FPGA Implementation of a Multi-Channel Continuous-Throughput FFT Processor

Elia Fankhauser
Intelligent Sensors and Networks Laboratory (ISN)
Lucerne University of Applied Sciences and Arts (HSLU)
Lucerne, Switzerland
elia.fankhauser@hslu.ch

Jürgen Wassner
Intelligent Sensors and Networks Laboratory (ISN)
Lucerne University of Applied Sciences and Arts (HSLU)
Lucerne, Switzerland
juergen.wassner@hslu.ch

***Abstract*—The Fast Fourier Transform (FFT) is at the heart of many signal processing systems, e.g. those using orthogonal frequency-division multiplexing (OFDM) modulation. For such systems the FFT is typically implemented on dedicated hardware, e.g. field-programmable gate arrays (FPGAs), to meet the high throughput and latency requirements. This paper describes a 2-parallel radix-2 FFT core embedded into an overall processing architecture that allows data interfaces to be tailored to application-specific needs. The proposed solution can dynamically switch between different FFT sizes while maintaining its maximum theoretical throughput. Our experiments show that a two-fold increase in throughput can be achieved without doubling resource usage, when taking into account FPGA-specific features for several optimizations.**

***Index Terms*—FFT, FPGA, Throughput, Butterfly**

I. Introduction

The discrete Fourier transform (DFT) is a core component of wireless and wire-based broadband digital communication systems that use orthogonal frequency-division multiplexing (OFDM) modulation, e.g. mobile radios (LTE), digital television (DTV), and power-line communication (PLC). Due to the increasing throughput requirements of such systems, an efficient hardware implementation of the one-dimensional DFT

$$X[k] \;=\; \sum_{n=0}^{N-1} x[n] e^{-j2\pi n \frac{k}{N}} \;, \quad k = 0, 1, \ldots, N-1 \qquad (1)$$

is of ongoing interest.

Existing work in this domain can be broadly divided into approaches based on systolic arrays [1] and those using variants of the fast Fourier transform (FFT) algorithm [2] [3].

In the former category mainly two-dimensional (2-D) systolic array architectures are of interest, because only those can be utilized to reduce the computational complexity of (1) from $\mathcal{O}(N^2)$ to $\mathcal{O}(N(N_1 + N_2))$ by factoring $N = N_1 N_2$. According to the results in [1], such 2-D systolic arrays achieve high throughput with comparable hardware cost up to $N = 2048$, but do not scale well for larger DFT sizes, such as those required by our target application, see section II-A. This non-linear scaling of 2-D systolic arrays is evaded by the divide-and-conquer approach of the FFT, which reduces the computational complexity to $\mathcal{O}(N \log_2(N))$. Since the constraint on N to be a power of two is feasible for our application, we focus on FFT algorithms.

In the context of OFDM, several architectures for a pipelined FFT processor have been proposed already in [4], but none of them meets the throughput requirements of modern systems. Most of the existing work on FFT implementation targets certain efficiency metrics but neglects application data input and output interfaces. Our work deliberately includes such interfaces into the design.

A low-complexity FFT processor was suggested in [5] to reduce power consumption at the cost of higher latency. The control scheme suggested with this processor infers stall cycles when dynamically changing the FFT length, which is detrimental to latency and throughput. Radix-2^k architectures for fixed-length FFT were studied in [6]. Such architectures provide very good throughput but result in high hardware costs, which might not be suitable for FPGA implementation. The same is true for the highly-parallel architecture suggested in [7]. A real-time reconfigurable processor was proposed in [8], which only covers FFTs with $\log_2(N) \bmod 2 = 0$ because of its radix-4 architecture.

With respect to digital noise produced by scaling and rounding, the present paper builds on [9], which studied the effect of a decimation-in-time (DIT) vs. a decimation-in-frequency (DIF) decomposition. The authors of [9] conclude that with optimal rounding, DIF produces less noise than DIT, and that DIF results in less noise for frequency indexes $N/2 \leq k \leq N-1$, while DIT features a noise spectrum symmetrical around $k = N/2 - 1$.

