5	20	21	W_4	1	21	0	0	21	1.00	1.00	0.52
6	21	22	W_{16}	21059	37	1	14	22	1.29	1.29	0.67
7	22	23	W_4	1	23	0	0	23	1.00	1.00	0.67
8	23	24	W_{4096}	161575	42	0	18	24	0.62	0.62	0.41
9	24	25	W_4	1	25	0	0	25	1.00	1.00	0.41
10	25	26	W_{16}	21059	41	1	14	26	1.29	1.29	0.53
11	26	27	W_4	1	27	0	0	27	1.00	1.00	0.53
12	27	28	W_{1M}	2^{37}	55	0	37	28	1.00	1.00	0.53
13	28	29	W_4	1	29	0	0	29	1.00	1.00	0.53
14	29	30	W_{16}	21059	45	1	14	30	1.29	1.29	0.68
15	30	31	W_4	1	31	0	1	30	0.50	1.00	0.68
16	30	31	W_{256}	212000	49	0	19	30	0.40	0.81	0.55
17	30	31	W_4	1	31	0	1	30	0.50	1.00	0.55
18	30	31	W_{16}	21059	46	1	15	30	0.64	1.29	0.70
19	30	31	W_4	1	31	0	1	30	0.50	1.00	0.70
20	30	31	-	-	31	0	1	30	0.50	1.00	0.70

parts of the number to 0.70 and its maximum magnitude to 1, which guarantees that no operation in the FFT will overflow. For the lower limit of 0.35 we know that if the scaling makes the signal smaller than that value, then the most significant bit (MSB) is unused, and we can remove the MSB instead of a least significant bit (LSB) to set the scaling again between 0.35 and 0.70. This makes an efficient use of the bits. Note that in some stages in Table II, one MSB is removed instead of one LSB for this reason.

As shown in Table II, the maximum word length at the input of any stage is 30. In general, it is a good practice to increase the word length by one bit in the first stages of the FFT and then keep it constant [35] by truncating one LSB more at each stage, as is done for the proposed FFT. However, to decide when to stop increasing the input word length to the stages is not straightforward. In our design, we have taken this decision by observing the influence of the word length on the SQNR and the data memory. On the one hand, the data memory for the proposed SDF FFT is calculated as

$$\text{Data memory (Bits)} = 2 \sum_{s=1}^{20} 2^{20-s} \cdot WL_B(s) \quad (24)$$

where the number 2 corresponds to the real and imaginary part of the data.

On the other hand, the SQNR is a measure of accuracy and is calculated as [35]

$$\text{SQNR (dB)} = 10 \cdot \log_{10} \left(\frac{E \{ |X_{ID}|^2 \}}{E \{ |X_Q - X_{ID}|^2 \}} \right), \quad (25)$$

where E represents the expected value, X_Q is the output of the quantized FFT, and X_{ID} is the output of the ideal FFT without quantization.

By combining SQNR and data memory, Fig. 15 shows the variation of these parameters as a function of the maximum word length at the input of the FFT stages or, in other words, at which word length we decide to keep the word length constant for the remaining stages of the FFT. As the

Fig. 15. SQNR and data memory as a function of the maximum word length at the input of the FFT stages.

maximum word length increases, the SQNR also increases until it remains approximately constant. The same occurs with the data memory. Note that the last stages of the FFT have small buffers. Thus, a few bits more in those buffers does not represent a significant change in the total memory of the FFT. According to this, the value of 30 is chosen as the value where the SQNR does not grow more. Furthermore, the SQNR of the proposed architecture is

$$\text{SQNR} = 95.6 \text{ dB}, \quad (26)$$

which is a high value, and the data memory is

$$\text{Data memory} = 37.75 \text{ Mbit}. \quad (27)$$

Fig. 16 shows the power spectrum of the proposed FFT for a sinusoid with frequency $f = 1024$ and maximum amplitude. Note that, due to the scale, this frequency appears at the very left of the figure, overlapping with the figure frame. The power is represented in decibels and it is normalized to the power

Fig. 16. Power spectrum of the proposed FFT for a sinusoid with frequency $f = 1024$ and maximum amplitude.

