World's Fastest FFT Architectures: Breaking the Barrier of 100 GS/s

Mario Garrido, *Member, IEEE*, Konrad Möller, and Martin Kumm, *Member, IEEE*

***Abstract*—This paper presents the fastest FFT hardware architectures so far. The architectures are based on a fully parallel implementation of the FFT algorithm. In order to obtain the highest throughput while keeping the resource utilization low, we base our design on making use of advanced shift-and-add techniques to implement the rotators, and on selecting the most suitable FFT algorithms for these architectures. Apart from high throughput and resource efficiency, we also guarantee high accuracy in the proposed architectures. For the implementation, we have developed an automatic tool that generates the architectures as a function of the FFT size, input word length and accuracy of the rotations. We provide experimental results covering various FFT sizes, FFT algorithms and FPGA boards. These results show that it is possible to break the barrier of 100 GS/s for FFT calculation.**

***Index Terms*—Fast Fourier transform (FFT), fully parallel, pipelined architecture.**

I. Introduction

With the increasing demand on throughput of current signal processing applications, pipelined parallel FFT architectures have become very popular in the last years [1]–[11]. These architectures are able to process a continuous data flow of several samples in parallel. The main types of parallel pipelined FFTs are multi-path delay commutator (MDC) [1]–[8] and multi-path delay feedback (MDF) [9]–[11]. Both of them allow for high throughput in the range from hundreds of mega samples per second (MS/s) to tens of giga samples per second (GS/s).

In this work, we aim for the highest possible throughput. In order to achieve it, we consider the use of fully parallel FFTs [4], [12]–[16]. These architectures calculate an N-point FFT in a continuous flow of $P = N$ samples in parallel. Thus, they correspond to the direct implementation of the FFT flow graph, i.e., each addition/rotation in the flow graph is directly translated into an adder/rotator in hardware. This represents the maximum parallelization that an N-point FFT can have.

Fully parallel FFTs are already used in applications such as compensation of chromatic dispersion inherent in optical fibers [12]–[14] and radar [17]. For compensation of chromatic dispersion, filters with samples rates of tens of GS/s are required. The filter length scales more or less linearly with the fiber length. Therefore, for long fibers, hundreds or even thousands of taps are required. An efficient way to implement these filters is to make use of fully parallel FFTs [12]–[14]. For radar applications, the high throughput of fully parallel FFTs allow for object detection over large bandwidths [17]. Another area of interest of fully parallel FFTs is for applications where iterative FFTs are implemented [18]–[20]. Note that in iterative FFTs [18]–[20], the butterfly of the processing element (PE) usually consists of an r-point fully parallel FFT, where r is the FFT radix. For high radices [19], a hardware-efficient fully parallel FFT can reduce significantly the hardware cost of the PE in the iterative FFT. Finally, given the high demands of 5G systems due to the use of multiple antennas [21], fully parallel architectures could be potentially used in future communication systems.

Although fully parallel FFTs have been known for a long time, no previous work in the literature addresses in detail the technical challenges of designing fully parallel FFT architectures.

A first technical challenge is the implementation of the rotators as shift-and-add operations. The high parallelization of fully parallel FFTs demands the use of a large number of rotators. Without a proper design of these rotators, the area of the FFT can increase considerably. The fact that all rotators in a fully parallel FFT rotate by a constant angle allows for using advanced shift-and-add constant multiplication techniques to minimize their resource complexity. In our approach, we exploit the use of existing methods for constant multiplications, including single constant multiplication (SCM) [22], [23], multiple constant multiplication (MCM) [24]–[28] and constant matrix multiplication (CMM) [29]–[31]. With the aim of designing the most efficient rotators, we also exploit different approaches to implement rotators in hardware. A number of them are discussed in [32].

A second challenge related to the design of the rotators is the accuracy of the FFT calculations. Here, the coefficient selection [33], [34] plays an important role in the design of rotators. A good coefficient selection results in coefficients that calculate accurate rotations using few adders [34]. This guarantees high accuracy for the entire FFT.

A third challenge related to rotations is the selection of the FFT algorithms. FFT algorithms based on the Cooley-Tukey approach only differ in the rotations at different stages [35]. A good selection of the FFT algorithm will reduce the number of rotations and, therefore, the area of the FFT architecture.

A fourth challenge in the design of very high-throughput FFTs is pipelining. High throughput demands deep pipelining. However, with the high parallelization of fully parallel FFTs, pipelining increases the area of the FFT. In order to minimize

M. Garrido is with the Department of Electrical Engineering, Linköping University, SE-581 83 Linköping, Sweden, e-mail: mario.garrido.galvez@liu.se

K. Möller and M. Kumm are with the Digital Technology Group, University of Kassel, 34121 Kassel, Germany, e-mails: konrad.moeller@uni-kassel.de, kumm@uni-kassel.de

This work was supported by the Swedish ELLIIT Program.

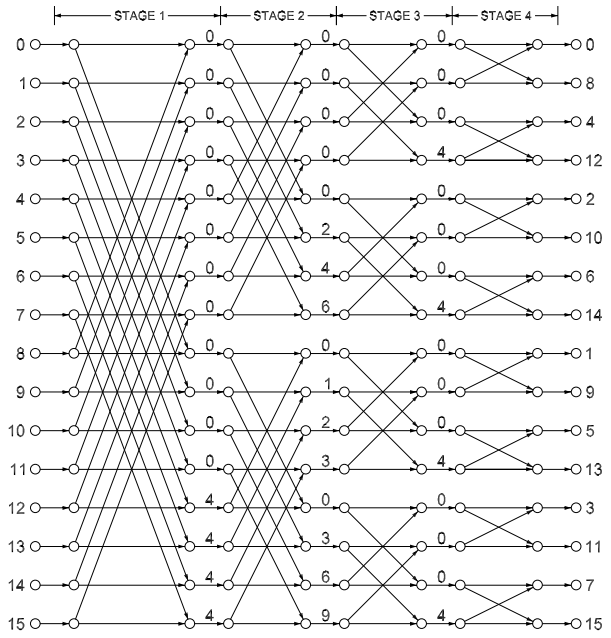

Fig. 1. Flow graph of a 16-point radix-2^2 DIF FFT.

the amount of pipelining, we have taken into account the depth of the rotators in the FFT. As a result, our rotators only require three clock cycles to calculate the rotations, achieving a critical path of only one adder. This not only increases the clock frequency, but also keeps the pipelining at reasonable levels.

A final challenge is the generation of the FFT architectures automatically. This challenge is a consequence of the implementation of the rotators as shift-and-add. Due to the large number, complexity and variety of rotators, it is unfeasible to implement them by hand or using the *generate* command in VHDL. As a result, it is needed to create a tool that generates the architectures automatically.

In this paper, we address the previous design challenges in order to achieve very high-throughput FFT architectures. This results in fully parallel architectures with throughput over 100 GS/s, which is the highest throughput for FFTs reported so far. Furthermore, the architectures have high accuracy and a relatively small area cost for the provided throughput.

The paper is organized as follows. In Section II, we review the FFT algorithm and discuss different FFT radices. In Section III, we review key concepts related to rotations in fixed-point arithmetic. In Section IV, we explain the proposed fully parallel FFT architectures. In Section V, we provide experimental results and compare them to previous works. Finally, in Section VI we provide the main conclusions of the paper.

II. The FFT algorithm

The N-point DFT of an input sequence $x[n]$ is defined as

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

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

TABLE I
Number of non-trivial rotations in FFT algorithms

N	Radix-2	Radix-2^2	Radix-2^3	Radix-2^4	Split Radix
8	2	2	2	2	2
16	10	8	10	8	8
32	34	28	28	30	26
64	98	76	80	76	72
128	258	204	216	200	186
256	642	492	504	480	456
512	1538	1196	1208	1200	1082
1024	3586	2732	2872	2672	2504

To calculate the DFT, the FFT based on the Cooley-Tukey algorithmm [36] is mostly used. This reduces the number of operations from $O(N^2)$ for the DFT to $O(N \cdot \log_2 N)$ for the FFT.