This paper describes a multi-channel FFT processor that optimally utilizes time-shared FPGA resources. With regard to previously proposed FFT architectures our work is closest to [7] and [8], but provides the following new contributions:

1) A multi-channel FFT processor that can be tailored to application-specific interface requirements.
2) A decentralized control scheme for radix-2 FFTs that supports dynamically changing N without stall cycles.
3) An implementation of butterfly operations optimized for FPGA-specific hardware resources.
4) A method for efficient twiddle factor storage with on-the-fly resolution enhancement.

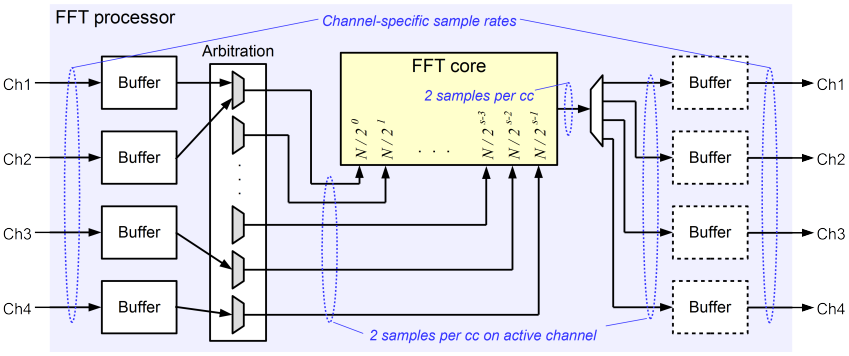

Fig. 1. Architecture of the FFT processor. Input data is buffered per channel and forwarded under priority control to the corresponding input of the general-purpose FFT core. Result data is multiplexed to the corresponding channel. Optional output buffers are used for data reordering and sample rate adaptation.

The remainder of the text is organized as follows. In section II the overall architecture of the FFT processor is derived from the target application. The generic FFT core is then explained top-down in section III, providing details on several optimizations that have been performed. These optimizations are experimentally evaluated with respect to latency, digital noise and hardware cost in section IV and conclusions are drawn in section V.

II. Application-specific FFT Processor

A. Target Application and Problem Statement

[11] specifies two types of OFDM physical layer procedures for broadband communication over power line networks in smart grid, transportation and in-home applications. In situations where strict requirements regarding electro-magnetic emissions must be met, the FFT-based physical layer is preferred over wavelet OFDM since it allows the realization of deep frequency notches without additional transmit filters.

The FFT physical layer version of [11] requires three FFT instances: A combined 4096/512-point inverse FFT (IFFT) for generation of modulated transmit symbols and preamble mini-symbols, and two separate 4096- and 512-point FFTs for receive symbol demodulation. All of these are working with the baseband PLC signal. For the specific low-latency FPGA implementation of [11] we used four different instances of the commercially available FFT core [12] with a target clock frequency of 400 MHz. This core serves as a reference point in the remainder of this text. From the protocol point of view, all four FFT instances could time-share the same hardware in our target system as they are guaranteed to not be used at the same time during all stages of transmission and reception. However, since the processing pipeline of [12] must be flushed every time the FFT length N is changed, throughput drops and latency increases beyond our system requirements. We therefore opted for a solution that optimally utilizes hardware resources time-shared between FFT channels.

Although our starting point was PLC as defined in [11], the proposed solution is applicable to any system that requires multiple FFTs or IFFTs of equal or different length N to be performed with low latency on various data streams in a time-multiplexed fashion.

B. FFT Processor Architecture

Fig. 1 shows the top-level architecture of the proposed solution, which consists of a general-purpose FFT core and the surrounding FFT processor infrastructure.

The FFT processor functionality can be statically configured for the targeted application. For our particular OFDM system we used four channels, but the general concept has no such limitation. Each of the channels can be statically configured to any FFT length N supported by the FFT core. This approach assumes that the different FFT lengths required are known a priori, but maintains the maximum FFT core throughput when changing processing from one FFT length to another. This is conceptually different to [12], where the FFT length can be dynamically changed, but any such change incurs stall cycles and a drop in throughput.

