# A 4096-Point Radix-4 Memory-Based FFT Using DSP Slices

Mario Garrido Gálvez, Miguel Angel Sanchez, Maria Luisa Lopez-Vallejo and Jesus Grajal

**Journal Article**

# A 4096-point Radix-4 Memory-Based FFT using DSP Slices

Mario Garrido, *Member, IEEE*, M.A. Sánchez, M.L. López-Vallejo and J. Grajal

***Abstract*—This brief presents a novel 4096-point radix-4 memory-based FFT. The proposed architecture follows a conflict-free strategy that only requires a total memory of size $N$ and few additional multiplexers. The control is also simple, as it is generated directly from the bits of a counter. Apart from the low complexity, the FFT has been implemented on a Virtex 5 FPGA using DSP slices. The goal has been to reduce the use of distributed logic, which is scarce in the target FPGA. With this purpose, most of the hardware has been implemented in DSP48E. As a result, the proposed FPGA is efficient in terms of hardware resources, as is shown by the experimental results.**

***Index Terms*—Fast Fourier Transform (FFT), Memory-based Architecture, Radix-4, Very-large-scale integration (VLSI), Field Programmable Gate Array (FPGA).**

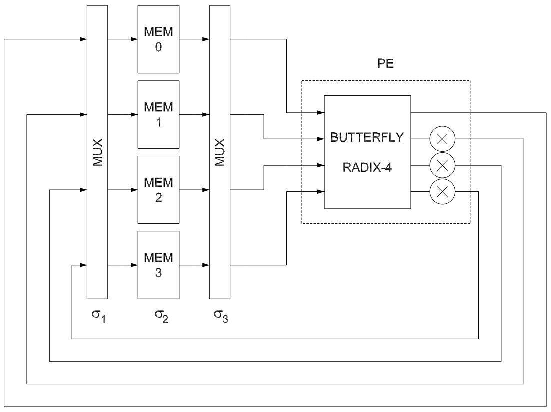

Fig. 1. Proposed memory-based radix-4 FFT architecture. The processing element (PE) is composed of a radix-4 butterfly and three rotators.

## I. Introduction

The fast Fourier transform (FFT) is one of the most important algorithms in the field of digital signal processing, used to calculate the discrete Fourier transform (DFT) efficiently. The FFT is part of numerous systems in a large variety of applications. Sometimes the system demands the computation of the FFT at a very high rate. For this purpose pipelined FFTs are mainly used [1], [2]. In other systems, the demands in terms of performance are not so strict. Instead, there are demands in terms of area or hardware resources occupied by the architecture. Under these circumstances the designers usually resort to memory-based FFTs [3]–[11], also called in-place or iterative FFTs.

Memory-based FFTs consists of a memory or bank of memories that store the data. These data are read from memory, processed by butterflies and rotators, and stored again in memory. This process repeats iteratively until all the stages of the FFT algorithm are calculated. The advantage of memory-based FFTs is the reduction in the number of butterflies and rotators, as they are reused for different stages of the FFT.

There exist numerous memory-based FFT architectures in the literature. They mainly differ in the size of the processing element (butterflies and rotators). The most typical approach is to use a radix-2 butterfly [3]–[5], but there are cases of radix-4 [6], [7] and other radices [8]–[11]. Memory-based FFTs also differ in the access strategy to the memory, which in most cases provides conflict-free access. The amount of memory used in an $N$-point memory-based FFT is generally $N$ or $2N$. Apart from the memory, the access strategy may demand extra multiplexers [7], buffers or cache memories.

This paper presents a novel radix-4 memory-based FFT. The proposed design has several advantages. With respect to previous radix-4 approaches, it uses the minimum memory of $N$ samples and few additional multiplexers. Furthermore, the proposed approach has been implemented using DSP48E slices on FPGA. The implementation allows to integrate the components of the architecture in the DSP48E, which reduces the hardware, especially the amount of distributed logic. As a result, the proposed approach is a compact solution for FPGA that takes advantage of the use of DSP48E slices, leaving room in the FPGA for other complex and area demanding elements.