Figure 1 shows the flow graph of a 16-point radix-2^2 FFT according to the Cooley-Tukey algorithm, decomposed using decimation in frequency (DIF) [35]. The FFT consists of $n = \log_2 N$ stages. At each stage $s \in \{1, \ldots, n\}$ of the graph, 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:

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

As a consequence, samples for which $\phi \in \{0, \frac{N}{4}, \frac{N}{2}, \frac{3N}{4}\}$ must be rotated by $0°$, $270°$, $180°$ or $90°$, which corresponds to complex multiplications by 1, $-j$, -1 and j, respectively. These rotations are considered trivial, because they can be carried out by interchanging the real and imaginary components and/or changing the sign of the data.

Different radices only differ in the rotations at the FFT stages [35], whereas the butterflies are the same. The most common algorithms are radix-2, radix-2^2, radix-2^3 and radix-2^4 [35], [37] in their decimation in time (DIT) and DIF versions. For a fully parallel FFT we also consider the split radix algorithm [38]–[40]. The advantage of split radix is its smaller number of non-trivial rotations, whereas the advantage of radices of 2^k, $k \geq 2$ is that some of their stages only include trivial rotations.

Table I shows the number of non-trivial rotations for split radix and radix-2^k algorithms for different FFT sizes. These numbers are equal to the number of rotators in a fully parallel FFT.

III. Rotations in fixed-point arithmetic

A rotation in a digital system can be described as [33]

$$\begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} C & -S \\ S & C \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}, \qquad (3)$$

where $X + jY$ is the result of the rotation and C and S are obtained from the rotation angle as

$$\begin{aligned} C &= R \cdot (\cos\alpha + \varepsilon_c), \\ S &= R \cdot (\sin\alpha + \varepsilon_s), \end{aligned} \qquad (4)$$

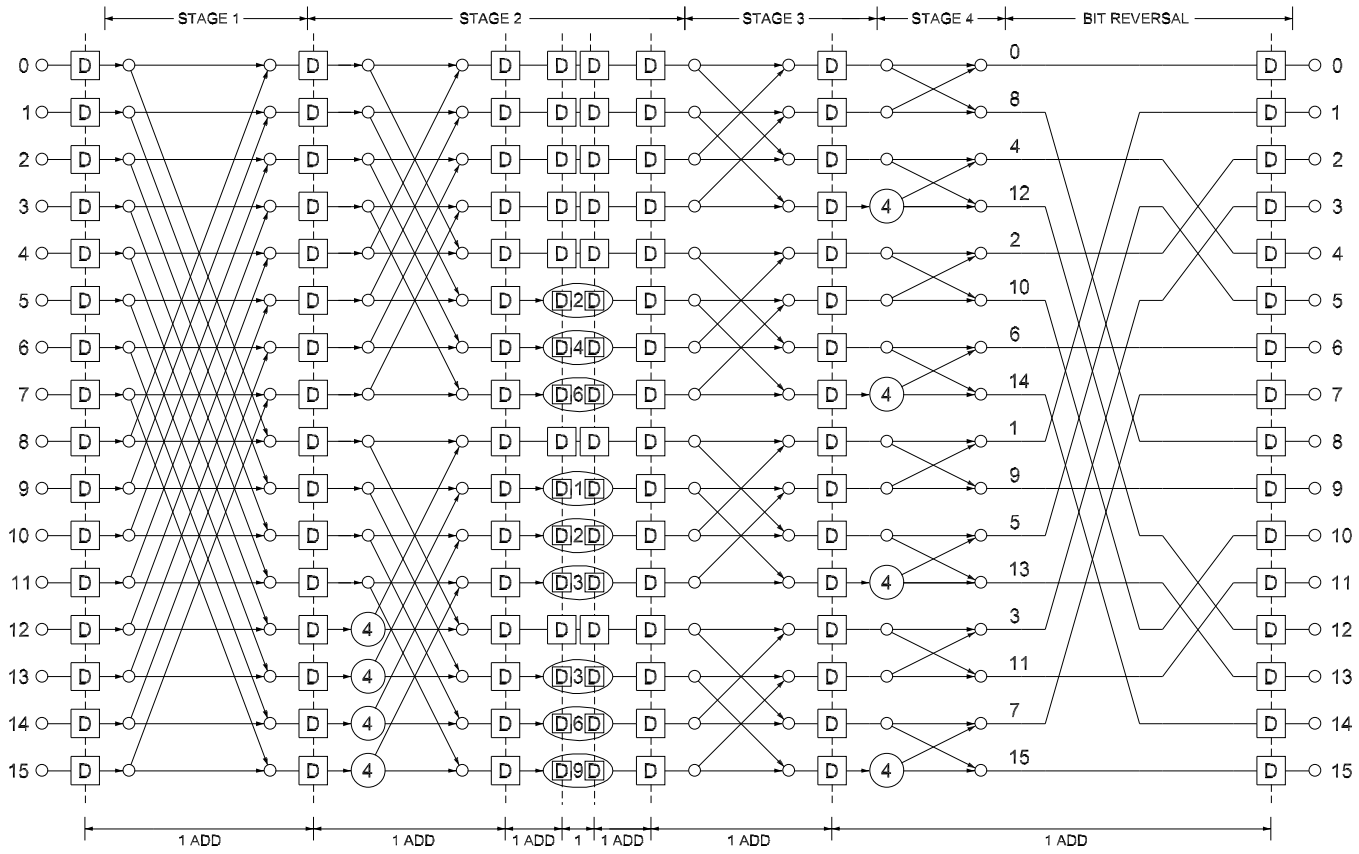

Fig. 2. Hardware architecture of a 16-point radix-2^2 fully parallel FFT.

where α is the rotation angle, ε_c and ε_s are the relative quantization errors of the cosine and sine components, respectively, and R is the scaling factor. The output $X + jY$ is also scaled by R. The rotation error for a single constant rotation (SCR) is calculated as [33]

$$\varepsilon = \sqrt{\varepsilon_c^2 + \varepsilon_s^2}. \tag{5}$$

The effective word length (WL_E) is a measure of the rotator accuracy that indicates the number of output bits that are guaranteed to be accurate. It is obtained from the rotation error as [34]

$$WL_E = -\log_2 \frac{\varepsilon}{2\sqrt{2}} = -\log_2 \varepsilon + \frac{3}{2}. \tag{6}$$

IV. Proposed fully parallel FFT architectures

Fig. 2 shows the proposed 16-point radix-2^2 fully parallel FFT architecture. This architecture is a direct implementation of the FFT flow graph in Fig. 1 in the sense that each addition in the flow graph is translated into an adder and each rotation into a rotator. The architecture consist of butterflies, delays (D) and rotators. Contrary to other FFT architectures, fully parallel FFTs do not include circuits for data management. However, it involves other design challenges related to the design of the rotators, the selection of the FFT algorithms and the implementation in VHDL. Next section present these challenges, as well as the proposed solutions.

A. Design of the rotators

In fully parallel FFTs, rotators take the largest part of the area. As they consist of constant multiplications, the best way to reduce the area of the FFT is to implement them as shift-and-add. Low-depth shift-and-add implementations also reduce the number of adders in series in the rotators. This reduces the amount of pipelining required for high-throughput. Furthermore, the accuracy of the rotators has effect in the accuracy of the entire FFT. In order to achieve accurate FFTs it is necessary to select coefficients with small rotator error. To achieve these goals, we take into account the coefficient selection, we explore different architectures for the rotators and we make use of advanced shift-and-add algorithms. The details are explained next.