TABLE III
PLACE AND ROUTE RESULTS FOR THE 1 MILLION-POINT FFT

Resource	Used	Available	Utilization (%)
Flip Flops	30 299	1 075 200	3
LUTs	16 827	537 600	3
I/O	96	832	12
BRAMs	1 157	1 728	67
DSP slices	30	768	4
Clock Frequency	233 MHz		
On-chip Power	3.436 W		
Junction Temperature	27.8 °C		

Device: xcvu095-ffvd1924-2-e-es1

at the frequency of the sinusoid. It can be observed that the spectrum has spurious with power below -100 dB. The highest spurious has a power of -100.42 dB.

IV. EXPERIMENTAL RESULTS AND COMPARISON

The 1 million-point FFT has been implemented on the Xilinx Virtex Ultrascale VCU107 Evaluation Board which hosts the xcvu095-ffvd1924-2-e-es1 FPGA. Table III shows an extract from the Xilinx Vivado 2014.3.1 place and route report. The report shows that the proposed 1 million-point FFT fits on a single FPGA, leaving a large amount of resources available. The highest utilization is on block random access memories (BRAM). This was expected due to the large number of points that the FFT processes. For the rest of resources the utilization is low. Regarding digital signal processing (DSP) slices, the number is small because only the W_{1M} , the W_{4096} and the two W_{256} use DSPs. The W_{16} rotators in the architecture are implemented as shift-and-add using LUTs instead of DSPs, which is more hardware efficient due to the small size of these rotators.

Table IV compares the proposed design to previous 1 million-point FFTs on FPGAs and application-specific integrated circuits (ASIC). For the 1 million-point FFT, the use of a FPGAs has two main advantages with respect to ASICs. First, they offer reconfigurability, i.e., the hardware can be updated later, which is not possible in ASICs. Second, the

applications where the 1 million-point FFT is used generally require a few units. In this context, ASICs become extremely expensive. This is due to the fact that the high production cost of ASICs is distributed among all the produced units. Thus, ASICs only achieve a low cost per unit when millions of them are produced.

Each architecture in Table IV is different, including a sparse FFT, a memory-based FFT, an FFT that uses parallel processing and our pipelined FFT. The highest throughput is achieved by [4]. However, it requires 12 FPGAs (8 for the FFT computations and 4 for control). If we normalize the number of FFT calculations per second by the number of devices, the proposed approach achieves 222 FFTs/Device/s, which is higher than in [5] and [4].

Regarding the sparse FFT in [6], it achieves higher throughput and lower latency than the proposed FFT. However, the work in [6] is an algorithm different to the FFT that can only be used when the signals are sparse. Therefore, the work in [6] simplifies the computations when the signal is sparse, but cannot be used to calculate a conventional FFT.

As a result, the proposed approach achieves the highest performance for a conventional 1 million-point FFT so far. Furthermore, the proposed design achieves a high SQNR and a power consumption of 3.436 W. This power consumption is taken from the power report in Vivado and is generated from the routed design.

V. CONCLUSIONS

In this work we have presented a 1 million-point FFT calculated on a single FPGA. The proposed design is the result of a careful study of the problem that takes into account the algorithm selection, the implementation of an efficient W_{1M} rotator and the adjustment of the scaling throughout the FFT stages.

As a result, the proposed 1 million-point FFT has high accuracy with a SQNR of 95.6 dB and is able to process 233 MS/s, which is equal to 222 full FFT calculations per second. Furthermore, the implementation not only fits on a single FPGA, but also leaves a large amount of resources available.

Compared to previous 1 million-point FFTs the proposed approach achieves the highest performance normalized to the number of devices so far.