As indicated in Fig. 1, the selected FFT core architecture supports a continuous throughput of two complex-valued samples per clock cycle, which is twice the theoretical throughput of the reference core [12]. The choice of a two-parallel FFT core architecture resulted from a trade-off analysis, which revealed that the linearly increasing hardware cost of higher FFT parallelization is not justified in our OFDM application, because system-level latency would only marginally decrease. Although the particular FFT architecture has been selected based on the requirements of our target application, the FFT core can be reused in other applications, either stand-alone (for single-channel applications) or by making use of the FFT processor infrastructure (for multi- and single-channel applications).

The FFT processor employs input buffers to ensure that once the FFT core has started to process a particular channel, two complex-valued input samples per clock cycle from this channel can be provided to the core. With this method the processing of each channel can be started at the optimal moment in time. This optimal time is the moment when the minimum fill level of the input buffer required for continuous core throughput, independent of the data rate of the specific channel, is reached. The arbitration between channels with different N is handled by the de-centralized control scheme explained in section III-B.

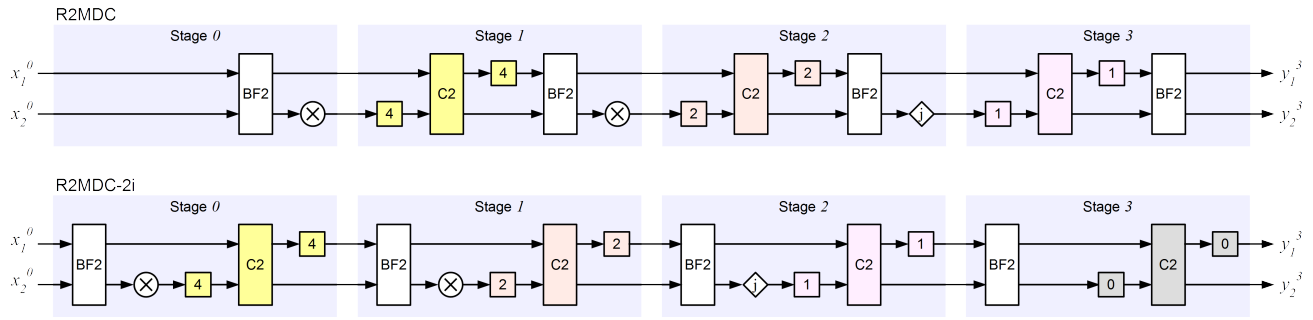

Fig. 2. Comparison of the 2-parallel R2MDC (top) and the proposed R2MDC-2i (bottom) architecture for $N = 16$. Moving criss-cross elements C2 to the preceding stage prepares for the localized control concept required for continuous throughput. The dummy element C2 added to the last stage does not consume any hardware resources.

When two or more channels are configured for the same FFT length, a dedicated input buffer for each channel is used. These channels are then arbitrated to the same core input. Assuming that the channel input sample rate is high enough, the FFT processor architecture ensures that the maximum theoretical throughput can be maintained at the FFT core output at all times.

Output buffers can be optionally instantiated per channel and used for reordering the bit-reversed FFT core output and/or sample rate adaptation. A channel-specific ID is propagated through the FFT core to allow proper multiplexing at the output buffers.

III. General-purpose FFT Core

A. Pipelined FFT Core Architecture

Starting off from the 8-parallel radix-2 multi-path delay commutator (R2MDC) architecture proposed in [7], two modifications were introduced, which allow the target throughput of two samples per clock cycle to be maintained even when changing the FFT length N. First, the 8-parallel R2MDC architecture from [7] has been scaled down to the 2-parallel architecture shown in Fig. 2 (top), which realizes the target throughput. However, its irregular structure across the $\log_2(N)$ stages excludes a generic localized control concept for each stage required to maintain maximum throughput as explained in section III-B. Second, by moving each criss-cross commutator C2 into the preceding stage as shown in Fig. 2 (bottom), such a control concept becomes feasible. The dummy element C2 added to the last stage does not consume any hardware resources.

The latency $L(N)$ of the 2-parallel architecture is

$$L(N) = \frac{N}{2} + L_S \cdot log_2(N) + L_W \tag{2}$$

because $N/2$ clock cycles are required to input all N values, the last of which has to pass through all $log_2(N)$ stages for the first result value $X[0]$ to become available at the output. L_S and L_W are constant latency values given by the pipelining depth of the data path within each stage and the first twiddle factor generation unit, respectively. In our particular implementation we used $L_S = 6$ and $L_W = 8$.