1) Coefficient selection: The rotation coefficients $C + jS$ are selected by taking into account that a fully parallel FFT has N parallel edges and, therefore, it calculates N rotations in parallel. Some of them are by 0° or other trivial rotations, and other ones are by non-trivial rotations. As the number of trivial rotations, specially those by 0°, is large in FFT algorithms, it is convenient that the rotators have unity scaling [34], which corresponds to scaling by a power-of-two. This scaling can be compensated by a hard-wired bit shift. By doing this, rotations by 0° do not require any hardware.

Fig. 3 shows the coefficient selection for SCR and unity scaling used in the proposed FFTs. For each angle $\alpha = -2\pi\phi/N$, it searches for the coefficients $C + jS$ that approx-

TABLE II
ADDER COST OF THE ALTERNATIVES TO CALCULATE CONSTANT ROTATIONS IN HARDWARE

Structure in Fig. 4	Angle and Coefficient			
	General Case $P = C + jS$	$\alpha = m \cdot \pi/2 + \pi/4$ $P = C + jC$	$\alpha = m \cdot \pi$ $P = C$	$\alpha = m \cdot \pi + \pi/2$ $P = S$
SCM I rotator (a)	$SCM(S) + SCM(C+S) + SCM(\lvert C-S \rvert) + 3$	$SCM(S) + SCM(C+S) + 2$	$2 \cdot SCM(C)$	$2 \cdot SCM(S)$
SCM II rotator (b)	$SCM(C) + SCM(C+S) + SCM(\lvert C-S \rvert) + 3$	$SCM(C) + SCM(C+S) + 2$		
MCM rotator (c)	$2 \cdot MCM(C,S) + 2$	$2 \cdot SCM(C) + 2$		
CMM rotator (d)	$CMM([C,-S],[C,S])$	$2 \cdot SCM(C) + 2$		

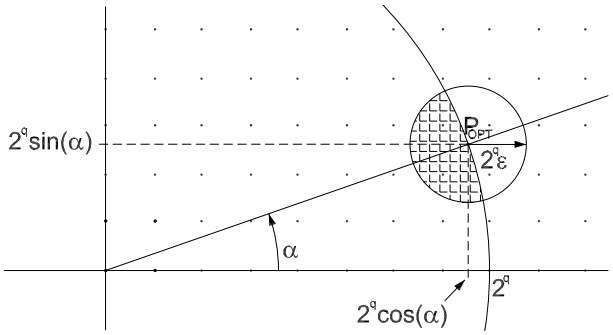

Fig. 3. Coefficient selection for SCR and unity scaling.

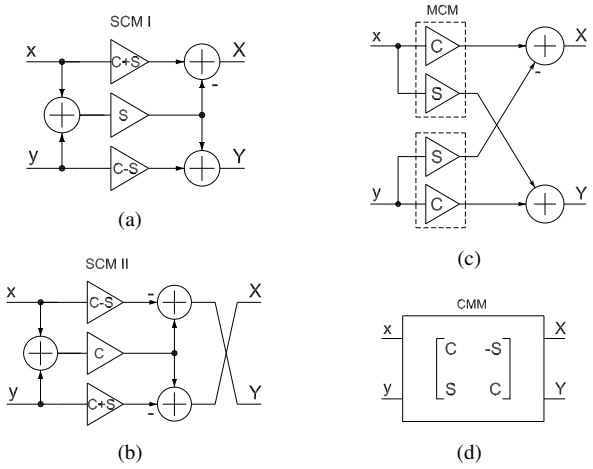

Fig. 4. Architectures of the constant rotators used in the fully parallel FFT. (a) Complex constant multiplication using 3 real multipliers and 3 adders, version I. (b) Complex contant multiplication using 3 real multipliers and 3 adders, version II. (c) Complex constant multiplication using 4 real multiplipliers and 2 adders. (d) General case of a complex constant multiplication.

imate the complex number $P_{OPT}(\alpha) = 2^q \cdot (\cos\alpha + j\sin\alpha)$ with an accuracy WL_E or higher, for $q = 1, \ldots, WL_C - 1$, where WL_C represents the maximum word length of the rotation coefficients. This corresponds to a circle of radius $2^q\varepsilon$ around $P_{OPT}(\alpha)$. Note that the maximum error ε that defines the circle is calculated from WL_E according to (6). Additionally, we include the condition that the the magnitude of the coefficients must be smaller than or equal to 2^q. This condition guarantees that the result of the rotators will not overflow. As a consequence, the selected coefficients are the complex numbers $C + jS$ with $C, S \in \mathbb{Z}$ that lie in the shaded region of Fig. 3.

The reason to do this selection is to have a set of coefficients with an accuracy WL_E or higher. It would be easier to obtain a single coefficient by rounding the real and imaginary part of P_{OPT}. However, to have a set of coefficients allows us to explore which of them can be calculated by using the smallest number of adders. This enables further optimization while meeting the accuracy requirements.

TABLE III
SHIFT-AND-ADD ALGORITHMS USED TO CALCULATE THE FFT ROTATORS

Shift-and-add Algorithm	Rotator Configuration	Reference
RPAG-CMM	CMM	[31]
HCUB	MCM	[26]
DiffAG	MCM	[25]
MAG2	MCM	[24]
Optimal SCM	SCM I	[22]
Optimal SCM	SCM II	[22]

2) Exploration of rotator implementations: Equation (3) defined a rotation as a complex constant multiplication, which can be calculated in different ways. The most straightforward approach is to do the calculations according to (3), which requires 4 multipliers and 2 adders. Other alternatives with 3 adders and 3 multipliers [41] are also possible. The 3-multiplier cases are shown in Figs. 4(a) and 4(b), whereas the 4-multiplier one is shown in Fig. 4(c).

Table II shows how to calculate the number of adders for the different constant rotators in Fig. 4. The most general case is a rotation by $P = C + jS$. However, when $C = S$, $C = 0$ or $S = 0$, the rotators can be simplified, leading to the costs in columns three to five in the table.

3) Shift-and-add implementation: The constant multiplications in the rotator architectures in Fig. 4 are implemented as shift-and-add. For the real constant multiplications in Figs. 4(a) and 4(b) we make use of the SCM algorithms in [22], [23]. For the constant multiplications in Fig. 4(c) we evaluate different MCM techniques [24]–[28]. Finally, we address the calculation in Fig. 4(d) as a CMM problem [31]. The exact algorithms that we consider for the implementation of the rotators are shown in Table III.

4) Selection of the best rotators: For each rotation angle in the FFT, the previous steps provide a number of coefficients with certain WL_E, implemented according to the multiple architectures in Fig. 4, which are evaluated according to a number of shift-and-add algorithms. Among all these cases, Table IV shows the best coefficients that we obtain for each rotation angle. The coefficients in the table are selected as those that require the smallest number of adders for an accuracy of at least WL_E bits using coefficients of up to WL_C bits. Different cases for WL_E and WL_C are presented in the table.