REFERENCES

- [1] M. P. Quirk, M. F. Garyantes, H. C. Wilck, and M. J. Grimm, "A wide-band high-resolution spectrum analyzer," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 36, no. 12, pp. 1854–1861, Dec. 1988.
- [2] C. He, W. Qiang, G. Zhenbin, and W. Hongxing, "A pipelined memory-efficient architecture for ultra-long variable-size FFT processors," in *Proc. IEEE Int. Conf. Comput. Science Information Tech.*, Aug. 2008, pp. 357–361.
- [3] G. Morris and H. Wilck, "JPL 2²⁰ channel 300 MHz bandwidth digital spectrum analyzer," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, vol. 3, Apr. 1978, pp. 808–811.
- [4] T. Kamazaki, S. K. Okumura, Y. Chikada, T. Okuda, Y. Kurono, S. Iguchi, S. Mitsuishi, Y. Murakami, N. Nishimuta, H. Mita, and R. Sano, "Digital spectro-correlator system for the Atacama compact array of the Atacama large millimeter/submillimeter array," *Publ. Astron. Soc. Japan*, vol. 64, no. 2, p. 29, Apr. 2012.

TABLE IV
COMPARISON OF 1 MILLION-POINT FFTS

	[6]	[5]	[4]	Proposed
FFT size	1M	1M	1M	1M
Device	Virtex 6	ASIC	Virtex 4	Virtex Ultrascale
Architecture	Sparse	Memory-based	Parallel processing	Pipelined
f_{CLK} (MHz)	100	500	-	233
Throughput (MS/s)	860	70	2091	233
Processing time (ms)	1.16	14.48	0.5	4.29
FFTs/s	820	67	2000	222
Number of devices	1	1	12	1
FFTs/Device/s	820	67	166	222
WL_I	32	32	3	16
WL_O	32	32	-	30
SQNR (dB)	-	-	-	95.6
Power (W)	-	-	-	3.436

