ESKİŞEHİR TECHNICAL UNIVERSITY JOURNAL OF SCIENCE AND TECHNOLOGY
A- APPLIED SCIENCES AND ENGINEERING

2023, 25(1), pp. 156-167, DOI: 10.18038/estubtda.1401022

MEMORY-BASED SELF-ORDERING FFT FOR EFFICIENT I/O SCHEDULING

Zeynep KAYA [1], Erol SEKE [2*]

[1] Dept. of Electical-Electronics Eng., Engineering, Şeyh Edebali University, Bilecik, Turkey
[2] Dept. of Electical-Electronics Eng., Engineering and Architecture, Osmangazi University, Eskişehir, Turkey

ABSTRACT

A complex-valued self-ordering radix-2 memory-based Fast Fourier Transform (FFT) architecture suitable for low end Field Programmable Gate Arrays (FPGA) is presented. Employing a self-ordering algorithm within the data flow, both input and output data are kept in normal sequential order, not in digit-reversed-order. This way, with an appropriate scheduling, last stage of the FFT and I/O operations are performed in parallel with no wait states. Self-ordering FFT algorithms are generally designed for software implementations. We designed and implemented one on FPGA (hardware), showing that considerable number of clock cycle savings can be obtained compared to unordered FFT counterparts. The approach is implemented on various FPGAs. The results are compared with similar radix-2 architectures in terms of required clock cycles and resource usage, confirming the advantage of the approach.

Keywords: FFT, DFT, memory-based FFT, self-ordering, FPGA, radix-2

1. INTRODUCTION

Discrete Fourier Transform (DFT) is almost always implemented as FFT (Fast Fourier Transform) and used in wide range of signal processing applications like high performance communications and image processing. Since it is computationally intensive, hardware implementations find a valuable place when real-time applications require it. Some researchers prefer pipeline FFT architectures to perform continuous data flow and high speed [1,3-5,15]. Other researchers employ memory-based FFT architectures because of the lower resource requirements and lesser occupied chip area [6,12,14,19]. Low resource and low power usage is especially important for hand-held battery powered devices. Figure 1 illustrates the recursive structure of the FFT, which enables designers create various software and hardware implementations depending on the applications' requirements and availability of the resources.

The term memory-based refers to the hardware design that continuously reads relatively small chunks of data from memory for intermediate FFT operations and writes the results back to the memory, repeating this until the complete FFT result appears in the memory. This is similar to software approach but done with hardware with higher parallelism and speed. Many researchers use memories with a capacity of $2N$ [8,18] or greater $2N+$ [14] for N-point FFT, in order to improve speed and/or to avoid memory conflicts. It should be noted that aiming minimum memory (size N) gains importance when N is large and posing problems in small FPGAs. However, it is possible to reduce memory down to N with the efficient addressing algorithms [6,10,12].

Researchers try to improve performance by improving parallelism (reading/writing larger chunks, higher radix FFT) by proposing various memory addressing schemas, by efficient data feed-in/out to/from this structure and/or by reducing the resource (memory, chip area etc.) requirements [11,17,19]. The studies of Xiao *et al.* [20,21] improved the address generation logic that has critical path independent of the transform size, hence suggested for large transforms by the authors. In the study of Ma *et al.* [12], researchers used two processing elements (PEs) working in parallel for radix-2 FFT on

*Corresponding Author: eseke@ogu.edu.tr,
Received: 06.12.2023 Published: 28.03.2024

real-valued input data. Garrido *et al.* [6] proposed a radix-4 algorithm on 4 blocks of memory of size N/4 each. FFT results are inherently in bit-reversed order and may require reordering at the end. However, using high-radix FFT increase the amount of the hardware and parallel PEs complicates the addressing algorithm. In our study, we achieve radix-2 FFT with self-ordering addressing algorithm without any conflict by using simple counters and one PE, in order to show that obtaining already-ordered output reduces both the total number of required clock cycles and memory.

