Multidimensional DFT IP Generator for FPGA Platforms

Chi-Li Yu, *Student Member, IEEE*, Kevin Irick, Chaitali Chakrabarti, *Senior Member, IEEE*, and Vijaykrishnan Narayanan, *Senior Member, IEEE*

***Abstract*—Multidimensional (MD) discrete Fourier transform (DFT) is a key kernel algorithm in many signal processing applications. In this paper we describe an MD-DFT intellectual property (IP) generator and a bandwidth-efficient MD DFT IP for high performance implementations of 2-D and 3-D DFT on field-programmable gate array (FPGA) platforms. The proposed architecture is generated automatically and is based on a decomposition algorithm that takes into account FPGA resources and the characteristics of off-chip memory access, namely, the burst access pattern of the synchronous dynamic RAM (SDRAM). The IP generator has been integrated into an in-house FPGA development platform, AlgoFLEX, for easy verification and fast integration. The corresponding 2-D and 3-D DFT architectures have been ported onto the BEE3 board and their performance measured and analyzed. The results shows that the architecture can maintain the maximum memory bandwidth throughout the whole procedure while avoiding matrix transpose operations used in most other MD DFT implementations. To further enhance the performance, the proposed architecture is being ported onto the newly released Xilinx ML605 board. The simulation results show that 2 K $\times$ 2 K images with complex 64-bit precision can be processed in less than 27 ms.**

***Index Terms*—Discrete Fourier transform (DFT), dynamic RAM (DRAM), field-programmable gate array (FPGA), multidimensional signal processing.**

I. Introduction

DISCRETE Fourier transform (DFT) is widely used in digital signal processing (DSP) and scientific computing applications. More specifically, multidimensional (MD) DFT is used in imaging applications which need frequency-domain analysis, such as image watermarking, finger print recognition, synthetic aperture radar (SAR) processing and medical imaging. The image sizes in many of these applications have become larger over the years. In SAR imaging, for instance, the image size could be as large as 2048×2048 [1], and in medical imaging, the data size could be $512 \times 512 \times 384$ [2]. Therefore, there is a need for new algorithms and architectures to support MD DFT of large-sized data.

Manuscript received February 21, 2010; revised July 09, 2010; accepted August 24, 2010. This work is supported in part by the Defense Advanced Research Projects Agency (DARPA) under Grant W911NF-05–1-0248 and in part by the National Science Foundation (NSF) under Grant 0916887 and Grant 0903432. This paper was recommended by Associate Editor B. Shi.

C.-L. Yu and C. Chakrabarti are with the School of Electrical, Computer and Energy Engineering, Arizona State University, Tempe, AZ 85287 USA (e-mail: chili.yu@asu.edu; chaitali@asu.edu).

K. Irick and V. Narayanan are with the Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA 16802 USA (e-mail: irick@cse.psu.edu; vijay@cse.psu.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSI.2010.2078750

Existing MD DFT implementations include software which are optimized for general-purpose processors, such as FFTW [3], [4], Spiral [5], [6], Intel MKL [7], and IPP [8], or even cluster computers [9], [10]. Software solutions are very flexible and can be ported to users' applications easily and quickly. However, these platforms usually consume power that is high for embedded applications.

Several application-specific integrated circuits (ASIC) implementations for 1-D DFT [11]–[16], and 2-D DFT [17], [18] have been proposed over the years. They exploit the high regularity and parallelism of the DFT algorithm and are quite efficient. While DFT ASICs offer high performance while consuming minimal power, the manufacturing cost of these chips is quite high, and once a chip is manufactured, its functionality and performance cannot be changed anymore. This is why field-programmable gate array (FPGA) has become an attractive alternative for MD DFT implementations [19]–[24]. In this paper, we concentrate on FPGA-based architectures since they can be reconfigured according to user-specified design parameters and offer flexibility while maintaining high performance.

Many of the existing hardware solutions for 2-D DFT are based on row-column (RC) decomposition [18], [20], [21], where the 2-D DFT is computed by successively applying 1-D DFT along rows and then along columns. This works fine for static RAM (SRAM) based designs [20], [21], where data access along rows and along columns have the same cost. However, for systems with dynamic memory, e.g., synchronous dynamic RAM (SDRAM) which is typically adopted for its large storage density, RC decomposition has low performance. This is because SDRAM only favors burst access, and thus while data stored in consecutive locations in memory can be retrieved very efficiently, accessing data along columns is a lot more expensive. To avoid memory access with large strides, the transpose operation is used in [18] to realign the column data into contiguous addresses. While this enables a long burst size to be maintained, the transpose operation take additional time. When the dimension is higher, e.g., in 3-D DFT, the transpose operation becomes harder to implement because of the limited local memory on the FPGA.

There are other hardware solutions such as [19], [23], and [25] that are not based on RC decomposition. Here the multistage DFT flow gragh is folded into one or more butterfly units. These approaches are efficient when the data size is small and can fit in the on-chip memory. For large data sizes, these architectures will have to optimize data access from external memory to sustain high performance. Finally, hardware solutions are not as versatile as software solutions. They cannot support 2-D and 3-D DFTs in one package, and some [18], [20] are even fixed to some specific image sizes.

In this paper, we propose an automated MD DFT hardware IP generator for implementing 2-D and 3-D DFT on FPGA platforms. The optimized MD DFT architecture maximizes the external memory bandwidth. It achieves this by accessing the memory data in a way that avoids transpose operations and utilizes the burst access pattern of the SDRAM memory. Specifically, a strip of data is loaded from the SDRAM onto the local memory, where the width of the strip matches the burst size. For 2-D DFT, if the local memory is not large enough to hold whole columns, smaller sized subcolumns are loaded. At the algorithm level, this translates to computing an equivalent 2-D DFT. The memory access pattern based on row-wise bursts is used for computing 3-D DFT as well. As a result, the proposed MD DFT architecture can maintain the maximum memory bandwidth and, thereby, achieve high performance throughout the computation of 2-D/3-D DFT.

The proposed MD DFT architecture is automatically generated by an in-house IP generator. Based on the image size, dimensionality, and FPGA hardware constraints, the IP generator produces optimized HDL code that utilizes the on-chip FPGA resources efficiently. The IP generator is integrated into a newly developed FPGA development platform, AlgoFLEX, and the architecture mapped on to the BEE3 FPGA board [26]. AlgoFLEX offers versatile support on both hardware and software levels, thereby enabling fast control and verification of the proposed MD DFT IP. The performance of the DFT IPs have been evaluated on the BEE board for different sized 2-D (square, rectangular) and 3-D (cubic, cuboid) images. The results prove that our architecture does not require transpose operations and can maintain the maximum memory throughput rate for the entire computation. To further improve the performance, we are porting our system onto the newly released ML605 platform [27] which is equipped with a Virtex-6 FPGA and a DDR3-SDRAM. Simulation results show that the proposed MD DFT can outperform the previous MD DFT approaches that require multiple banks of SRAM. The MD DFT FPGA architecture presented here was first proposed in [28]. In this paper, we provide more insights into the algorithm modifications, more details of the FPGA architecture, especially the on chip memory organization, numerical accuracy analysis, the newly developed IP generator, the AlgoFLEX platform, and finally more detailed performance comparison with competing implementations.

The rest of the paper is organized as follows. In Section II, the proposed MD DFT algorithm is derived. Section III describes in detail the proposed DFT architecture. The AlgoFLEX FPGA development framework is described in Section IV. The implementation details and evaluation results are discussed in Section V, and concluding remarks are given in Section VI.