The paper is organized as follows. Section II describes the proposed memory-based FFT. Section III explains the implementation using DSP slices. Section IV compares the proposed FFT to previous memory-based FFTs. Section V presents an application where the proposed FFT has been used. Section VI shows the experimental results on FPGA. Finally, Section VII summarizes the main conclusions of the paper.

## II. Proposed Memory-Based FFT

### A. Basic Architecture

The basic architecture of the proposed 4096-point FFT is shown in Fig. 1. The architecture uses radix-4 and computes the FFT algorithm iteratively in 6 iterations, which comes from the fact that in a radix-r memory-based FFT the number of iterations is

$$It = \frac{\log_2 N}{\log_2 r} = \frac{n}{\log_2 r}, \tag{1}$$

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

M.A. Sánchez and M.L. López-Vallejo are with the Department of Electrical Engineering, Universidad Politécnica de Madrid, 28040 Madrid, Spain, e-mail: masanchez@die.upm.es, marisa@die.upm.es

J. Grajal is with the Department of Signal, Systems and Radiocommunications, Universidad Politécnica de Madrid, 28040 Madrid, Spain, e-mail: jesus@gmr.ssr.upm.es

This work was supported by the the Swedish ELLIIT Program.

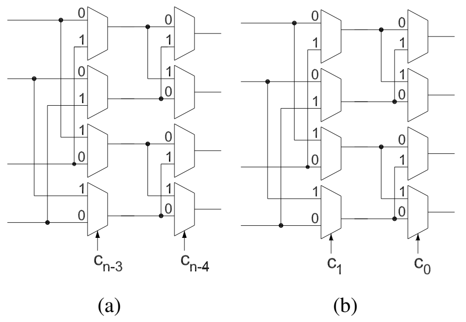

Fig. 2. Circuits for the permutations in the multiplexers. They only consists of multiplexers controlled by the bits of the counter. (a) $\sigma_1$. (b) $\sigma_3$.

The proposed design includes four memories of $N/4$ samples in parallel instead of a single memory of $N$ samples. This allows to read and write data simultaneously in all the memories, which reduces the latency and increases the throughput of the circuit. Thus, at every clock cycle the processing element (PE) receives and provides four samples in parallel, one from and to each memory.

## B. Conflict-free Access

As all four memories are accessed simultaneously it must be assured that the four samples processed in the PE every clock cycle come from different memories. This demands a conflict-free memory access strategy as shown next. The notation in the paper is the same one used in previous works [12], [13].

Initially, samples are stored in natural order in the memories: Samples 0 to $N/4-1$ are stored in MEM0, $N/4$ to $N/2-1$ in MEM1 and so on. If we number the samples from 0 to $N-1$ with an index $I \equiv b_{n-1}b_{n-2}\dots b_0$, the position in memory of each sample $I$ is

$$P_1 \equiv \underbrace{b_{n-3}, b_{n-4}\dots, b_0}_{serial(address)} | \underbrace{b_{n-1}, b_{n-2}}_{parallel(memory)} \tag{2}$$

Bits $b_{n-1}$ and $b_{n-2}$ indicate in which of the four memories the sample is stored, whereas bits $b_{n-3}, \dots, b_0$ are the address.

In a radix-2 FFT the butterfly at stage $s$ operates on samples whose index differ in the bit $b_{n-s}$ [2]. For radix-4, the butterfly at stage $s$ operates on samples that differ in bits $b_{n-2s-1}b_{n-2s}$. At each iteration of the FFT these samples must arrive in parallel to the PE. For the first stage/iteration, bits $b_{n-1}b_{n-2}$ are already in different memories according to (2). For the second iteration samples that differ in bits $b_{n-3}b_{n-4}$ need to arrive in parallel to the PE. This is achieved by calculating the permutation

$$\sigma(u_{n-1}, \dots, u_2|u_1, u_0) = u_{n-3}, u_{n-4}, \dots, u_0|u_{n-1}, u_{n-2} \tag{3}$$

on the data in position $P_1$, which leads to the position

$$P_2 \equiv \underbrace{b_{n-5}, b_{n-6}, \dots, b_0, b_{n-1}, b_{n-2}}_{serial} | \underbrace{b_{n-3}, b_{n-4}}_{parallel} \tag{4}$$

For the rest of iterations the same permutation is carried out to enable the correct samples into the PE.