TABLE IV
BEST ROTATORS FOR $N = 256$ FOR DIFFERENT WL_E AND WL_C

	WL_E / WL_C											
	8/12			12/16			16/20			20/20		
Angle (ϕ)	Coeff. ($C+jS$)	WL_E (bits)	Add. Rot.	Coeff. ($C+jS$)	WL_E (bits)	Add. Rot.	Coeff. ($C+jS$)	WL_E (bits)	Add. Rot.	Coeff. ($C+jS$)	WL_E (bits)	Add. Rot.
0	$1-j0$	Inf	0	$1-j0$	Inf	0	$1-j0$	Inf	0	$1-j0$	Inf	0
1	$1023-j32$	8.71	3	$16379-j400$	14.44	8	$16379-j402$	18.74	10	$524130-j12867$	22.05	14
2	$511-j24$	10.25	5	$4091-j200$	13.52	8	$4091-j201$	17.36	10	$261828-j12863$	21.18	14
3	$255-j18$	9.67	5	$16336-j1213$	12.41	8	$16339-j1205$	16.08	11	$522867-j38569$	21.75	15
4	$127-j12$	9.08	5	$1019-j100$	12.91	8	$130440-j12847$	18.64	12	$521763-j51389$	21.62	15
5	$63-j8$	8.38	3	$4065-j500$	13.01	6	$520345-j64176$	19.21	12	$520345-j64178$	21.62	14
6	$63-j9$	8.51	5	$4051-j602$	13.24	10	$259305-j38460$	17.23	12	$518613-j76928$	20.37	14
7	$63-j11$	11.11	7	$8071-j1400$	15.12	10	$129141-j22408$	18.07	12	$516569-j89632$	20.13	14
8	$1000-j191$	8.21	5	$251-j50$	12.84	7	$16069-j3197$	16.09	11	$514213-j102284$	20.40	16
9	$249-j56$	9.85	6	$999-j224$	12.90	8	$63943-j14359$	18.20	12	$127887-j28718$	20.60	14
10	$124-j31$	10.87	4	$3972-j993$	12.14	6	$63570-j15925$	16.34	11	$508575-j127392$	20.73	12
11	$123-j35$	8.60	6	$31580-j8737$	15.00	10	$31581-j8739$	17.14	12	$505295-j139835$	20.00	14
12	$61-j18$	8.17	6	$31354-j9509$	14.40	10	$62713-j19024$	17.43	12	$31357-j9512$	20.75	13
13	$485-j160$	10.11	7	$31104-j10287$	12.79	9	$497811-j164448$	16.24	12	$248913-j82230$	21.54	14
14	$30-j11$	8.47	6	$3855-j1380$	12.85	9	$30852-j11039$	17.35	12	$123410-j44157$	20.94	15
15	$1912-j733$	10.41	8	$15284-j5893$	13.45	10	$244577-j94336$	16.42	12	$122289-j47172$	20.32	14
16	$59-j24$	8.48	7	$30255-j12544$	12.24	9	$484368-j200639$	17.01	12	$242189-j100318$	20.49	14
17	$29-j13$	8.46	7	$14976-j6639$	14.20	10	$239651-j106228$	17.16	12	$479309-j212462$	20.27	16
18	$231-j110$	10.04	8	$1851-j875$	12.95	10	$473949-j224160$	19.09	14	$236975-j112081$	20.94	15
19	$114-j57$	9.14	6	$29265-j14728$	13.82	10	$234149-j117856$	16.47	12	$234153-j117863$	20.82	15
20	$113-j60$	9.98	5	$14449-j7724$	15.91	10	$115593-j61788$	17.21	12	$115595-j61787$	20.80	14
21	$889-j508$	9.57	6	$1782-j1009$	13.62	10	$28511-j16151$	18.30	12	$114044-j64605$	21.17	14
22	$219-j132$	10.02	8	$28096-j16839$	12.88	10	$56211-j33693$	17.08	12	$449697-j269537$	20.62	15
23	$27-j17$	9.50	7	$3460-j2191$	14.17	10	$221472-j140247$	19.00	12	$442946-j280493$	21.21	16
24	$1701-j1134$	10.42	7	$27235-j18200$	12.95	10	$435929-j291264$	16.61	12	$435929-j291278$	20.54	14
25	$1673-j1176$	10.67	8	$26784-j18855$	12.63	9	$214323-j150946$	18.26	14	$428649-j301890$	20.32	15
26	$409-j306$	9.20	7	$6577-j4880$	12.98	10	$26319-j19520$	17.44	12	$210556-j156159$	24.33	16
27	$403-j310$	8.17	7	$3229-j2520$	17.38	10	$3229-j2520$	17.38	10	$206660-j161279$	20.85	15
28	$99-j80$	8.23	7	$12664-j10395$	14.93	10	$202633-j166295$	16.15	13	$101320-j83151$	20.83	14
29	$24-j21$	8.50	6	$3101-j2674$	12.92	10	$198494-j171217$	16.31	12	$198498-j171225$	20.75	16
30	$47-j43$	8.75	7	$1517-j1375$	13.27	11	$48557-j44013$	16.11	13	$388471-j352091$	20.43	15
31	$23-j22$	8.91	7	$2967-j2823$	12.97	10	$47463-j45191$	16.66	13	$379713-j361518$	20.41	16
32	$45-j45$	8.97	6	$181-j181$	14.69	8	$92681-j92681$	18.15	10	$370728-j370727$	20.97	12
Avg:			**5.97**			**8.91**			**11.52**			**14.06**

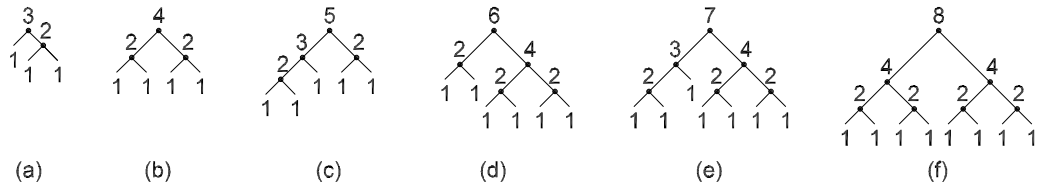

Fig. 5. Binary trees of the implemented fully parallel FFTs.

TABLE V
AVERAGE ADDERS PER ROTATOR FOR DIFFERENT ALGORITHMS AS A FUNCTION OF WL_E AND WL_C

	WL_E/WL_C			
Algorithm	8/12	12/16	16/20	20/20
RPAG-CMM	6.03	9.06	12.36	16.18
HCUB	6.67	9.21	11.82	14.67
DiffAG	6.67	9.39	12.24	15.94
MAG2	6.67	9.15	-	-
SCM	7.52	10.42	12.48	14.45
CSD	7.94	12.73	17.70	22.12
BIN	12.42	19.82	27.03	34.91
Selected Coeff.	**5.97**	**8.91**	**11.52**	**14.06**

The angles $\phi = 0, \ldots, 32$ in the first column of Table IV correspond to $\alpha = -2\pi\phi/256$, which are the angles in $[-45°, 0°]$ of a 256-point FFT. This serves to design any FFT of size up to 256 points. Note that the rotations for N smaller than 256 are the angles $\phi = [0, \ldots, N/8] \cdot 256/N$ of a 256-point FFT. For instance, a 32-point FFT uses the angles $\phi = \{0, 8, 16, 24, 32\}$ in the table. Appendix A explains how to obtain all the rotation angles from those in $[-45°, 0°]$.

