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

Mario Garrido, K. Möller and M. Kumm

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-155881

N.B.: When citing this work, cite the original publication.
https://doi.org/10.1109/TCSI.2018.2886626

Original publication available at:
https://doi.org/10.1109/TCSI.2018.2886626
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.
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.

Copyright (c) 2018 IEEE. Personal use of this material is permitted. However, permission to use this material for any other purposes must be obtained from the IEEE by sending an email to pubs-permissions@ieee.org
where \( W \) experimental results and compare them to previous works. In Section V, we provide fully parallel FFT architectures. In Section III, we review key concepts related to rotations in the FFT algorithm and discuss different FFT radices. In Section V, 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

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

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

![Flow graph of a 16-point radix-2^2 DIF FFT.](image)

To calculate the DFT, the FFT based on the Cooley-Tukey algorithm [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 \( \phi \) in between the stages indicates a rotation by:

\[
W_N^k = e^{-j \frac{2\pi}{N} \phi}. \tag{2}
\]

As a consequence, samples for which \( \phi \in \{0, \frac{N}{4}, \frac{N}{2}, \frac{3N}{4}\} \) must be rotated by \( 0^\circ, 270^\circ, 180^\circ \) or \( 90^\circ \), 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.

### TABLE I

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

To calculate the DFT, the FFT based on the Cooley-Tukey algorithm [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.

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 \( \phi \) in between the stages indicates a rotation by:

\[
W_N^k = e^{-j \frac{2\pi}{N} \phi}. \tag{2}
\]

As a consequence, samples for which \( \phi \in \{0, \frac{N}{4}, \frac{N}{2}, \frac{3N}{4}\} \) must be rotated by \( 0^\circ, 270^\circ, 180^\circ \) or \( 90^\circ \), 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]

\[
\begin{bmatrix}
X \\
Y
\end{bmatrix} =
\begin{bmatrix}
C & -S \\
S & C
\end{bmatrix}
\begin{bmatrix}
x \\
y
\end{bmatrix},
\tag{3}
\]

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

\[
C = R \cdot (\cos \alpha + \varepsilon_c),
\]

\[
S = R \cdot (\sin \alpha + \varepsilon_s),
\tag{4}
\]

where \( R \) is the magnitude of rotation, \( \alpha \) is the rotation angle, and \( \varepsilon_c \) and \( \varepsilon_s \) are small errors.
where $\alpha$ is the rotation angle, $\varepsilon_c$ and $\varepsilon_s$ 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 $\varepsilon = \sqrt{\varepsilon_c^2 + \varepsilon_s^2}$. (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 $WL_E = -\log_2 \varepsilon - \log_2 \epsilon + 3/2$. (6)

IV. PROPOSED FULLY PARALLEL FFT ARCHITECTURES

Fig. 2 shows the proposed 16-point radix-$2^4$ 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 consists 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. Furthermore, the accuracy of the rotators has an 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 $0^\circ$ or other trivial rotations, and other ones are by non-trivial rotations. As the number of trivial rotations, specially those by $0^\circ$, 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 $0^\circ$ 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

<table>
<thead>
<tr>
<th>Structure in Fig. 4</th>
<th>General Case</th>
<th>Angle and Coefficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCM I rotator (a)</td>
<td></td>
<td>$\alpha = m \cdot \pi/2 + \pi/4$</td>
</tr>
<tr>
<td>SCM II rotator (b)</td>
<td></td>
<td>$P = C + jS$</td>
</tr>
<tr>
<td>MCM rotator (c)</td>
<td></td>
<td>$P = C - jS$</td>
</tr>
<tr>
<td>CMM rotator (d)</td>
<td></td>
<td>$P = C$</td>
</tr>
</tbody>
</table>

| Angle and Coefficient | $\alpha = m \cdot \pi$ | $\alpha = m \cdot \pi + \pi/2$ |

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

<table>
<thead>
<tr>
<th>Shift-and-add Algorithm</th>
<th>Rotator Configuration</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPAG-CMM</td>
<td>CMM</td>
<td>[31]</td>
</tr>
<tr>
<td>HCUB</td>
<td>MCM</td>
<td>[26]</td>
</tr>
<tr>
<td>DiffAG</td>
<td>MCM</td>
<td>[25]</td>
</tr>
<tr>
<td>MAG2</td>
<td>MCM</td>
<td>[24]</td>
</tr>
<tr>
<td>Optimal SCM</td>
<td>SCM I</td>
<td>[22]</td>
</tr>
<tr>
<td>Optimal SCM</td>
<td>SCM II</td>
<td>[22]</td>
</tr>
</tbody>
</table>

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 constant multiplication using 3 real multipliers and 3 adders, version II. (c) Complex constant multiplication using 4 real multipliers and 2 adders. (d) General case of a complex constant multiplication.

Animate 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 $\varepsilon$ that defines the circle is calculated from $WL_E$ according to (6). Additionally, we include the condition that 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.

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.

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 $[-45^\circ, 0^\circ]$ 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 $[-45^\circ, 0^\circ]$.

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 $W_{LE}$ and $W_{LC}$, which justifies the selection of the best case for each angle. Furthermore, we can get an estimate of the savings of the proposed approach in terms of adders with respect to conventional

![Binary trees of the implemented fully parallel FFTs.](image)

![Diagram](image)
approaches. With respect to canonical-signed-digit (CSD), the proposed approach saves 25% 30%, 35% and 36% of the adders for $\text{WL}_E$ 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 consists of obtaining the $\phi$ values required at the stages of each FFT algorithm, and then sum the adder cost of the rotators that each algorithm includes. The $\phi$ 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

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

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.