The permutation in equation (3) is carried out in three steps:

$$\sigma = \sigma_3 \circ \sigma_2 \circ \sigma_1 \tag{5}$$

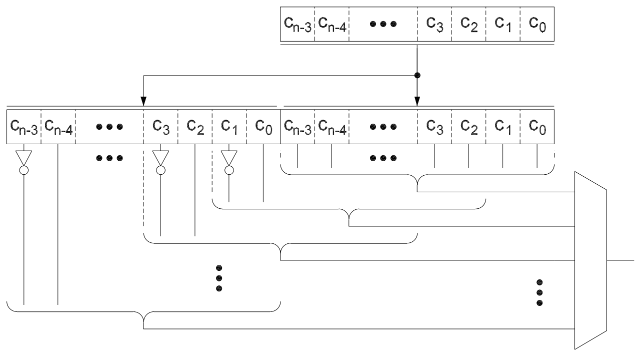
c_{n-3} c_{n-4} ••• c_3 c_2 c_1 c_0
c_{n-3} c_{n-4} ••• c_3 c_2 c_1 c_0 c_{n-3} c_{n-4} ••• c_3 c_2 c_1 c_0

Fig. 3. Generation of the memory address for MEM2. The counter is replicated twice and the corresponding bits are selected by the multiplexer. Note that the circuit only consists of NOT gates and the multiplexer.

The first permutation

$$\sigma_1(u_{n-1}, \dots, u_0) = u_{n-1}, \dots, u_2|u_{n-1} \oplus u_1, u_{n-2} \oplus u_0 \tag{6}$$

is the permutation of the multiplexers before the memories, where $(\oplus)$ represents the logic XOR function. The circuit that calculates this permutation is shown in Fig. 2(a). This permutation determines the memory in which data is going to be stored. The permutation $\sigma_1$ is controlled by the bits $c_{n-3}$ and $c_{n-4}$. These are the two MSBs of the control counter with bits $c_{n-3}, \dots, c_0$. Thus, the control is simple, as it is taken directly from the bits of the counter.

The second permutation

$$\begin{aligned}\sigma_2(u_{n-1}, \dots, u_0) &= \\ &= u_{n-3}, \dots, u_2, u_{n-1} \oplus u_1, u_{n-2} \oplus u_0|u_1, u_0\end{aligned} \tag{7}$$

is carried out by the memories. It only affects the content of the memories. The memory address is obtained as

$$\begin{aligned}&W_{i+1} = R_i = \\ &c_{2i-1} \oplus m_1, c_{2i-2} \oplus m_0, \dots c_1 \oplus m_1, c_0 \oplus m_0, c_{n-3}, \dots, c_{2i}\end{aligned} \tag{8}$$

where $m_1m_0$ are the bits that indicate the memory, $c_i$ are the bits of the control counter, $W_{i+1}$ is the writing address at iteration $i+1$ and $R_i$ is the reading address at iteration $i$. Note that $W_{i+1} = R_i$. This means that at each iteration data are written in the addresses that are emptied in the previous iteration. This optimization allows to only use a total memory of $N$. For the first iteration, $W_1 = R_0$ is equal to the control counter. The generation of the memory address is shown in Fig. 3 for MEM2, for which $m_1m_0 = 10$.

The third permutation

$$\sigma_3(u_{n-1}, \dots, u_0) = u_{n-1}, \dots, u_2|u_3 \oplus u_1, u_2 \oplus u_0 \tag{9}$$

is the permutation of the multiplexers after the memories, shown in Fig. 2(b). As $\sigma_1$, it only determines the memory in which data is stored.

Finally, the first time that samples are read from the memory, the permutation $\sigma_3$ is disabled, i.e., the control signals are set to 00. This happens because samples in the memory are already in the order demanded by the PE.