II. MD DFT Algorithm

The general form of 2-D DFT can be described in matrix form as follows. Here input U and output V_2 are of size $N_2 \times N_1$; F_{N_1} and F_{N_2} are twiddle factor matrices for row and column DFT computations

$$V_2 = F_{N_2} \cdot U \cdot F_{N_1}. \quad (1)$$

In (1), $V_1 = U \cdot F_{N_1}$ can be done by applying 1-D DFT along the rows of U, and $(F_{N_2} \cdot V_1)$ by applying 1-D DFT along the columns. This is the traditional row-column (RC) decomposition. If the data along the rows of U are stored in a one-dimensional memory, say SDRAM, we can efficiently access data in consecutive location while executing row DFT. However, adjacent data along the columns are no longer in consecutive addresses, and the long strides that are necessary to gather data along columns could make the access latency about 10 times longer during column DFT computation [18]. To reduce access latency of column data, in many implementations, the data is transposed after row operation [18] so that the column-wise data can be realigned into adjacent addresses in the memory and can still be accessed in a burst manner when executing column DFT. However, transpose operations cost extra time, and during matrix transpose operation, the computation unit remains idle. Furthermore, when computing higher dimensional DFT, such as 3-D DFT, much larger on-chip local memory is required for the transpose operation, which cannot be supported on some smaller FPGAs.

In the rest of this section, we first describe the memory-aware 2-D DFT algorithm, which is transpose-free. Then, we also show how the proposed operations in 2-D DFT can be reused in 3-D DFT.

A. Proposed 2-D DFT Algorithm

To maintain row-wise burst for column DFT computations, we utilize the local memory on FPGA and store multiple columns of data. Let S be the size of the local memory, then we can store S/N_2 columns. If S/N_2 is less than the SDRAM's burst size B, then the SDRAM bandwidth is not utilized completely. For instance, if $B = 32$, $S = 16\,384$ and $N_2 = 1024$, then $S/N_2 = 16 < 32$, the burst size. To utilize burst size, B, we decompose the column computations of size N_2 into 1-D computation of size m followed by 1-D computation of size p, where $N_2 = m \cdot p$. Let $L = S/B$, then $p \leq L$. The procedure can be represented mathematically as follows:

$$V_2 = \underbrace{P_{N_2,m} \cdot (I_m \otimes F_p) \cdot \underbrace{\tilde{D}_{N_2} \cdot (F_m \otimes I_p) \cdot \underbrace{U \cdot F_{N_1}}_{\text{Step 1-}a}}_{\text{Step 1-}b}}_{\text{Step 2}}. \quad (2)$$

where I_m and I_p are the identity matrices of sizes $m \times m$ and $p \times p$, respectively, $\tilde{D}_{N_2}$ is a diagonal matrix of twiddle factors, $P_{N_2,m}$ is the column permutation, and $\otimes$ is the Kronecker product. Equation (2) can be summarized as follows:

2-D DFT Procedure

- **Step 1. Row Operations:** This step includes two operations:
  - — Step 1-a. Row DFT $(U \cdot F_{N_1})$: Compute the DFT along the rows of array U.
  - — Step 1-b. Column stride DFT: Column DFT of size m followed by twiddle multiplications.
- **Step 2. Column Local DFT**: Column DFT of size p followed by column-wise permutation.

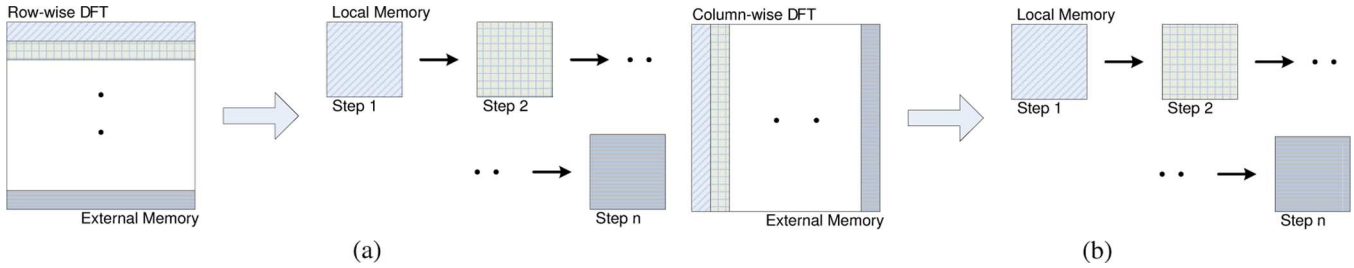


Fig. 1. External memory access pattern for traditional row-column 2-D DFT.

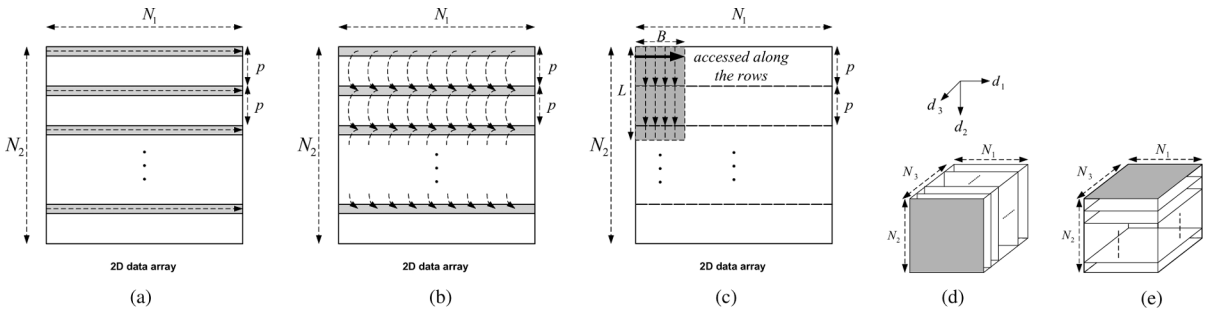


Fig. 2. The proposed data access pattern for 2-D/3-D DFT.

TABLE I
COMPARISONOF COMPUTATION TIMES FOR 2-D DFT

Direct RC		Transpose		Proposed	
Operations	Time	Operations	Time	Operations	Time
Row DFT	T	Row DFT	T	Row operations	αT
Column DFT	KT	Transpose	T	Column local DFT	T
		Row DFT	T		
		Transpose	T		
Total	(K+1)T	Total	4T	Total	$(\alpha+1)$T

Note that Step 1-b and Step 2 form the column DFT. The data access pattern for the 2-D DFT is illustrated in Fig. 2. In Fig. 2(a), m rows spaced p rows apart are selected from the data array and $N_1 \times m$ data is sent to the FPGA local memory. Since the local memory is of size S, $S \geq N_1 \times m$, DFT of N_1 points is executed along each of these rows. Then, column-wise m-point DFT followed by twiddle factor multiplication is applied as shown in Fig. 2(b). The result is stored back in the same location of the data array in the SDRAM. After p iterations of Step 1-a & 1-b, all row DFTs and column stride DFTs are completed. Then Step 2, which consists of p-point local DFT, is computed. As shown in Fig. 2(c), L rows with B elements per row are stored in the local memory. Thus $S \geq B \times L$, where $L \geq p$. This enables p-point local DFT to be computed on these subcolumns. Note that if $N_2 \leq L$, column stride DFT can be skipped, because the whole column can fit in the local memory and the decomposition is not required. Next, the data along the rows have to be stored back to the correct row locations (based on the column-wise permutation, $P_{N_2,m}$) in the SDRAM.

We compare the theoretical time consumptions of three different 2-D DFT solutions in Table I. Direct RC implementation only needs to access the data array in the SDRAM two times. However, the column access is K times longer, where K could be as high as 10 [18]. The transpose-based solution needs to access the SDRAM four times, because of the two additional transpose operations. Note that during the transpose operation, the computation units are idle. In contrast, the proposed design only needs to access the SDRAM two times, for row operations (Step 1) and for column local DFT (Step 2). The row operations could take two times longer $(\alpha = 2)$ than the column local DFT if column stride DFT is activated. However, if the row operations can be fully overlapped with data accesses to the external memory, as is done in our proposed method, α can be reduced to 1. Thus, under the same hardware constraints, the proposed method can be up to two times faster than the transpose-based solution and much faster than the direct-RC implementation.

