SFF—The Single-Stream FPGA-Optimized Feedforward FFT Hardware Architecture

Carl Ingemarsson¹ · Oscar Gustafsson¹

Received: 8 December 2017 / Revised: 15 March 2018 / Accepted: 18 April 2018 / Published online: 1 May 2018
© The Author(s) 2018

Abstract

In this paper, a fast Fourier transform (FFT) hardware architecture optimized for field-programmable gate-arrays (FPGAs) is proposed. We refer to this as the single-stream FPGA-optimized feedforward (SFF) architecture. By using a stage that trades adders for shift registers as compared with the single-path delay feedback (SDF) architecture the efficient implementation of short shift registers in Xilinx FPGAs can be exploited. Moreover, this stage can be combined with ordinary or optimized SDF stages such that adders are only traded for shift registers when beneficial. The resulting structures are well-suited for FPGA implementation, especially when efficient implementation of short shift registers is available. This holds for at least contemporary Xilinx FPGAs. The results show that the proposed architectures improve on the current state of the art.

Keywords Fast Fourier transform (FFT) · Field-programmable gate arrays (FPGAs) · Pipeline FFT · FPGA optimization · Single-stream FFT

1 Introduction

Hardware implementations of cores for computation of the discrete Fourier transform (DFT) continuously find more applications. Examples are, for instance, in contemporary communications schemes that use orthogonal frequency division multiplexing (OFDM) and in spectral analysis. The fast Fourier transform (FFT) is a collection of algorithms for efficient computation of the DFT.

Field-programmable gate array (FPGA) technology continues to become more mature, and are to an increasing degree used in signal processing applications. Therefore there is also an increasing need for efficient FPGA implementation of different algorithms, among these, the FFT.

Over the years, many different architectures for hardware implementation of the FFT have been proposed [11].

They range from fully parallel structures with an excess of processing elements and maximal throughput at minimal latency, to memory based structures that sequentially computes the DFT using only one or a few processing elements. As an intermediate class, pipeline FFT architectures have been proposed [23]. As the name suggests, such FFT cores consists of a pipeline of stages that are fed either one (single-stream [13]) or several (parallel-stream [12]) sample(s) per clock cycle.

The radix-2 single-path delay feedback (SDF) architecture was the first single-stream pipeline FFT architecture to be proposed [13]. Since then several single-stream FFT architectures have been suggested [2, 3, 6, 10, 18, 21, 24]. Some decades later, a major milestone in this field came with the introduction of the radix-2² algorithm [14]. In the literature one can also find numerous publications that consider the implementation of single-stream FFTs in FPGAs, e.g. [1, 5, 17, 19, 25, 26]. However, there has not been much research performed on the architectural level on FFT with the particular aspect of FPGA-mapping in consideration. Most earlier FFT architecture works have focused on ASIC implementation. However, that an architecture works well in an ASIC context certainly does not guarantee that it also works well in an FPGA. This was to some degree shown in [15, 16], where also transformations allowing a better FPGA-mapping of radix-2 SDF pipeline FFTs were proposed.

Special Issue on fast Fourier transform (FFT) hardware implementations

✉ Carl Ingemarsson
carl.ingemarsson@liu.se

Oscar Gustafsson
oscar.gustafsson@liu.se

¹ Division of Computer Engineering, Department of Electrical Engineering, Linköping University, SE-581 83, Linköping, Sweden

In this paper, a new single-stream radix-2 pipeline FFT architecture is proposed. We refer to this architecture as the single-stream FPGA-optimized feedforward (SFF) architecture. In itself this architecture provides a reduction in adders required, as compared to the SDF and SDC architectures, at the price of an increase in the amount of memory required compared to the SDF architecture. In fact, the resulting number of adders is minimal and these adders are fully utilized. The SFF architecture has the same control structure and data ordering as the SDF and SDC architectures. We can therefore also propose a combined SFF and SDF architecture. We refer to this architecture as SFF/SDF. Why and when this is beneficial is discussed later in the paper. The results show that the proposed architectures are well-suited for FPGA implementation, at least in FPGAs where short shift registers can be efficiently implemented, such as contemporary Xilinx FPGAs.

The paper is organized as follows: in Section 2, the SFF and SFF/SDF architectures are described. Then, in Section 3, these architectures are compared with previously proposed single-stream pipeline FFT architectures. In Section 4, the FPGA-mapping of the proposed architectures is discussed and in Section 5 a proof-of-concept parameterized HDL design for FPGA implementation is described. Implementation results are reported and discussed in Section 6. Finally, Section 7 summarizes the contributions, and concludes the paper.

