Optimal Dynamic Data Layouts for 2D FFT on 3D Memory Integrated FPGA

Ren Chen, Shreyas G. Singapura and Viktor K. Prasanna*

University of Southern California, Los Angeles, CA 90089, USA
{renchen, singapur, prasanna}@usc.edu

Abstract. FPGAs have been widely used for accelerating various applications. For many data intensive applications, the memory bandwidth can limit the performance. 3D memories with through-silicon-via connections provide potential solutions to the latency and bandwidth issues. In this paper, we revisit the classic 2D FFT problem to evaluate the performance of 3D memory integrated FPGA. To fully utilize the fine grained parallelism in 3D memory, optimal data layouts so as to effectively utilize the peak bandwidth of the device are needed. Thus, we propose dynamic data layouts specifically for optimizing the performance of the 3D architecture. In 2D FFT, data is accessed in row major order in the first phase whereas, the data is accessed in column major order in the second phase. This column major order results in high memory latency and low bandwidth due to high row activation overhead of memory. Therefore, we develop dynamic data layouts to improve memory access performance in the second phase. With parallelism employed in the third dimension of the memory, data parallelism can be increased to further improve the performance. We adopt a model based approach for 3D memory and we perform experiments on the FPGA to validate our analysis and evaluate the performance. Our experimental results demonstrate up to **40**x peak memory bandwidth utilization for column-wise FFT, thus resulting in approximately **97%** improvement in throughput for the complete 2D FFT application, compared to the baseline architecture.

1 Introduction

FPGAs have been used as accelerators for many applications such as Signal Processing, Image Processing, Packet classification etc. The general purpose processors cannot keep up with the demands of these applications in terms of performance. Even with the high performance of FPGAs, meeting the throughput requirement of these applications is a challenging task. Most of the applications are data intensive and this translates to frequent accesses to the memory. The bottleneck in these cases is the low bandwidth and high latency of the memory.

3D memory has been widely studied in the research community with the high bandwidth and short latency access being the important parameters. 3D memories consist of stack of layers connected using Through Silicon Vias (TSVs) [9].

* This material is based in part upon work supported by the National Science Foundation under Grant Number ACI-1339756

The high speed vertical TSVs along with the third dimension of memory result in short latencies and packs in large memory sizes compared to the conventional 2D memories. Although 3D memories are expected to provide **10**× bandwidth compared to 2D memory, this is subject to the ideal conditions. These include data layouts which reduce row activation overhead, high page hit rate for stride access, etc. These problems are similar to the issues in the conventional planar memories. But, employing the solutions in the context of 3D memory is not trivial due to the structure and organization of 3D memory.

In this paper, we target 2D FFT application on 3D memory integrated FPGA and evaluate its performance with throughput and latency as the target metrics. 2D FFT is a data intensive application with stride memory access patterns. 2D FFT consists of two phases and the access patterns in the two phases require mutually conflicting data layouts. The ideal data layout in the first phase is row major data layout whereas, the second phase requires a column major data layout. Therefore, a static data layout trying to improve the performance in one phase will lower the performance in the other phase. The main reasons for this low performance are high number of row activations and low page hit rate. Therefore, with a static data layout the true capability of 3D memory cannot be realized. We address this problem by extending our solution of dynamic data layouts [6] to 3D memory. The main contributions in this paper are:

1. Model the 2D FFT application on 3D memory integrated FPGA.
2. Develop optimal dynamic data layouts to optimize performance of 2D FFT on 3D memory.
3. Evaluation of optimized and baseline implementation with throughout and latency as the performance metrics.

2 Related Work

As the well-known simplest multidimensional FFT algorithm, the row-column algorithm has been commonly used to implement 2D FFT by performing a sequence of 1D FFTs [10,15]. In this algorithm, input elements hold by an $\boldsymbol{N} \times \boldsymbol{N}$ array are stored in row-major order in the external memory such as DRAM. One major issue in the implementation of the 2D FFT architecture is the considerable delay caused by DRAM row activation which are mainly introduced by the strided memory access in the column wise 1D FFTs. To solve this problem, the authors in [2] propose a tiled data mapping method to improve the external memory bandwidth utilization. They logically divide the input $\boldsymbol{N} \times \boldsymbol{N}$ input array into $\frac{\boldsymbol{N}}{\boldsymbol{k}} \times \frac{\boldsymbol{N}}{\boldsymbol{k}}$ tiles and map the elements in each tile to consecutive memory locations. They conclude that the DRAM bandwidth utilization is maximized when the size of each tile is set to be the size of the DRAM row buffer. However, this solution introduces non-trivial on-chip hardware resource cost for local transposition. Various traditional 2D memory based 2D FFT architectures achieving high throughput performance have been developed in [10,16]. In [10], the authors propose a 2D decomposition algorithm which enables local 2D FFT on sub-blocks. In this way, the times of DRAM row activation is