B. 3-D DFT Algorithm

3-D DFT is a simple extension of 2-D DFT. As illustrated in Fig. 2(d), the 2-D DFT algorithm (described in Section II-A) is computed on each of the N_3 2-D slices parallel to the d_1-d_2 plane. Next, 1-D DFT of size N_3 is done along the d_3-axis. Since adjacent data along the d_3 dimension are not in consecutive addresses in the SDRAM, to utilize the burst along d_1 dimension, we access data on a 2-D slice parallel to d_1-d_3 plane and apply the decomposition along d_3 dimension, as shown in Fig. 2(e). The mathematical description of this procedure is given by

$$\begin{aligned} V_{1,3} &= F_{N_3} \cdot U_{1,3} \\ &= \underbrace{P_{N_3,m'} \cdot (I_{m'} \otimes F_{p'}) \cdot \underbrace{\tilde{D}_{N_3} \cdot (F_{m'} \otimes I_{p'}) \cdot U_{1,3}}_{\text{Step 2-}a}}_{\text{Step 2-}b}, \end{aligned} \quad (3)$$

where $U_{1,3}$ represents an input 2-D array parallel to d_1-d_3 plane and $V_{1,3}$ is the output array; $N_3 = m' \times p'$. The whole 3-D DFT procedure can be summarized as follows:

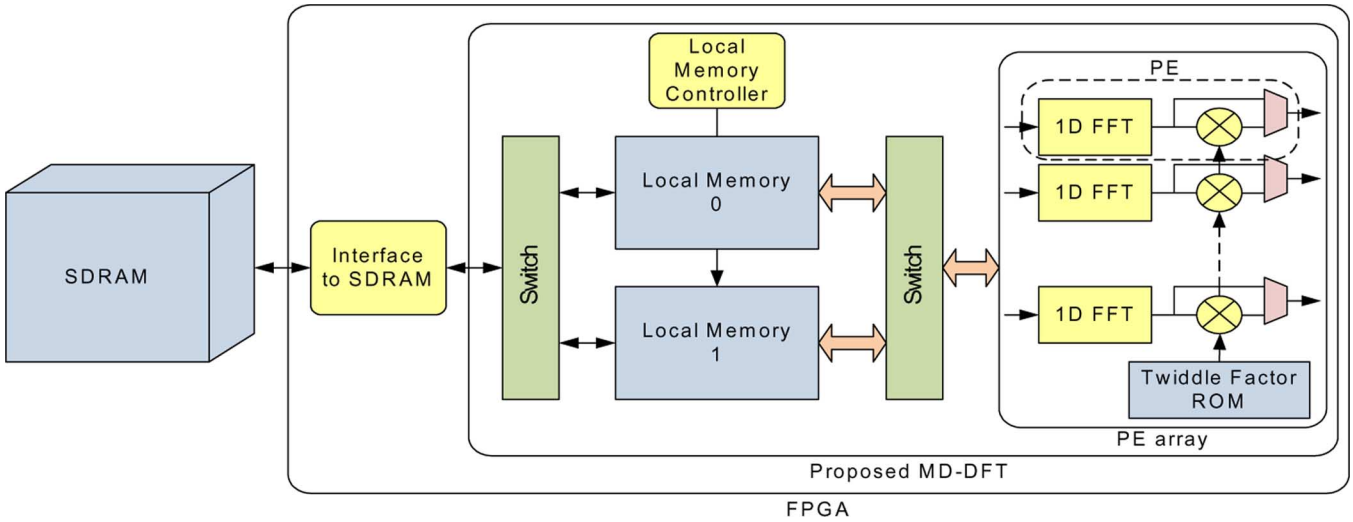


Fig. 3. The proposed MD DFT architecture.

3-D DFT Procedure

- **Step 1. Run [2-D DFT Procedure] on N_3 slices**: Compute 2-D DFT on every 2-D slice parallel to d_1-d_2 plane.
- **Step 2. DFT along d_3 dimension (3)**:
  — Step 2-a. Stride DFT: Compute m'-point stride DFT followed by twiddle multiplications on the 2-D slice parallel to d_1-d_3 plane.
  — Step 2-b. Local DFT: Compute p'-point local DFT followed by the permutation on the 2-D slice parallel to d_1-d_3 plane.

Step 2 has to be iterated N_2 times. After Step 3, the final results have to be stored to the correct locations in a different part of the SDRAM. Similar to the 2-D DFT procedure, Step 2 can be skipped if $N_3 \leq L$.

Note that the purpose of the decompositions along d_2 (column)- and d_3-dimension is to utilize the burst access along d_1-dimension (row) throughout the whole 3-D DFT computation. With this decomposition technique, we can avoid the data transpose required in other 3-D DFT implementations [10], [29] and achieve higher performance.

III. Proposed DFT Architecture

A. Architecture Overview

The architecture proposed for MD DFT is shown is Fig. 3. The main components are an FPGA to do the processing and an SDRAM to store the data. The SDRAM controller fetches the input data from SDRAM and sends it to the local memory on the FPGA. The processing elements (PE) read this data, process it, and store the results back to the local memory. The SDRAM controller then reads these results from the local memory and stores them back to the SDRAM. The main components of the architecture are described below:

1) Processing Elements (PEs): A PE consists of a 1-D DFT module followed by a complex multiplier. The 1-D DFT module can support a maximum of N_1-point DFT for computing along rows, but it can also compute column stride/local DFTs of other lengths $L \leq N_1$. In our design, we adopt Xilinx's streaming fast Fourier transform (FFT) IP core [30]. The complex multiplier is used to compute the twiddle multiplication after column stride DFT. The number of PEs depends on the required data throughput as well as the hardware resources available on the FPGA.

2) SDRAM: SDRAM is the main memory used to store the multidimensional data. In our implementation, a 2 GB DDR2–400 DIMM is adopted. For 2-D or 3-D images, consecutive data along d_1 dimension is stored in consecutive locations in the SDRAM. We use AlgoFLEX (described in Section IV) to transfer the data between SDRAM and FPGA. The data is always accessed along d_1 dimension, thereby fully exploiting SDRAM's bandwidth efficiency.

3) Dual Local Memory: There are two identical local memories of size S that serve as ping pong buffers. These local memories are implemented with dual-port block RAMs on the FPGA. Unlike the SDRAM, nonconsecutive addresses in the Block RAM can be accessed in contiguous clock cycles without performance penalty. Each local memory consists of multiple banks, so that multiple data can be received within one cycle from the SDRAM, and data in the local memory can be accessed by multiple 1-D FFT IP cores at the same time. To support simultaneous accesses from multiple (up to r) PEs, each local memory is divided into r banks, as illustrated in Fig. 4. If S is the size of the local memory, for the row operation, the SDRAM controller fetches the first S/m consecutive data (i.e., the first $S/(m \times N_1)$ rows) from the SDRAM and stores them into the banks starting from Bank 0. Then, the SDRAM controller fetches the next S/m consecutive data starting from the pth row and stores them starting from Bank 1, and so on. Such a storage scheme enables r PEs to access the data that were in the same bank. The arrows in Fig. 4 show how the conflict-free accesses work for row DFT and column stride DFT. The same addressing scheme also works for column local DFT. Note that the local memory can also receive multiple input data via a wider bus or SDRAM interface with the multiple-banked organization.