Fig. 4 shows the data management for a 16-point FFT. The upper part of the figure shows the data orders at the different stages of the circuit below. First, data are stored in memory.

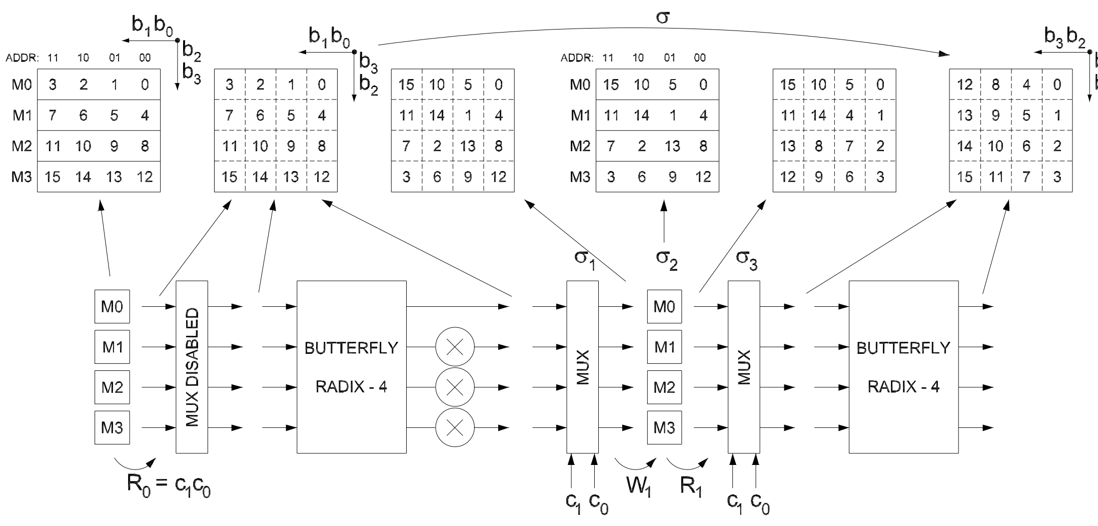

Fig. 4. Data management example for a 16-point memory-based FFT. In the figure $W_1 = R_0 = c_1c_0$ and $R_1 = c_1 \oplus m_1, c_0 \oplus m_0$.

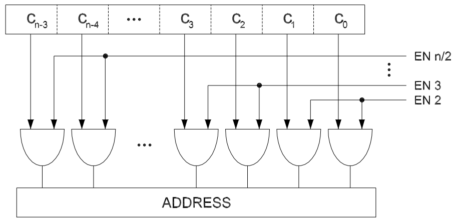

Fig. 5. Generation of the address for the rotation memories. $c_{n-1}, \ldots, c_0$ are the bits of the counter. Each enable signal EN $i$ is activated at iteration $i$ and the iterations after that one.

The figure shows the content of the different addresses in M0, M1, M2 and M3. These data are read according to $R_0 = c_1c_0$, i.e., data from the memories are read in order. The data bypass the multiplexer after the memory, which is disabled in this iteration, and inputs the butterfly. The data order does not change until after the multiplexer that calculates $\sigma_1$. This multiplexer is controlled by $c_{n-3}c_{n-4} = c_1c_0$ and only permutes parallel data that arrive in the same clock cycle according to $\sigma_1$. The data at the output of the multiplexer is stored again in memory. The writing address is $W_1 = c_1c_0$, which is equal to $R_0$ to avoid memory access conflicts, as shown in (8). The memories are read according to $R_1 = c_1 \oplus m_1, c_0 \oplus m_0$. Note that the writing plus the reading in the memories leads to a permutation $\sigma_2$ on data that arrive in series to the memories. Finally, a second parallel permutation, $\sigma_3$, provides the order required at the input of the butterfly for the second iteration. The output of the butterfly provides the output of the FFT.

