AREA-DELAY EFFICIENT FFT ARCHITECTURE USING PARALLEL PROCESSING AND NEW MEMORY SHARING TECHNIQUE

Yousri Ouerhani, Maher Jridi, Ayman Alfalou

▶ **To cite this version:**

Yousri Ouerhani, Maher Jridi, Ayman Alfalou. AREA-DELAY EFFICIENT FFT ARCHITECTURE USING PARALLEL PROCESSING AND NEW MEMORY SHARING TECHNIQUE. Journal of Circuits, Systems, and Computers, 2012, 21 (6), 1240018 (15 p.). hal-00782711

HAL Id: hal-00782711
https://hal.science/hal-00782711

Submitted on 31 Jan 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Journal of Circuits, Systems, and Computers
© World Scientific Publishing Company

AREA-DELAY EFFICIENT FFT ARCHITECTURE USING PARALLEL PROCESSING AND NEW MEMORY SHARING TECHNIQUE

YOUSRI OUERHANI, MAHER JRIDI and AYMAN ALFALOU

Equipe Vision, laboratoire L@bISEN de ISEN-Brest
20 Rue Cuirasse Bretagne, CS 42807, 29228 Brest Cedex 2, France.

In this paper we present a novel architecture for FFT implementation on FPGA. The proposed architecture based on radix-4 algorithm presents the advantage of a higher throughput and low area-delay product. In fact, the novelty consists on using a memory sharing and dividing technique along with parallel-in parallel-out Processing Elements (PE). The proposed architecture can perform N-point FFT using only 4/3N delay elements and involves a latency of N/4 cycles. Comparison in terms of hardware complexity and area-delay product with recent works presented in the literature and commercial IPs has been made to show the efficiency of the proposed design. Moreover, from the experimental results obtained from a FPGA prototype we find that the proposed design involves an execution time of 56% lower than that obtained with Xilinx IP core and an increase of 19% in the throughput by area ratio for 256-point FFT.

Keywords: Digital hardware implementation; VLSI; embedded signal processing.

1. Introduction

The Discrete Fourier Transform (DFT) is one of the most important tools used in Digital signal and image processing applications. It has been widely implemented in digital communication systems such as Radars, Ultra Wide Band (UWB) receivers and many other image processing applications. The direct realization of this algorithm using N-sample input, requires a large number of operations (N^2 complex multiplications and $N(N-1)$ complex additions). Since the DFT algorithm is computation-intensive, several improvements have been proposed in literature for computing it efficiently and rapidly. To reduce the number of operations a fast algorithm has been introduced by Cooley-Tukey[1] and called Fast Fourier Transform (FFT). The latter, consists on decomposing DFT computing into small building blocks called radix-2 by using efficiently the symmetry and the periodicity of the twiddle factors. This decomposition reduces complexity from $O(N^2)$ to $O(NlogN)$. Since the work of Cooley-Tukey, several algorithms have been proposed to further reduce computational requirement including radix-4 [2], split radix [3], prime factor [4].
Due to the fact that radix-based FFT algorithms divide the computation into odd- and even-half parts recursively [5], many block RAM are required to save these intermediate data.
FFT algorithm can be implemented on multiple software platforms including General Purpose Processor (GPPs) and Digital Signal Processors (DSPs) and in hardware circuits such as Field Programmable Gate Arrays (FPGAs) and Application Specific Integrated Circuits (ASICs). FPGA design of FFT is often tailored to fit high speed on low-power specification due to the fact that FPGAs have grown in capacity and performance and

decreased in cost.
In literature, several architectures for implementation FFT on FPGA have been proposed in order to improve speed and reduce the high memory usage. It is found that there are two main implementations of FFT on FPGA: memory-based design and pipelined architectures. Memory-based FFT uses only one butterfly and large memories for data storage. However, pipelined architecture uses many butterflies to improve speed. Several architectures have been proposed to implement both method (memory-based design and pipelined design) in order to improve speed and minimize area.
In[6] authors present an FPGA implementation and a comparison of six memory-based architectures. The authors give the name of RX2-B1, RX4-B1, RX2-B2, RX4-B2, RX2-B4, MXRX-B4 for these architectures. Where the RX presents the radix used and Bi indicate that the architecture presents i outputs in parallel. The techniques used for each architecture are based on memory sharing[7], Conflict free memory addressing[8], In-place memory processing[9 10], Continuous flow design[10 11], N-word memory size, Fixed-point arithmetic and Pre-computed twiddle factors stored in ROM. It was found that the fastest processors are the RX2-B4 and MXRX-B4 processors which can process four data samples per clock cycle.
Regarding the pipelined architectures, many works have presented optimizations to achieve high performance and low area occupation. The famous architectures of the pipeline implementation are Multi-path Delay Commutator (R2MDC)[12], Radix-2 Single-Path Delay Feedback [13], Radix-4 Single-Path Delay Commutator[14], and Radix-2^2 Single-Path Delay Feedback (R2^2SDF)[15]. The main difference between these architectures is about the number of inputs and outputs and the butterfly used. In[16], an optimized implementations of two different pipelined FFT processors are proposed and validated on Xilinx Spartan-3 and Virtex-4 FPGAs.
Although there have been several efficient designs, there is an inherent drawback to existing studies related to the area (and consequently power) overhead.
In [17], we presented an optimized architecture for low cost FPGA. The architecture proposed is based on modified radix-4 architecture and sharing memory between different blocks. In this paper, we present an extended work related to the optimization of this architecture to improve speed and throughput and to minimize the consumed silicon area.
This paper is organized as follows. In section II, definition and architecture of N-point FFT based on radix-4 algorithm are introduced. Section III is devoted to the proposed architecture. We detail the principle of sharing memories between different stages and the structure of N-point FFT. Next, Section IV presents the complexity of the proposed architecture. Section V shows the implementation results and comparison with prior works. Finally, we summarize and conclude this paper in section VII.