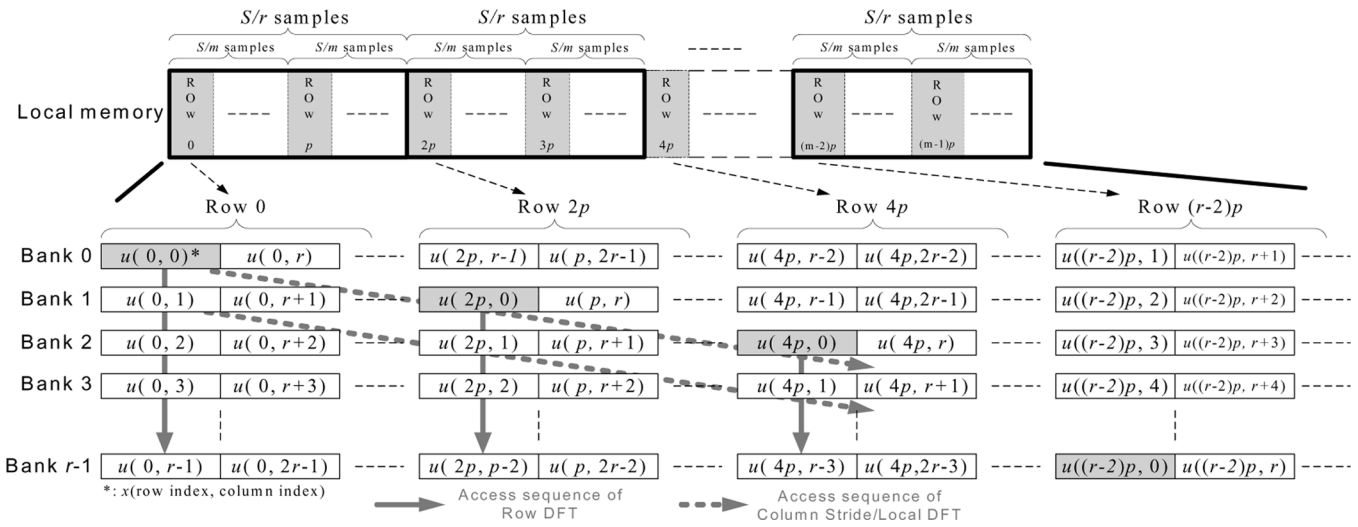


Fig. 4. Data organization and the access patterns of the local memory.

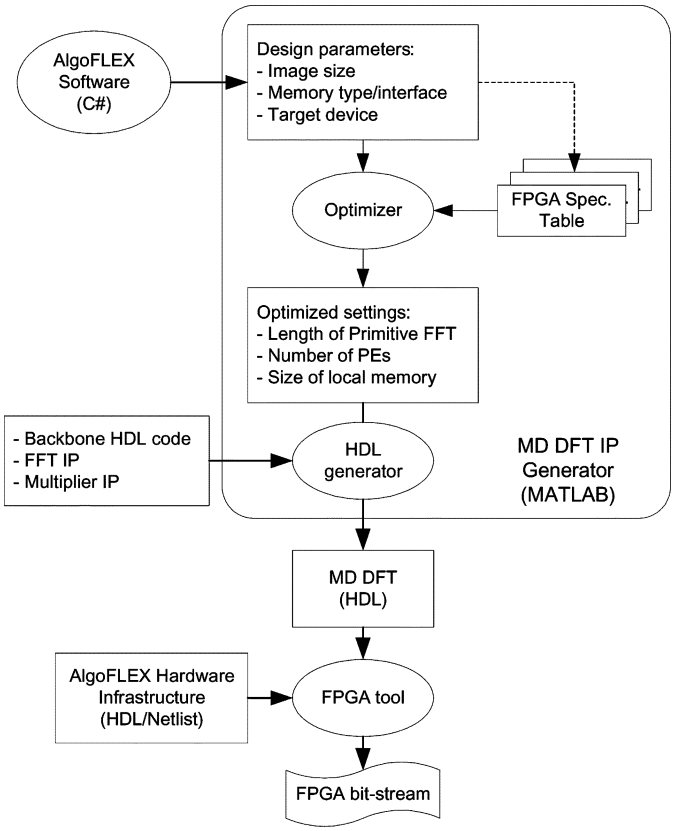


Fig. 5. Automation flow of generating architecture for the proposed MD DFT.

IV. Test Platform and MD DFT IP Generator

To implement and evaluate the proposed MD DFT design on different FPGAs, we have developed an automated MATLAB-based MD DFT IP generator. The IP generator automatically calculates the size of the FFT IP in the PE, and the optimal number of PEs, N_{PE}, based on image size, FPGA resources and external memory bandwidth. Then, it generates the corresponding HDL (Verilog) files which can be fed into the FPGA tool to produce the final configuration bit-files. The automation flow of the MD DFT IP generator is shown in Fig. 5.

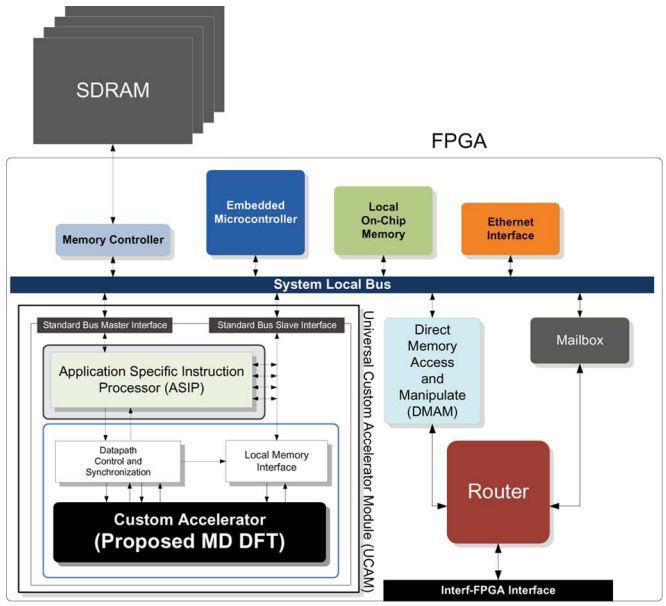


Fig. 6. AlgoFLEX Hardware Infrastructure.

In order to integrate the MD DFT cores with other programmable and customized cores for large system design, we have also developed a framework called AlgoFLEX. This framework provides a backbone hardware platform for seamless integration of customized cores. Further, it provides a front-end GUI interface and back-end synthesis software chain for mapping the desired set of IP blocks onto a target FPGA. We only explain the portions of this framework essential to understanding the integration of MD DFT core on to this platform.

A. AlgoFLEX Platform

The AlgoFLEX framework has been designed for integrating multiple cores to aid fast development of complete applications on FPGA based hardware platforms. On the hardware side, it provides various standard interfaces, like system bus and SDRAM controller for plug-n-play user-defined modules. On the software side, the platform incorporates a unified graphical