### C. Rotations

The rotations of the FFT are preformed by three complex multipliers. Each of them is connected to a rotation memory which is a ROM of $N/4$ addresses that store the sine and cosine components of the rotation angle $\phi = -m\frac{2\pi}{N}i$, where $m \in \{1, 2, 3\}$ is the memory and $i$ is the memory address. The memory address of the rotation memories is generated in a simple way from the control counter, as shown in Fig. 5. The address is the same for all three memories and is obtained by enabling bits of the counter depending on the iteration.

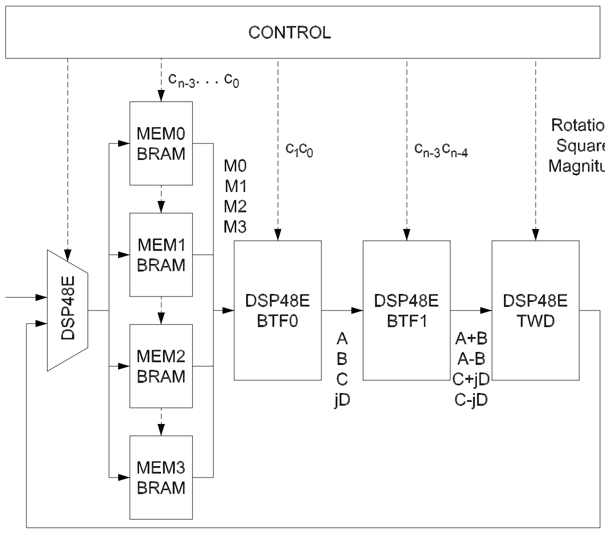

Fig. 6. Proposed FFT implemented using DSP48E.

## III. Implementation using DSP slices

The proposed architecture has been implemented on a Virtex-5 XC5VSX95T FPGA. The VSX family is characterized by including a large number of DSP48E and a small amount of distributed logic. Thus, we have pursued to maximize the use of DSP48E and minimize the use of distributed logic by implementing on DSP48E all the elements of the architecture except the memories. An advantage of using DSP slices is that they can be clocked at high clock frequencies. Furthermore, the implementation on DSP slices allows for large word lengths without reducing the clock frequency, compared to designs implemented in distributed logic, where the clock frequency may be reduced when increasing the word length. Fig. 6 shows the architecture.

BRAM refers to the 4 memories of the architecture. They are implemented using the block RAMs. Each memory has 1024 addresses and each address stores a sample of 24 + 24 bits for the real and imaginary parts, respectively.

The module BTF0 consists of 2 DSP48E in which the use

TABLE I
OUTPUTS OF THE MODULE BTF0 AS A FUNCTION OF THE CONTROL BITS.

| Control bits | Outputs | | | |
|---|---|---|---|---|
| $c_1c_0$ | Output 0 | Output 1 | Output 2 | Output 3 |
| 00 | A=M0+M2 | B=M1+M3 | C=M0-M2 | D=M1-M3 |
| 01 | A=M1+M3 | B=M0+M2 | C=M1-M3 | D=M0-M2 |
| 10 | A=M2+M0 | B=M3+M1 | C=M2-M0 | D=M3-M1 |
| 11 | A=M3+M1 | B=M2+M0 | C=M3-M1 | D=M2-M0 |

TABLE II
OUTPUTS OF THE MODULE BTF1 AS A FUNCTION OF THE CONTROL BITS.

| Control bits | Outputs | | | |
|---|---|---|---|---|
| $c_{n-3}c_{n-4}$ | Output 0 | Output 1 | Output 2 | Output 3 |
| 00 | A+B | A-B | C+jD | C-jD |
| 01 | A-B | A+B | C-jD | C+jD |
| 10 | C+jD | C-jD | A+B | A-B |
| 11 | C-jD | C+jD | A-B | A+B |

of the multiplier has been disabled and allows to work with 4 inputs of 24 + 24 bits. The ALU inside the DSP48E is configured in mode SIMD=TWO24. This way the real and imaginary components of the data are operated jointly.