2. The FFT algorithm

For a given sequence x of n samples, the DFT frequency components $X(k)$ may be defined by Eq. (1)

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk} \tag{1}$$

where $W_N = e^{\frac{-2j\pi}{N}}$ are the twiddle factor, n and k are respectively the time and frequency indexes, $0 \leq k \leq$ N-1 , $0 \leq n \leq$ N-1 and N is the DFT length.

2.1. *Radix-4 algorithm*

It is obvious that the direct realization of N-point DFT in hardware device is inefficient. To overcome this drawback we can use the principle of decomposing the FFT into sequences of smaller FFTs such as radix-2[1], radix-4[2], Split Radix[3], prime factor[4]. In

this work we are interested to the radix-4 architecture because it represent an efficient solution. In fact, not only it has a higher throughput since it permits to compute four outputs on the same time but also it has a fewer stages compared to radix-2 and split radix designs. However, to apply the radix-4 decomposition, N should be expressed as $N = 4^v$ where v is an integer.
The Processing Element (PE) of the radix-4-based FFT algorithms is the 4-point FFT. Fig. 1 illustrates the Signal Flow Graph (SFG) of the butterfly. The four butterfly

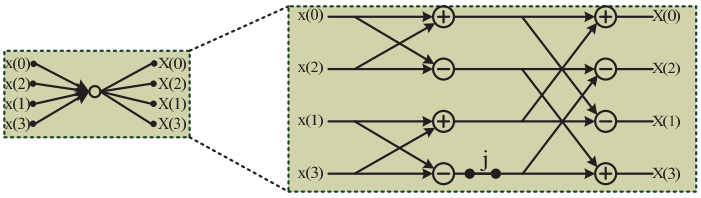

Fig. 1. Radix-4 butterfly

outputs ($X(0)$, $X(1)$, $X(2)$ and $X(3)$) obtained using inputs ($x(0)$, $x(1)$, $x(2)$ and $x(3)$) can be performed using Eq. (2):

$$\begin{aligned} X(0) &= x(0) + x(2) + x(1) + x(3) \\ X(1) &= x(0) - x(2) - j(x(1) - x(3)) \\ X(2) &= x(0) + x(2) - x(1) - x(3) \\ X(3) &= x(0) - x(2) + jx(1) - jx(3) \end{aligned} \tag{2}$$

2.2. *N-point FFT architecture based on radix-4 algorithm*

When the decomposition in multiple building blocks is applied, the N-point FFT is realized by using several stages each one contains many butterflies. For N-point FFT, we need $s = log_4 N$ stages and $b = \frac{N}{4}$ butterflies per stage. An example of 64-point Radix-4 FFT diagram is shown in Fig. 2 where $s = 3$ and $b = 16$.
It can be seen that the computation of the FFTs in the second stage is dependent on the first stage computation. The same concept is applied to the computation of the third stage which depends on the second stage computation. More generally, the data-dependant computation is observed between successive stages for N-point FFT.
Therefore, two architectures are possible. The first one is a parallel realization. An example of that is shown in Fig. 2. For N-point FFT, we need $\frac{N}{4} log_4 N$ radix-4 butterflies which is equivalent to $3\frac{N}{4} log_4(N)$ complex multipliers and $8\frac{N}{4} log_4 N$ complex adders. Although this solution does not make an efficient use of the resources, it is very simple and offers a higher throughput.
The second realization is a recursive one. An example of the SFG is illustrated in Fig. 3 where the N-point FFT can be performed using one radix-4 butterfly. This architecture is interesting in terms of the use of arithmetic operators but suffers from a low operating frequency.