- [5] F. Han, L. Li, K. Wang, F. Feng, H. Pan, B. Zhang, G. He, and J. Lin, "An ultra-long FFT architecture implemented in a reconfigurable application specified processor," *IEICE Electron. Express*, vol. 13, no. 13, pp. 1–12, Jun. 2016.
- [6] A. Agarwal, H. Hassanieh, O. Abari, E. Hamed, D. Katabi, and Arvind, "High-throughput implementation of a million-point sparse Fourier transform," in *Proc. Int. Workshop Field-Programmable Logic Applications*, Sep. 2014, pp. 1–6.
- [7] M. Garrido, F. Qureshi, J. Takala, and O. Gustafsson, "Hardware architectures for the fast Fourier transform," in *Handbook of Signal Processing Systems*, 3rd ed., S. S. Bhattacharyya, E. F. Deprettere, R. Leupers, and J. Takala, Eds. Springer, 2019.
- [8] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. Comput.*, vol. 19, pp. 297–301, 1965.
- [9] M. Garrido, "A new representation of FFT algorithms using triangular matrices," *IEEE Trans. Circuits Syst. I*, vol. 63, no. 10, pp. 1737–1745, Oct. 2016.
- [10] H.-Y. Lee and I.-C. Park, "Balanced binary-tree decomposition for area-efficient pipelined FFT processing," *IEEE Trans. Circuits Syst. I*, vol. 54, no. 4, pp. 889–900, April 2007.
- [11] F. Qureshi and O. Gustafsson, "Generation of all radix-2 fast Fourier transform algorithms using binary trees," in *Proc. European Conf. Circuit Theory Design*, Aug. 2011, pp. 677–680.
- [12] L. Yang, K. Zhang, H. Liu, J. Huang, and S. Huang, "An efficient locally pipelined FFT processor," *IEEE Trans. Circuits Syst. II*, vol. 53, no. 7, pp. 585–589, Jul. 2006.
- [13] S. He and M. Torkelson, "Design and implementation of a 1024-point pipeline FFT processor," in *Proc. IEEE Custom Integrated Circuits Conf.*, May 1998, pp. 131–134.
- [14] M. Garrido, R. Andersson, F. Qureshi, and O. Gustafsson, "Multiplierless unity-gain SDF FFTs," *IEEE Trans. VLSI Syst.*, vol. 24, no. 9, pp. 3003–3007, Sep. 2016.
- [15] A. Cortés, I. Vélez, and J. F. Seviliano, "Radix r^k FFTs: Matricial representation and SDC/SDF pipeline implementation," *IEEE Trans. Signal Process.*, vol. 57, no. 7, pp. 2824–2839, Jul. 2009.
- [16] S.-N. Tang, J.-W. Tsai, and T.-Y. Chang, "A 2.4-GS/s FFT processor for OFDM-based WPAN applications," *IEEE Trans. Circuits Syst. II*, vol. 57, no. 6, pp. 451–455, Jun. 2010.
- [17] H. Liu and H. Lee, "A high performance four-parallel 128/64-point radix-2⁴ FFT/IFFT processor for MIMO-OFDM systems," in *Proc. IEEE Asia Pacific Conf. Circuits Syst.*, Nov. 2008, pp. 834–837.
- [18] L. Liu, J. Ren, X. Wang, and F. Ye, "Design of low-power, 1GS/s throughput FFT processor for MIMO-OFDM UWB communication system," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2007, pp. 2594–2597.
- [19] J. Lee, H. Lee, S. in Cho, and S.-S. Choi, "A high-speed, low-complexity radix-2⁴ FFT processor for MB-OFDM UWB systems," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2006, pp. 210–213.
- [20] N. Li and N. van der Meijs, "A Radix 2² based parallel pipeline FFT processor for MB-OFDM UWB system," in *Proc. IEEE Int. SoC Conf.*, Sep. 2009, pp. 383 – 386.
- [21] S.-I. Cho, K.-M. Kang, and S.-S. Choi, "Implementation of 128-point fast Fourier transform processor for UWB systems," in *Proc. Int. Wireless Comm. Mobile Comp. Conf.*, Aug. 2008, pp. 210–213.
- [22] X. Liu, F. Yu, and Z. Wang, "A pipelined architecture for normal I/O order FFT," *Journal of Zhejiang University - Science C*, vol. 12, no. 1, pp. 76–82, Jan. 2011.
- [23] Y.-N. Chang, "An efficient VLSI architecture for normal I/O order pipeline FFT design," *IEEE Trans. Circuits Syst. II*, vol. 55, no. 12, pp. 1234–1238, Dec. 2008.
- [24] —, "Design of an 8192-point sequential I/O FFT chip," in *Proc. World Congress Eng. Comp. Science*, vol. II, Oct. 2012.
- [25] Z. Wang, X. Liu, B. He, and F. Yu, "A combined SDC-SDF architecture for normal I/O pipelined radix-2 FFT," *IEEE Trans. VLSI Syst.*, vol. 23, no. 5, pp. 973–977, May 2015.
- [26] M. Garrido, J. Grajal, M. A. Sánchez, and O. Gustafsson, "Pipelined radix-2^k feedforward FFT architectures," *IEEE Trans. VLSI Syst.*, vol. 21, no. 1, pp. 23–32, Jan. 2013.
- [27] M. Garrido, S. J. Huang, and S. G. Chen, "Feedforward FFT hardware architectures based on rotator allocation," *IEEE Trans. Circuits Syst. I*, vol. 65, no. 2, pp. 581–592, Feb. 2018.
- [28] Y.-W. Lin and C.-Y. Lee, "Design of an FFT/IFFT processor for MIMO OFDM systems," *IEEE Trans. Circuits Syst. I*, vol. 54, no. 4, pp. 807–815, Apr. 2007.
- [29] S. Li, H. Xu, W. Fan, Y. Chen, and X. Zeng, "A 128/256-point pipeline FFT/IFFT processor for MIMO OFDM system IEEE 802.16e," in *Proc. IEEE Int. Symp. Circuits Syst.*, Jun. 2010, pp. 1488–1491.
- [30] M. Garrido, O. Gustafsson, and J. Grajal, "Accurate rotations based on coefficient scaling," *IEEE Trans. Circuits Syst. II*, vol. 58, no. 10, pp. 662–666, Oct. 2011.
- [31] O. Gustafsson, "On lifting-based fixed-point complex multiplications and rotations," in *Proc. IEEE Symp. Comput. Arithmetic*, 2017.
- [32] M. Garrido, P. Källström, M. Kumm, and O. Gustafsson, "CORDIC II: A new improved CORDIC algorithm," *IEEE Trans. Circuits Syst. II*, vol. 63, no. 2, pp. 186–190, Feb. 2016.
- [33] Y. Voronenko and M. Püschel, "Multiplierless multiple constant multiplication," *ACM Trans. Algorithms*, vol. 3, pp. 1–39, May 2007.
- [34] M. Garrido, F. Qureshi, and O. Gustafsson, "Low-complexity multiplierless constant rotators based on combined coefficient selection and shift-and-add implementation (CCSSI)," *IEEE Trans. Circuits Syst. I*, vol. 61, no. 7, pp. 2002–2012, Jul. 2014.
- [35] D. Guinart and M. Garrido, "SQNR in FFT hardware architectures," Under review.