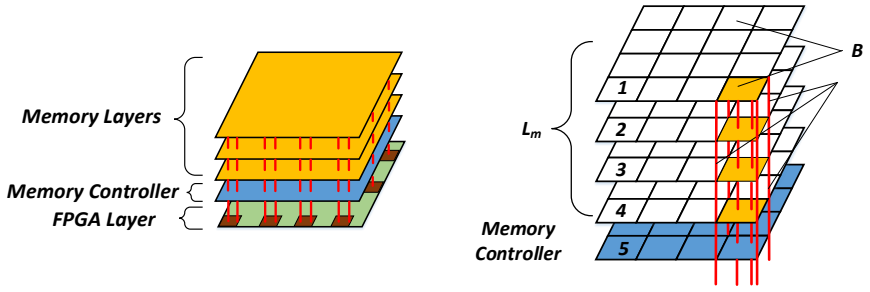

Fig. 1: (a) 3D MI-FPGA Architecture (b) 3D Memory

minimized. Vector radix 2D FFT in [16] presents a general structure theorem to construct a multi-dimensional FFT architecture by decomposing a large size problem into small size 2D FFTs. The external memory row activation overhead is not considered.

3D memory is expected to provide bandwidth higher than the 2D memory by an order of magnitude. There have been many works which have focused on this aspect of 3D memory. [17] implements matrix multiplication and 2D FFT on a Logic-in-Memory architecture. The architecture consists of a logic layer is interleaved between two segments of memory layers to form a 3D architecture. The performance metrics are energy efficiency and bandwidth. In [8], the authors develop power efficient FFT on an architecture consisting of memory layers stacked on multiple FPGA layers. The authors focus on energy efficiency improvement while moving to a 3D architecture from a 2D architecture.

3 3D Memory Integrated FPGA (3D MI-FPGA)

Our model of 3D architecture consists of 3D memory integrated with FPGA interacting through TSVs. We extend our previous work on 3D architectures [13, 14]. Here, we provide a brief overview of 3D Memory Integrated FPGA (3D MI-FPGA). The architecture consists of three components: 3D memory, FPGA and TSVs. Fig. 1 illustrates the architecture of 3D memory integrated FPGA. The memory is composed of several layers ($\boldsymbol{L}$) vertically stacked one above the other. Each of these layers is partitioned into several banks. Vaults are defined as the group of banks (1, 2, 3, 4 in Fig. 1) across layers which share a set of interconnects (TSVs). This set of banks residing on one layer which belong to the same vault ($\boldsymbol{B}$) is analogous to the number of banks in a chip in the 2D memory. The reason being these set of banks share the bus in 2D memory and they share the TSVs in the 3D memory architecture. This set of TSVs shared by the banks in a vault is denoted by $\boldsymbol{N_{tsv}}$. Each vault has a dedicated memory controller which handles the memory accesses to that particular vault. These memory controllers form a separate layer in the memory. Vaults can be activated at the same time as they do not share the TSVs. On the other hand, the banks in a given vault share the TSVs and the activation of these banks has to be

pipelined or interleaved as in the case of 2D memory. Denoting by $\boldsymbol{BW_{vault}}$ the bandwidth of a vault, the total bandwidth of 3D memory is $\boldsymbol{V \times BW_{vault}}$. The FPGA architecture is similar to that of the conventional FPGA consisting reconfigurable logic, DSP blocks, on-chip memory (Block RAM and Distributed RAM) and memory controllers. The difference is that we model the FPGA to interact with the memory through the set of TSVs connecting the FPGA and the memory. These TSVs are between memory controllers on FPGA and those in the memory. FPGA accesses the data in the memory through the TSVs which are high speed, low latency vertical interconnects. The TSVs are characterized by the number of TSVs and latency of data transfer across them. These two parameters affect the amount of data that can be transferred between memory and FPGA in a given unit of time. Each TSV can transfer 1 bit of data at a time. Therefore, higher the number of TSVs, higher the bandwidth.