B. FFT Stage Architecture with Decentralized Control

Fig. 3 shows a more detailed diagram of a single stage from Fig. 2. Each stage $s = 0, \ldots, \log_2(N) - 1$ of the FFT core includes a local controller of identical structure which de-multiplexes input data from either of two sources to the butterfly processing block (BF2), and directs the twiddle factor generator and criss-cross element. The two data sources are drawn as vertical and horizontal inputs in Fig. 3.

The vertical input of stage s is used by the surrounding FFT processor to provide data for a specific FFT length $N/2^s$. For this, each stage controller handshakes via request and ready signals with the corresponding input buffer of the FFT processor.

The horizontal input of stage s is used by the upstream stage $s-1$ to provide pre-processed data for further processing. Such processing must be requested at the local controller of stage s precisely

$$T_{\mathrm{LA}}(s) = \frac{N}{2^{s+1}} + L_W \tag{3}$$

clock cycles in advance. This look-ahead time $T_{\mathrm{LA}}(s)$ allows stage controller s to operate its ready signal for the vertical data input, such that FFT frames from both inputs can be processed back-to-back without idle cycles for the local butterfly and criss-cross elements. It also allows stage controller s to prevent the start of a new vertical-input FFT, which would collide with future horizontal input data. In contrast to vertical interfaces, there is no need for horizontal ready signals, since requests from upstream stages are always handled with higher priority then external requests.

Using this look-ahead signaling scheme, the chain of local stage controllers optimally orchestrates data flow through the FFT core without any central control instance. It avoids flushing the processing pipeline when dynamically changing the FFT length N as in [12], and always maintains the maximum theoretical throughput at the FFT core output, provided that the input buffers of the FFT processor do not run empty.

C. Butterfly Structure Optimizations

There are two major types of radix-2 butterfly operations, the decimation-in-frequency (DIF) and the decimation-in-time (DIT) butterfly structure [2]. While both combine two

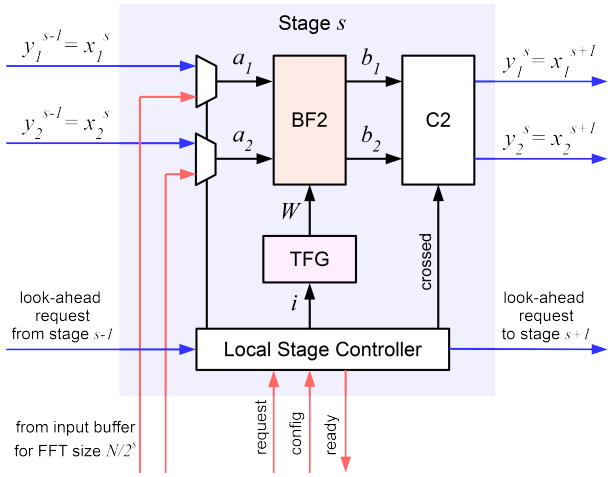

Fig. 3. Generic FFT stage architecture with local controller, de-multiplexing vertical and horizontal data inputs to the radix-2 butterfly (BF2), directing the twiddle factor generator (TFG) and the criss-cross element (C2).