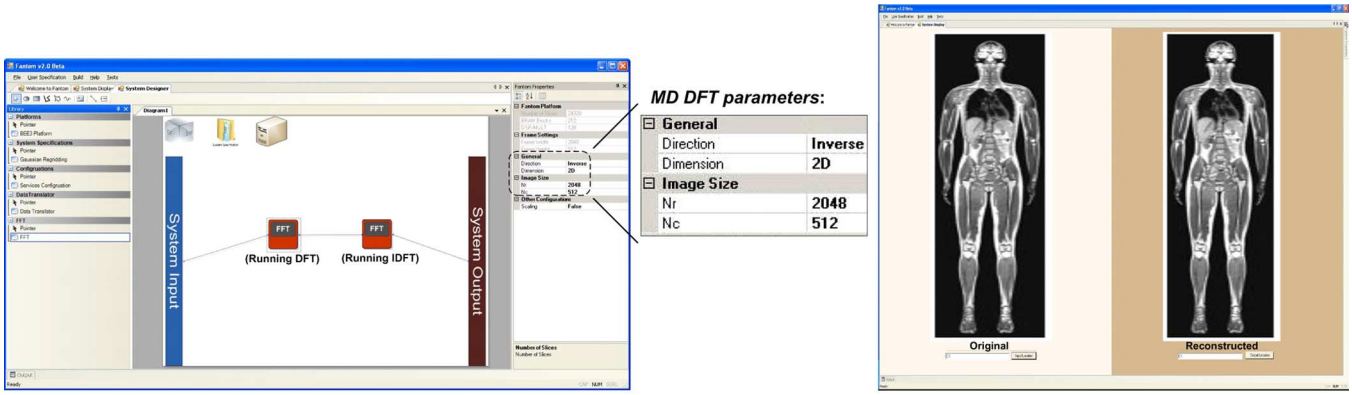
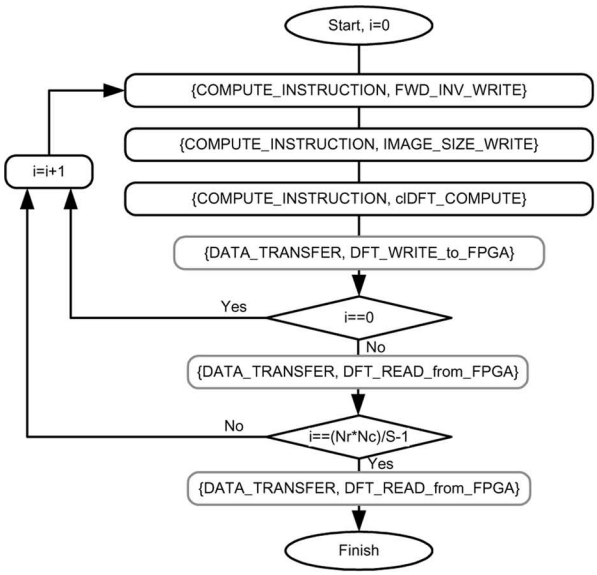


Fig. 7. Graphical user interface of AlgoFLEX.



Command Option	Parameter	Description
rDFT_COMPUTE = 2	--	Run row DFT
csDFT_COMPUTE = 3	--	Run column stride DFT
rDFT_csDFT_COMPUTE = 4	--	Run row operations
clDFT_COMPUTE = 5	--	Run column local DFT
NULL_COMPUTE = 6	--	No computation (For memory read/write tests)
csDFT_START_rADDR_WRITE = 7	Row address of the first sample	Specify starting address for twiddle factor generating in column stride DFT
IMAGE_SIZE_WRITE = 8	{log2(Nc), log2(Nr)}	Specify image size
FWD_INV_WRITE = 9	IDFT:0, DFT:1	Choose IDFT or DFT

(b)

Fig. 8. (Left) A pseudocode of the algorithm specified using command sequence; (Right) Commands exposed by the accelerator to the algorithm designer.

user interface (GUI) which allows the user to modify the configurations, execute the FPGA implementation flow and display the results.

The AlgoFLEX hardware infrastructure, shown in Fig. 6, provides a system designer with the capability to plug-n-play a collection of custom accelerator blocks, such as our MD DFT core, without requiring intimate knowledge of how those blocks will be composed. As shown, the platform consists of a hybrid communication network comprised of a System Local Bus (SLB) and a packet-based on-chip router along with memory controllers, accelerators, and other system components. Currently, we adopt Xilinx's 128-bit Processor Local Bus (PLB) [31] as SLB and 64-bit multiport memory controller (MPMC) [32] as the memory controller. While the SLB serves as the communication backbone within a single FPGA, the router is used to coordinate a secondary channel for both intra- and inter-FPGA communication.

To integrate accelerators such as our MD DFT into the base hardware configuration of AlgoFLEX, we have designed a Universal Custom Accelerator Module (UCAM) wrapper. It defines a standard interface between the common system level facilities and an instance of an accelerator module. It takes advantage of the fact that many signal processing applications can be characterized by the following: 1) at the application level a sequence of subtasks are repeatedly performed on incoming data and 2) in each subtask, a static set of compute-intensive operations are performed whose sequence can vary across different invocations. For example, an optimized MD DFT implementation may selectively perform decomposition of data into 1-D DFT operations based on the dimensionality, but the sequence of these operations may be radically different across different problem specifications.

TABLE II
HARDWARE RESOURCE UTILIZATION OF THE MD DFT IP ON A XILINX VIRTEX-5 XC5LX155T FPGA

Slices	DSP48Es	Block RAMs
25% (8,273/33,088)	53% (68/128)	41% (87/212)

Furthermore, each UCAM is equipped with an Application Specific Instruction Processor, ASIP, that facilitates fetching, decoding, and preprocessing of accelerator specific instruction sequences. It also provides a set of built-in Command Handlers that are useful for many algorithm accelerators. For the case of

TABLE III
MEASURED COMPUTATION TIME OF 2-D DFT ON BEE3

Shape	Image size ($N_1 \times N_2$)	Row operations (ms)	Column local DFT (ms)	Total (ms)
Square	128×128	0.89	0.90	1.79
	256×256	3.01	3.04	6.05
	512×512	12.14	12.72	24.86
	1024×1024 *	50.21	52.42	102.63
	2048×2048 *	202.45	209.65	412.10
Rectangle	512×2048 *	50.34	52.37	102.71
	2048×512	50.11	52.47	102.58

(*: Column stride DFT is required.)

2-D and 3-D DFT, the window fetch handler supports optimized fetching and storing of n-dimensional data with programmable column size, number of rows and columns, row and column offset, interrow stride, intercolumn stride, and interdimension offset. The provisioning of such features within the AlgoFLEX framework facilitates reuse of design effort across modules as well as helps in quickly adapting the customized cores to a different target FPGA with changes only in a small set of base platform modules.

AlgoFLEX also provides a drag-and-drop graphical interface that allows the user to compose a system by diagramming the algorithmic specification onto a canvas using AlgoLets, as shown in Fig. 7(a). An AlgoLet is an abstraction of an algorithmic entity that, when synthesized, is implemented in one or more associated UCAM modules. The dataflow between AlgoLets is described by the user by drawing connections between several AlgoLets. AlgoFLEX automatically infers control flow, maps UCAMs to FPGA resources, instantiates direct memory access and manipulate (DMAM) module and stream operators, and executes synthesis scripts necessary to generate bitstreams for each FPGA in the platform. An example command set and execution sequence for 2-D DFT can be seen in Fig. 8. In addition to hardware, software is generated for the embedded microcontroller to perform initialization tasks such as configuring routing tables, performing memory allocation, and memory mapping for each UCAM. The executables for each microcontroller are loaded using a JTAG debug interface while the system initialization data is loaded through an Ethernet interface. For instance, in Fig. 7(a), two MD DFT AlgoLets are dragged and linked on the canvas. The former one runs DFT, and the latter one runs IDFT. This setting is for the MD DFT function verification. In Fig. 7(b), the GUI displays the input and output images, which if identical, implies that the MD DFT/IDFT works correctly.