3.1 Timing Parameters

Bandwidth and latency of accesses to the 3D memory depend on a certain set of timing parameters and we discuss these in this section. Data in the 3D memory is stored in rows which combine to form a bank and which group together to form a vault. Therefore, each row belongs to a specific bank and vault. When memory is accessed, depending on the address a specific row, bank and vault are activated. Therefore, although some of the parameters overlap with that of the 2D memory, certain additional parameters have to be defined taking into account the architecture and different accesses possible in the context of a 3D memory. We model the 3D memory using the following parameters:

1. $\boldsymbol{t_{diff-row}}$: minimum time required between issuing two successive activate commands to different rows in the same bank
2. $\boldsymbol{t_{diff-bank}}$: minimum time required between successive activate commands to different rows in different banks in same or different vaults
3. $\boldsymbol{t_{in-row}}$: minimum time required between successive accesses to elements in the same row in the same bank
4. $\boldsymbol{t_{in-vault}}$: minimum time required between accesses to different rows in different banks in the same vault

The values of the above parameters have a significant impact on latency and bandwidth of the 3D memory. In general, accessing data from different vaults causes zero latency. Hence, a parameter such as $\boldsymbol{t_{diff-vault}}$ is not defined. This is because, since vaults are completely independent and can be active at the same time, this parameter is equal to zero. Since the banks located in different layers but belonging to the same vault can be activated in a pipeline, this latency ($\boldsymbol{t_{in-vault}}$) is lower than that of accessing data from banks belonging to the same layer and same vault. Other parameters are similar to the parameters of 2D memory. Therefore, accessing data from the same row in a bank ($\boldsymbol{t_{in-row}}$) is faster than accessing data from two rows in different banks ($\boldsymbol{t_{diff-bank}}$). The highest latency is seen when we access data from two different rows in the same bank in the same vault denoted by $\boldsymbol{t_{diff-row}}$.

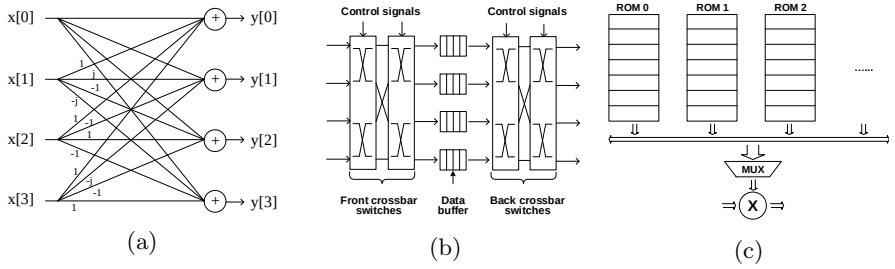

Fig. 2: (a) Radix-4 block (b) Data permutation unit (c) Twidlle factor coeffcient unit

4 2D FFT Architecture

4.1 1D FFT Kernel

An N-point (floating-point) 1D FFT kernel is implemented by concatenating several basic components including radix block, data path permutation (DPP) unit, and twiddle factor computation (TFC) unit. The design of each architecture component relies on the FFT algorithm in use. Implementation details of those components will be introduced next. We applied several energy optimizations discussed in [3–5] onto the design components to reduce their energy consumption. The 1D FFT kernel supports processing continuous data streams so as to maximize design throughput and the memory bandwidth utilization.

Radix block The radix block is used to perform a butterfly computation on some input samples. For example, the radix block for radix-4 FFT takes four input samples, performs the butterfly computation and then generates four results in parallel. Each radix block is composed of complex adders and subtractors. The structure of a radix block is determined by the FFT algorithm in use. Fig. 2a shows the structure of radix block for radix-4 FFT.

DPP Unit DPP unit is used for data permutation between butterfly computation stages in FFT. A DPP unit is composed of multiplexers and data buffers. In subsequent clock cycles, data from previous butterfly computation stage are first multiplexed and written into several data buffers. Each stored data element will be buffered with a certain number of clock cycles and then read out. Outputs from data buffers will also be multiplexed and fed into the next butterfly computation stage. Fig. 2b shows the DPP unit used for a radix-4 based FFT design. Each DPP unit consists of eight 4-to-1 multiplexers and four data buffers. In each cycle, a data buffer may be read and written simultaneously on different addresses. The size of each data buffer depends on the ordinal number of its present butterfly computation stage and the FFT problem size. Note that each data element is a complex number including both its real part and imaginary part, hence the data width is 64 bit.