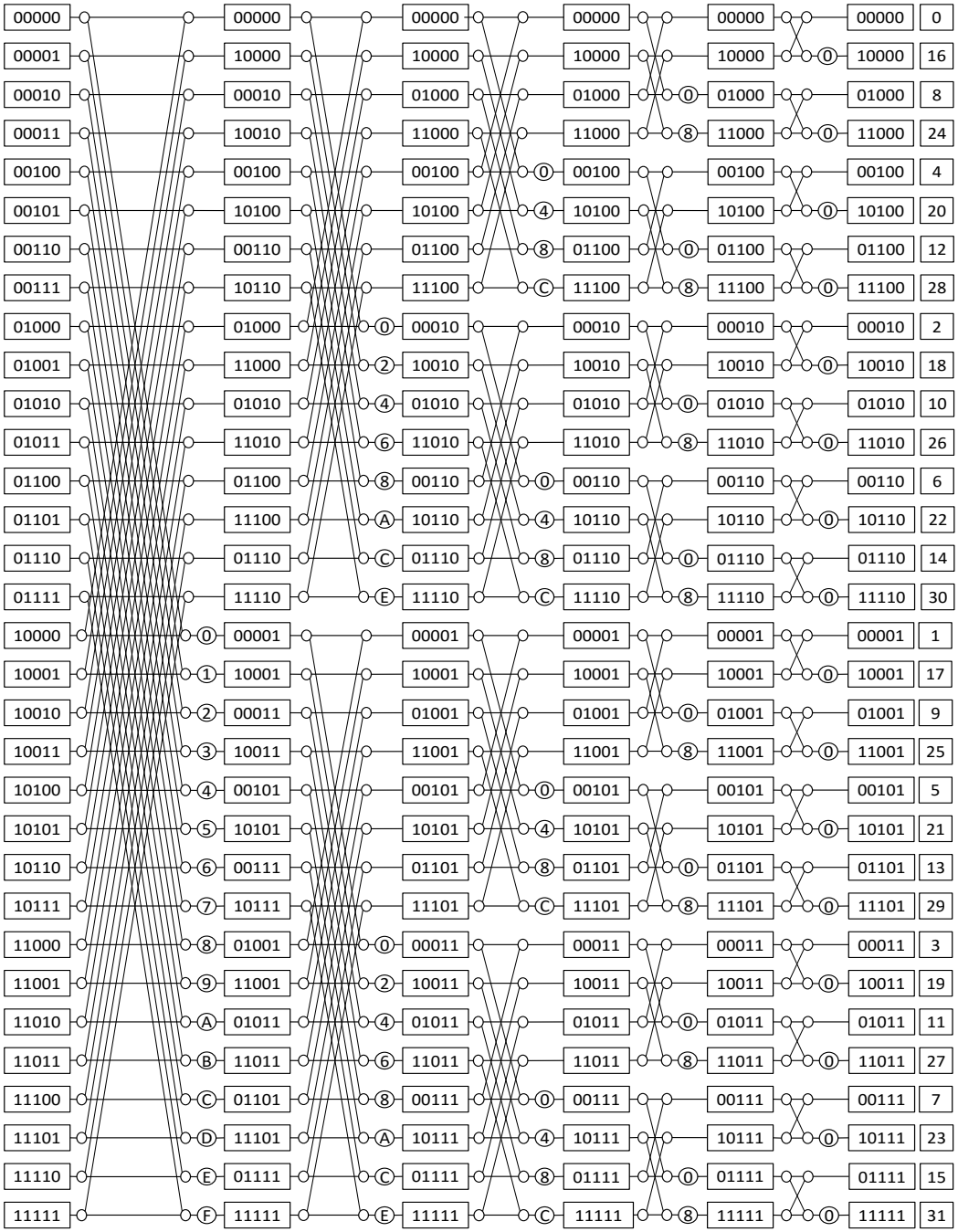


Figure 1. Radix-2 *N*=32 DIF FFT process tree showing actual memory addresses in binary to expose the self-ordering process

Memory addressing schemes are available for general memory-based FFT architectures which facilitate normal ordered FFT outputs [7,14,16]. There exist normal ordered input/output designs [7] implemented

in ASIC for special high throughput requirements. The radix-16 512-point FFT chip in the work of Huang and Chen [7] uses 16 blocks of memory and is not customized for FPGA implementations for different needs. Each memory block has only 32 locations, hardly holding the "memory-based FFT" title. It requires 16 complex floating-point multipliers, not suitable for small FPGAs.

Two methods for self-sorting FFT algorithm are presented in the works of Chu and George [2] and Johnson and Burrus [9]. The work of Chu and George [2] requires in-place swapping of data after each stage, whereas swapping in the latter [9] is performed on the write-addresses. However, both methods are designed for software implementations and is not restricted by resource count limitations faced by hardware implementations. For example, there can never be a memory access conflict in a software implementation.

Currently, all practical FFT implementations on hardware platforms reorder the output data after FFT processing is finished, either by transferring the unordered data to the ordering circuit or performing the sorting in place and preventing the new data loading during the sort. Since the output data is in bit-reversed order, it is not possible to start a new in-place FFT cycle during these operations.

In this study, we aim for re-configurable low-capacity FPGA needs and ordered input/output capability. For ordered input/output, we implemented the methods of [2,9] in hardware by performing stage operations in a special order that prevents memory access conflicts while doing one PE calculation per clock cycle. We took the advantage of ordered i/o and used it for further parallelism by parallelizing input, output and last stage processing.

In the following subsections, basics of memory based FFT and self-sorting algorithm are given briefly. Section 2 details the implemented FFT hardware where each subsection describes a specific part (problems in and solutions developed for) of the circuit. Since the rotator (twiddle factors) generation can be handled as a separate part of the problem, we did not spare a specific section for that as the purpose of this paper is to develop an addressing method for self-ordering and scheduling. However, in implementations we used a quarter cycle look-up table for providing real and imaginary parts of the twiddle factors. Clock cycle comparisons are given with similar FFT hardware implementations in Section 3.