The last row in Table IV shows the average number of adders among all the angles in the column. These numbers are compared in Table V to the average costs of the individual algorithms. It can be observed that there is no algorithm that perform best for all combinations of WL_E and WL_C, which justifies the selection of the best case for each angle. Furthermore, we can get an estimate of the savings of the pro-

TABLE VI
ADDER COST OF THE PROPOSED FULLY PARALLEL FFT ARCHITECTURES

FFT Algorithm	WL_E / WL_C 8/12	12/16	16/20	20/20
		$N = 8$		
Radix-2 (a)	60 (6.0)	64 (8.0)	68 (10.0)	72 (12.0)
Split Radix	60 (6.0)	64 (8.0)	68 (10.0)	72 (12.0)
		$N = 16$		
Radix-2	192 (6.4)	212 (8.4)	236 (10.8)	256 (12.8)
Radix-2^2 (b)	180 (6.5)	196 (8.5)	216 (11.0)	232 (13.0)
Split Radix	180 (6.5)	196 (8.5)	216 (11.0)	232 (13.0)
		$N = 32$		
Radix-2	536 (6.4)	608 (8.5)	696 (11.1)	776 (13.4)
Radix-2^3,2^2 (c)	496 (6.3)	556 (8.4)	628 (11.0)	696 (13.4)
Split Radix	484 (6.3)	540 (8.5)	608 (11.1)	672 (13.5)
		$N = 64$		
Radix-2	1380 (6.2)	1616 (8.7)	1876 (11.3)	2104 (13.6)
Radix-2^2 (d)	1244 (6.3)	1432 (8.7)	1636 (11.4)	1808 (13.7)
Radix-2^3	1256 (6.1)	1452 (8.6)	1652 (11.1)	1832 (13.3)
Split Radix	1216 (6.2)	1396 (8.7)	1588 (11.4)	1752 (13.7)
		$N = 128$		
Radix-2	3396 (6.2)	4056 (8.8)	4748 (11.5)	5352 (13.8)
Radix-2^2,2	3056 (6.2)	3576 (8.7)	4120 (11.4)	4608 (13.8)
Radix-2^3,2^4 (e)	3036 (6.2)	3536 (8.7)	4052 (11.3)	4496 (13.5)
Split Radix	2948 (6.2)	3428 (8.8)	3928 (11.5)	4368 (13.8)
		$N = 256$		
Radix-2	8080 (6.2)	9784 (8.9)	11508 (11.5)	13048 (13.9)
Radix-2^2	7156 (6.2)	8480 (8.9)	9808 (11.6)	10976 (14.0)
Radix-2^4 (f)	7137 (6.3)	8329 (8.8)	9571 (11.4)	10655 (13.7)
Split Radix	6924 (6.2)	8148 (8.9)	9372 (11.6)	10472 (14.0)

posed approach in terms of adders with respect to conventional approaches. With respect to canonical-signed-digit (CSD), the proposed approach saves 25% 30%, 35% and 36% of the adders for WL_E 8, 12, 16 and 20, respectively. Likewise, with respect to the sum of the bits that are equal to '1' in the binary representation (BIN), the proposed approach saves 52%, 55%, 57% and 60% of the adders for the same cases. These saving have direct impact on the area of the FFT.

B. Exploration of FFT algorithms

As FFT algorithms differ in the rotations throughout the FFT stages, to design the fully parallel FFT we have explored all the FFT algorithms that can be represented by the binary tree representation [37]. Furthermore, we have considered the split radix algorithm [38]–[40], as it leads to the smallest number of non-trivial rotations.

The exploration consist of obtaining the ϕ values required at the stages of each FFT algorithm, and then sum the adder cost of the rotators that each algorithm includes. The ϕ rotations are calculated according to [37], and the adder costs are taken from Table IV.

The exploration of the FFT algorithms leads to the following conclusions:

- The number of adders is approximately proportional to the number of non-trivial rotations (NTR), according to

$$\text{Adders}_{\text{ROT}} \approx 0.7 \cdot WL_E \cdot \text{NTR}. \quad (7)$$

  Thus, the FFT algorithms with the smallest number of non-trivial rotations (refer to Table I) also lead to the smallest number of adders. These algorithms are the split radix algorithm and the algorithms based on radix-2^4 and radix-2^2.
- Among all the binary tree FFT algorithms, radix-2 is the algorithm with the highest cost in terms of non-trivial rotations and in terms of adders.
- The DIF and DIT decompositions of the algorithms require the same number of adders. The reason is that DIF and DIT use the same rotations: The rotations at stage $s = 1, \ldots, n-1$ of the DIF decomposition appear at stage $n - s$ of the DIT version.

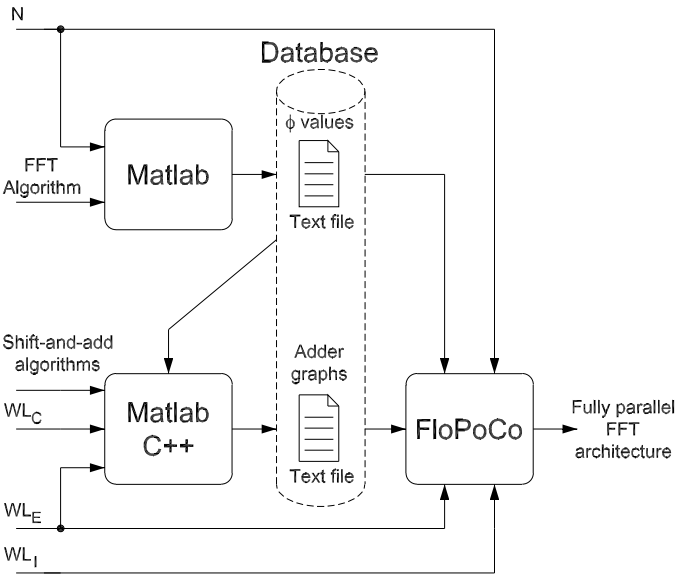

Fig. 6. Complete tool flow for the proposed fully parallel FFT architectures.

Based on these observations, we have collected in Table VI the results of the best FFT algorithms together with the maximum cost set by radix-2. The table shows the adder cost of fully parallel FFT architectures for various configurations of WL_E and WL_C. The numbers in parenthesis are the average number of adders per (non-trivial) rotator. The binary tree algorithms with the smallest number of adders are marked with the letter (a) to (f) in the table and their corresponding binary trees are shown in Fig. 5.

By considering that the total number of real adders in the butterflies of the FFT is

$$\text{Adders}_{\text{BUTT}} = 2N \cdot \log_2(N), \quad (8)$$

the total number of adders in fully pipelined FFTs is obtained as the sum of equations (7) and (8), i.e.,

$$\text{Adders}_{\text{FFT}} \approx 0.7 \cdot WL_E \cdot \text{NTR} + 2N \cdot \log_2(N). \quad (9)$$

This equation may be useful for other FFT algorithms and/or to estimate other values of WL_E that are not collected in Table VI.

C. Implementation

To implement the proposed architectures we have developed tools that automate this task. The reason for this is that it is not feasible to implement all the rotators by hand. Fig. 6

shows the block diagram of the complete tool flow used to implement the architectures. First, Matlab is used to calculate the ϕ values for all the rotations of a given FFT algorithm. These values are stored in a text file. Second, Matlab together with algorithms implemented in C++ were used to obtain the adder graphs of the best rotators in Table IV. An adder graph is a directed acyclic graph in which each vertex, except the input, represents an adder/subtractor computing a certain multiple of the input. The adder graphs are stored in a text file containing standardized strings for each required rotator. The text files for the ϕ values and the adder graphs only have to be generated once and serve as a database for the code generator.

For the implementation of the fully parallel FFT, we have developed a VHDL code generator with the help of the FloPoCo library [42]. FloPoCo provides an environment to easily develop VHDL code generators for arithmetic circuits in C++. We made our implementation available as open source in the "uni_ks" branch of the FloPoCo Git repository [43].

The configuration parameters that have to be provided to the code generator are the FFT size, N, the input word length WL_I, and the effective word length of the rotators, WL_E. The rotation matrix with the ϕ values of the FFT algorithm, and the adder graph are then looked-up in the text files.

The output word length WL_O has been selected to be the same as the input word length WL_I. Therefore, the butterfly additions/subtractions are done by using $WL_I + 1$ bits. The result of each butterfly is provided to the rotator after it. The rotation is calculated without truncating any bit. After each rotator, the data word length is reduced to the WL_I most significant bits by truncation. These WL_I bits are provided to the butterfly of the next stage.

The FFT output is generated in bit-reversed order. However, as the architecture is fully parallel, all the outputs are provided in parallel. Therefore, the bit reversal is hard wired during code generation and does not have any hardware cost.