Both sets of multiplexers in Fig. 2 would represent a significant cost if they were implemented in distributed logic. In order to avoid this, the connections between the memory and the processing elements are static, i.e., the outputs of the memories are always connected to the PE without multiplexing. This demands to modify the PE: The module BTF0 calculates the first crossed terms of the radix-4 butterfly and incorporates the multiplexers in Fig. 2(b). Thus, the BTF0 changes the operations of the DSP48E depending on the 2 LSBs of the control counter, according to table I.

The module BTF1 is analogous to BTF0. It consists of 2 DSP48E and calculates the second part of the radix-4 butterfly. The operations that are executed depend on the bits $c_{n-3}c_{n-4}$ of the control counter, as shown in table II.

The module TWD calculates the multiplications by the twiddle factors. The twiddle factors are stored in ROM memory, which is used in all the iterations of the FFT. While in the first iteration the coefficients are read one by one in order, in the rest of iterations the LSBs of the control counter are canceled in order to determine the address [14], as shown in Fig. 5. As the outputs of BTF1 are shuffled, the twiddle factors are also provided in this shuffled order. During the last iteration, the module TWD does not need to calculate any rotation. Thus, the multipliers of this module are used to calculate the squared magnitude of the complex values, i.e., $|C^2 + S^2|$, in order to determine the power at each output frequency.

## IV. COMPARISON

Table III compares various memory-based FFTs. In memory-based FFTs there is a trade-off between the amount of resources of the architecture and the processing time, $T_{PROC}$. This depends on the radix: The higher the radix, the larger the PE. A large PE increases the area, but reduces $T_{PROC}$:

$$T_{PROC} = \frac{N}{r} It \cdot T_{MEM} \tag{10}$$

where $T_{MEM}$ is the access time to the memory. Thus, radix-2 FFTs in Table III need less resources [3]–[5], whereas high-radix FFTs [8]–[10] achieve higher throughput.

Radix-4 is in the middle between radix-2 and high radices. Therefore, it presents a trade-off between resources and performance. Among radix-4 designs [6], [7], [10], the proposed approach is characterized by the use of the least amount of resources, while keeping the same $T_{PROC}$ as other radix-4 designs. Specifically, it only needs a total memory of $N$ samples compared to $2N$ samples [6] or $2N(\phi + 1)$ samples [10]. Note, however, that the approach in [6] is intended for continuous flow whereas our approach is not. Compared to [7] the proposed design reduces significantly the number of multiplexers to only 16 2-input multiplexers, compared to 16 16-input multiplexers and demultiplexers in [7], which is equivalent to 240 2-input multiplexers.

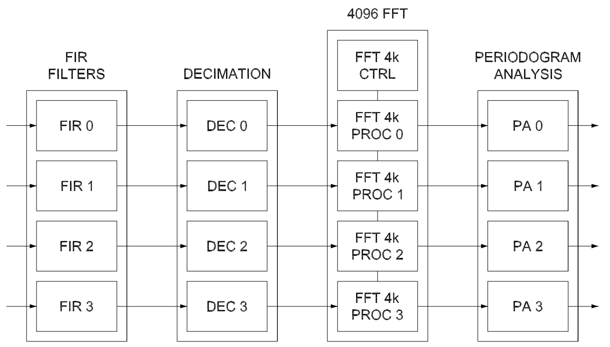

Fig. 7. Block diagram of the spectrum analyzer. The FFT is included in the system and consists of a control (CTRL) block and four processing (PROC) blocks, one for each of the channels.

## V. APPLICATION CASE

The proposed FFT has been used for spectrum analysis. Fig. 7 shows the block diagram of the spectrum analyzer. The system includes four channels. Each channel consists of a finite-impulse response (FIR) filter, a decimation stage (DEC), a 4096-point iterative FFT and a periodogram analysis (PA).

The FIR filter is in charge of the bandwidth adaptation. The filtering is done with appropriate coefficients to avoid aliasing, and to reduce the band interferences and noise.

After the filter, data are decimated a factor $L = 8$. The DEC block provides 1 sample every 20 ns to the FFT.