1.1. Memory-Based FFT

Assuming that the data to be processed is already loaded into memory of size-N where N is an integer power of r, radix-r memory based FFT algorithm with single processing element reads r memory locations, performs radix-r FFT on them, and writes the r sub-results back into the memory. Data flow from/to memory for $N = 2^5$=32 radix-2 FFT is given in Figure 1, as an example. Connections within stages show the memory locations read (from left) and written (to right). When radix-2 FFT results are written back to the locations originally occupied by the two-input data of the elementary 2in-2out computation, the final FFT results are normally in bit-reversed order and can be ordered with additional processing and possibly with additional hardware resources. With the implementation of self-sorting, the memory addresses are swapped at each stage as shown in binary in Figure 1. Note that, although the FFT tree shown in Figure 1 remains the same, the output data memory locations are in the same order with the output data index (normal ordered data indexes).

We claim that, in this paper, by applying a self-ordering algorithm within stage processing to obtain the results in normal order, reading the final results out and writing in new data for the next round of FFT can also be done in parallel. That is, data from the last stage calculations need not be written back into memory, instead the ordered data can be sent to output. Simultaneously, since the intermediate data in these memory locations will no longer be needed, new data can be loaded. This means, clock cycles required for data read-in/read-out are completely eliminated from the total clock count. Therefore, we

claim that the benefits of applying a self-ordering algorithm are; ordered data at the output, considerable savings on total clock cycles. It should be noted that, without self-ordered outputs, reading in new input data could not have been performed simultaneously with reading out FFT data as the memory locations to write would have been busy until the end of read-out.

Clock cycle savings can be seen by comparing Figure 2a and Figure 2d. The total clock cycles required for radix-2 N-point FFT is $N/2\,log_2(N)$. In the usual approach, it would have been $N/2\,log_2(N) + N_{load} + N_{unload} + N_{sort}$, where N_{load} and N_{unload} are clock cycles required to load in unprocessed data and unload processed data respectively. N_{load} and N_{unload} are usually equal to the the number of input samples, N. However, by parallelizing flows to/from memory blocks, they can be made equal to N/N_B where N_B is the number of independently accessible memory blocks. N_{sort} is the number of clock cycles to order the output data when the application requires.

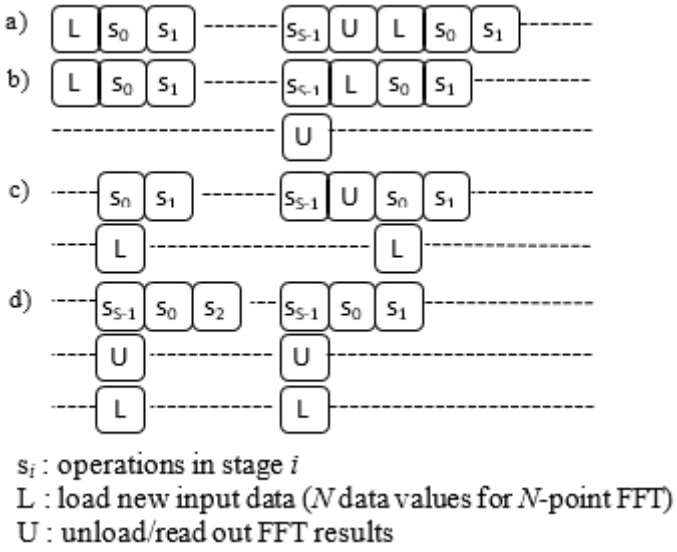


Figure 2. Possible timing diagrams of Memory-Based FFTs, (a) Conceptual timing for basic Load-process-Unload case, (b) Unloading is performed in parallel with the last stage, (c) Loading is performed in parallel with the first stage, (d) Loading, Unloading and the last stages are performed in parallel

1.2. Self-Ordering Algorithm

The method proposed in this paper adapts the “self-sorting in-place” algorithm, described for software FFT implementations [2,9] like Stockham FFT, to limited FPGA (hardware) resources. A difficulty/confusion arouses when the word "cycle" in software implementations is tried to be mapped to "clock cycle" in hardware. In a software cycle, both memory access count and number of calculations are virtually unlimited, whereas FPGA counterpart is limited by the design of the memory component and other resources. For dual-port block memory components in FPGAs, only two addresses can be accessed for reading and/or writing on a single clock pulse.