2 The SFF Architecture

The SFF architecture is a radix-2 pipeline FFT architecture that consists of $\log_2 N$ stages, where N is the transform length. Most earlier proposed FFT architectures include so called butterfly processing elements consisting of a pair of one complex adder and one complex subtractor operating in parallel. The stages in an SFF pipeline instead of butterfly units contains a new processing element that with a single complex adder-subtractor unit performs the additions and subtractions in a time-interleaved manner. An SFF stage is depicted in Fig. 1.

In Fig. 2, the signal flow graph (SFG) of a 16-point radix- 2^2 decimation in frequency FFT is depicted. As can be

Figure 1 An SFF stage. l denotes the length of the two shift registers.

Figure 2 Signal flow graph of 16-point radix- 2^2 FFT.

seen, in the first stage the distance between the samples to be added and subtracted is eight. Therefore in the first stage of a 16-point SFF FFT core the length of the shift registers is eight. The structure of one whole radix- 2^2 SFF 16-point FFT core is depicted in Fig. 3. Configured in this fashion the FFT core has normal input order and bit-reversed output order. It is of course possible to construct an SFF FFT core with normal ordered output and bit-reversed input, as for other FFT architectures [13]. In that case the largest register will be in the end of the pipeline instead.

To make the data flow of an SFF stage more clear a data flow diagram is shown in Table 1. For sake of a more compact diagram the data flow of a stage with shift registers of length two is depicted. This could, e.g., be the third stage in the 16-point SFF pipeline, or the first stage in a four-point pipeline.

As can be seen in Table 1, the output order of an SFF stage is exactly the same as that of a corresponding SDF or SDC stage. It can also be seen that the control signal needed for each stage is the same as that of the stages in an SDF or SDC pipeline. The bits from a $\log_2 N$ bit synchronous counter is sufficient to control the multiplexers in each stage. These two facts makes it possible to combine SDF, SDC, and SFF stages in the same pipeline. As there are no cases where the SDC architecture is better than the SDF, we only consider combining SDF and SFF. We refer to such a pipeline as having an SFF/SDF architecture. The choice of which type of stage to use for a given stage number is discussed later on.

Figure 3 A 16-point radix-2² SFF pipeline FFT core.

3 Comparison with Earlier Proposed Architectures

Although previously proposed FFT architectures have not targeted FPGA implementation as such, many of the FPGA implementations that have been reported made use of these architectures. In the following, we summarize those contributions and also compare them with the proposed architectures. In Table 2, the proposed and previously published architectures for single-stream FFT computation are compared. The data in Table 2 is for the case with normal ordered input with no requirement on the output order. Included are the number of complex operations required as well as the switching complexity, i.e., how complicated multiplexing is required by the architecture, where simple means few multiplexers and complex many.

In Table 2, it is seen that the radix-2 SDF [13] and radix-4 SDF [6] architectures uses $N - 1$ memory slots, which is the minimal number of memory slots needed. This is an advantage of these architectures. However, both use more adders that are not fully utilized. The radix-4 SDC architecture [2] uses fewer adder, although not minimal. This comes at a cost of twice the amount of memory as compared with the SDF architectures. Although not explicitly defined in [2], the radix-2 SDC architecture can be easily derived from the same concepts. The resulting architecture is illustrated in, e.g., [4]. Again, higher memory complexity is obtained.

Table 1 Data flow of third stage in 16-point or first stage in four-point SFF pipeline.

Clock cycle	1	2	3	4	5	6
S	0	0	1	1	0	0
Input:	x_0	x_1	x_2	x_3		
SR 1			x_0	x_1	x_2	x_3
SR 2					x_0	x_1
Mux out			x_2	x_3	x_0	x_1
Additions			$\times$	$\times$		
Subtractions					$\times$	$\times$
Output			$x_0 + x_2$	$x_1 + x_3$	$x_0 - x_2$	$x_1 - x_3$

In parallel with the evolution of different pipeline FFT architectures, improvements to FFT algorithms have also been proposed. It was a major milestone in this field when the radix-2² algorithm was proposed [14]. This algorithm uses a novel twiddle factor decomposition scheme to combine the low number of non-trivial multiplications of the radix-4 algorithm with the lower number of adders and memory of the radix-2 SDF architecture. Since then more research have been conducted resulting in many other efficient twiddle factor decompositions. In [4], a generalized approach for radix- r^k was proposed. In [20], the generation of all possible radix-2 twiddle factor decompositions based on binary trees was described and analyzed, which covers a larger design space compared to [4]. The amount of possible decompositions was further extended by an approach corresponding to moving the twiddle factor multiplications between stages in [9]. The proposed architecture, as most other earlier proposed architectures, can be combined with any of these approaches.