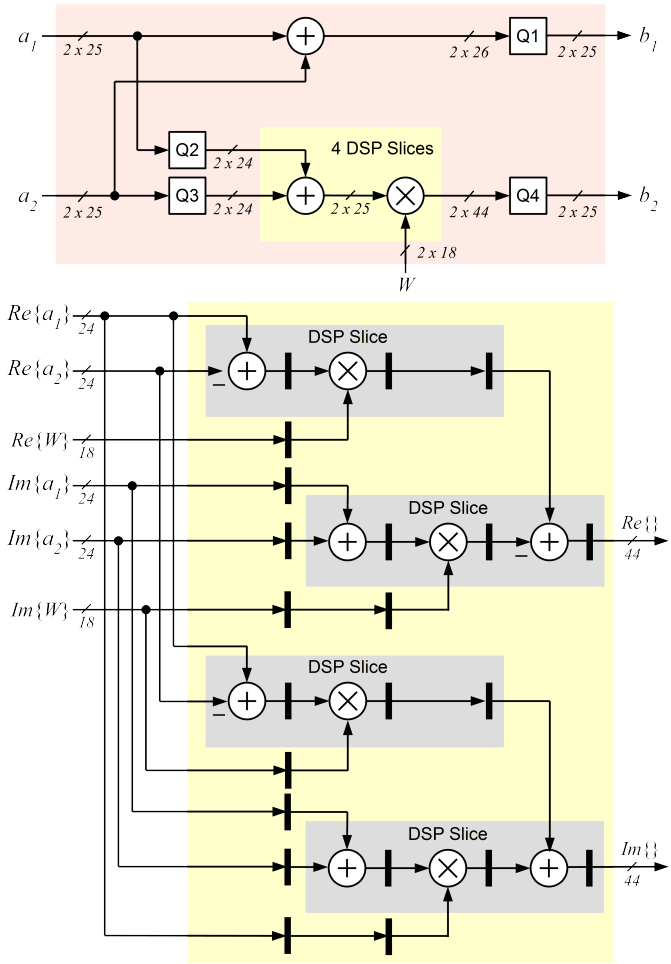

Fig. 4. Radix-2 decimation in frequency (DIF) butterfly with one complex-valued addition and multiplication mapped to DSP slices and quantization blocks Q1...Q4 (top). Configuration of the four DSP slices required per butterfly maximizing pre-adder utilization (bottom).

complex-valued data inputs into two complex-valued outputs by means of two additions/subtractions and one multiplication with a twiddle factor, they differ in the order of these operations. The DIF butterfly performs multiplication for one of the outputs only, whereas the DIT butterfly multiplies one of its inputs with the twiddle factor.

This difference in the multiplication order results in different digital noise propagation behavior of DIF and DIT, since unavoidable product re-quantization is the main source of digital noise. The DIF butterfly as shown in Fig. 4 (top) has been selected because it introduces less digital noise [9]. Re-quantization blocks Q1...Q4 are systematically placed in order to make optimal use of the given multiplier word width within the DSP slices of the target FPGA technology. The effect of these quantization blocks on digital noise is investigated in section III-D.

Another advantage of the DIF butterfly is that the pre-adder present in every DSP slice can be utilized for one of the complex addition operations, thus saving FPGA fabric resources. This is not possible with the DIT butterfly, because the complex addition is performed after the multiplication, with the post-adder of two DSP slices already used for re-combination within real and imaginary parts. Following this scheme, Fig. 4 (bottom) shows how the arithmetic logic of the complex multiplication and addition of the DIF butterfly can be implemented solely using DSP slice resources. As indicated, some fabric pipeline registers are required for proper timing.

[10] proposed to implement a complex multiplication with three multipliers. This approach has not been used, because the order of operations required is unfavorable with respect to digital noise and also does not allow to map all adder logic to DSP slices.

D. Digital Noise Reduction

A base word width of $B = 25$ bits has been selected for the butterfly arithmetic as shown in Fig. 4 (top). Both butterfly input and output data is quantized with B bits and the radix point is scaled such that no overflow occurs. The value of B has been selected based on the target DSP slice architecture which supports a 18x25-bit multiplication with full precision. Since the pre-adder used in each of the DSP slices increases the word width by one bit, both inputs of the DSP slice are quantized by Q2 and Q3 to $B-1$ bits. Although it would be possible to save Q2 and Q3 and work with butterfly inputs and outputs of $B-1$ bits, a 3 dB drop in the signal-to-quantization-noise ratio (SQNR) was found in this scenario. Subsequently, Q1 and Q4 quantize the butterfly outputs to B bits.

For each of the four quantization blocks two different methods for decreasing the word width have been considered: truncation (i.e. rounding towards $-\infty$) and rounding towards nearest (ties round towards $+\infty$ and $-\infty$ for positive and negative numbers, respectively). While truncation infers no hardware cost, true rounding is expected to provide more accurate results.

To evaluate the effect of rounding and truncation on digital noise, a bit-accurate simulation model of the fixed-point butter-

TABLE I
SQNR FOR DIFFERENT ROUNDING SCHEMES IN QUANTIZATION BLOCKS Q1...Q4, RANDOM AND OFDM INPUT SIGNALS, AND 16/18-BIT RESOLUTION FOR TWIDDLE FACTORS AND INPUT SIGNALS.