In this self-sorting schema, $S = log_2(N)$ being the number of address bits and the number of stages in a radix-2 FFT at the same time, two inputs of the radix-2 PE are read from

$$\begin{aligned} R_0 &= a_{S-1}a_{S-2}\cdots a_1a_0 \\ R_1 &= b_{S-1}b_{S-2}\cdots b_1b_0 \end{aligned} \tag{1}$$

addresses where R_0 and R_1 differs only in one bit whose position is determined by the stage number $s = 0 \ldots S-1$. For example, in first stage of a DIF (Decimation in Frequency) FFT implementation

these addresses would be $0a_{S-2}\cdots a_1a_0$ and $1a_{S-2}\cdots a_1a_0$. In the second stage the addresses would be $a_{S-1}0\cdots a_1a_0$ and $a_{S-1}1\cdots a_1a_0$, and so on. However, while writing the PE results back into memory, this differing bit is swapped with its symmetric LSB bit in both R_0 and R_1. For example, the write addresses for the first stage would be $a_0a_{S-2}\cdots a_10$ and $a_0a_{S-2}\cdots a_11$. In the second stage, they would be $a_{S-1}a_1\cdots 0a_0$ and $a_{S-1}a_1\cdots 1a_0$ and so on for stages up to $S_h = int(S/2)$. After stage S_h, since all permutations/relocations are completed, the rest of the stages can be performed by "write where it is read from" approach.

When a complete PE cycle per clock pulse is aimed for radix-r FFT, $2r$ memory accesses are required in a single clock cycle; r reads and r writes. For a radix-2 in-place algorithm, this minimally calls for two dual port memory blocks in which a total of 4 ports can be independently read or written provided that no more than two accesses are required for each. Self-ordering introduces additional addressing problems, on the other hand. We have proven that, with a simple example in appendix A, "self-sorting in-place" algorithm as given by [2,9] for radix-2 FFT aiming one PE per clock cycle cannot be generalized efficiently for hardware implementations with less than four dual port memory-blocks. This is because, at some points of the algorithm, it becomes necessary to access 3 address locations of a dual port memory at the same time, which is not possible. We have discussed this in section "Multi-Block Memory Access".

The following sections describe the proposed solutions to problems in hardware (FPGA) implementation of self-sorting radix-2 FFT.

2. HARDWARE DESIGN OF PROPOSED APPROACH

Widely used illustration of a memory-based FFT architecture diagram that uses dual-port memory blocks is shown in Figure 3. Address generator circuit is the main controller of the operation flow. It generates the physical access addresses for the memory blocks, physical memory block selection/activation signals, multiplexer selection signals to route the appropriate memory output data to PE and proper signals for the twiddle factor generation. It is also necessary to generate input-ready (Ird) and data-ready (Ord) signals to indicate the circuit is ready to receive new input data and ready to spit out the calculated FFT data respectively.

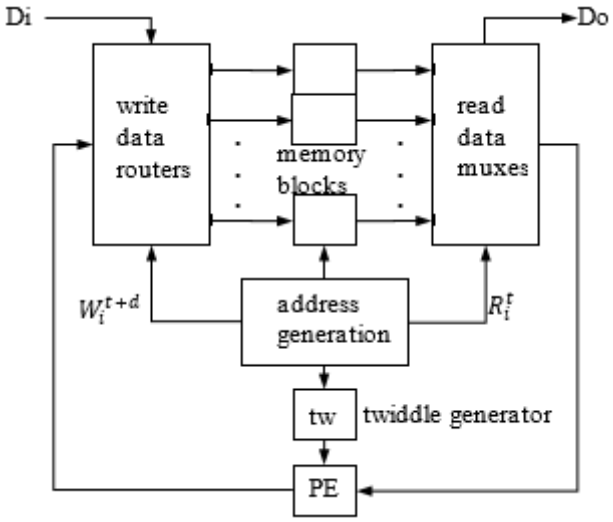


Figure 3. General block diagram of a memory-based FFT circuit. *d* represents an appropriate delay for feeding write addresses to memory blocks after read addresses

The following subsections describe in detail how these signals are generated for the proposed method. We skipped the description of twiddle factor generation as it is optimization-goal dependent. However, in the actual tests, we employed quarter wave look-up table method for practical reasons. That is, sine samples for 1/4th of a full period are stored in a memory look-up table, and the real and imaginary parts of the twiddle factors are generated using symmetry of sine wave.

2.1. Address Generation

The earliest time the PE results can be written back into the memory is the next clock cycle that follows the read operation of its input data. That is, when the memory read and write addresses are (R_0^t, R_1^t) and (W_0^t, W_1^t) where $t = 0,1,2 \ldots$ indicating clock cycle number, (W_0^t, W_1^t) are the write addresses of the PE data calculated using the data obtained by reading addresses (R_0^{t-1}, R_1^{t-1}). It is obvious that the data in the (W_0^t, W_1^t) must have already been read, otherwise they will be overwritten and lost. Since we opted to use a set of data pipeline registers between the complex adders and the multiplier within PE circuit, as shown in Figure 4, a delay of two clock cycles on PE outputs is obtained.