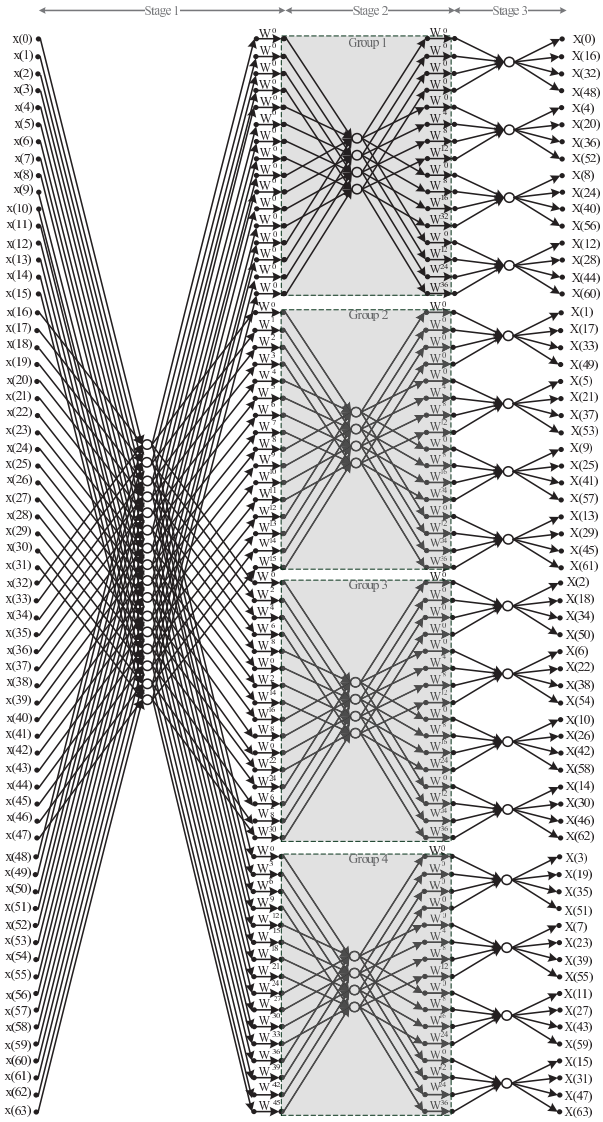

Fig. 2. SFG of 64-point FFT

3. Proposed radix-4 based N-point FFT architecture

3.1. *Prior work*

We have made in [17], a compromise between architecture of Fig. 2 and Fig. 3. Our objective was to balance the memory size and the Maximum Operating Frequency (MOF)

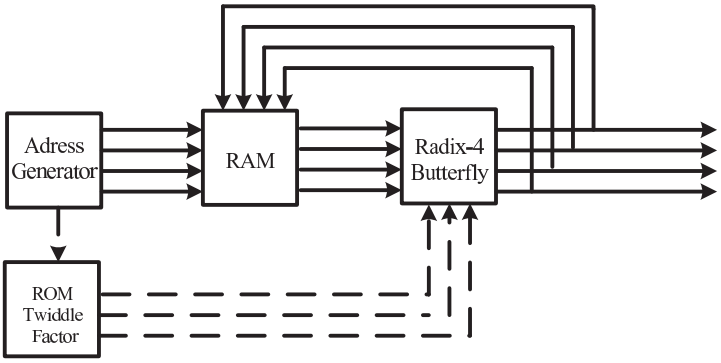

Fig. 3. SFG of 64-point FFT

of the FFT design. The basic idea was to employ one butterfly per stage instead of $\frac{N}{4}log_4 N$ for the design of Fig. 2 or only one butterfly for the design of Fig. 3. This architecture is divided into four stages as mentioned in Fig. 4 for $N = 256$. Each stage is composed of one butterfly and one multiplier block. To store the outputs of each stage, the obvious idea consists in using one N-point memory after each stage. The basic novelty of the architecture proposed in [17] is to use one N-point memory for the N-point FFT and to share this memory between all stages. Then, in order to reduce the number of simultaneous memory access (which consists on reducing the number of memories), we have modified the radix-4 architecture in order to use a serial-in serial-out PE. The memory is divided into 4 blocks and used to store intermediate data between stages. Another advantage of this modification consists on using only one constant complex multiplication with the generated phase (twiddle factor). We mentioned in [17] that with this architecture, the area-delay product presents a slight decrease.