A key to high-performance circuits on FPGAs is pipelining. Therefore, a register is placed after each butterfly adder. Moreover, the multiplierless rotators are fully pipelined, which means that each adder is followed by a register. To get a valid pipeline in stages with different rotator depths, balancing registers are placed after rotators with shorter pipeline depth. This is done by using the pipeline framework of FloPoCo.

As an example, the VHDL for the 16-point radix-2^2 fully parallel FFT architecture in Fig. 2, which corresponds to the case marked with (b) in Tables VI and VII, is obtained by the following FloPoCo call

```
flopoco FullyParallelFFT wIn=16 wE=16 N=16 alg=BT
```

where `wIn` and `wE` stand for WL_I and WL_E, respectively, `N` specifies the FFT size and `alg=BT` indicates that the selected FFT algorithm is the best binary tree algorithm. Alternatively, the split radix algorithm is selected by setting `alg=SR`.

V. Experimental results

Table VII shows the experimental results for the proposed FFT architectures (non-shaded rows) and compares them to previous implementations (shaded rows) of fully parallel FFTs. The results are obtained for $WL_E = 16$ and for FFT sizes $N = 8, 16, 32, 64, 128$ and 256. The experimental results include both the split radix algorithm and the best binary tree algorithms marked with the letters (a) to (f) in Table VI. Furthermore, for easier comparison to previous works, we show the results on different FPGA devices, which are provided in the second column of the table and are detailed at the bottom of Table VII.

For all designs and FPGA devices, the experimental results include area, performance in terms of f_{CLK}, latency and throughput, dynamic power consumption and signal-to-quantization-noise ration (SQNR) [44]. All power results are obtained from a worst-case 50% input activity using XPower (Virtex 5 and 6) and Vivado's power estimator (Ultrascale). To evaluate the accuracy, the SQNR was determined by running intensive simulations of the generated VHDL.

For the same N and the same FPGA, the results for split radix and for the best binary tree algorithm are very similar in terms of the number of slices. The main difference between them is in the smaller latency of the latter. The reason is that the binary tree algorithm only has trivial rotators in odd stages, which reduces the number of pipelining registers in those stages. Conversely, the split radix algorithm has non-trivial rotators in all the stages, which leads to extra pipelining registers, many of them forming shift registers. Based on this, it could be expected that the split radix FFT requires a larger number of FF. However, in the FPGAs used here, shift registers are implemented by using LUTs [45]. This also explains why some split radix FFTs use more LUTs than the corresponding binary tree FFT, even though the number of adders is smaller in split radix.

Compared to previous works for $N = 16$ to 64 and for the same FPGA, the main advantage of the proposed FFTs is the reduction of hardware resources. Whereas previous approaches require a large number of DSPs, the proposed designs do not use any DSP. The cost is an increase in the number of slices that is small compared to the amount of DPSs that are saved.

Compared to previous 256-point fully parallel FFTs, the proposed architectures achieve more than double throughput than previous approaches, reaching 138.5 GS/s. For a word length of 16 bits both for the real and imaginary parts of the data, this corresponds to 4.4 Tbit/s. To the best of the authors' knowledge, this is the highest throughput for FFT architectures reported so far.

Regarding area, the proposed 256-point fully parallel FFTs are more efficient than previous approaches in terms of LUTs. Note that the word length of the proposed architectures is 33% larger than the word length in [15], whereas the number of LUTs is only 2% larger. With respect to FFs, the proposed architectures use a large number of them due to the deep pipelining that they have, which allows for the extremely high values of throughput.

The power consumption of the proposed designs grows proportional to the size of the architectures. Although values around 30 W for $N = 256$ may seem large, they are not when normalizing by the number of bits and frequency. By doing

TABLE VII
COMPARISON OF FULLY PARALLEL FFT HARDWARE ARCHITECTURES.

Fully parallel FFT	FPGA (⋆)	WL_I (bits)	WL_E (bits)	Area					f_{CLK} (MHz)	Latency		Th. (GS/s)	Dyn. Pow. (W)	SQNR (dB)
				LUTs	FFs	Slices	BRAM	DSP		(cyc.)	(ns)			
N = 8														
Radix-2 (a)	V5	16	16	1566	1574	499	0	0	451	7	16	3.61	0.12	80.2
Radix-2 (a)	V6	16	16	1568	1564	443	0	0	469	7	15	3.75	0.62	80.2
Radix-2 (a)	VU	16	16	1380	1636	272	0	0	742	7	9	5.93	0.33	80.2
Split Radix	V5	16	16	1556	1532	528	0	0	425	7	16	3.40	0.76	79.4
Split Radix	V6	16	16	1507	1564	429	0	0	488	7	14	3.90	0.09	79.4
Split Radix	VU	16	16	1380	1636	275	0	0	847	7	8	6.77	0.31	79.4
N = 16														
Garrido [4]	V6	16	–	–	–	1612	0	48	322	9	28	5.15	–	–
Dillon [16]	V5	16	–	–	–	1283	0	20	400	–	–	6.40	–	–
Radix-2^2 (b)	V5	16	16	4730	5377	1776	0	0	414	8	19	6.62	0.25	75.9
Radix-2^2 (b)	V6	16	16	4791	5377	1640	0	0	450	8	18	7.20	0.16	75.9
Radix-2^2 (b)	VU	16	16	4376	5586	801	0	0	636	8	13	10.17	0.60	75.9
Split Radix	V5	16	16	5216	5335	1732	0	0	411	12	29	6.58	1.56	75.8
Split Radix	V6	16	16	5272	5367	1622	0	0	468	12	26	7.48	0.16	75.8
Split Radix	VU	16	16	4632	5586	810	0	0	675	12	18	10.80	0.75	75.8
N = 32														
Dillon [16]	V5	16	–	–	–	5068	0	80	400	–	–	12.80	–	–
Radix-2^3,2^2 (c)	V5	16	16	15270	17191	5297	0	0	368	15	41	11.78	3.28	72.7
Radix-2^3,2^2 (c)	V6	16	16	15639	17095	4680	0	0	417	15	36	13.34	2.40	72.7
Radix-2^3,2^2 (c)	VU	16	16	13317	19412	2543	0	0	655	15	23	20.96	1.91	72.7
Split Radix	V5	16	16	15592	16452	5139	0	0	383	19	50	12.26	3.68	72.3
Split Radix	V6	16	16	15709	16356	4340	0	0	415	19	46	13.28	2.35	72.3
Split Radix	VU	16	16	13418	18520	2472	0	0	596	19	32	19.07	1.74	72.3
N = 64														
Garrido [4]	V6	16	–	–	–	6590	0	384	414	16	39	26.49	–	–
Dillon [16]	V5	16	–	–	–	9875	0	208	400	–	–	25.60	–	–
Radix-2^2 (d)	V5	16	16	38847	48622	14777	0	0	302	16	53	19.32	9.50	69.4
Radix-2^2 (d)	V6	16	16	40513	48542	12178	0	0	409	16	39	26.17	5.55	69.4
Radix-2^2 (d)	VU	16	16	35057	51414	6594	0	0	579	16	28	37.06	4.65	69.4
Split Radix	V5	16	16	41897	47723	14918	0	0	318	27	85	20.35	8.70	68.7
Split Radix	V6	16	16	43299	47636	12256	0	0	386	27	70	24.69	5.04	68.7
Split Radix	VU	16	16	36044	50565	6420	0	0	590	27	46	37.76	5.20	68.7
N = 128														
Radix-2^3,2^4 (e)	V5	16	16	113954	97730	35728	0	0	221	21	95	28.28	16.01	65.9
Radix-2^3,2^4 (e)	V6	16	16	101977	119844	32488	0	0	312	21	67	39.94	12.91	65.9
Radix-2^3,2^4 (e)	VU	16	16	88876	127243	16630	0	0	538	21	39	68.86	10.57	65.9
Split Radix	V5	16	16	105565	119896	36222	0	0	255	35	137	32.64	21.00	65.6
Split Radix	V6	16	16	108298	119639	32413	0	0	273	35	128	34.94	15.11	65.6
Split Radix	VU	16	16	90292	127251	15817	0	0	531	35	66	67.97	11.44	65.6
N = 256														
Polat [13]	V6	5	$\approx$ 10	213700	123406	–	0	0	312 (†)	9	54	43.00	–	–
Kim [15]	VU	11-13	$\approx$ 12	205530	131883	–	0	0	182	19	105	46.59	–	45
Radix-2^4 (f)	VU	16	16	210215	303484	40895	0	0	541	25	46	138.5	29.93	62.7
Split Radix	VU	16	16	218403	312146	39988	0	0	541	43	79	138.5	33.78	62.5