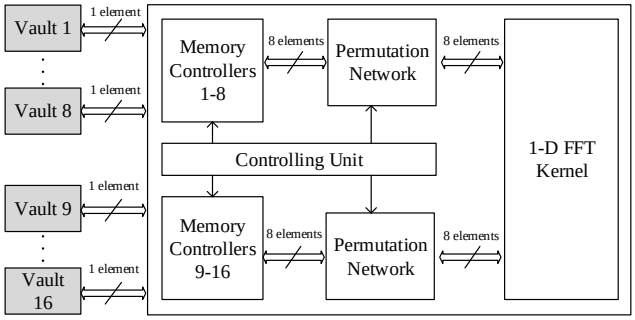

Fig. 3: 2D FFT Processor Architecture

TFC Unit A TFC unit consists of two parts: the TFC generation logic and the complex number multiplier. As shown in Fig. 2c, the TFC generation logic includes several lookup tables (functional ROMs) for storing twiddle factor coefficients, where the data read addresses will be updated with the control signals. The size of each lookup table is determined by the ordinal number of its present butterfly computation stage and the FFT problem size. Each lookup table can be implemented using a BRAM or distributed RAM (dist. RAM) on FPGA [1]. Each complex number multiplier consists of four real number multipliers and two real number adders/subtractors.

The proposed 2D FFT architecture is shown in Fig. 3, in which a controlling unit (CU) and a permutation network are introduced. The permutation network is developed based on our work in [7]. The CU is responsible for reconfiguring the permutation network to achieve the dynamic data layout.

4.2 Baseline Architecture

In baseline architecture, when performing column-wise 1D FFTs, memory address is increased with a stride equals to FFT problem size $\boldsymbol{N}$ after each memory access. However, a minimum activate-to-activate delay exists when successively accessing two rows in the same bank, same vault or accessing in two banks in the same vault. This delay results in a decline in 3D memory bandwidth utilization, thus the entire system throughput is impaired.

4.3 Optimized Architecture

In the optimized architecture, the controlling unit is responsible for reconfiguring the permutation network dynamically to ensure data results of row-wise 1D FFTs are mapped onto the different vaults using the optimal dynamic data layout. Through this data remapping, vault row activation will be only needed after several successive accesses on the same row rather than every memory access. Thus, the impact of vault row activations on the entire system throughput

will almost be minimized. Furthermore, to reduce the times of vault row activation, data inputs of several consecutive column-wise 1D FFTs will be moved from vaults to local memory together, without waiting for the completion of the current executed 1D FFT.

4.4 Optimal Dynamic Data Layouts

Our work in this paper is based on the dynamic data layouts (DDL) developed for the traditional 2D external memory in [12]. In this approach, the data layout in memory is dynamically reorganized during computation. After reorganizations, non-unit stride accesses are converted to unit stride accesses, thereby reducing cache misses. The data layout is *optimal* from the performance point of view as it maximizes the memory bandwidth utilization. However, the data reorganization overhead with regarding to latency and on-chip SRAM buffer consumption has not been considered. In [6] , we proposed the *optimal dynamic data layouts* for 2D memory such that peak memory bandwidth utilization is achieved with minimal data reorganization overhead. The data reorganization overhead is evaluated using the reorganizing latency and the on-chip buffer consumption. We further optimized our approach in [6] so that this technique is applicable for 3D memory based architecture. In the baseline, row major order data layouts are employed. In our approach, instead of mapping results of row-wise FFTs to 3D memory in row major order, we employ block-based dynamic data layout, and the results are read block-by-block by the column-wise 1D FFT. The dynamic data layout is organized into blocks, each of size $\boldsymbol{w} \times \boldsymbol{h}$. $\boldsymbol{w}$ and $\boldsymbol{h}$ represent the width and height, respectively. $\boldsymbol{w}$ is dynamically determined by the stride permutation to be performed in 1D FFTs. We assume the row buffer size in each 3D memory vault is $\boldsymbol{s}$, the number of memory banks in each vault is $\boldsymbol{b}$, the number of valuts to be accessed in parallel is $\boldsymbol{n_v}$. To achieve the *optimal dynamic data layout*, $\boldsymbol{h}$ is calculated based on the equation below:

$$\boldsymbol{h} = \begin{cases} \boldsymbol{n_v \cdot sb/m} & \text{if } \boldsymbol{0 < m < sb\frac{t_{diff\_row}}{t_{in\_row}}}; \\ \boldsymbol{n_v \cdot t_{diff\_bank}/t_{in\_row}} & \text{if } \boldsymbol{sb\frac{t_{diff\_row}}{t_{in\_row}} \leq m < sb}\ ; \\ \boldsymbol{n_v \cdot t_{diff\_row}/t_{in\_row}} & \text{if } \boldsymbol{m \geq sb.} \end{cases} \tag{1}$$

Note that $\boldsymbol{w = s/h}$. The permutation network will be employed for permuting the data in these blocks locally. Due to the limitation of space, we cannot give all the relevant details. For more information, please refer our previous work in [6].