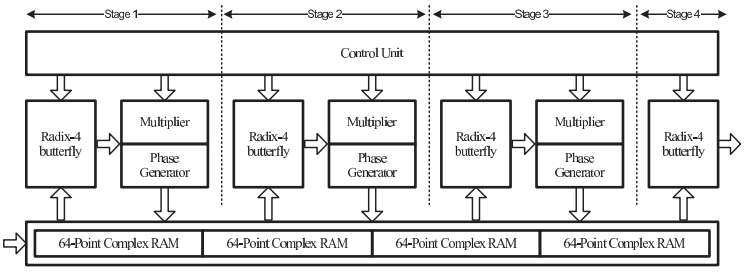

Fig. 4. 256-point FFT architecture proposed in[17]

3.2. *Principle of the proposed architecture*

3.2.1. *Data management*

To further improve the area-delay product of N-point FFT we propose to decrease the computation time of the radix-4 component by using a parallel data processing. This means that we use a parallel-in parallel-out systolic array for deriving the PE on each stage of the FFT. It is observed that the computation time decreases from 10 clock cycles [17] to 2 clock cycles. However, the gain in term of delay is relatively paid by an increase in the number of multipliers. In fact, to maintain the pipeline way, the block multiplier should compute four constant complex multiplications per clock cycle. This number of nontrivial complex multipliers is reduced to three since the first constant is equal to 1. Another drawback of the proposed delay minimization is related to the memory usage. To cope with this problem a novel memory sharing technique is proposed.

3.2.2. *Memory sharing technique*

In the following, we detail the principle of the memory sharing and dividing techniques. The N inputs of N-point FFT are stored in a RAM called Memory_1. As shown in Fig. 2, the first stage of N-point FFT uses $\frac{N}{4}$butterflies. Let's $b1 \in [0, \frac{N}{4} - 1]$ denotes the index of the butterfly of the stage 1. The input addresses of the butterfly b1 are $b1$, $b1 + \frac{N}{4}$, $b1 + \frac{N}{2}$ and, $b1 + 3\frac{N}{4}$. In order to have a simultaneous access to these inputs, they should be stored in four different memories. For this reason, the Memory_1 is divided into 4 blocks. Moreover, the outputs of the first stage are stored in the same memory at the same addresses. This means that Memory_1 and Memory_2 are the same. The computation of the second stage is decomposed into 4 groups. In each group, the butterfly is used $\frac{N}{4^2}$ times (in Fig. 2, the butterflies of group 1 in stage 2 are used $\frac{N}{16} = 4$ times). Let's $b2 \in [0, \frac{N}{16} - 1]$ denotes the index of the butterfly of the group 1 of stage 2. The inputs addresses of that butterfly are $b2$, $b2 + \frac{N}{16}$, $b2 + \frac{N}{8}$ and $b2 + 3\frac{N}{16}$ (for example the addresses of the first butterfly of group 1 in Fig. 2 are 0, 4, 8, 12). Since we need to access to these inputs simultaneously, they should be localized in different memories. Hence, the first quarter of Memory_1 should be divided into 4 memories. According to this, the second, third and fourth quarter of Memory_1 are respectively divided into 4 memories to store outputs of the second, third and fourth group of stage 2. Consequently, the Memory_1 is divided into 16 small memories.
On the other hand, the outputs of group 1 of stage 2 are stored in an $\frac{N}{4}$-point memory called Memory_3 in Fig. 5. Hence, Memory_3 is used in write mode for the group 1 of stage 2. When the computation of group1 of stage 2 is finished, Memory_3 is used in both write and read mode. Indeed, stage 3 is divided into 16 groups each one is composed of 4 butterflies and 3 multipliers. The Memory_3 is used in write mode for 4 groups of stage 3 and in read mode for group2 of stage 2. When the computation of group1 to group4 of stage 3 is finished (which corresponds to the end of computation of group2 in stage2), the Memory_3 is used in write mode for group3 of stage 2 and in read mode for group5 to group8 of stage 3, and so one.
Also, the same principle of dividing Memory_1 into 16 parts is applied again for Memory_3. The goal is to minimize the latency without duplicating memories and increasing area and power consumption. Finally, the process of creating, dividing and sharing memories is repeated as necessary. Comparing to [17] the proposed technique consumes less than $\frac{N}{3}$-point additional memory. However, the most relevant advantage is that it offers a reduced latency since the latency of PEs has been reduced.

3.2.3. *Proposed architecture*

One possible implementation of the proposed architecture based on the new sharing memory technique of Fig. 5 is depicted in Fig. 6 for $N = 256$. It consists of four stages