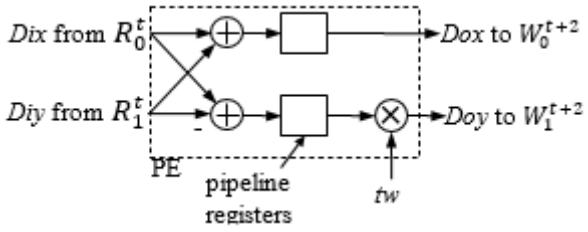


Figure 4. General design of radix-2 PE circuit. W_i^{t+2} addresses are generated two clocks after R_i^t addresses

Consequently, write addresses are also delayed by two clock cycles using address pipeline registers. Let us now analyze the first stage read/write addresses;

$$\begin{aligned} R_0^t &= 0a_{S-2} \cdots a_1 a_0, R_1^t = 1a_{S-2} \cdots a_1 a_0, \\ W_0^t &= a_0 a_{S-2} \cdots a_1 0, W_1^t = a_0 a_{S-2} \cdots a_1 1 \end{aligned} \tag{2}$$

Since the only differing bits are lsb and msb bits (a_0), consecutively performing PE pairs for which only the values of these bits differ, such that;

$$\begin{aligned} R_0^t &= 0a_{S-2} \cdots a_1 0, R_1^t = 1a_{S-2} \cdots a_1 0, \\ W_0^t &= 0a_{S-2} \cdots a_1 0, W_1^t = 0a_{S-2} \cdots a_1 1, \\ R_0^{t+1} &= 0a_{S-2} \cdots a_1 0, R_1^{t+1} = 1a_{S-2} \cdots a_1 1, \\ W_0^{t+1} &= 1a_{S-2} \cdots a_1 0, W_1^{t+1} = 1a_{S-2} \cdots a_1 1 \end{aligned} \tag{3}$$

completes the access requirements for these four addresses for the current stage. For the first PE pair in the first stage of N=32 for example, these addresses would be 00000, 10000, 00001 and 10001. By pairwise processing of PEs, entire first stage can be completed without any need for additional large temporary storage and the stage can be completed in-place.

Logically, we assumed that $a_{S-2} \cdots a_1 a_0$ bits are generated by a binary counter with $S-1$ bits. In that case, for the first stage, the a_0 bit automatically handles the consecutive pairing. For other stages up to S_h, a_0 should be swapped with s^{th} bit, so that pairings described above will occur.

2.2. Multi-Block Memory Access

One or more of the address bits should be assigned as memory-block-selection bits. For example, in a 4 block-memory design, 2 address lines are to be used for that purpose.

We are forced to handle pairwise processing of PEs as explained in the previous section. Write addresses delayed by two clock cycles will not conflict with the current read addresses when a_1 of the counter with the one at the center of address bits. This bit and one of its neighbors (selection is arbitrary) are used as two selection signals for 4 memory blocks. For N=32 for example, a_0 and a_1 are used from counter bits $a_3a_2a_1a_0$. However, when a_1 is one of the swapped bits for self-ordering, it is swapped with a_2 first and used as one of the block selection signals. With an additional bit that identifies one of R_0 or R_1, the final address lines would be $R_0 = 0a_3a_1a_2a_0$ and $R_1 = 1a_3a_1a_2a_0$. The pair a_1a_2 will be used as block selection signals. Write addresses are then, $W_0 = a_0a_3a_1a_20$ and $W_1 = a_0a_3a_1a_21$. Since this approach will work only for odd number of address bits, the proposed schema will handle FFTs with $N = 2^b$ where b is odd (N=32,128,512,2048...). But it will handle FFTs with even b if additional write-only clock cycles are inserted before starting and after finishing the stage $s = S_h$. It adds some complexity in the control circuit, hence we kept this option out of this paper. Complete flow of the address generation circuit is given Figure 5 with one additional bit inversion which will be described in the Section 2.3.

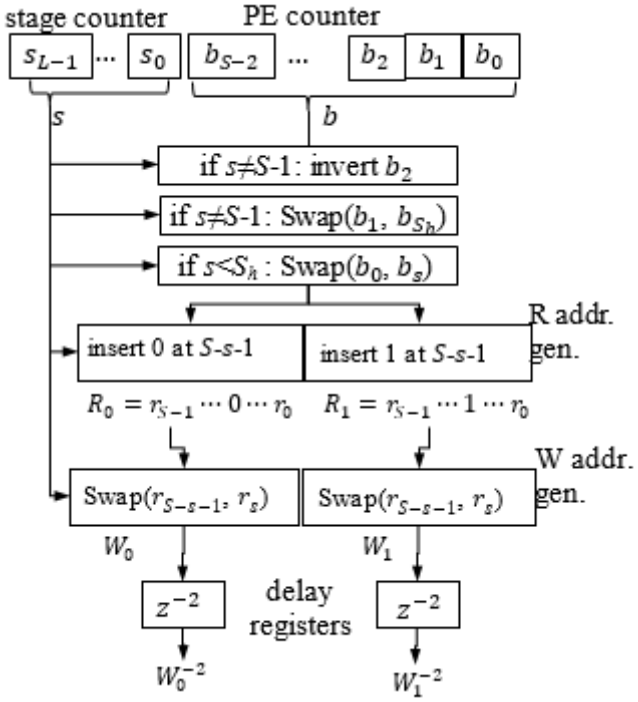


Figure 5. Generation of read/write addresses. Swap and insert functions are implemented as multiplexers (combinatorial logic)