B. Automated MD DFT IP Generator

In this section, we describe the operations in the MD DFT IP generator shown in Fig. 5. It receives the user-selected device, memory type/interface, the required image size, and dimensionality from the AlgoFLEX GUI. Based on the image size, it chooses the size of the Xilinx 1-D FFT IP. Then, based on the hardware resources of the target FPGA, the IP generator picks the number of PEs, $N_{\text{PE\_Resource}}$. Note that the external memory type/interface also affects the number of PEs, since the pipelined Xilinx FFT IP [30] can input 1 datum/cycle. For instance, if the memory interface can transfer 128 bits/cycle and an image sample is 64-bit wide, only 2 samples from the external memory can be read in one clock cycle, and the performance of the MD DFT cannot be improved with more

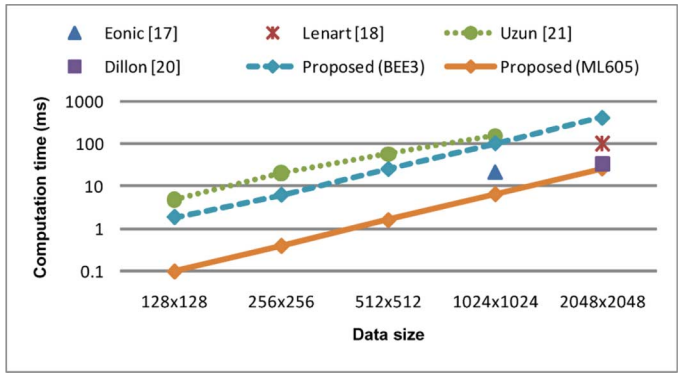


Fig. 9. Comparison of computation times of 2-D DFT architectures. Note that the computation times are normalized for the same data width, 2×32 bits.

PEs. Let $N_{\text{PE\_Bandwidth}}$, be the number of PEs based on the memory bandwidth. Then, the number of PEs is the minimum of $N_{\text{PE\_Resource}}$ and $N_{\text{PE\_Bandwidth}}$, i.e.,

$$N_{\text{PE}} = \min\{N_{\text{PE\_Resource}}, N_{\text{PE\_Bandwidth}}\}. \quad (4)$$

The MD DFT IP generator also has the capability to determine if the image size can be supported by the FPGA platform or not. It computes the maximum image size based on the following equations: First, let the 1-D FFT IP's maximum size, $N_{\text{FFT\_IP}}$ be the maximum value of N_1, which is provided by the user through the GUI. So

$$N_{\text{FFT\_IP}} = N_{1\_\max}. \quad (5)$$

The local memory size, S, decides the number of rows, m, which can be loaded into the FPGA in row DFT/column stride DFT computations, where $m = S/N_{1\_\max}$. The burst size, B, which is an SLB parameter, affects the sizes of N_2 and N_3. For column local DFT, L rows with B samples per row are loaded onto the local memory, i.e., $L = S/B$. L is actually the maximum size of column local DFT, p, and $N_2 = p \times m$. Therefore, the maximum value of N_2 is

$$N_{2\_\max} = L \times m = \frac{S^2}{N_{1\_\max} \times B}. \quad (6)$$

In other words,

$$N_{1\_\max} \times N_{2\_\max} = \frac{S^2}{B}. \quad (7)$$

A similar analysis holds for N_3, and

$$N_{1\_\max} \times N_{3\_\max} = \frac{S^2}{B}. \quad (8)$$

For example, in our implementation, the local memory size S on a Virtex-5 LX155T FPGA is 16384, and the burst size B is 32 samples. Based on (7)–(8), the MD DFT implementation can support up to 2048×4096 2-D DFT and $2048 \times 4096 \times 4096$ 3-D DFT. If the user inputs N_2 and N_3 larger than $N_{2\_\max}$ and $N_{3\_\max}$, the GUI would show a warning message and suggest the user to reduce the image size.

V. EVALUATION

The proposed MD DFT architecture has been generated by the AlgoFLEX platform and its functionality has been verified on the BEE3 board [26], which is equipped with a Virtex-5

TABLE IV
COMPARISON OF 2-D DFT IMPLEMENTATIONS WITH RESPECT TO HARDWARE CONFIGURATION AND PERFORMANCE

		Eonic[17]	Lenart[18]	Uzun[21]	Dillon[20]	Proposed (BEE3$^{(1)}$)	Proposed (ML605$^{(2)}$)
	Technology	ASIC, 180nm	ASIC, 130nm	Virtex-E, 180nm	Virtex-II-Pro, 130nm	Virtex-5, 65nm	Virtex-6, 40nm
	External memory	Quad SDRAM	Dual SDRAM	Quad SRAM	Dual SRAM	Single SDRAM	Signal SDRAM
	Clock freq. (MHz)	128	250	26	120	100	100
	Data Width (bits)	2×32	2×16	2×16	2×8	2×32	2×32
Computation times for different Image sizes (ms)	128×128	–	–	2.4 (4.8)†	–	1.8	0.1
	256×256	–	–	10.1 (20.2)	–	6.1	0.4
	512×512	–	–	28.6 (57.2)	–	24.9	1.6
	1024×1024	21.0	–	76.9 (153.8)	–	102.6	6.5
	2048×2048	–	50.0 (100.0)	–	8.3 (33.3)	412.1	26.2

(1): The results are measured from BEE3 board; (2): The results are simulated based the hardware configuration of ML605.
(†: The computation times in the brackets are normalized for a data width of 2×32 bits.)

TABLE V
MEASURED ACCURACY OF THE PROPOSED 2-D DFT

Image	Size	SNR(dB)	Maximum norm error
Lena$^{(1)}$	128×128	136.26	1.23e-4
Lena$^{(1)}$	256×256	133.93	2.37e-4
Lena$^{(1)}$	512×512	131.04	4.94e-4
MRI-brain$^{(2)}$	1024×1024	128.02	1.21e-3
SAR$^{(3)}$	2048×2048	126.59	2.07e-3

(1): From USC-SIPI Image Database [33].
(2): From University of Virginia Health System [34], and the image is resized to 1024×1024.
(3): From EUSAR06 [35], and the image is resized to 2048×2048.

TABLE VI
MEASURED COMPUTATION TIME OF 3-D DFT ON BEE3

Number of samples	Image size ($N_1 \times N_2 \times N_3$)	Total (ms)
2^{21}	$128 \times 128 \times 128$	348.24
2^{22}	$64 \times 256 \times 256$	757.21
	$256 \times 64 \times 256$	646.81
	$256 \times 256 \times 64$	697.09

LX155T FPGA. We assume that $N_{1\_\max} = 2048$ in our experiments. For this configuration, the FPGA can only accommodate one PE, which consists of a 2048-point FFT IP and a complex multiplier, since this occupies more than half (53%) of DSP48Es. The local memory size, S, is 16 384 samples. The ping pong buffer and other memory in the 2048-point FFT IP consume 41% of Block RAMs, and the other Block RAMs are needed to support AlgoFLEX's infrastructure. The hardware resource utilization of the MD DFT IP is summarized in Table V. Since there is only one PE, single-banked local memory would have been sufficient. However, we choose to divide it into 2 banks, since in each cycle the 128-bit SLB can transfer 2 complex samples with single precision and store them into the local memory. For maximum performance, SLB's burst size is set to the largest value: 16 cycles. Thus, the effective burst size, B, is $16 \times 2 = 32$ complex samples. In our implmentation, m is set to 8, because 8 rows of length 2048 can fit in the local memory. The clock frequency is set to 100 MHz. A timer on the FPGA is used to count the clock cycles elapsed during the computations.

A. 2-D DFT

The computation times (measured) of 2-D DFT for square and rectangular images of different sizes are listed in Table III. First, these results show that column local DFT takes almost the same time as the row operations. This means that the row-wise burst access mode for the column local DFT computations achieve the same bandwidth efficiency as the row operations. Secondly, the computation time is proportional to the image size, that is, if the computation time is T for an $N \times N$ image, it is $4T$ for a $2N \times 2N$ image. So a 512×512 2-D DFT takes about four times longer than 256×256 2-D DFT; a 1024×1024 2-D DFT takes almost the same time as 2048×512 2-D DFT, because their data sizes are the same.