Hans Kanders completed his M.Sc. degree in applied physics and electrical engineering in 2018 at Linköping University. After that he has worked with FFT architectures and FPGAs at the Department of Electrical engineering at Linköping University and also as a lab assistant in digital electronics. He is currently working with DSP architectures at Mediatek.

Tobias Mellqvist completed his M.Sc. degree in applied physics and electrical engineering in 2018 at Linköping University. He is currently working at MediaTek where he is part of the development of DSP processors for wireless modems aimed at future 5G technology.

Mario Garrido (M'07) received the M.Sc. degree in electrical engineering and the Ph.D. degree from the Technical University of Madrid (UPM), Madrid, Spain, in 2004 and 2009, respectively. In 2010 he moved to Sweden to work as a postdoctoral researcher at the Department of Electrical Engineering at Linköping University. Since 2012 he is Associate Professor at the same department.

His research focuses on optimized hardware design for signal processing applications. This includes the design of hardware architectures for the calculation of transforms, such as the fast Fourier transform (FFT), circuits for data management, the CORDIC algorithm, and circuits to calculate statistical and mathematical operations. His research covers high-performance circuits for real-time computation, as well as designs for small area and low power consumption.

ation of transforms, such as the fast Fourier transform (FFT), circuits for data management, the CORDIC algorithm, and circuits to calculate statistical and mathematical operations. His research covers high-performance circuits for real-time computation, as well as designs for small area and low power consumption.

Kent Palmkvist (M'99) received the M.Sc. and Ph.D. degrees from Linköping University, Sweden, in 1991 and 1999 respectively. He is currently an Associate Professor at the Computer Engineering Division, Department of Electrical Engineering, Linköping University. His research interests the design and implementation process of digital signal processing systems from algorithm modifications down to architecture and register transfer level.

Oscar Gustafsson (S'98–M'03–SM'10) received the M.Sc., Ph.D., and Docent degrees from Linköping University, Linköping, Sweden, in 1998, 2003, and 2008, respectively.

He is currently an Associate Professor and Head of the Computer Engineering Division, Department of Electrical Engineering, Linköping University. His research interests include design and implementation of DSP algorithms and arithmetic circuits. He has authored and coauthored over 170 papers in international journals and conferences on these topics.

Dr. Gustafsson is a member of the VLSI Systems and Applications and the Digital Signal Processing technical committees of the IEEE Circuits and Systems Society. He has served as an Associate Editor for the IEEE Transactions on Circuits and Systems Part II: Express Briefs and Integration, the VLSI Journal. Furthermore, he has served and serves in various positions for conferences such as ISCAS, PATMOS, PrimeAsia, ARITH, Asilomar, Norchip, ECCTD, and ICECS.