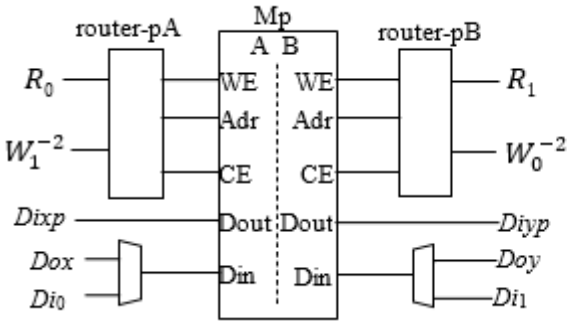
2.3. Address Routing Circuits

With the address bit arrangements described in the previous section, no more than 2 internal address locations are required to be accessed for each dual-port memory block. Accesses to each of 8 ports could be managed by using 4-to-1 multiplexers on each port. Further circuit simplification can be achieved by managing the order of processing at each stage. It is possible to have any single port to see only two of $\dot{R}_0, \dot{R}_1, \dot{W}_0, \dot{W}_1$ at any given cycle, where dotted address notations ($\dot{R}_0, \dot{R}_1, \dot{W}_0, \dot{W}_1$) indicate the physical

addresses (two block selection bits excluded) and necessary delays are applied for write addresses. For example, port A of all memory blocks will see either one of $\dot{R}_0$ or $\dot{W}_1$, or the port is not used in the current cycle. Similarly, the other port (B) will see either $\dot{R}_1$ or $\dot{W}_0$. This arrangement suggests the circuit illustrated in Figure 6, per memory block.

In Figure 6, Mp is one of the four memory blocks where p is either 0, 1, 2 or 3. A and B indicates the two ports of the dual-port memory. mR_0, mR_1, mW_0 and mW_1 are $S_h{}^{\text{th}}$ and 1^{st} bits of the respective addresses. The routers simply multiplexes $\dot{R}$ or $\dot{W}$ inputs to the address lines depending on these bits. Since, with the described address generation algorithm, only one of them is allowed at any time, the routers will allow $\dot{W}$ to pass if its mW bits equal to p (and activate WE -write enable- output), otherwise it will let $\dot{R}$ pass through. The routers will also activate CE (memory enable) signal if any of its address inputs point to p.

Stage transitions (when a new stage is started) pose a problem in managing multiplexer-friendly (use of 2 input mux instead of 4) addressing of block memories. Since consecutive pairings are managed by a_1 counter bit, pair ordering in consecutive stages is handled by just inverting one of the higher counter bits, for example a_2. This is shown in Figure 5 as a first step after generating counter values. These inversion and bit swapping are not performed in the last stage since the last stage must be performed in normal order. Outputs of the last stage will not be written back to memory anyway, instead new data will be loaded in normal order, and no conflict will occur.



router truth table for port A

mW_1	mR_0	Adr	WE	CE
p	-	$\dot{W}_1$	1	1
else	p	$\dot{R}_0$	0	1
else	else	-	0	0

Figure 6. Address routing circuits for pth dual-port memory block. Truth table shows signal routing for port A. All routers are identical except that address inputs of B ports receive R_1 and W_0. p is a two bits value for selecting p^{th} memory block. *mW* and *mR* are $S_h{}^{\text{th}}$ and S_h-1^{th} bits of *W* and *R* inputs of the router. The remaining bits are $\dot{W}_1$ and $\dot{R}_0$. '-' indicate don't-care, therefore PE outputs (*Dox*, *Doy*) can directly be routed to the *Din* inputs except when new data are being filled into memories

Memory output data (*Dixp* and *Diyp*, where p is one of 0, 1, 2 or 3) shall also be multiplexed into PE inputs *Dix* and *Diy* using mR_0 and mR_1 two bit selectors, respectively. However, since data comes out from memories after a clock cycle, mR_0 and mR_1 selectors should also be delayed by 1 clock cycle.

Two additional signals (Ird, Ord) are generated by the address generation circuit given in Figure 5 (these signals are not shown) indicating that the FFT operation is in the last stage so that the calculated results should be taken out from PE outputs and new data for FFT should be provided. Input data multiplexers controlled with this signal are shown in Figure 6.

This completes the general design of the radix-2 self-ordered in-place FFT with the total clock cycles of

$$N_{clk} = N/2 \log_2(N) \tag{4}$$

and no additional clock cycles are required for re-ordering and data inputs & outputs.