ID	Rounding Method		SQNR [dB]			
			Random		OFDM	
	floor()	round()	16 bit	18 bit	16 bit	18 bit
R1	Q1,2,3,4		84.9	91.3	83.2	85.8
R2	Q2,3,4	Q1	85.5	93.6	85.2	88.3
R3	Q1,2,4	Q2	85.6	91.7	83.9	86.0
R4	Q1,2,4	Q3	85.1	90.5	83.1	84.9
R5	Q1,2,3	Q4	85.3	93.4	84.2	89.0
R6	Q2,3	Q1,4	86.1	94.8	85.5	89.5
R7		Q1,2,3,4	87.1	92.6	86.0	85.9

fly arithmetic described in section III-C has been implemented, and the SQNR defined as

$$\text{SQNR [dB]} = 10 \cdot \log_{10} \left(\frac{\sum_{i=0}^{N-1} |X_{\text{flt}}[i]|^2}{\sum_{i=0}^{N-1} |X_{\text{flt}}[i] - X_{\text{fix}}[i]|^2} \right) \quad (4)$$

is used as performance metric. FFT outputs X_{flt} and X_{fix} were obtained from double floating-point precision and bit-accurate fixed-point simulation, respectively.

Table I summarizes the results obtained for $N = 4096$. First, all quantization blocks are set to truncation. Then, each block is set to rounding mode individually. Finally, all blocks are set to rounding mode jointly. In each case, the SQNR is obtained for two different real-valued input signals with $-3\,\text{dBFS}$, i.e. random uniform noise and a typical OFDM symbol with a peak-to-average power ratio (PAPR) of $12\,\text{dB}$. As indicated in Table I, both input signals and twiddle factors W have been simulated with 16- and 18-bit resolution. These values were chosen in order to evaluate the trade-off between increased hardware cost for 18-bit twiddle factors and fully utilizing the second input of the DSP slice multiplier.

The results in Table I show that 18-bit twiddle factors significantly increase SQNR by about 6 to $8\,\text{dB}$ for random noise input compared to 16-bit twiddle factors. For OFDM signals the SQNR gain is less. This is due to the high PAPR of OFDM signals, which prevents a full-scale butterfly input for almost all samples, and thus makes the additional twiddle factor resolution less effective. It has been found that the best SQNR would be achieved for 18-bit twiddle factors with Q1/Q4 set to rounding, and Q2/Q3 set to truncation.

Furthermore, we found the same noise characteristic for DIF as seen in [9], when using truncations for Q1...Q4. However, with Q1 set to rounding, a noise spectrum symmetrical around carrier index $k = N/2 - 1$ was observed, which was only achieved for DIT in [9].

E. Twiddle Factor Generation

A dedicated twiddle factor generator (TFG) is connected to the butterfly unit in each stage, as shown in Fig. 3. Of the $N/2$ twiddle factors