4.5 Metrics of Evaluation

We evaluate the performance of 2D FFT on 3D memory integrated FPGA with respect to the metrics throughput and latency for the entire application.

Throughput: defined as the maximum bandwidth of the memory supported by the application. It is measured in Giga Bytes per second (GB/s). Since our architecture is streaming data every cycle, the bandwidth at which memory

Table 1: Throughput Comparison: Column-wise FFT

	1024 × 1024 2D FFT	**4096 × 4096** 2D FFT	**8192 × 8192** 2D FFT
Throughput of column-wise FFT (Baseline)	6.4 Gb/s	3.2 Gb/s	3.2 Gb/s
Peak bandwidth utilization	1.00%	0.5%	0.5%
Throughput of column-wise FFT (Optimized)	32 GB/s	25.6 GB/s	23.04 GB/s
Peak bandwidth utilization	40.0%	32.0%	28.8%

operates determines the total execution time of the application. Therefore, higher the throughput, lower the execution time.

Latency: defined as the time elapsed between accessing first input from the memory and the time at which the first output is generated by the FFT kernel. We measure latency in the unit of ns. This penalty is paid just once and at the beginning of the processing. As we employ a streaming architecture, after the first output is generated, the subsequent outputs are generated every cycle of operation.

5 Experimental Results

Before evaluating the performance of the entire design, we separately estimate the throughput for both the baseline and the optimized architecture for the 3D architecture described in Section 3. Table 1 shows the throughput performance of the 3D memory before and after the proposed optimization. There is no much performance difference between the baseline architecture and the optimized architecture regarding memory access by row-wise 1D FFTs. The reason for that is the system throughput is almost not affected by row-wise 1D FFTs in both architectures. From the Table 1, it shows that performance loss for column-wise FFT increases with a larger problem size. Through the proposed optimization, the peak bandwidth utilization is improved to 40.0%, 32.0%, and 28.8% for **1024 × 1024**, **4096 × 4096** and **8192 × 8192** size 2D FFTs respectively.

In order to give a thorough view of the performance of the complete 2D FFT implementation, we evaluate the entire system architecture based on our memory model and our actual implementation of 2D FFT design on FPGA. Table 2 presents the throughput and latency performance comparison between the baseline 2D FFT architecture and the optimized 2D FFT architecture. It shows that the optimized 2D FFT architecture achieves 32.0, 25.6 and 23.0 GB/s in throughput for **1024 × 1024**, **4096 × 4096** and **8192 × 8192** problem sizes, respectively. The throughput performance is improved by 95.1%, 97.0%, 96.6% for **1024×1024**, **4096×4096** and **8192×8192** point 2D FFT, respectively. The

Table 2: Performance Comparison: Entire 2D FFT application

	Baseline architecture			Optimized architecture			
FFT size	Throughput (GB/s)	Latency (ns)	Data Parallelism # elements	Throughput (GB/s)	Latency (ns)	Data Parallelism # elements	Performance improvement (throughput)
1024 × 1024	16.4	1.60 ms	32	32.0	524 μs	32	95.1%
4096 × 4096	13.0	7.48 ms	32	25.6	2.4 ms	32	97.0%
8192 × 8192	11.7	145.4 ms	32	23.0	46.6 ms	32	96.6%

latency is reduced by up to 3x by using our proposed optimizations. Comparing the results of the throughput in the 1D FFT kernel and the entire 2D FFT architecture, we observe that the optimization for 3D memory access makes a major contribution in the performance improvement. Moreover, the sustained throughput of the optimized 2D FFT architecture achieves up to 40% of the *peak memory bandwidth*, which is an upper bound on the performance of the chosen FFT algorithm and 3D system architecture. Note that when calculating the *peak memory bandwidth*, we ignored the run-time behavior of the target applications.

6 CONCLUSION

In this paper, we proposed dynamic data layout optimizations to obtain a high throughput 2D FFT architecture on 3D memory integrated architecture. The proposed architecture achieves high throughput by maximizing and balancing the bandwidth between the external memory and FFT kernel on FPGA. By proposing the dynamic data layouts realized with the on-chip permutation network, the delay caused due to row activation overhead is highly reduced, thus leading to significant performance improvement. The experimental results comparing with the baseline architecture show that our implementation outperforms in throughput and latency. In the future, we plan to build a design framework targeted at throughput-oriented signal processing kernels, which enables automatic data layout optimizations addressing new 3D memory technologies.