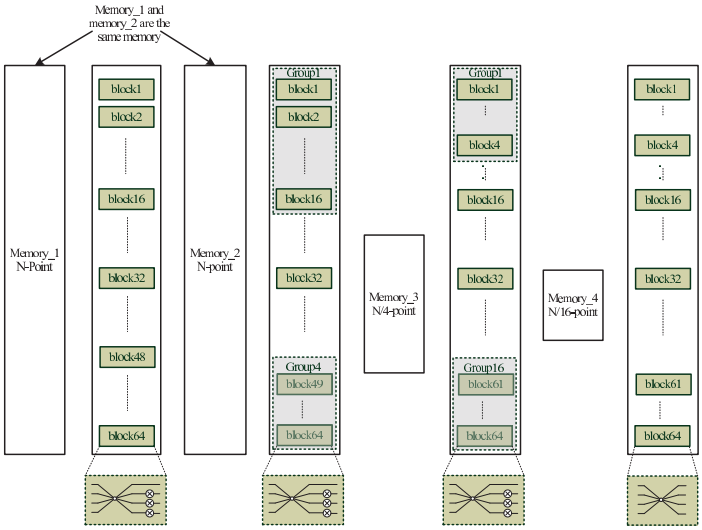

Fig. 5. Memory sharing principle for N=256

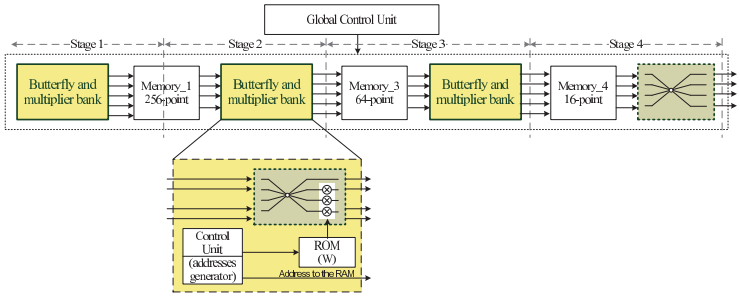

Fig. 6. 256-point FFT proposed architecture

containing three RAMs, one radix-4 (PE) and three blocks of butterfly and multiplier bank. Input values are fed in parallel to the PE. It yields its first 4 outputs two cycles after the first inputs arrive. PE's outputs will be multiplied by a constant multiplication by using a ROM of twiddle factors addressed by a control unit entity. Besides, the control unit generates addresses to the RAM to indicate the position of multipliers outputs. All these blocks are controlled by a global control unit.

4. Timing and hardware complexity analysis

4.1. *Timing analysis*

In this section we will present the latency of the proposed architecture. Let's T_A and T_M represent the computation time of one adder and one multiplier respectively. Accordingly,

the computation time of a butterfly block in stage 4 is equal to $2T_A$ and the computation time of the butterfly and multiplier is about to $2T_A + T_M$. Hence, for N-point-FFT we need $((log_4(N)-1)\times(2T_A+T_M))$ for all the butterfly and multiplier bank. On the other hand, let's T_{Mem} represents the period of one delay element. The number of required delay elements can be expressed as: $3\frac{N}{16} + 3\frac{N}{64} + \cdots + 3\frac{N}{N} = N - 3\frac{N}{4} = \frac{N}{4}$. We can compute the relative latency L as the time elapsed from the computation beginning to the first output. Under these conditions, L is expressed by:

$$L = \frac{N}{4}T_{Mem} + (log_4(N) - 1) \times (T_M + 2T_A) + 2T_A \tag{3}$$

4.2. *Comparison with efficient designs*

The hardware complexityto he proposed architecture is listed along with those of the existing structures in Table 1.

The table shows the tradeoff between area and timing performance. The area is measured by the number of complex multipliers, adders and memory size, whereas the timing performance is represented by the throughput and the latency. The structures presented in this table are parallel-in serial-out design [13 14 15 18 19 20] or parallel-in parallel-out design like the proposed one and designs of [22 12 21]. The number of complex multiplier and adder of the paralel-in serial-out designs is three times lower than parallel-in parallel-out designs. Compared with parallel-in parallel-out schemes, the proposed design provides a lower absolute latency. This latency is approximated without the number of order $logN$ as in all references cited in Table. 1. Furthermore, the proposed architecture involves nearly half of memory size of [21] with same number of adders and multipliers.

Table 1. Hardware requirements ressources comparison of pipeline FFT architecture

References	Structure	Complex Multipliers	Complex Adders	Memory size	Nbr Samples/cycle	Absolute Latency
R2SDF[13]	Radix-2	log_2N-2	$2log_2$N	N-1	1	N
R2^{2}SDF[15]		log_4N-1	$4log_4$N	N-1	1	N
R2^{2}SDF[18]		log_4N-1	log_2N	2(N-1)	1	N
FB_Radix-2[19]		log_4N-1	$2log_4$N	$\frac{4}{3}$N	1	N
R4SDF[20]	Radix-4	log_4N-1	$8log_4$N	N-1	1	N
R4SDF[14]		log_4N-1	$3log_4$N	2(N-1)	1	N
R2MDC[12]	Radix-2	log_2N-2	$2log_2$N	$\frac{3}{2}$ N-2	1	N
FF_Radix-2[21]	Radix-2	2(log_4N-2)	$2log_2$N	4 N	2	N
R4MDC[22]	Radix-4	3(log_4N-1)	$8log_4$N	$\frac{5}{2}$N-2	4	N
FF_Radix-4[21]		3(log_4N-1)	$8log_4$N	$\frac{8}{3}$N	4	$\frac{N}{3}$
Design of[17]		log_4N-1	$8log_4$N	N	1	N
Proposed		3(log_4N-1)	$8log_4$N	$\frac{4}{3}$N	4	$\frac{N}{4}$