Once all the samples are loaded into the FFT module, the calculation of the FFT starts. The FFT module applies a window to the input sequence, then calculates the FFT and finally the squared magnitude of the FFT (periodogram) is calculated to obtain the power of the signals.

## VI. EXPERIMENTAL RESULTS

Table IV summarizes the figures of merit of the proposed 4096-point memory-based FFT shown in Fig. 7. It processes 4 channels of 4096 points each with a word length of 24 + 24 bits. The clock frequency is 200 MHz. Higher clock frequency up to 400 MHz is also supported to reduce the processing time $T_{PROC}$. Higher input rate up to 4 samples per clock cycle is also feasible in order to reduce the load time $T_{LOAD}$. Area is measured in terms of Slices, Slice LUTs, DSP48E and BRAM.

The implementation has been done on an FPGA. This differs from most memory-based FFTs in the literature, which have

TABLE III
COMPARISON OF ACCESS STRATEGIES IN MEMORY-BASED FFTS.

| Approach | Radix | Parallel Process. † | Mem. Size | Mem. Banks | Iterations †† | Cycles per It. | $T_{PROC}$††† (Cycles) | Observations |
|---|---|---|---|---|---|---|---|---|
| [3] | 2 | 2 | $N$ | 2 | $\log_2 N$ | $N/2$ | $N(\log_2 N)/2$ | - |
| [4] | 2 | 2 | $N$ | 4 | $\log_2 N$ | $N/2$ | $N(\log_2 N)/2$ | Barrel shifter for RD/WR address |
| [5] | 2 | 4 | $N$ | 4 | $\log_2 N - 1$ | $N/4$ | $N(\log_2 N - 1)/4 + 1$ | For real-valued data |
| [6] | 4/2 | 4 | $2N$ | 4 | $(\log_2 N)/2$ | $N/4$ | $N(\log_2 N)/8$ | Continuous flow |
| [7] | 4 | 4 | $N$ | 4 | $(\log_2 N)/2$ | $N/4$ | $N(\log_2 N)/8$ | Large amount of multiplexers |
| [8] | $2/2^2/2^3$ | 2 | $2N$ | 4 | $(\log_2 N)/3$ | $N/4$ | $N(\log_2 N)/12$ | Uses pipelined MDC FFTs as PE |
| [9] | 16 | 16 | $N$ | 16 | $(\log_2 N)/4$ | $N/16$ | $N(\log_2 N)/64$ | For high throughput |
| [10] | $r$ | $r$ | $2N(\phi+1)$ | $2r(\phi+1)$ | $\log_r N$ | $N/r$ | $(N \log_r N)/r$ | The radix $r$ is a power of 2 |
| Proposed | 4 | 4 | $N$ | 4 | $(\log_2 N)/2$ | $N/4$ | $N(\log_2 N)/8$ | Implemented in DSP48E slices |

† Parallel Process. = Number of data that are processed in parallel. This means how many data are read in parallel from the memories in each clock cycle.
††Related to the radix: Bigger butterflies calculate a larger part of the FFT flow graph at each iteration, reducing the number of iterations.
†††$T_{PROC}$ = processing time, which is the product of the number of iterations and the number of cycles per iterations.

TABLE IV
FIGURES OF MERIT OF THE PROPOSED MEMORY-BASED FFT ON A VIRTEX-5 XC5VSX95T FPGA.

| Parameter | Value |
|---|---|
| N | 4096 |
| Channels | 4 |
| Radix | 4 |
| Iterations | 6 |
| Word Length | 24 + 24 bits |
| $f_{CLK}$ | 200 MHz |
| Input Rate | 1 sample / 20 ns |
| $T_{LOAD}$ | 81.92 $\mu$s |
| $T_{PROC}$ | 61.44 $\mu$s |
| Output Rate | 4 samples / 10 ns |
| $T_{EMPTY}$ | 10.24 $\mu$s |
| Slices | 1135 (7.7 %) |
| Slice LUT | 1407 |
| Slice FF | 1163 |
| DSP48E | 98 (15.3 %) |
| BRAM | 31 (4.2 %) |