(†): The multipliers of the architecture run at 312 MHz, but other parts of the circuit run at lower frequency.
(⋆): V5 stands for Virtex 5 XC5VLX330T-2FF1738 using Xilinx ISE 14.7; V6 stands for Virtex 6 XC6VLX760-FF1760 using Xilinx ISE 14.7; VU stands for Virtex UltraScale XCVU190-FLGA2577-2-E using Xilinx Vivado 2018.1.

this, we obtain:

$$P_{\text{NORM}} = \frac{29.93}{256 \cdot 32 \cdot 541} = 6.75 \ \mu\text{W/bit/Hz}. \qquad (10)$$

Finally, the SQNR of the proposed architectures is high and approximates to the following equation:

$$\text{SQNR} \approx 90 - 3.5 \cdot \log_2 N \ \text{(dB)}. \qquad (11)$$

Note also that the SQNR of the proposed 256-point FFTs is 17.5 dB larger than that in [15]. This improvement comes from the use of 16 bits for data and for WL_E.

VI. CONCLUSIONS

In this paper, we have presented the fastest FFT architectures so far. The architectures correspond to a fully parallel implementation of the FFT. The design of these architectures involves the design of hardware-efficient and accurate shift-and-add rotators, the selection of rotator-efficient FFT algorithms, a proper pipelining of the architecture and the design of a tool that generates the architectures automatically. Experimental results for the implemented FFTs lead to throughput rates up to 138.5 GS/s, which breaks the barrier of 100 GS/s.

TABLE VIII
MAPPING OF THE FULL RANGE ANGLE $\alpha \in [-180°, 180°]$ TO THE REDUCED RANGE $\alpha' \in [-45°, 0°]$ AND ITS MODIFICATION ON THE ROTATOR CONSTANT

α	α'	ϕ	ϕ'	Rot. Const.
$[-180°, -135°]$	$-180° - \alpha$	$[3/8N, N/2]$	$N/2 - \phi$	$-C+jS$
$[-135°, -90°]$	$\alpha + 90°$	$[N/4, 3/8N]$	$\phi - N/4$	$-S+jC$
$[-90°, -45°]$	$-90° - \alpha$	$[N/8, N/4]$	$N/4 - \phi$	$S+jC$
$[-45°, 0°]$	α	$[0, N/8]$	ϕ	$C+jS$
$[0°, 45°]$	$-\alpha$	$[7/8N, N]$	$N - \phi$	$C-jS$
$[45°, 90°]$	$\alpha - 90°$	$[3/4N, 7/8N]$	$\phi - 3/4N$	$S-jC$
$[90°, 135°]$	$90° - \alpha$	$[5/8N, 3/4N]$	$3/4N - \phi$	$-S-jC$
$[135°, 180°]$	$\alpha + 180°$	$[N/2, 5/8N]$	$\phi - N/2$	$-C-jS$

VII. ACKNOWLEDGEMENTS

We want to thank the authors of the works [22], [24]–[26], [31] for making available their approaches for shift-and-add constant multiplications.

APPENDIX A
SYMMETRIES IN FFT ROTATIONS

Throughout the paper, all rotators were considered only for angles in the range $\alpha' \in [-45°, 0°]$. The rotator for the full angle range $\alpha \in [-180, 180°]$ can be constructed from the solution of the limited range rotator by using either trivial rotations (by $\pm 90°$ or $180°$) and/or mirroring the rotator among the real axis. As a consequence, some outputs of rotators have to be negated or swapped. Negation is done by shifting the negation of the output to the input of adders/subtractors of the subsequent butterfly. Swapping is done by simply re-wiring the outputs during VHDL generation.

Table VIII shows the rules on how to obtain the complex rotator constant. From a given full range angle $\alpha = -2\pi\phi/N$ or its corresponding ϕ, the reduced range α' and ϕ' are obtained as given in Table VIII. Next, the complex rotator coefficient for the limited range is obtained for a given accuracy from best rotators in Table IV and transformed according to the last column of Table VIII.

Take, e.g., a rotation by $\phi = 116$ of an $N = 256$ point FFT which has a corresponding angle of $\alpha = -163.125°$. Its reduced angles are obtained by Table VIII to be $\phi' = N/2 - \phi = 12$ and $\alpha' = -180° - \alpha = -16.875°$. For $WL_E = 8$, the corresponding best rotator is obtained from Table IV and has the value $C + jS = 61 - j18$. From Table VIII, the resulting rotator is then $-C + jS = -61 - j18$ which provides the required rotation by $\alpha = -163.125°$.

REFERENCES

[1] A. X. Glittas, M. Sellathurai, and G. Lakshminarayanan, "A normal I/O order radix-2 FFT architecture to process twin data streams for MIMO," *IEEE Trans. VLSI Syst.*, vol. 24, no. 6, pp. 2402–2406, Jun. 2016.

[2] K.-J. Yang, S.-H. Tsai, and G. Chuang, "MDC FFT/IFFT processor with variable length for MIMO-OFDM systems," *IEEE Trans. VLSI Syst.*, vol. 21, no. 4, pp. 720–731, Apr. 2013.

[3] 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.

[4] M. Garrido, M. Acevedo, A. Ehliar, and O. Gustafsson, "Challenging the limits of FFT performance on FPGAs," in *Int. Symp. Integrated Circuits*, Dec. 2014, pp. 172–175.

[5] M. G. Kim, S. K. Shin, and M. H. Sunwoo, "New parallel MDC FFT processor with efficient scheduling scheme," in *Proc. IEEE Asia-Pacific Conf. Circuits Syst.*, Nov. 2014, pp. 667–670.

[6] J. K. Jang, M. G. Kim, and M. H. Sunwoo, "Efficient scheduling scheme for eight-parallel MDC FFT processor," in *Int. SoC Design Conf.*, Nov. 2015, pp. 277–278.

[7] T. Ahmed, M. Garrido, and O. Gustafsson, "A 512-point 8-parallel pipelined feedforward FFT for WPAN," in *Proc. Asilomar Conf. Signals Syst. Comput.*, Nov. 2011, pp. 981–984.

[8] 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.

[9] J. Wang, C. Xiong, K. Zhang, and J. Wei, "A mixed-decimation MDF architecture for radix- 2^k parallel FFT," *IEEE Trans. VLSI Syst.*, vol. 24, no. 1, pp. 67–78, Jan. 2016.

[10] 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.

[11] S.-I. Cho and K.-M. Kang, "A low-complexity 128-point mixed-radix FFT processor for MB-OFDM UWB systems," *ETRI J.*, vol. 32, no. 1, pp. 1–10, Feb. 2010.

[12] F. de Dinechin, H. Takeugming, and J. M. Tanguy, "A 128-tap complex FIR filter processing 20 giga-samples/s in a single FPGA," in *Proc. Asilomar Conf. Signals Syst. Comput.*, Nov. 2010, pp. 841–844.