5. Implementation results

5.1. *Hardware complexity*

In this section we describe the material complexity of FFT architectures detailed in section 2.2 and section 3. The basic criterion used for comparison is the area-delay product. The hardware and time complexities of the proposed structure using the parallel-in parallel-out radix-4 and the memory sharing technique are listed along with those of the existing structures in Table. 2. All the structures mentioned in Table. 2 are coded

using the VHDL language and synthesized using Xilinx ISE and Spartan-3 FPGA device. Also, we used the propagation delay with the execution time as time complexity criteria. The execution time is defined by absolute latency and obtained by multiplying the relative latency by the duration of one cycle.
The design of Fig. 2 is the direct realization of 64-point FFT using radix-4 PE. This structure involves the lowest execution time but the highest number of slices. On the other hand, the recursive structure of the memory-based design[6] has the highest execution time and a low number of slices. The proposed design and our design of[17] have nearly the same propagation delay but the last one involves more than double the execution time.

Table 2. Area-delay comparison

Architecture	Slices	Propagation Delay (ns)	Execution time(s)	Slice-Delay product (Slice.s)
Design of Fig. 2	43623	15.3	0.15	6543.4
Memory-based[6]	2281	30.3	8.3	18932.3
Design of [17]	1155	9.4	2.03	2344.6
Proposed	2345	8.4	0.8	1876

5.2. *Synthesis Results*

To analyse show the efficiency of the proposed architecture, a comparison of the synthesis results obtained from a Spartan-3 implementation with several architectures has been made. Table. 3 and Table 4 present a comparison with prior art of the pipelined architectures, based-memory architectures and Xilinx IP core. The performance of different architectures is analyzed in terms of area (slice number), Maximum Operating Frequency (MOF), throughput (M samples/s), execution time and area-delay product.

Table 3. Synthesis results of 64-point FFT

Architecture	Structure	Slices	MOF (MHz)	Throughput (MS/s)	Execution time(s)	Area-Delay product Slice.μs
R4SDC[16]	Radix-4	2662	107	107	1.2	3194
R2SDF[16]	Radix-2	2520	98	98	1.4	3528
RX4-B1[6]	Radix-4	2281	33	33	8.3	18932
RX4-B2[6]	Radix-4	3323	33	66	4.6	15285
MXRX-B4[6]	Split-Radix	6655	33	132	2.69	17901
Xilinx IP[23]	Radix-2	1477	152	152	1.68	2481
Design of[17]	Radix-4	1155	106	106	2.03	2344
Proposed	Radix-4	2345	118	472	0.8	1876

It can be found that the proposed design involves an execution time about 56%, 36% and 26% lower than that of Xilinx IP core[23], R2^{2}SDC[16] and R4SDC[16] respectively for 256-point FFT. Regarding the area-delay product, it can be seen that the proposed architecture can achieve about 11% and 10% of reduction in the area-delay product compared to R4SDC[16] and R2^{2}SDF[16] respectively and about 73% compared to MXRX-B4[6] for 256-point FFT.
On the other hand, comparison in term of Throughput by Slice ratio between the pro-

Table 4. Synthesis results of 256-point FFT

Architecture	Structure	Slices	MOF (MHz)	Throughput (MS/s)	Execution time(s)	Area-Delay product Slice.μs
R4SDC[16]	Radix-4	4555	111	111	4.6	20953
R2^{2}SDF[16]	Radix-2	3868	98	98	5.36	20732
RX4-B1[6]	Radix-4	2281	33	33	39.3	89643
RX4-B2[6]	Radix-4	3323	33	66	20	66460
MXRX-B4[6]	Split-Radix	6655	33	132	10.2	67881
Xilinx IP[23]	Radix-2	2455	148	148	7.6	18854
Design of[17]	Radix-4	1924	91	91	10.1	19432
Proposed	Radix-4	5525	103	412	3.4	18785

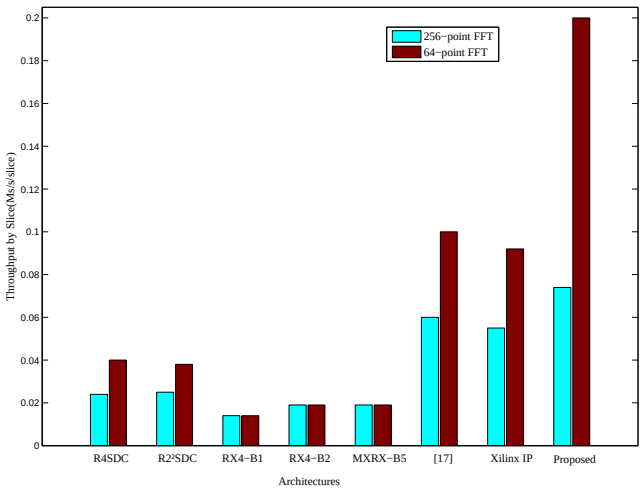

Fig. 7. Throughput by Slice ratio comparison

posed architecture and those listed in table 3 for 64-point and 256-point FFT is presented in Fig. 7. It is indicated that our design still has advantage over the others. In fact, our proposed design present an increase of 26% and 19% in term of throughput by slice ratio compared to the architecture proposed on[17] and to Xilinx IP[23] for 256-point FFT.