The radix-2 SD²F was proposed in [24]. As can be seen in Table 2 this architecture needs the minimal number of adders at the price of a larger amount of memory as compared to the SDF architectures. This also holds for the proposed SFF architecture. One major drawback for FPGA implementation of the radix-2 SD²F architecture is though that its stages include a large number of multiplexers. The implementation cost of multiplexers is much higher in FPGAs than in ASIC implementations [8]. This makes it prohibitive to consider radix-2 SD²F for highly optimized FPGA implementation.

Chang [3], proposed a radix-2 single-stream SDC FFT architecture. This architecture is similar to the proposed SFF architecture in the sense that it also uses time-multiplexing to reduce the number of adders needed. This is achieved by dividing the sample stream into two half-word streams, so that the adders and subtractors can be time-multiplexed over the two halves of the words.

Another radix-2 single-stream SDC pipeline FFT architecture was proposed by Liu et al. in [18]. Again, the purpose is to reduce the number of additions by time-multiplexing. This architecture separates the complex samples to time-multiplex the adders and subtractors over the real-imaginary dimension.

Table 2 Characteristics of the proposed architectures with previously published single-stream architectures for normal ordered input.

Architecture	# Multipliers	# Adders	Memory size	Switching
Radix-2 SDF [13]	$2(\log_4 N - 1)$	$4 \log_4 N$	$N - 1$	Simple
Radix-4 SDF [6]	$\log_4 N - 1$	$8 \log_4 N$	$N - 1$	Simple
Radix-2 SDC [2, 4]	$2(\log_4 N - 1)$	$4 \log_4 N$	$2N - 2$	Very simple
Radix-2 ² SDC [2, 4]	$\log_4 N - 1$	$4 \log_4 N$	$2N - 2$	Very simple
Radix-4 SDC [2]	$\log_4 N - 1$	$3 \log_4 N$	$2N - 2$	Moderate
Radix-2 ² SDF [14]	$\log_4 N - 1$	$4 \log_4 N$	$N - 1$	Simple
Radix-2 SD ² F [24]	$\log_4 N - 1$	$2 \log_4 N$	$\frac{4}{3}(N - 1)$	Complex
Radix-2 SDC [3]	$2(\log_4 N - 1)$	$2 \log_4 N$	$\frac{3}{2}N + 2(\log_2 N - 2)$	Complex
Radix-2 SDC [18]	$2(\log_4 N - 1)$	$2 \log_4 N$	$\frac{3}{2}N - 2$	Complex
Radix-2 SDC-SDF [21]	$\log_4 N - 1$	$2 \log_4 N + 1$	$\frac{3}{2}(N + \log_2 N - 1)$	Complex
Radix-2 SC [10]	$\log_4 N - 1$	$2 \log_4 N$	$\frac{3}{2}N + 5 \log_2 N - 2$	Complex
Proposed Radix-2 SFF	$2(\log_4 N - 1)$	$2 \log_4 N$	$2N - 2$	Very simple
Proposed Radix-2 SFF/SDF	$2(\log_4 N - 1)$	$2 \log_4 N + k$	$N - 2 + \frac{N}{2^k}$	(Very) simple
Proposed Radix-2 ² SFF	$\log_4 N - 1$	$2 \log_4 N$	$2N - 2$	Very simple
Proposed Radix-2 ² SFF/SDF	$\log_4 N - 1$	$2 \log_4 N + k$	$N - 2 + \frac{N}{2^k}$	(Very) simple

In [21] a combined SDF-SDF radix-2 single-stream pipeline FFT architecture was proposed. Also this architecture uses time-multiplexing and splitting of the complex samples to reduce the number of adders/subtractors and multipliers needed.

In [10], the serial commutator (SC) FFT was proposed by Garrido et al. The SC architecture is similar to those proposed in [3, 21] in the sense that minimal number of adders/subtractors is achieved by time-sharing over the real-imaginary dimension. Note, that the number of memory slots for the SC FFT differs between what is reported in [10]. For an arbitrary input order, the approach in [10] has lower memory requirements. Also, in our table the memory required inside the processing element is included.

As can be seen in Table 2, all four architectures proposed in [3, 10, 18, 21] use (almost) the minimal number of additions, however at the cost of a higher number of memory words, about $\frac{3}{2}N$, more than what is required for the radix-2 SD²F architecture. All these architecture also contain a large number of multiplexers making them less suitable for implementation in FPGAs.