$$W_N^i = e^{-j\frac{2\pi}{N}i}, \quad i = 0, 1, \ldots, N/2 - 1 \quad (5)$$

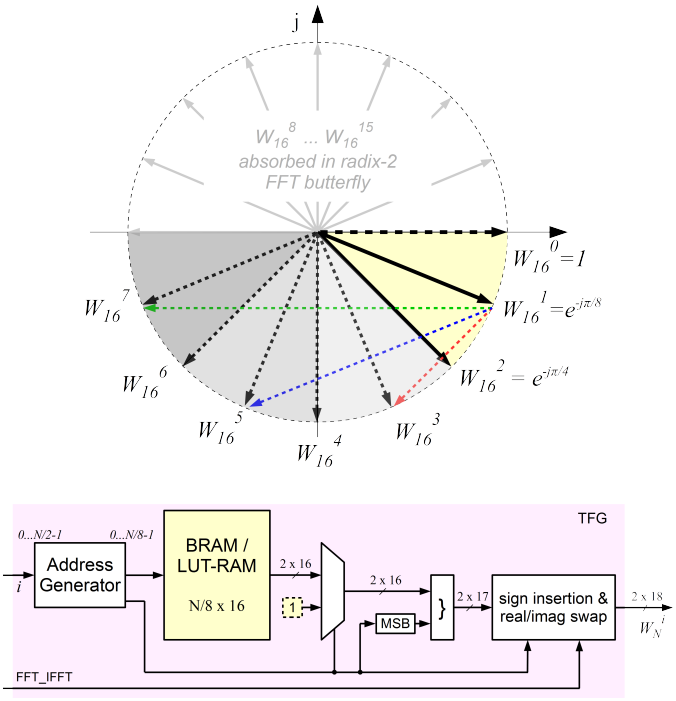

Fig. 5. Principle of twiddle factor generation for $N = 16$ (top). Only the $N/8 + 1 = 3$ twiddle factors from the yellow area must be stored. Twiddle factors in the three gray areas are derived from stored values by swapping real/imaginary parts with sign inversion on both (red), swapping real/imaginary parts with sign inversion of real part only (blue), or simple sign inversion for the real part only (green). To optimize block-RAM usage, only the 16 LSBs of the real and imaginary part of $N/8$ twiddle factors are stored, while additional MSBs are appended on the fly to produce twiddle factors with higher resolution (bottom).

required for calculation in the stage corresponding to N, only the value of the real and imaginary part of $N/8 + 1$ twiddle factors must be known. In particular, the real and imaginary parts of $W_N^1, \ldots, W_N^{N/8}$ are stored in a look-up table. The trivial value of $W_N^0 = 1$ is hard-coded in order to make the number of entries in the look-up table a power of two, and thus make efficient use of all addressable memory resources. All other twiddle factors can then be deduced from those $N/8 + 1$ numerical values by means of appropriate addressing, real/imaginary-part swapping, and/or sign inversion. This process is exemplified in the top part of Fig. 5 for $N = 16$. The other concept used to ensure better utilization of BRAM resources is shown in the bottom part of Fig. 5. The $N/8+1$ values are stored with a 16-bit resolution, and then are extended on the fly by two additional bits in order to achieve a total resolution of 18 bits; thereby fully utilizing the 18x25-bit DSP slice multiplier. An additional sign inversion is performed on the imaginary part, if IFFT-mode is active.

Alternatively, CORDIC functions could be used for twiddle factor generation. This would require less memory resources at the cost of more algebraic operations and higher latency, L_W, which according to (2) would also slightly increase the overall latency, L.

TABLE II
DIFFERENCE IN LATENCY AND FPGA RESOURCE USAGE BETWEEN REFERENCE CORE [12] AND THE PROPOSED FFT CORE WITH COMPARABLE SQNR FOR $N = 64$ AND $N = 4096$.

	N	FFT [12]	FFT core	Diff [%]
SQNR [dB]	64	95.3	95.4	$\approx +0$
	4096	82.3	85.8	$\approx +0$
Latency [cc]	64	161	76	-53
	4096	4256	2128	-50
Throughput [value/cc]	64	1	2	$+50$
	4096	1	2	$+50$
DSP slices	64	12	16	$+33$
	4096	30	40	$+33$
BRAM [2KB]	64	0	0	/
	4096	17	20	$+18$
LUT-RAM	64	591	387	-35
	4096	1299	1421	$+9$
LUT	64	1342	1488	$+11$
	4096	3001	3441	$+15$
Flip-Flops	64	3460	3266	-6
	4096	7460	7317	-2

IV. EVALUATION ON FPGA

The FFT processor and core concepts described above have been implemented in VHDL and integrated into a Xilinx xc7z045-2 device for a target clock frequency of 400 MHz. The twiddle factors are stored with 16-bit resolution and extended to 18 bits on the fly in order to fully utilize DSP slice multipliers. Rounding scheme R1 from Table I has been used. The VHDL implementation has been verified using cycle-true and bit-accurate Matlab/Simulink models based on fixed-point numbers. The continuous-throughput feature has been tested by simulating multiple FFTs of different length back-to-back.

Table II compares the IP-core [12] with our proposed FFT core. All latency and SQNR results were obtained by simulation. The hardware resource numbers have been taken from the FPGA utilization reports of 400 MHz implementation runs. For fair comparison of hardware resource usage, [12] has been configured to achieve a slightly lower SQNR as the proposed FFT core.