5.3. *Signal-to-quantization noise ratio (SQNR)*

Since we use fixed-point operators, some truncations are needed to maintain the dynamic range of intermediate signals and outputs. These truncations can affect the accuracy of the FFT algorithm by introducing the quantization noise. It is thus necessary to evaluate this noise and to compute the Signal to Quantization Noise Ratio (SQNR). Many research have been treated the problems of truncation noise for fixed-point operators used in orthogonal transforms as in FFT[5], DCT[24] and FIR[25]. Our objective in this section is to evaluate the effects of truncation. For our implementation, we have performed

the truncation only for the constant multipliers (not for adders). In fact, input data are encoded using n bits. The computation of these inputs implies a bit growth of up to 2 bits after each butterfly block. However, for the multipliers, the output width is equal to the input width. Indeed, for a multiplication with twiddle factor encoded using c bits, we apply truncation by using c right shifts to the outputs. Consequently, the output width of the proposed N-point FFT architecture is equal to $n + 2log_4 N$ bits.
To evaluate the SQNR, we applied a sine wave with input frequency of 32 kHz and a sampling frequency of 100 kHz. Two FFTs have been computed. The "exact" FFT ($X_{fl}(k)$) is with a floating-point arithmetic obtained by Matlab 64-bit precision. The second ($X_{fx}(k)$) is with the proposed architecture. The SQNR is defined by:

$$SQNR = \frac{\sum_k |X_{fx}(k)|^2}{\sum_k (|X_{fl}(k)| - |X_{fx}(k)|)^2} \tag{4}$$

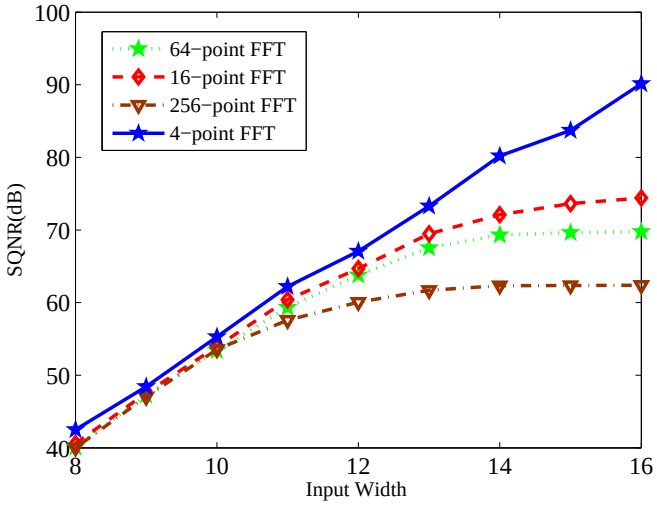

Fig. 8. SQNR variation for different FFT size and different input width

We present in Fig. 8 the SQNR evaluation versus FFT size and data width of inputs (the twiddle factor width is set to 12 bits). It can be clearly observed that larger is the input width higher is the SQNR. Moreover, for a large size of FFT the SQNR decrease. This is due to quantization noise propagation. Finally, it should be pointed out that the maximum Mean Square Error (MSE) between X_{fx} and X_{fl} obtained with 256-point FFT is about 1%.

6. Conclusion

In this paper we have proposed a novel architecture of N-point FFT based on radix-4 algorithm suitable for FPGA implementation. The novelty of the proposed architecture

consists on using the memory sharing and dividing techniques along with parallel-in parallel-out processing in order to reduce latency and minimize the area occupation. We compared favorably our proposed architecture with some recent works quoated in litterature. We find that the proposed architecture has several advantages in terms of speed and throughput performances and saving of silicon area. Power consumption should be evaluated, which is being studied.

Acknowledgments

This research was supported by a grant from the Interface Concept Company. For more details, please consult http://www.interfaceconcept.com.