The parameter k for the proposed SFF/SDF architecture in Table 2 is a trade-off parameter. The expressions hold when k is the number of SDF stages in the beginning of the pipeline. We see that this architecture is unique in Table 2 in providing such a trade-off between the number of adders and amount of memory needed. To illustrate the implications of this trade-off, a plot of the number of additions and memory for a transform length equal to 1024 and for different values of k are shown in Fig. 4. Horizontal lines corresponding to the memory and adder requirements of the SFF, radix-2 SDF and radix-2 SD²F architectures

are also plotted as reference. We see that when $k = 2$ the amount of memory is smaller than for radix-2 SD²F from [24] using only two more adders.

4 FPGA Mapping

When mapping an architecture on an FPGA it is important to consider the hardware structure of the targeted FPGA to achieve efficient results [7, 16]. Different FPGA families have both similarities and differences in their hardware structure. In the following, we will mainly restrict our attention toward contemporary Xilinx FPGAs with six-input LUTs.

Figure 4 Memory and adder requirements of SFF/SDF for a 1024-point FFT.

Many FPGA vendors have introduced pre-adders in front of the embedded multipliers in their most recent generations of FPGAs. Although mainly aimed at the implementation of linear-phase finite impulse response (FIR) filters, these pre-adders can also be used for other purposes. In our earlier work [15, 16], it was shown that these pre-adders can be used in FPGA implementation of radix-2 SDF stages. However, this requires that transformations are applied to the SDF stage. As an SFF stage has its adder directly connected to the following twiddle factor multiplier, mapping of this adder to the pre-adder of a following multiplier block is straightforward without any transformations. This of course requires, both for the transformed SDF and SFF case, that the stage is followed by a general multiplier and hence depends on which twiddle factor decomposition is being used.

As has been shown in previous sections, the SFF and SFF/SDF architectures reduce the number of adders by an increase of the amount of memory needed as compared with the SDF architecture. This is not obviously beneficial but can certainly be in some situations. FPGA implementation, at least in contemporary Xilinx devices, is one such case.

For those stages in the SFF pipeline that have small feedforward shift registers or all stages in a sufficiently short pipeline it is possible to map the shift registers to SRL primitives in Xilinx devices. Contemporary Xilinx FPGAs with six-input look-up tables (LUTs) allow two one bit wide shift registers, equally long, and with length less or equal to 16 to be mapped to the same LUT6 resource. An adder implemented in such an FPGA requires one complete LUT6 per output bit. This means that the LUT cost of a short shift register is half that of an adder. Because of this, the SFF architecture is very suitable for FPGA implementation of short pipeline FFT processors. For longer pipelines that contain shift registers of larger sizes it is better to use the SFF/SDF architecture with SFF stages where the shift registers are short and SDF stages where the shift registers are longer.

In Figs. 5 and 6, the required number of LUTs for, one ordinary SDF stage, one ordinary SDC stage, one optimized SDF stage from [16] (denoted as SDF'), and one SFF stage, are indicated. The indicated LUT resource usage assumes that the shift registers are mapped to SRL primitives and that the length l is less than or equal to 16. In Fig. 5, it is further assumed that pre-adder mapping is not possible. This could, e.g., be the case for the last stage in the pipeline or for a stage followed by a trivial twiddle factor multiplier. In Fig. 6, the situation when pre-adder mapping is possible is considered. For the ordinary SDF and SDC stages, pre-adder mapping is not possible, and hence this stage is not included in Fig. 6.

In Table 3, the number of LUTs required for the different types of stages and different lengths of the shift register(s)

are shown. In Table 3, SRL implementation of the shift register(s) is assumed. We see that SFF is better than SDF for the cases when pre-adder mapping is not possible, which for the radix-2² algorithm is more than half of the stages. It is also clear that there are no cases where the SDC architecture is beneficial.

In Fig. 7, the best FPGA mapping of a 16-point radix-2² SFF pipeline FFT core is illustrated.

Figure 5 LUT6 resources required for SDF, SDC, SDF', and SFF without pre-adder mapping with $l \leq 16$. Dashed indicates one LUT6 per output bit, and dotted one half.

Figure 6 LUT6 resources required for SDF' and SFF with pre-adder mapping with $l \leq 16$. Dashed indicates one LUT6 per output bit, dotted one half, and dash-dotted pre-adder equipped multiplier block.

5 Parameterized FPGA Implementation

To demonstrate the efficiency in FPGA implementation of the proposed architectures a parameterized FPGA design was constructed. It is parameterized in transform length, data word length, and coefficient word length. It also contains a parameter that controls if an SFF/(SDF), SDF', or SDF architecture should be used. This allow the implementation results to be fairly compared with what was proposed in [16], as well as with FPGA implementation

of the ordinary SDF architecture. We will return to this comparison when discussing the implementation results in the following section. The parameterized design also has another parameter that selects if a radix-2 or radix-2² twiddle factor decomposition should be used. As evident from Table 3 there are no cases where the standard SDC stage may be beneficial, and, hence, no SDC architecture was implemented.