In the current implementation on the BEE3 board, the SDRAM's peak performance is not fully exploited due to the slow SLB (Xilinx's PLB) and the memory controller, MPMC. In the 2048×2048 DFT, the average data transfer rate is only 325.69 MB/s, while the peak transfer rate of the SDRAM is 3200 MB/s. To achieve a much higher performance, the proposed architecture is currently being ported onto the Xilinx ML605 FPGA board [27], which is equipped with a Virtex-6 LX240T FPGA and a DDR3–800 SDRAM SO-DIMM. This FPGA can accommodate 8 PEs, which can easily deal with the full SDRAM bandwidth. In addition, if a dedicated SDRAM controller is designed that can utilize 80% of the SDRAM's bandwidth, the proposed design can complete 2048×2048 DFT in 26.2 ms.

With ML605, the proposed architecture can outperform other 2-D DFT solutions listed in Table IV. To make a fair comparison, we extrapolate the performances of the different architectures for the same data width, namely, 2×32 bits (single precision complex data). We assume that in all cases the performance is constrained by the data transfer between the external memories and FPGA/ASIC, and that the bandwidth of the external memory is the same as the original implementation. So the normalized time consumption is 64/(data width) of the reported computation time. Under this scenario, both Dillon's implementation [20] and ML605 (simulated) have very low computation times, around 30 ms for 2048×2048 data. While our design requires only one SDRAM, Dillon's solution [20] utilizes multiple SRAM modules and a memory controller that is optimized for an image size of 2048×2048. Uzun's 2-D DFT [21] supports multiple image sizes. However, since it requires transpose operations and runs at a lower frequency, its performance is lower. Other competing solutions such as Lenart's [18] also requires transpose operations, and the one from Eonic [17] requires multiple SDRAMs (up to 4 banks). The performances of the 2-D DFT architectures have been illustrated in Fig. 9. We see that the proposed 2-D DFT on ML605 is the fastest implementation for different data sizes. Also, the straight lines in this log-scaled plot imply that the performances of the 2-D DFTs on BEE3 and ML605 are proportional to the data sizes.

TABLE VII
COMPARISON OF 3-D DFT IMPLEMENTATIONS WITH RESPECT TO HARDWARE CONFIGURATION AND PERFORMANCE

		Sasaki[22]	Proposed (BEE3$^{(1)}$)	Proposed (ML605$^{(2)}$)
	Technology	Virtex-II, 180nm	Virtex-5, 65nm	Virtex-6, 40nm
	External memory	Single SDRAM	Single SDRAM	Single SDRAM
	Clock freq. (MHz)	100	100	100
	Data width (bits)	2×64	2×32	2×32
Computation times for different image sizes (ms)	$128 \times 128 \times 128$	441	348 (696)†	22 (44)
	$256 \times 256 \times 256$	4,027	2,327 (4,654)	148 (296)
	$512 \times 512 \times 512$	–	19,241 (38,482)	1,223 (2,447)

(1): Except for $128 \times 128 \times 128$ 3D FFT, the other results are extrapolated.
(2): The results are simulated based the hardware configuration of ML605.
(†: The computation times in the brackets are normalized for the same data width, 2×64 bits.)

To analyze the precision of the 2-D DFT, we first use MATLAB's 2-D FFT function to transform different sized images to the frequency domain. Then we use the spectral data to reconstruct the images using our 2-D DFT BEE3 implementation. In Table V, we record the images' Signal-to-Noise-Ratio (SNR) and maximum reconstructive error, where SNR is defined as

$$SNR(dB) = 10 \log_{10} \frac{P_{\text{original image}}}{P_{\text{quantization noise}}} \tag{9}$$

$P_{\text{original image}}$ is the power of original image (the ideal result), and $P_{\text{quantization noise}}$ is the power of quantization noise. We see that the SNR is around 130 dB, which is mainly due to Xilinx 1-D FFT IP, whose SNR is about 140 dB [30]. Secondly, the maximum errors of all the images are fairly small. We conclude that the proposed architecture has high accuracy and can be used in most DFT-based image reconstruction applications.

B. 3-D DFT

The performance of the 3-D DFT implementation is summarized in Table VI. The results presented here are measured values, but they match very well with estimated result derived from 2-D DFT measurements. In 128^3 DFT, for example, 2-D DFT on the 128 d_1-d_2 planes takes $128 \times 1.79 = 229.12$ ms, and column DFTs on the 128 d_1-d_3 planes takes $128 \times 0.9 = 115.2$ ms. Thus, the total estimated time of 128^3 3-D DFT is 344.32 ms, which is pretty close to the measured result of 348.24 ms. This means SDRAM's bandwidth efficiency in 2-D DFT is maintained in 3-D DFT. As in the 2-D case, the time-consuming transpose operations have been avoided. Note that, due to driver issues, we can currently only measure performance of data volumes up to 2^{22} samples on the BEE3 board.

Table VII compares the performance of our architecture on the BEE3 and ML605 with Sasaki's architecture [22]. In [22], the computation kernel consists of three double-precision adders and two double-precision multipliers, which can implement a butterfly computation in two cycles. It utilizes the memory bandwidth efficiently and is computation-bound. To make the 3-D DFTs comparable, we normalize our implementations to double precision. Since the implementations on BEE3 and ML605 are communication-bound, their computation times would be doubled due to 2x wider data width. The implementation on BEE3 is constrained by the data transfer as mentioned before. On ML605 board, however, the proposed architecture could be at least 10x faster than [22]. Since images larger than $1024 \times 1024 \times 1024$ with single precision require at least 8 GB memory, they cannot fit in the 4 GB SO-DIMM, on the ML605 platform. Thus, we only simulate image sizes up to $512 \times 512 \times 512$.

VI. CONCLUSION

An MD DFT IP has been proposed that is based on a decomposition algorithm which takes into account the burst access pattern of the SDRAM and the available FPGA resources. It does not require long stride memory accesses or transpose operations and is able to maintain the maximum SDRAM bandwidth throughout the computation. The MD DFT IP is automatically generated and the MD DFT IP generator integrated into the AlgoFLEX development platform. The input specifications such as image size, dimensionality, FPGA resources, memory bandwidth are input through the AlgoFLEX GUI, and the optimized HDL code is produced by the IP generator. The resulting architecture has been ported onto the BEE3 FPGA board and validated for different sized 2-D and 3-D data. To achieve higher performance, the architecture is being ported onto the newly released Xilinx ML605 FPGA board. Simulation results demonstrate the superior performance of these architectures compared to existing DFT implementations.

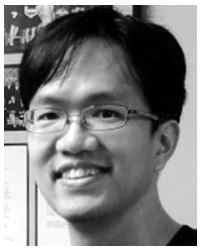
ACKNOWLEDGMENT

The authors gratefully acknowledge A. Al Maashri and S. Park of Pennsylvania State University, and Dr. X. Sun and Dr. N. Pitsianis of Duke University for their valuable assistance.