[13] G. Polat, S. Ozturk, and M. Yakut, "Design and implementation of 256-point radix-4 100 Gbit/s FFT algorithm into FPGA for high-speed applications," *ETRI Journal*, vol. 37, no. 4, pp. 667–676, Aug. 2015.

[14] A. Kovalev, O. Gustafsson, and M. Garrido, "Implementation approaches for 512-tap 60 GSa/s chromatic dispersion FIR filters," in *Proc. Asilomar Conf. Signals Syst. Comput.*, Oct. 2017, pp. 1779–1783.

[15] J. Kim, J. Lee, and K. Cho, "Design of 256-point FFT processor for 100 Gb/s coherent optical OFDM system," in *Proc. IEEE Int. Symp. Consumer Electron.*, Sep. 2016, pp. 61–62.

[16] "Dillon Engineering - Parallel FFT," Jul. 2017. [Online]. Available: http://www.dilloneng.com/parallel-fft.html

[17] M. Jamali, J. Downey, N. Wilikins, C. R. Rehm, and J. Tipping, "Development of a FPGA-based high speed FFT processor for wideband direction of arrival applications," in *Proc. IEEE Radar Conf.*, May 2009, pp. 1–4.

[18] M. Garrido, M. Sánchez, M. López-Vallejo, and J. Grajal, "A 4096-point radix-4 memory-based FFT using DSP slices," *IEEE Trans. VLSI Syst.*, vol. 25, no. 1, pp. 375–379, Jan. 2017.

[19] S.-J. Huang and S.-G. Chen, "A high-throughput radix-16 FFT processor with parallel and normal input/output ordering for IEEE 802.15.3c systems," *IEEE Trans. Circuits Syst. I*, vol. 59, no. 8, pp. 1752–1765, Aug 2012.

[20] Z.-G. Ma, X.-B. Yin, and F. Yu, "A novel memory-based FFT architecture for real-valued signals based on a radix-2 decimation-in-frequency algorithm," *IEEE Trans. Circuits Syst. II*, vol. 62, no. 9, pp. 876–880, Sept 2015.

[21] S. Malkowsky, J. Vieira, L. Liu, P. Harris, K. Nieman, N. Kundargi, I. C. Wong, F. Tufvesson, V. Öwall, and O. Edfors, "The world's first real-time testbed for massive MIMO: Design, implementation, and validation," *IEEE Access*, vol. 5, pp. 9073–9088, May 2017.

[22] O. Gustafsson, A. G. Dempster, K. Johansson, M. D. Macleod, and L. Wanhammar, "Simplified design of constant coefficient multipliers," *Circuits Syst. Signal Process.*, vol. 25, no. 4, pp. 225–251, Apr. 2006.

[23] J. Thong and N. Nicolici, "Time-efficient single constant multiplication based on overlapping digit patterns," *IEEE Trans. VLSI Syst.*, vol. 17, no. 9, pp. 1353–1357, Sep. 2009.

[24] A. G. Dempster and M. D. Macleod, "Multiplication by two integers using the minimum number of adders," in *Proc. IEEE Int. Symp. Circuits Syst.*, vol. 2, May 2005, pp. 1814–1817.

[25] O. Gustafsson, "A difference based adder graph heuristic for multiple constant multiplication problems," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2007, pp. 1097–1100.

[26] Y. Voronenko and M. Püschel, "Multiplierless multiple constant multiplication," *ACM Trans. Algorithms*, vol. 3, pp. 1–39, May 2007.

[27] L. Aksoy, E. Günes, and P. Flores, "Search algorithms for the multiple constant multiplications problem: Exact and approximate," *Microprocessors & Microsystems*, vol. 34, no. 5, pp. 151–162, Aug. 2010.

[28] M. Kumm, P. Zipf, M. Faust, and C. H. Chang, "Pipelined adder graph optimization for high speed multiple constant multiplication," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2012, pp. 49–52.

[29] A. Dempster, O. Gustafsson, and J. O. Coleman, "Towards an Algorithm for Matrix Multiplier Blocks," in *Proc. European Conf. Circuit Theory Design*, Sep. 2003, pp. 1–4.

[30] O. Gustafsson, H. Ohlsson, and L. Wanhammar, "Low-Complexity Constant Coefficient Matrix Multiplication Using a Minimum Spanning

Tree Approach," in *Proc. IEEE Nordic Signal Process. Symp.*, Feb. 2004, pp. 141–144.
[31] M. Kumm, M. Hardieck, and P. Zipf, "Optimization of constant matrix multiplication with low power and high throughput," *IEEE Trans. Comput.*, vol. 66, no. 12, pp. 2072–2080, 2017.
[32] M. D. Macleod, "Multiplierless implementation of rotators and FFTs," *EURASIP J. Adv. Signal Process.*, vol. 2005, no. 17, pp. 2903–2910, Oct. 2005.
[33] 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.
[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] 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.
[36] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. Comput.*, vol. 19, pp. 297–301, Apr. 1965.
[37] 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.
[38] R. Yavne, "An economical method for calculating the discrete Fourier transform," in *Fall Joint Comp. Conf., Part I*, Dec. 1968, pp. 115–125.
[39] P. Duhamel and H. Hollmann, "Split radix FFT algorithm," *Electronics Letters*, vol. 20, no. 1, pp. 14–16, Jan. 1984.
[40] S. G. Johnson and M. Frigo, "A modified split-radix FFT with fewer arithmetic operations," *IEEE Trans. Signal Process.*, vol. 55, no. 1, pp. 111–119, Jan. 2007.
[41] A. Wenzler and E. Luder, "New structures for complex multipliers and their noise analysis," in *Proc. IEEE Int. Symp. Circuits Syst.*, vol. 2, Apr. 1995, pp. 1432–1435.
[42] F. de Dinechin and B. Pasca, "Designing Custom Arithmetic Data Paths with FloPoCo," *IEEE Design & Test of Computers*, vol. 28, no. 4, pp. 18–27, Jul. 2012.
[43] "FloPoCo," Oct. 2018, http://flopoco.gforge.inria.fr.
[44] D. Guinart and M. Garrido, "SQNR in FFT hardware architectures," *IEEE Trans. Circuits Syst. I*, Under review.
[45] "Using look-up tables as shift registers (SRL16) in Spartan-3 generation FPGAs," May 2005, https://www.xilinx.com/support/documentation/ application_notes/xapp465.pdf.

Mario Garrido (M'07) received the M.S. 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.

Konrad Möller received the B.Eng. degree in electrical engineering from the University of Applied Sciences Fulda, Germany in 2011 and the M.Sc. degree in electrical engineering from University of Kassel, Germany in 2012. In 2017 he received his Ph.D. (Dr.- Ing.) degree from the University of Kassel, Germany were he worked in the Digital Technology Group. He currently works as electronic design engineer. His research interests are reconfigurable computing on FPGAs and model based design for FPGAs.

Martin Kumm received the Dipl.-Ing. degree in electrical engineering from the University of Applied Sciences Fulda, Germany, and the Technical University of Darmstadt, Germany, in 2003 and 2007, respectively. From 2003 to 2009, he was with GSI Darmstadt, working on digital RF control systems for particle accelerators. In 2015 he received his Ph.D. (Dr.-Ing.) degree from the University of Kassel, Germany. He currently has a post-doctoral position in the Digital Technology Group of the University of Kassel.

Dr. Kumm is member of the VLSI Systems & Applications Technical Committee of the IEEE Circuits and Systems Society and TPC member of the ARITH, ISCAS, AICAS and ICECS conferences. His research interests are arithmetic circuits and their optimization as well as high-level synthesis, all in the context of reconfigurable systems.