In the SFF/SDF architecture, optimized stages from [16] are used in the SDF part of the pipeline. To stress this, we henceforth denote this architecture SFF/SDF'. It will be clear from the results that the implementation of SFF/SDF' with ordinary SDF stages is not preferable. Both for SDF' and SFF, pre-adder enabled stages are used if the stage is followed by a non-trivial twiddle factor multiplication. In case of the SFF/SDF' architecture pipelines for transform length 64 or less uses only SFF stages. For longer pipelines, SFF stages are used for the stages with shift registers up to and including 32. As seen earlier in Table 3, one SFF stage with shift registers of length 32 costs slightly more than the equivalent SDF' stage. However, we anyhow choose to have this as an SFF stage in SFF/SDF' so that the 64-point FFT is implemented with only SFF stages.

For constant data word length through the pipeline, in each stage, a division by two or, equivalently, a one bit right shift is introduced after the adder-subtractor. After this, the least significant bit is dropped. This corresponds to safe scaling [22] and truncation as quantization scheme.

The SFF stages as described above have an inverter on the carry input of the adder-subtractor. In order not to have this in the critical path reducing the maximal clock frequency, $f_{\max}$, instead, the control signal was inverted. For pipelines with only SFF stages, a counter that counts down from $N - 1$ to zero was used. For other pipelines an ordinary up-counter was used. For SFF/SDF' pipelines an inverter was used to

Table 3 Required number of LUT6 per stage and output bit.

	SDF	SDC	SDF'	SFF
Pre-adder mapping not possible				
Figure	Figure 5a	Figure 5b	Figure 5c	Figure 5d
LUT6/stage	$0.5\lceil l/16 \rceil + 3$	$\lceil l/16 \rceil + 2.5$	$0.5\lceil l/16 \rceil + 2$	$\lceil l/16 \rceil + 1$
for $l \leq 16$	3.5	3.5	2.5	2
for $l = 32$	4	4.5	3	3
for $l = 64$	5	6.5	4	5
Pre-adder mapping possible				
Figure	–	–	Figure 6a	Figure 6b
LUT6/stage	–	–	$0.5\lceil l/16 \rceil + 1$	$\lceil l/16 \rceil + 0.5$
for $l \leq 16$	–	–	1.5	1.5
for $l = 32$	–	–	2	2.5
for $l = 64$	–	–	3	4.5

Figure 7 Best mapping of 16-point radix- 2^2 SFF pipeline. Dashed means one LUT6 per output bit, dotted one half, and dash-dotted pre-adder multiplier block.

invert the control signals between the SDF' and SFF part of the pipeline.

In order to increase $f_{\max}$, pipeline registers were introduced at the output of each stage. Pipeline registers were also added after each trivial multiplication that is positioned before an SFF stage. Trivial multiplications before both ordinary and optimized SDF stages can be integrated in the same LUT as the adder/subtractor in the stage [16]. Hence, adding pipeline registers there is not a good idea. Because of this SFF(SDF') pipelines will have slightly higher latency than the corresponding SDF and SDF' ones.

The complex multipliers were implemented using four DSP48E1 blocks each. All pipeline register in the DSP blocks were enabled allowing the DSP blocks to run at their maximal clock frequency of 639 MHz. This gives a latency through the complex multipliers of four clock cycles. The DSP blocks were instantiated in order to achieve the desired pre-adder mapping, especially that involving the bypass function needed by the SDF' stages [16]. All other circuitry was described using RTL level VHDL code.

Shift registers of length 32 and shorter were mapped to distributed resources (SRLs or registers), while longer shift registers were mapped to block RAMs.

6 FPGA Implementation Results

The parameterized design was implemented for power-of-four transform lengths from 16 to 1024, and for different choices regarding architecture. Two sets of pipelines was considered, one set using radix-2 twiddle factor decompositions and the other set instead using radix- 2^2 . The Xilinx Virtex-6 device xc6vsx315t-3 was used as target. In all implementations, a 16-bit data word length for real and imaginary part respectively was chosen. The coefficient word length was chosen to be equal to that of the data. In Table 4, the FPGA implementations results are shown. It should be noted that the SDF' design was compared with earlier works in [16] and shown to be the current state of the art. Hence, results for other previous works are not included here. The generic SDF design is included to illustrate the difference with a non-FPGA optimized design.