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 $\text{WL}_E$ and $\text{WL}_C$. 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.,

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

This equation may be useful for other FFT algorithms and/or to estimate other values of $\text{WL}_E$ 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

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Adders</th>
<th>WL_E/WL_C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radix-2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Split Radix</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Radix-2$^2$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Split Radix</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Radix-2$^3$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Split Radix</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Radix-2$^4$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Split Radix</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table VI: Adder cost of the proposed fully parallel FFT architectures.

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

- 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.

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 $\text{WL}_E$ and $\text{WL}_C$. 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.,

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

This equation may be useful for other FFT algorithms and/or to estimate other values of $\text{WL}_E$ 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

1. 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.
2. 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.
3. 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 $\text{WL}_E$ and $\text{WL}_C$. 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.
4. 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.,

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

This equation may be useful for other FFT algorithms and/or to estimate other values of $\text{WL}_E$ 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

1. 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.
2. 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.
3. 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 $\text{WL}_E$ and $\text{WL}_C$. 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.
4. 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.,

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

This equation may be useful for other FFT algorithms and/or to estimate other values of $\text{WL}_E$ that are not collected in Table VI.
the $\phi$ 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 $\phi$ 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 $W_L_I$, and the effective word length of the rotators, $W_L_E$. The rotation matrix with the $\phi$ values of the FFT algorithm, and the adder graph are then looked-up in the text files.

The output word length $W_L_O$ has been selected to be the same as the input word length $W_L_I$. Therefore, the butterfly additions/subtractions are done by using $W_L_I + 1$ 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 $W_L_I$ most significant bits by truncation. These $W_L_I$ 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 wI=16 wE=16 N=16 alg=BT
```
where $wI$ and $wE$ stand for $W_L_I$ and $W_L_E$, 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 $W_L_E = 16$ 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 DSPs 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 Gb/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 this, we obtain:

$$P_{\text{NORM}} = \frac{29.93}{256 \cdot 32 \cdot 541} = 6.75 \, \mu\text{W/bit/Hz}. \quad (10)$$
Finally, the SQNR of the proposed architectures is high and approximates to the following equation:

\[
\text{SQNR} \approx 90 - 3.5 \cdot \log_2 N \text{ (dB)}.
\]  

(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 \(W_L E\).

### 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.

### 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.
TABLE VIII

Mapping of the full range angle $\alpha \in [-180^\circ, 180^\circ]$ to the reduced range $\alpha' \in [-45^\circ, 0^\circ]$ and its modification on the rotator constant

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>$\alpha'$</th>
<th>$\phi$</th>
<th>$\phi'$</th>
<th>Rot. Const.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[-180^\circ, -135^\circ]$</td>
<td>$-180^\circ - \alpha$</td>
<td>$[3/8, N/2]$</td>
<td>$N/2 - \phi$</td>
<td>$-C + jS$</td>
</tr>
<tr>
<td>$[-135^\circ, -90^\circ]$</td>
<td>$-90^\circ - \alpha$</td>
<td>$[N/4, 3/8, N/4]$</td>
<td>$\phi - N/4$</td>
<td>$-S + jC$</td>
</tr>
<tr>
<td>$[-90^\circ, -45^\circ]$</td>
<td>$-45^\circ - \alpha$</td>
<td>$[N/8, N/4]$</td>
<td>$N/4 - \phi$</td>
<td>$S + jC$</td>
</tr>
<tr>
<td>$[-45^\circ, 0^\circ]$</td>
<td>$0^\circ - \alpha$</td>
<td>$[0, N/8]$</td>
<td>$\phi$</td>
<td>$C + jS$</td>
</tr>
<tr>
<td>$[0^\circ, 45^\circ]$</td>
<td>$45^\circ - \alpha$</td>
<td>$[7/8, N, N/8]$</td>
<td>$N - \phi$</td>
<td>$C - jS$</td>
</tr>
<tr>
<td>$[45^\circ, 90^\circ]$</td>
<td>$90^\circ - \alpha$</td>
<td>$[3/4N, 7/8N]$</td>
<td>$\phi - 3/4N$</td>
<td>$S - jC$</td>
</tr>
<tr>
<td>$[90^\circ, 135^\circ]$</td>
<td>$135^\circ - \alpha$</td>
<td>$[5/8N, 3/4N]$</td>
<td>$3/4N - \phi$</td>
<td>$S - jC$</td>
</tr>
<tr>
<td>$[135^\circ, 180^\circ]$</td>
<td>$180^\circ - \alpha$</td>
<td>$[N/2, 5/8N]$</td>
<td>$\phi - N/2$</td>
<td>$-C - jS$</td>
</tr>
</tbody>
</table>

APPENDIX A

Symmetries in FFT Rotations

Throughout the paper, all rotators were considered only for angles in the range $\alpha' \in [-45^\circ, 0^\circ]$. The rotator for the full range angle $\alpha \in [-180^\circ, 180^\circ]$ can be constructed from the solution of the limited range rotator by using either trivial rotations (by $\pm 90^\circ$ or $180^\circ$) 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 $\phi$, the reduced range $\alpha'$ and $\phi'$ 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 $\alpha = -163.125^\circ$. Its reduced angles are obtained by Table VIII to be $\phi' = N/2 - \phi = 12$ and $\alpha' = -180^\circ - \alpha = -168.75^\circ$. For $W_{L_E} = 8$, 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 $\alpha = -163.125^\circ$.

REFERENCES

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. He currently works 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.