been implemented on ASICs. A previous memory-based FFT implemented on FPGAs is shown in [5]. The work in [5] requires 2863 slice LUTs, 2992 slice FF, 24 DSP48E and 8 BRAM. Four of this memory based FFT would use 11452 slice LUTs, 11968 slice FF, 96 DSP48E and 32 BRAM. The number of DSP48E and BRAM are comparable to the 98 DSP48E and 31 BRAM of the proposed design. However, in the proposed approach the use of distributed logic is only 1407 slice LUTs and 1163 slice FF, compared to the 11452 slice LUTs, 11968 slice FF in [5]. This is an important saving in terms of distributed logic, and agrees with the goal of reducing distributed logic in the proposed design. Furthermore, with this hardware the proposed architecture calculates a complex FFT, whereas [5] is only valid for real-valued signals.

## VII. CONCLUSIONS

The proposed 4096-point radix-4 memory-based FFT architecture presents a novel conflict-free access strategy. The new strategy requires the minimum amount of memory and few multiplexers. This reduces the amount of hardware with respect to previous radix-4 memory-based FFTs. Furthermore, the proposed FFT has been implemented efficiently on an FPGA making use of the DSP slices. The proposed design requires less distributed logic than previous results on FPGA, while keeping a comparable amount of DSP slices and BRAM.

## REFERENCES

[1] S. He and M. Torkelson, "Design and implementation of a 1024-point pipeline FFT processor," in *Proc. IEEE Custom Integrated Circuits Conf.*, May 1998, pp. 131–134.

[2] M. Garrido, J. Grajal, M. A. Sánchez, and O. Gustafsson, "Pipelined radix-$2^k$ feedforward FFT architectures," *IEEE Trans. VLSI Syst.*, vol. 21, pp. 23–32, Jan. 2013.

[3] D. Cohen, "Simplified control of FFT hardware," *IEEE T. Acoust. Speech.*, vol. 24, no. 6, pp. 577 – 579, Dec 1976.

[4] Y. Ma and L. Wanhammar, "A hardware efficient control of memory addressing for high-performance FFT processors," *IEEE Trans. Signal Process.*, vol. 48, no. 3, pp. 917 –921, Mar 2000.

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

[6] B. Jo and M. Sunwoo, "New continuous-flow mixed-radix (CFMR) FFT processor using novel in-place strategy," *IEEE Trans. Circuits Syst. I*, vol. 52, no. 5, pp. 911 – 919, May 2005.

[7] X. Xiao, E. Oruklu, and J. Saniie, "Fast memory addressing scheme for radix-4 FFT implementation," in *IEEE Int. Conf. on Electro/Information Tech.*, June 2009, pp. 437–440.

[8] P.-Y. Tsai and C.-Y. Lin, "A generalized conflict-free memory addressing scheme for continuous-flow parallel-processing FFT processors with rescheduling," *IEEE Trans. VLSI Syst.*, vol. 19, no. 12, pp. 2290–2302, Dec. 2011.

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

[10] D. Reisis and N. Vlassopoulos, "Conflict-free parallel memory accessing techniques for FFT architectures," *IEEE Trans. Circuits Syst. I*, vol. 55, no. 11, pp. 3438 – 3447, Nov 2008.

[11] C.-F. Hsiao, Y. Chen, and C.-Y. Lee, "A generalized mixed-radix algorithm for memory-based FFT processors," *IEEE Trans. Circuits Syst. II*, vol. 57, no. 1, pp. 26 –30, Jan 2010.

[12] M. Garrido, "Efficient hardware architectures for the computation of the FFT and other related signal processing algorithms in real time," Ph.D. dissertation, Universidad Politécnica de Madrid, Dec. 2009.

[13] M. Garrido, J. Grajal, and O. Gustafsson, "Optimum circuits for bit reversal," *IEEE Trans. Circuits Syst. II*, vol. 58, no. 10, pp. 657–661, Oct. 2011.

[14] M. Garrido and J. Grajal, "Efficient memoryless CORDIC for FFT computation," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, vol. 2, Apr. 2007, pp. 113–116.