One major difference between the shorter and longer pipelines is that longer pipelines contain block RAMs. The

output delay and routing delay from a shift register gets larger if the shift register is mapped to a block RAM instead of distributed resources. This also holds for twiddle factor memories implemented using block RAMs. Because of this, the pipelines containing block RAMs have a significantly smaller $f_{\max}$. This could be mitigated by mapping the last position in the shift register to a slice register instead of mapping it to the block RAM. Similar techniques can naturally be used for twiddle factor memories implemented in block RAMs. This slice register can then be placed closer to the following carry chains. This holds for all three kinds of pipelines.

For pipelines with the same architecture, the critical path should be of similar length since the pipeline is just a repetition of similar stages and there are pipeline registers between the stages. For all considered pipelines, the implementation flow was executed for many different timing constraints to find $f_{\max}$. However, the tool uses a heuristic when mapping the design to FPGA components. Hence, when the pipelines get longer, the difficulty in this problem gets larger, and therefore it is less likely to find an as high $f_{\max}$ for the longer pipelines as for the shorter ones. $f_{\max}$ is also limited either by the maximal frequency of the DSP blocks at 639 MHz or, in cases where such are used, block RAMs at 600 MHz. Primarily, Xilinx ISE 14.4 was used, although for some cases version 13.2 was used instead.

In Table 4, we clearly see how the SFF(SDF') architecture is superior both to the optimized SDF (SDF') and ordinary SDF architecture. Although the relative slice savings of SFF compared with SDF' are less for longer pipelines, the stages with shorter shift registers always offer reductions if implemented with SFF stages. This is a significant part, even for longer pipelines. For a 4096-point processor, the SFF part will be half of the pipeline.

In all cases, the SFF or SFF/SDF' architecture performs better compared with only SDF' in terms of area. This is archived at the expense of a slight addition in latency. However, in longer pipelines, the two types of stages are good complements to each other in the SFF/SDF' architecture. Therefore the optimized stages from [16] still has a role to play. Also, in FPGAs from other vendors or older Xilinx FPGAs, where the SFF might not result in area savings, some version of the optimized SDF stages should still be usable. We note that the clock frequency is

Table 4 FPGA implementation results for proposed architecture, SFF(/SDF'), FPGA optimized SDF architecture, SDF' [16], and generic SDF architecture, SDF.

	16-point			64-point			256-point			1024-point		
	SFF	SDF'	SDF	SFF	SDF'	SDF	SFF/SDF'	SDF'	SDF	SFF/SDF'	SDF'	SDF
Radix-2												
# of slices	71	82	178	113	126	284	144	157	378	177	191	477
# of LUT6	266	289	660	434	459	1069	545	569	1420	661	683	1776
used as logic	132	219	595	215	332	962	302	417	1296	391	510	1634
used as memory	101	37	42	202	90	94	210	98	103	220	110	113
used as route-thrus	33	34	23	17	37	13	33	54	21	50	63	29
# of registers	423	358	419	663	593	716	784	719	906	918	848	1104
# of DSP48E1s	8	8	8	16	16	16	24	24	24	32	32	32
# of block RAMs	0	0	0	0	0	0	4	4	4	8	8	8
RAMB36E1s	0	0	0	0	0	0	0	0	0	2	2	2
RAMB18E1s	0	0	0	0	0	0	4	4	4	6	6	6
Latency, # clks	28	27	27	86	85	85	288	287	287	1066	1065	1065
Latency, ns @ $f_{\max}$	44.2	42.3	78.4	154	138	248	556	541	908	2210	2170	3520
$f_{\max}$, MHz	633	639	345	560	617	343	518	530	316	482	490	303
TP/slice, kS/s/slice	8916	7793	1938	4956	4897	1208	3597	3376	836	2723	2565	635
Radix-2 ²												
# of slices	79	91	170	128	139	265	164	181	349	208	223	436
# of LUT6	299	320	625	493	504	992	619	645	1305	776	794	1627
used as logic	163	267	563	263	419	885	399	556	1189	539	693	1499
used as memory	101	34	38	203	85	92	203	88	94	211	98	98
used as route-thrus	35	19	24	27	0	15	17	1	22	26	3	30
# of registers	468	369	399	720	587	648	862	724	817	1005	859	992
# of DSP48E1s	4	4	4	8	8	8	12	12	12	16	16	16
# of block RAMs	0	0	0	0	0	0	3	3	3	6	6	6
RAMB36E1s	0	0	0	0	0	0	0	0	0	5	5	5
RAMB18E1s	0	0	0	0	0	0	3	3	3	1	1	1
Latency, # clks	25	23	23	80	77	77	278	275	275	1052	1049	1049
Latency, ns @ $f_{\max}$	39.2	36.0	66.5	125	127	227	518	511	899	2080	2070	3450
$f_{\max}$, MHz	639	639	346	639	607	339	537	538	306	505	507	304
TP/slice, kS/s/slice	8089	7022	2035	4992	4367	1279	3274	2972	877	2428	2274	697

slightly lower for SFF(/SDF') than for the SDF' architecture. The main reason for this is that the path going through the multiplexer in the SFF stages gets quite long for the case when pre-adder mapping is enabled. Adding an extra pipeline cut here mitigates this problem at the expense of an increase in latency, but we have chosen not to apply this given that the increase in frequency anyhow is limited by the output paths from block RAM in most of these cases as discussed above.