References

1. J. W. Cooley and J. Tukey, An algorithm for the machine calculation of Complex Fourier series, *Math. Comput.*, vol. 19, April 1965, pp. 297–301.
2. A. V. Oppenheim, R. W. Schafer, and J. R. Buck, Discrete-Time Signal Processing, 2nd ed. Englewood Cliffs, NJ: *Prentice-Hall*, (1998).
3. H. Sorensen, M. Heindeman, and C. Burrus, On computing the split radix FFT, *IEEE Trans. Acoustics, Speech, Signal Process*, vol.34, (1986), pp. 152-156.
4. D. Kolba, T. Parks, A prime factor FFT algorithm using high-speed convolution, *IEEE Trans. Acoustics, Speech and Signal Processing*, vol.25, no.4, Aug 1977 pp. 281- 294.
5. W-H. Chang, T. Q. Nguyen, On the Fixed-Point Accuracy Analysis of FFT Algorithms, *IEEE Trans. Signal Processing*, vol.56, no.10, Oct. 2008, pp. 4673-4682.
6. H.G. Yeh, G. Truong, Speed and Area Analysis of Memory Based FFT Processors in a FPGA, *Wireless Telecommunications Symp.*, (2007), pp. 1-6.
7. B. S. Son, B. G. Jo, M. H. Sunwoo, and Y. S. Kim, A High-Speed FFT Processor For OFDM Systems, in *Proc. IEEE Int. Symp. Circuits and Systems (ISCAS)*, vol. 3, (2002), pp. 281-284.
8. L. G. Johnson, Conflict free memory addressing for dedicated FFT hardware, *IEEE trans. Circuits Syst. II., Analog Digital Signal Processing*, vol. 39, no. 5, May 1992, pp. 312- 316.
9. K. L. Heo, J. H. Baek, M. H. Sunwoo, B. G. Jo, and B. S. Son, New in-place strategy for a mixed-radix FFT processor, in *Proc. IEEE Int. [Systems-on-Chip] SOC Conference*, (2003), pp. 81 - 84.
10. B.G. Jo and M.H. Sunwoo, New Continuous-flow mixedradix (CFMR) FFT Processor using novel in-place strategy, *IEEE Trans. Circuits Sys.- I.*, vol. 52, May 2005, pp. 911 - 919.
11. R. Radhouane, P. Liu, and C. Modlin, Minimizing the memory requirement for continuous flow FFT implementation: continuous flow mixed mode FFT (CFMM-FFT), in *Proc. IEEE Int. Symp. Circuits and Systems (ISCAS)*, vol. 1, (2000), pp. 116-119.
12. L.R. Rabiner and B. Gold, Theory and Application of Digital Signal Processing, *Prentice-Hall*, Inc., (1975).
13. E. H. Wold and A.M. Despain, Pipeline and parallel-pipeline FFT processors for VLSI implementation, *IEEE Trans. Computers*, vol. C-33, no. 5, 1984, pp. 414-426.
14. G. Bi and E.V. Jones, A pipelined FFT processor for word-sequential data, *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. 37, no. 12, 1989, pp. 1982-1985.
15. S. He and M. Torkelson, A new approach to pipeline FFT processor, *Int. Parallel Processing Symposium (IPPS '96)*, Apr. 1996, pp. 766-770.
16. B. Zhou, Y. Peng, D. Hwang, Pipeline FFT Architectures Optimized for FPGAs, *Int. Journal of Reconfigurable Computing*, 2009.

17. Y. Ouerhani, M. Jridi, A. Alfalou, Implementation techniques of high-order FFT into low-cost FPGA, *2011 IEEE 54th International Midwest Symposium Circuits and Systems (MWSCAS),* Aug. 2011, pp. 1-4.
18. G. Bi; G. Li, Pipelined structure based on radix-22 FFT algorithm, *IEEE Conference on Industrial Electronics and Applications (ICIEA)*, 21-23 June 2011, pp. 2530-2533.
19. L. Yang, K. Zhang, H. Liu, J. Huang, and S. Huang, An efficient locally pipelined FFT processor, *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 53, no. 7, Jul. 2006, pp. 585-589.
20. A. M. Despain, Fourier transform computer using CORDIC iterations, IEEE Trans. Comput., C-23(10):993-1001, Oct. 1974.
21. M. A. Snchez, M. Garrido, M. L. Lpez, and J. Grajal, Implementing the FFT algorithm on FPGA platforms: A comparative study of parallel architectures, *Int. Conf. Design Circuits Integr.*, Syst. (2004).
22. L. R. Rabiner and B. Gold, Theory and Application of Digital Signal Processing, *Prentice-Hall*, Inc., (1975).
23. Xilinx Product Specification, High perfomance Complex FFT/IFFT V.7.0, (2009) [online]. Available on: http://www.xilinx.com/ipcenter.
24. L. Tan, L. Wang, Oversampling Technique for Obtaining Higher Order Derivative of Low-Frequency Signals, *IEEE Trans. on Instrumentation and Measurement*, vol.60, no.11,Nov. 2011, pp.3677-3684.
25. M. Hassan, F. Shalash, FPGA Implementation of an ASIP for high throughput DFT/DCT 1D/2D engine, *Circuits and Systems (ISCAS)*, May 2011, pp. 1255-1258.