References

1. Virtex-7 FPGA Family.
http://www.xilinx.com/products/virtex7
2. Akin, B., Milder, P., Franchetti, F., Hoe, J.: Memory Bandwidth Efficient Two-Dimensional Fast Fourier Transform Algorithm and Implementation for Large Problem Sizes. In: 20th International Symposium on Field-Programmable Custom Computing Machines,. pp. 188–191. IEEE (April 2012)

3. Chen, R., Park, N., Prasanna, V.K.: High Throughput Energy Efficient Parallel FFT Architecture on FPGAs. In: IEEE High Performance Extreme Computing Conference (HPEC). pp. 1–6. IEEE (2013)
4. Chen, R., Prasanna, V.K.: Energy-Efficient Architecture for Stride Permutation on Streaming Data. In: International Conference on Reconfigurable Computing and FPGAs. pp. 1–7 (2013)
5. Chen, R., Prasanna, V.K.: Energy efficient parameterized FFT architecture. In: International Conference on Field-programmable Logic and Application. pp. 1–7. IEEE (2013)
6. Chen, R., Prasanna, V.K.: DRAM Row Activation Energy Optimization for Stride Memory Access on FPGA-based Systems. In: International Conference on Applied Reconfigurable Computing, pp. 349–356. Springer (2015)
7. Chen, R., Prasanna, V.K.: Energy and Memory Efficient Bitonic Sorting on FPGA. In: International Symposium on Field-Programmable Gate Arrays. pp. 45–54. ACM/SIGDA (2015)
8. Gadfort, P., Dasu, A., Akoglu, A., Leow, Y.K., Fritze, M.: A Power Efficient Reconfigurable System-in-Stack: 3D Integration of Accelerators, FPGAs, and DRAM. In: International Conference on System-on-Chip Conference (SOCC). pp. 11–16. IEEE (2014)
9. Hybrid Memory Cube Consortium: Hybrid Memory Cube Specification, `http://hybridmemorycube.org/files/SiteDownloads/HMC_Specification%201_0.pdf`
10. Kim, J.S., Yu, C.L., Deng, L., Kestur, S., Narayanan, V., Chakrabarti, C.: FPGA Architecture for 2D Discrete Fourier Transform based on 2D Decomposition for Large-sized Data. In: IEEE Workshop on Signal Processing Systems. pp. 121–126. IEEE (Oct 2009)
11. Langemeyer, S., Pirsch, P., Blume, H.: Using SDRAMs for Two-Dimensional Accesses of Long $\mathbf{2^n \times 2^m}$-point FFTs and Transposing. In: International Conference on Embedded Computer Systems (SAMOS). pp. 242–248. IEEE (July 2011)
12. Park, N., Prasanna, V.: Dynamic Data Layouts for Cache-conscious Implementation of A Class of Signal Transforms. IEEE Transactions on Signal Processing 52(7), 2120–2134 (July 2004)
13. Singapura, S.G., Panangadan, A., Prasanna, V.K.: Performance Modeling of Matrix Multiplication on 3D Memory Integrated FPGA. In: 22nd Reconfigurable Architectures Workshop, IPDPDS. IEEE, To appear (2015)
14. Singapura, S.G., Panangadan, A., Prasanna, V.K.: Towards Performance Modeling of 3D Memory Integrated FPGA Architectures. In: International Conference on Applied Reconfigurable Computing, pp. 443–450. Springer (2015)
15. Wang, W., Duan, B., Zhang, C., Zhang, P., Sun, N.: Accelerating 2D FFT with Non-Power-of-Two Problem Size on FPGA. In: International Conference on Reconfigurable Computing and FPGAs . pp. 208–213. IEEE (Dec 2010)
16. Wu, H., Paoloni, F.: The Structure of Vector Radix Fast Fourier Transforms. IEEE Transactions on Acoustics, Speech and Signal Processing 37(9), 1415–1424 (Sep 1989)
17. Zhu, Q., Akin, B., Sumbul, H.E., Sadi, F., Hoe, J.C., Pileggi, L., Franchetti, F.: A 3D-Stacked Logic-in-Memory Accelerator for Application-Specific Data Intensive Computing. In: IEEE International Conference on 3D Systems Integration Conference (3DIC). pp. 1–7. IEEE (2013)