REFERENCES

[1] F. Tupin, B. Houshmand, and M. Datcu, "Road detection in dense urban areas using SAR imagery and the usefulness of multiple views," *IEEE Trans. Geosci. Remote Sens.*, vol. 40, pp. 2405–2414, 2002.
[2] A. Souza and R. Senn, "Model-based super-resolution for MRI," in *Proc. 30th Annu. Int. Conf. IEEE Eng. Med. Biol. Soc. 2008 (EMBS 2008)*, pp. 430–434.
[3] M. Frigo and S. Johnson, "FFTW: An adaptive software of the FFT," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 1998, vol. 3, pp. 1381–1384.
[4] M. Frigo and S. G. Johnson, "The design and implementation of FFTW3," *Proc. IEEE, Special Issue Program Gener., Optim., Adaptation Comput. Phys. Commun.*, vol. 93, no. 2, 2005.
[5] M. Püschel, J. M. F. Moura, J. Johnson, D. Padua, M. Veloso, B. Singer, J. Xiong, F. Franchetti, A. Gacic, Y. Voronenko, K. Chen, R. W. Johnson, and N. Rizzolo, "Spiral: Code generation for DSP transforms," *Proc. IEEE*, vol. 93, pp. 232–275, Feb. 2005.
[6] J. Johnson and X. Xu, "Generating symmetric DFTs and equivariant FFT algorithms ," in *ACM Int. Symp. Symbolic Algebr. Comput. (ISSAC)*., 2007, pp. 195–202.
[7] Intel Math Kernel Library (MKL) [Online]. Available: http://software.intel.com/en-us/intel-mkl/
[8] Intel Integrated Performance Primitives (IPP) [Online]. Available: http://software.intel.com/en-us/intel-ipp/
[9] M. Eleftheriou, B. G. Fitch, A. Rayshubskiy, T. J. C. Ward, and R. S. Germain, "Scalable framework for 3D FFTs on the blue gene/l supercomputer: Implementation and early performance measurements," *IBM J. Res. Develop.*, vol. 49, pp. 457–464, 2005.

[10] B. Fang, Y. Deng, and G. Martyna, "Performance of the 3D FFT on the 6D network torus QCDOC parallel supercomputer," *Comput. Phys. Commun.*, vol. 176, no. 8, pp. 531–538, Apr. 2007.
[11] Y.-W. Lin and C.-Y. Lee, "Design of an FFT/IFFT processor for MIMO OFDM systems," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 54, no. 4, pp. 807–815, 2007.
[12] Y.-N. Chang and K. K. Parhi, "An efficient pipelined FFT architecture," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 50, no. 6, pp. 322–325, 2003.
[13] Y. Chen, Y.-C. Tsao, Y.-W. Lin, C.-H. Lin, and C.-Y. Lee, "An indexed-scaling pipelined FFT processor for OFDM-based WPAN applications," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 55, no. 2, pp. 146–150, 2008.
[14] B. Baas, "A low-power, high-performance, 1024-point FFT processor," *IEEE J. Solid-State Circuits*, vol. 387, p. 380, Mar. 1999.
[15] W.-C. Yeh and C.-W. Jen, "High-speed and low-power split-radix FFT," *IEEE Trans. Signal Process.*, vol. 51, pp. 864–874, Mar. 2003.
[16] Y.-W. Lin, H.-Y. Liu, and C.-Y. Lee, "A 1-GS/s FFT/IFFT processor for UWB applications," *IEEE J. Solid-State Circuits*, vol. 40, pp. 1726–1735, Aug. 2005.
[17] PowerFFT ASIC [Online]. Available: http://www.eonic.com/index.asp?item=32
[18] T. Lenart, M. Gustafsson, and V. Öwall, "A hardware acceleration platform for digital holographic imaging," *J. Signal Process. Syst.*, vol. 52, no. 3, pp. 297–311, Sep. 2008.
[19] P. A. Milder, F. Franchetti, J.C. Hoe, and M. Püschel, "Formal datapath representation and manipulation for implementing DSP transforms ," in *Proc. Design Autom. Conf. (DAC)*, 2008, pp. 385–390.
[20] T. Dillon, "Two Virtex-II FPGAS deliver fastest, cheapest, best high-performance image processing system," *Xilinx Xcell J.*, vol. 41, pp. 70–73, 2001.
[21] I. Uzun, A. Amira, and A. Bouridane, "FPGA implementations of fast Fourier transforms for real-time signal and image processing," *IEE Proc.. Vis., Image, Signal Process.*, vol. 152, no. 3, pp. 283–296, Jun. 2005.
[22] T. Sasaki, K. Betsuyaku, T. Higuchi, and U. Nagashima, "Reconfigurable 3D-FFT processor for the car-parrinello method," *J. Comput. Chem., Jpn.*, vol. 4, no. 4, pp. 147–154, 2004.
[23] P. D'Alberto, P. A. Milder, A. Sandryhaila, F. Franchetti, J. C. Hoe, J. M. F. Moura, M. Püschel, and J. Johnson, "Generating FPGA accelerated DFT libraries ," in *Proc. IEEE Symp. Field-Programmable Custom Comput. Mach. (FCCM)*, 2007, pp. 173–184.
[24] P. Kumhom, J. Johnson, and P. Nagvajara, "Design, optimization, and implementation of a universal FFT processor ," in *Proc. IEEE ASIC/SOC Conf.*, 2000, pp. 182–186.
[25] H. Kee, S. S. Bhattacharyya, N. Petersen, and J. Kornerup, "Resource-efficient acceleration of 2-dimensional fast Fourier transform computations on FPGAs ," in *Proc. 3rd ACM/IEEE Int. Conf. Distrib. Smart Cameras (ICDSC 2009)*, Aug. 2009, pp. 1–8.
[26] The BEE3 Hardware Platform [Online]. Available: http://www.beecube.com/platform.html
[27] Virtex-6 FPGA ML605 Evaluation Kit [Online]. Available: http://www.xilinx.com/products/devkits/EK-V6-ML605-G.htm
[28] C.-L. Yu, C. Chakrabarti, S. Park, and V. Narayanan, "Bandwidth-intensive FPGA architecture for multi-dimensional DFT," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP 2010)*, pp. 1486–1489.
[29] D. Takahashi, "Efficient implementation of parallel three-dimensional FFT on clusters of PCs," *Comput. Phys. Commun.*, vol. 152, pp. 144–150, 2003.
[30] Xilinx FFT Logicore [Online]. Available: http://www.xilinx.com/products/ipcenter/FFT.htm
[31] Processor Local Bus [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/plb_v46.pdf
[32] Xilinx Multi-Port Memory Controller [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/mpmc.pdf
[33] The USC-SIPI Image Database [Online]. Available: http://sipi.usc.edu/database/
[34] 3T MRI [Online]. Available: http://www.healthsystem.virginia.edu/internet/physicians-direct/images/julystories/01MN_Tim_App_MR_002.jpg
[35] Dresden—Historical City Center [Online]. Available: http://www.dlr.de/hr/en/Portaldata/32/Resources/images/eusar/EUSAR2006-home-E-SAR-image.jpg



Chi-Li Yu (S'10) received the B.S. and M.S. degrees in electrical engineering from National Central University, Taiwan, in 1998 and 2000, respectively.

From 2001 to 2005, he was an Associate Engineer with the Industrial Technology Research Institute (ITRI), Taiwan. Currently, he is working toward the Ph.D. degree in the School of Electrical, Computer and Energy Engineering, Arizona State University, Tempe. His research interests include high-performance/low-power VLSI architectures for DSP, image processing, and communication applications.

Kevin Irick received the B.S. degree in electronics engineering technology from DeVry University, Atlanta, GA, in 2002 and the M.S. and Ph.D. degrees in computer science and engineering from Pennsylvania State University, University Park, in 2006 and 2009, respectively.

Currently, he is a Research Scientist in the Department of Computer Science and Engineering at Pennsylvania State University. His research interests include application-specific hardware accelerators, hardware assisted bioinspired image processing and recognition, and high-performance computing on FPGAs.

Chaitali Chakrabarti (SM'02) received the Ph.D. degree from University of Maryland, College Park, in 1990.

She is a Professor with the Department of Electrical Engineering, Arizona State University, Tempe. Her research interests are in the areas of low power embedded systems design and VLSI architectures and algorithms for signal processing, image processing, and communications.

Vijaykrishnan Narayanan (SM'99) received the Ph.D. degree from the University of South Florida, Tampa, in 1998.

He is a Professor of Computer Science and Engineering at Pennsylvania State University, University Park. His research interests are in power-aware architectures, embedded reconfigurable systems, and reliable systems.