3. RESULTS AND COMPARISONS

A general radix-2 complex-valued FFT schema contains $N/2$ PE in each of $log_2 N$ stages, making minimal clock count of $\frac{N}{2} log_2 N$, assuming that one PE calculation is performed in each clock cycle. This excludes clock cycles required for loading new data and unloading calculated FFT output. Since our proposed self-ordering method does loading, unloading and the last FFT stage all in parallel, the number of overhead clock cycles is zero. That is, requiring no overhead clock cycles, the number of clock cycles between new data loadings is $\frac{N}{2} log_2 N$. Clock cycles given in Table 1 does not include overhead clock cycles (but marked as T_{lu}) as referenced methods do not provide such numbers.

In Table 1, the work of Ma and Wanhammar [13] exhibits the minimum clock cycles, but do not include overhead cycles and the output data is in bit-reversed order. The method in [8] has the lowest cycles, however again, it does not include additional clock cycles and uses mixed radix. It also uses $2N$ memory. The method of Ma *et al.* [12] on the other hand uses N memory locations but designed for real-valued inputs. It is naturally expected to achieve half the required clock cycles for complex-valued FFTs anyway. Our proposed method requires lowest possible clock cycles in its category. It can be easily implemented on FPGAs with low resource counts.

Since the main target is low resource small FPGAs, the proposed schema is designed in VHDL for N=32, 128, 512 and 2048 and tested on Spartan 3E and Virtex-6 FPGAs, feeding it with stored data samples and real-time sampled data and observing the output. Input and output data were 18 bit fixed-point. This wordlength selection is arbitrary and has no effect on number of clock cycles. In tests/experiments on Spartan 3E, design has achieved 115 MHz clock frequency for N=512 with 116 FFs, 754 4-input LUTs. On Virtex-6, the clock frequency was 186 MHz for N=2048 with 174 slice registers and 607 LUTs. Obviously, clock speeds are not impressive but accepted as reasonable since no clock speed improvement measures (like pipelining combinatorial circuits) were taken in the design and the devices are low end. It was notable that N=2048 fixed-point FFT comfortably fits in a Spartan XC3S100E device. Since the initial target was low end FPGAs, the choice of radix-2 was appropriate as they have limited number of DSP-slices/multipliers. Therefore, we compared the method with the radix-2 methods found in the literature.

Table 1. Comparison with previous radix-2 memory-based methods

method	radix, #banks,M	#clock cycles	#clock cycles for N=2048	#cycles with parallel loading/unloading to 4 memory blocks
Ma and Wanhammar [13]	2,4,N	$\frac{N}{2}\log(N)+T_{lu}$	11264+T_{lu}	12288
Jo and Sunwoo [8] [1]	2-4,4,2N	$\frac{N}{8}\log(N)+T_{lu}$	3840+T_{lu}	4864
Ma *et al.* [12] [2]	2,4,N	$\frac{N}{4}\log(N)-\frac{N}{4}+1+T_{lu}$	6145+T_{lu}	7169
proposed[3].	2,4,N	$\frac{N}{2}\log(N)$	11264	11264

T_{lu}: clock cycles required for i/o and possible re-ordering
$T_{lu} = N/N_B$, N_B: number of memory blocks
[1]: radix-2 and radix-4 mixed (large PE), 2N memory
[2]: real-valued inputs
[3]: includes i/o clock cycles

All the methods given in Table 1 have similar/comparable resource utilization [8] uses twice the memory that of the others. It should be noted that the proposed method is not intended to reduce number of clock cycles spent in performing the actual FFT calculations. However it eliminates the clock cycles required

to load input data and unload the results after FFT. These clock cycles are shown as T_{lu} in Table 1. The numbers in the rightmost column of Table 1 gives the total number of clock cycles required for loading data, processing (FFT) them and reading out the processed data, assuming that all 4 memory blocks are loaded and unloaded in parallel. Even the state-of-the-art references do not discuss on these overhead cycles and assume that the input data is already loaded. It is generally $T_{lu} = 2N/N_B$ where N_B is the number of memory banks that can be read/written in parallel (assuming that no ordering is required). Proposed method requires load/unload processes too, but they are done in parallel to FFT calculation whereas in other methods parallel load/calculate/unload is not possible due to unordered placement of data in the memory.

In section 2.2, it is stated that for cases of $N = 2^b$ where b is even, additional clock cycles must be inserted between stages. For a particular case of N =1024, there are 10 stages and therefore 20 additional write-only clock cycles are needed to prevent addressing conflicts. This will make the total number of clock cycles 5140, which is 492 clock cycles smaller than the required clock cycles for [13]. This means that, inserting additional clock cycles does not really affect the performance but slightly increase the chip area usage due to the needed management circuitry for write-only cycles. In our example implementation, the increase was less than 1%, not even making into synthesizer statistics.