In Table 4, we also see that, by choosing a radix-2 or radix-2² twiddle factor decomposition, it is in all cases possible to trade DSP blocks against slices which can be

good in some situations. However, in most cases probably radix-2² is a better choice given the lower number of DSP blocks required.

Note that the SFF stages is applied when the SRLs can be efficiently used as seen from Table 3 and Figs. 5d and 6b. However, there may be cases where only one of two ports of a two-port block RAM is used. For these cases, the SFF stage can again be applied with a saving in LUTs, and no increase in block RAMs as long as the shift register fits. For the considered FPGA, this requires that the `read first` port mode is used. Also note that for all architectures it is possible to reduce block RAMs needed [16].

7 Conclusion

In this paper, SFF, a new single-stream pipeline FFT architecture is proposed. It is well suited for implementation in FPGAs with efficient shift register implementation, which at least includes contemporary Xilinx FPGAs. As the control signals are identical to SDF and SDC architectures, in addition, an architecture that combines SDF and SFF stages in the same pipeline is proposed. This architecture provides a trade-off between the required number of additions and required amount of memory. It is shown that the SFF architecture and the combined SFF and SDF architecture improves on the current state of the art.

This work also shows how efficient FPGA implementation results can be achieved by considering the mapping of the architecture to the course-grained hardware structure of the FPGA. Knowledge of this aspect of the mapping is generally crucial for efficient FPGA implementation results.

Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

References

1. Abdullah, S.S., Nam, H., McDermot, M., Abraham, J.A. (2009). A high throughput FFT processor with no multipliers. In *Proceedings of the IEEE international conference on computer design* (pp. 485–490). Lake Tahoe. <https://doi.org/10.1109/ICCD.2009.5413113>.
2. Bi, G., & Jones, E.V. (1989). A pipelined FFT processor for word-sequential data. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(12), 1982–1985. <https://doi.org/10.1109/29.45545>.
3. Chang, Y.N. (2008). An efficient VLSI architecture for normal I/O order pipeline FFT design. *IEEE Transactions on Circuits and Systems II*, 55(12), 1234–1238. <https://doi.org/10.1109/TCSII.2008.>
4. Cortes, A., Velez, I., Sevillano, J.F. (2009). Radix FFTs: matricial representation and SDC/SDF pipeline implementation. *IEEE Transactions on Signal Processing*, 57(7), 2824–2839. <https://doi.org/10.1109/TSP.2009.2016276>.
5. Derafshi, Z.H., Frounchi, J., Taghipour, H. (2010). A high speed FPGA implementation of a 1024-point complex FFT processor. In *International conference on computer network technology* (pp. 312–315). Bangkok. <https://doi.org/10.1109/ICCNT.2010.12>.
6. Despain, A.M. (1974). Fourier transform computers using CORDIC iterations. *IEEE Transactions on Computers*, C-23(10), 993–1001. <https://doi.org/10.1109/T-C.1974.223800>.
7. Ehliar, A. (2010). Optimizing Xilinx designs through primitive instantiation: guidelines, techniques, and tips. In *Proceedings of the FPGAworld conference* (pp. 20–27). <https://doi.org/10.1145/1975482.1975484>.
8. Ehliar, A., & Liu, D. (2009). An ASIC perspective on FPGA optimizations. In *Proceedings of the international conference on field-programmable logic and applications* (pp. 218–223). <https://doi.org/10.1109/FPL.2009.5272311>.
9. Garrido, M. (2016). A new representation of FFT algorithms using triangular matrices. *IEEE Transactions on Circuits and Systems I*, 63(10), 1737–1745. <https://doi.org/10.1109/TCSI.2016.2587822>.
10. Garrido, M., Huang, S.J., Chen, S.G., Gustafsson, O. (2016). The serial commutator FFT. *IEEE Transactions on Circuits and Systems II*, 63(10), 974–978. <https://doi.org/10.1109/TCSII.2016.2538119>.
11. Garrido, M., Qureshi, F., Takala, J., Gustafsson, O. (2018). Hardware architectures for the fast Fourier transform. In Bhattacharyya, S.S., Deprettere, E.F., Leupers, R., Takala, J. (Eds.) *Handbook of signal processing systems*, 3rd edn: Springer.
12. Gold, B., & Bially, T. (1973). Parallelism in fast Fourier transform hardware. *IEEE Transactions on Audio and Electroacoustics*, 21(1), 5–16. <https://doi.org/10.1109/TAU.1973.1162428>.
13. Groginsky, H.L., & Works, G.A. (1970). A pipeline fast Fourier transform. *IEEE Transactions on Computers*, C-19(11), 1015–1019. <https://doi.org/10.1109/T-C.1970.222826>.
14. He, S., & Torkelson, M. (1996). A new approach to pipeline FFT processor. In *Proceedings of the international parallel processing symposium* (pp. 766–770). <https://doi.org/10.1109/IPPS.1996.508145>.
15. Ingemarsson, C., Källström, P., Gustafsson, O. (2012). Using DSP block pre-adders in pipeline SDF FFT implementations in contemporary FPGAs. In *Proceedings of the international conference on field-programmable logic and applications* (pp. 71–74). <https://doi.org/10.1109/FPL.2012.6339243>.
16. Ingemarsson, C., Källström, P., Qureshi, F., Gustafsson, O. (2017). Efficient FPGA mapping of pipeline SDF FFT cores. *IEEE Transactions on VLSI Systems*, 25(9), 2486–2497. <https://doi.org/10.1109/TVLSI.2017.2710479>.
17. Li, J., Liu, F., Long, T., Mao, E. (2009). Research on pipeline R²SDF FFT. In *Proceedings of the IET international radar conference* (pp. 1–5). <https://doi.org/10.1049/CP.2009.0174>.
18. Liu, X., Yu, F., Wang, Z. (2011). A pipelined architecture for normal I/O order FFT. *Journal of Zhejiang University-Science C*, 12(1), 76–82. <https://doi.org/10.1631/JZUS.C1000234>.
19. Ma, Z., Yu, F., Ge, R., Wang, Z. (2011). An efficient radix-2 fast Fourier transform processor with ganged butterfly engines on field programmable gate arrays. *Journal of Zhejiang University-Science C*, 12(4), 323–329. <https://doi.org/10.1631/JZUS.C1000258>.
20. Qureshi, F., & Gustafsson, O. (2011). Generation of all radix-2 fast Fourier transform algorithms using binary trees. In *Proceedings of the European conference on circuit theory and design* (pp. 677–680).
21. Wang, Z., Liu, X., He, B., Yu, F. (2015). A combined SDC-SDF architecture for normal I/O pipelined radix-2 FFT. *IEEE Transactions on VLSI Systems*, 23(5), 973–977. <https://doi.org/10.1109/TVLSI.2014.2319335>.
22. Wanhammar, L. (1999). *DSP integrated circuits*. New York: Academic.
23. Wold, E.H., & Despain, A.M. (1984). Pipeline and parallel-pipeline FFT processors for VLSI implementations. *IEEE Transactions on Computers*, C-33(5), 414–426. <https://doi.org/10.1109/TC.1984.1676458>.
24. Yang, L., Zhang, K., Liu, H., Huang, J., Huang, S. (2006). An efficient locally pipelined FFT processor. *IEEE Transactions on Circuits and Systems II*, 53(7), 585–589. <https://doi.org/10.1109/TCSII.2006.875306>.

25. Zhong, G., Zheng, H., Jin, Z., Chen, D. (2011). Pang, Z.: 1024-point pipeline FFT processor with pointer FIFOs based on FPGA. In *Proceedings of the IEEE/IFIP international VLSI system-on-chip conference* (pp. 122–125). <https://doi.org/10.1109/VLSISOC.2011.6081654>.
26. Zhou, B., Peng, Y., Hwang, D. (2009). Pipeline FFT architectures optimized for FPGAs. *International Journal of Reconfigurable Computing*, 2009, 1–9. <https://doi.org/10.1155/2009/219140>.

Carl Ingemarsson received the M.Sc. degree from Linköping University, Linköping, Sweden in 2009 where he is currently pursuing his studies toward a Ph.D. degree at the Computer Engineering Division, Department of Electrical Engineering.

Mr. Ingemarsson's research interests focus on hardware implementation of digital signal processing and arithmetic circuits.

Oscar Gustafsson received the M.Sc., Ph.D., and Docent degrees from Linköping University, Linköping, Sweden, in 1998, 2003, and 2008, respectively.

He is currently an Associate Professor and Head of the Computer Engineering Division, Department of Electrical Engineering, Linköping University. His research interests include design and implementation of DSP algorithms and arithmetic circuits. He has authored and co-authored over 140 papers in international

journals and conferences on these topics.

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