As expected, our proposed approach reduces latency by a factor of two through its 2-parallel R2MDC architecture. Additionally, the decentralized control scheme guarantees the maximum theoretical throughput of two complex values per clock cycle even when changing between different FFT lengths. As can be seen in Table II, the halving of latency and the continuous-throughput feature have been achieved without doubling the utilization of any FPGA resource type. In particular, DSP utilization only increases by one third because of the butterfly optimizations explained in section III-C. The increase in combinational LUT usage is even less. This is due to the mapping of almost all butterfly logic to DSP-slices, which partly compensates for additional control and twiddle factor generation logic. For $N = 4096$, the BRAM and LUT-RAM usage of the proposed solution is slightly higher. This is because the decentralized control concept requires local storage for criss-cross elements C2 and twiddle factors, which is difficult to share between stages. For smaller N, sharing is less important, because no BRAMs of fixed size are used. In this case, the proposed twiddle factor generation concept requires less LUT-RAMs than [12].

V. CONCLUSION

It is feasible to double the throughput and cut the latency of state-of-the-art FFT FPGA-implementations by half, without doubling hardware resource utilization and without compromising on the signal-to-quantization-noise ratio. This requires careful selection of the FFT core architecture and several low-level optimizations such as butterfly reorganization for optimal DSP slice mapping and efficient twiddle factor generation. Using a decentralized control scheme, the proposed general-purpose FFT core can be embedded into a multi-channel processor architecture that maintains the maximum theoretical throughput, even when the FFT length is changed.

Future work will include BRAM optimizations by means of dual-port memories. Further investigations are required to explain the difference between our results and [9] regarding the spectral characteristics of digital noise.

ACKNOWLEDGMENT

The authors would like to thank Dr. David Perels for fruitful discussions and valuable comments on the topic of OFDM system implementation. The research leading to these results has partially received funding from the Swiss Hasler Foundation.

REFERENCES

[1] J. G. Nash, "Computationally efficient systolic architecture for computing the discrete Fourier transform," IEEE Transactions on Signal Processing, vol. 53, issue 12, December 2005.
[2] Richard G. Lyons, "Understanding digital signal processing," 3rd ed, Prentice Hall, 978-0-13-702741-5, August 2011, pp129-156.
[3] E. E. Jr. Swartzlander, "Systolic FFT processors: past, present and future," IEEE 17th International Conference on Application-specific Systems, Architectures and Processors, pp. 153-158, Sept. 2006.
[4] Shousheng He, and M. Torkelson, "Designing pipeline FFT processor for OFDM (de)modulation," 1998 URSI International Symposium on Signals, Systems, and Electronics. Conference Proceedings (Cat. No.98EX167), September 1998.
[5] Q. Lu, X. Wang, and J. Niu, "A low-power variable-length FFT processor base on radix-2^4 algorithm," Asia Pacific Conference on Postgraduate Research in Microelectronics Electronics (PrimeAsia), January 2009.
[6] M. Garrido, J. Grajal, M. A. Sanchez, and O. Gustafsson, "Pipelined radix-2^k feedforward FFT architectures," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 21, issue 1, January 2013.
[7] S. Mookherjee, L. DeBrunner, and V. DeBrunner, "A low power radix-2 FFT accelerator for FPGA," 49th Asilomar Conference on Signals, Systems and Computers, November 2015.
[8] M. Hasan, T. Arslan, and J. S Thompson, "A delay spread based low power reconfigurable FFT processor architecture for wireless receiver," 2003 International Symposium on System-on-Chip (IEEE Cat. No.03EX748), November 2003.
[9] I. Szolik, K. Kovac V, and Smiesko, "Influence of digital signal processing on precision of power quality parameters measurement, "Measurement Science Review, vol. 3, section 1, 2003.
[10] M. Hemnani, S. Palekar, P. Dixit, and P. Joshi, "Hardware optimization of complex multiplication scheme for DSP application," 2015 International Conference on Computer, Communication and Control (IC4), September 2015.
[11] "IEEE 1901-2010 -Standard for broadband over power line networks: medium access control and physical layer specifications," 2011.
[12] Xilinx, "Fast Fourier transform v9.0, logic core ip product guide," https://www.xilinx.com/support/documentation/ip_documentation/xfft/v9_0/pg109-xfft.pdf, October 2017.