4. CONCLUSION

A radix-2 memory-based FFT architecture with normal ordered input and output data and a novel application of a self-sorting algorithm is presented and experimented. Small FPGAs with low resource counts are targeted. With a single complex multiplier, it achieves minimum possible clock cycles ($\frac{N}{2}\log_2(N)$) between FFT cycles by eliminating clock cycles required for data feed-in and read-out. FFT processing circuit is never in a wait-for-data state. For comparisons with other memory-based radix-2 designs with minimum required memory of N and minimum number of clock cycles $\frac{N}{2}\log_2(N)$, it should be noted that it is not possible to do data feed-in and read-out simultaneously because of the digit-reversed ordering nature of the FFT algorithm. This is achieved with our proposed schema by embedding a self-ordering algorithm into the design. Although the proposed schema works for only FFTs with $N = 2^b$ where b is odd, it works for even bs with a few write cycles inserted between stages. It is obvious that this radix-2 FFT architecture with efficient self-sorting addressing method provides savings in total required clock cycles and reduction in hardware resources. It is also obvious that radix-2^k and quite possible that higher radix FFTs can be implemented using the proposed approach, by increasing the number of memory blocks accordingly.

CONFLICT OF INTEREST

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

AUTHORSHIP CONTRIBUTIONS

Authors have equal contribution in design of the system, HDL testing and manuscript preparation.

REFERENCES

[1] Chang YN, Parhi KK. An efficient pipelined FFT architecture, IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing 2003; 50-6: 322–325.

[2] Chu E, George A. FFT algorithms and their adaptation to parallel processing, Linear Algebra and its Applications 1998; 284-1-3: 95–124.

[3] Garrido M. A survey on pipelined FFT hardware architectures, Journal of Signal Processing Systems 2021; https://doi.org/10.1007/s11265-021-01655-1.

[4] Garrido M, Grajal J, Sanchez MA. Gustafsson O, Pipelined radix-2k feedforward FFT architectures, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 2013; 21: 23–32.

[5] Garrido M, Moller K, Kumm M. World's fastest FFT architectures: Breaking the barrier of 100 GS/s, IEEE Transactions on Circuits and Systems I: Regular Papers 2019; 66-4: 1507–1516.

[6] Garrido M, Sanchez MA, Lopez-Vallejo ML, Grajal J. A 4096-point radix-4 memory-based FFT using DSP slices, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 2016; 25: 375–379.

[7] Huang SJ, Chen SG. A high-throughput radix-16 FFT processor with parallel and normal input/output ordering for IEEE 802.15.3C Systems, IEEE Transactions on Circuits and Systems I: Regular Papers 2012; 59-8: 1752–1765.

[8] Jo BG, Sunwoo MH. New continuous-flow mixed-radix (CFMR) FFT processor using novel in-place strategy, IEEE Transactions on Circuits and Systems I: Regular Papers 2005; 52-5: 911–919.

[9] Johnson H, Burrus C. An in-order, in-place radix-2 FFT, ICASSP '84. IEEE International Conference on Acoustics, Speech, and Signal Processing 1984.

[10] Kaya Z, Seke E, A novel addressing algorithm of radix-2 FFT using single-bank dual-port memory, Circuit World 2020; 48: 64–70.

[11] Luo HF, Liu Y-J, Shieh M-D..Efficient memory-addressing algorithms for FFT processor design, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 2015; 23-10: 2162–2172.

[12] Ma ZG, Yin XB, Yu F. A novel memory-based FFT architecture for real-valued signals based on a radix-2 decimation-in-frequency algorithm, IEEE Transactions on Circuits and Systems II: Express Briefs 2015; 62-9: 876–880.

[13] Ma Y, Wanhammar L, A hardware efficient control of memory addressing for high-performance FFT processors, IEEE Transactions on Signal Processing 2000; 48-3; 917–921.

[14] Mao XB, Ma ZG, Yu F, Xing QJ. A continuous-flow memory-based architecture for real-valued FFT, IEEE Transactions on Circuits and Systems II: Express Briefs 2017; 64-11; 1352–1356.

[15] Nguyen NH, Khan SA, Kim CH, Kim JM. A high-performance, resource-efficient, reconfigurable parallel-pipelined FFT processor for FPGA platforms, Microprocessors and Microsystems 2018; 60: 96–106.

[16] Sorokin H, Takala J. Conflict-free parallel access scheme for mixed-radix FFT supporting I/o permutations, 2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) 2011.

[17] Takala JH, Jarvinen TS, Sorokin HT. Conflict-free parallel memory access scheme for FFT processors, Proceedings of the 2003 International Symposium on Circuits and Systems, 2003.

[18] Tsai PY, Lin CY. A generalized conflict-free memory addressing scheme for continuous-flow parallel-processing FFT processors with rescheduling, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 2011; 19-12: 2290–2302.

[19] Valencia D, Alimohammad A. Compact and high-throughput parameterisable architectures for memory-based FFT algorithms, IET Circuits, Devices and Systems 2019; 13-5: 696–703.

[20] Xiao X, Oruklu E, Saniie J, An efficient FFT engine with reduced addressing logic, IEEE Transactions on Circuits and Systems II: Express Briefs 2008; 55-11: 1149–1153.

[21] Xiao X, Oruklu E, Saniie J. Fast memory addressing scheme for radix-4 FFT implementation, IEEE International Conference on Electro/Information Technology 2009.