Modularized architecture of address generation units suitable for real-time processing MR data on an FPGA

Limin Li^{1,a} and Alice M. Wyrwicz^{1,2}

¹Center for Basic MR Research, NorthShore University HealthSystem Research Institute, Evanston, Illinois 60201, USA

²Department of Biomedical Engineering, Northwestern University, Evanston, Illinois 60208, USA

(Received 3 December 2015; accepted 19 May 2016; published online 7 June 2016)

In this paper, we describe a modular approach to the design of an Address Generation Unit (AGU). The approach consists of development of a generic Address Generation Core (AGC) as a basic building block and the construction of an AGU from the AGCs. We illustrate this concept with AGUs capable of handling 2D- and 3D-structured data, and as well as their setup for executing 2D and 3D FFT algorithms on a Field Programmable Gate Array (FPGA). The AGUs developed using our proposed method are simple and easily expandable. Furthermore, they can potentially support irregularly structured data which are often generated from the wide variety of pulse sequences in magnetic resonance imaging. Our experimental results show that these AGUs are capable of generating addresses with a user-predefined pattern automatically at the speed of one address per clock cycle and operate at clock rates up to 80 MHz. They can operate concurrently with other processes and thus do not introduce additional operation latencies. Although we focus on applying the developed AGUs to executing 2D and 3D FFT, we expect that the modular design method should have much wider applications. *Published by AIP Publishing.* [<http://dx.doi.org/10.1063/1.4953113>]

I. INTRODUCTION

Modern magnetic resonance imaging (MRI) experiments typically collect a large number of images with high temporal and spatial resolution. It is therefore desirable to develop more sophisticated hardware capable of processing the data in real time. Much effort has been devoted to developing hardware-accelerated processing systems based on multi-core central processing units (CPUs) or graphics processing units (GPUs),¹ or application-specific integrated circuit (ASIC) and Field Programmable Gate Array (FPGA) chips.^{2–17} Among these hardware devices, FPGAs are gradually becoming more widely utilized in MR image processing in real time.

FFT algorithms are indispensable for many image processing applications such as image reconstruction, and thus a variety of methods have been developed to speed up FFT execution on hardware processors.^{7–13} FFT algorithms also have been designed as IP modules for reconfigurable processors such as FPGAs.^{14,15} However, while effective, these processors are generally unable to achieve the performance necessary to process MR data in real time as their architectures are designed without considering the unique characteristics of MR data. One of these characteristics is that with high-speed MRI, time-domain signals typically are spatially encoded irregularly and thus sampled data are irregularly structured with reference to FFT-type computations. In designing FFT hardware processors to be built on an FPGA, strategies for efficient utilization of hardware resources, operation optimization, and minimizing power consumption are all

considerably affected by an address generation unit (AGU). Its architecture has a direct impact on the performance of the FFT processors. In this paper, we discuss the unique design challenges associated with designing AGUs suited for irregular MR data and describe a modular design method as an effective solution to these issues.

In order to understand how the irregular structures of MR data are related to the design of an AGU, we first define an irregular structure of data by using an example of 2D image reconstruction. A 2D DFT of a $N \times N$ data matrix can be expressed as

$$X(m_1, m_2) = \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} e^{-j2\pi n_1 m_1 / N} S(n_1, n_2) e^{-j2\pi n_2 m_2 / N}, \quad (1)$$

where $0 \leq m_1, m_2 \leq N-1$, $S(n_1, n_2)$ represents time-domain signal samples and $X(m_1, m_2)$ represents calculated image pixels. The linear dependence of the column-wise phase, $-2\pi n_1 m_1 / N$, and the row-wise phase, $-2\pi n_2 m_2 / N$, respectively, upon n_1 and n_2 constitutes a standard 2D Fourier decoding scheme, which cannot be modified once embedded into a 2D FFT hardware processor.

In MRI data acquisition, the time-domain signal is encoded by applying field gradients which are predefined by an imaging sequence. During the encoding process, the signal is frequency-modulated by means of two space-dependent phase factors, $e^{j2\pi k_x x}$ and $e^{j2\pi k_y y}$, in a k-space formalism.¹⁸ The dependence of the modulated phases on k_x and k_y constitutes a 2D encoding scheme. With k_x and k_y digitization, the modulated phases constitute a $N \times N$ phase grid. The encoding scheme can be described as the scanning path through all the phases with increasing time, i.e., a k-space trajectory.^{18,19} The pattern of a trajectory

^aAuthor to whom correspondence should be addressed. Electronic mail: LLi@northshore.org. Tel.: (224) 364-1405. FAX: (847)492-0731.

FIG. 1. The k-space trajectories for three commonly used imaging pulse sequences. The red arrow lines indicate the trajectory in an encoding process. The blue dot lines are used to illustrate how a trajectory passes through the k space over time. The spin-warp imaging sequence defines a linear advancement of the k-space trajectory along k_x and k_y axes, respectively (a). An EPI sequence defines a zigzag pattern of the k-space trajectory (b); the trajectory changes its direction from one k_y value to next. The RARE sequence defines a k-space trajectory that jumps across several lines along k_y axis (c), i.e., $\Delta k_y > 1$.

is determined solely by the applied imaging sequence. Fig. 1 illustrates the trajectories of three commonly used imaging sequences. For example, in a spin-warp imaging sequence,²⁰ the k-space trajectory follows the k_x axis and at the same time shifts line-by-line along the k_y axis (Fig. 1(a)). We refer to such a trajectory in Fig. 1(a) as a regular trajectory, in the sense that the modulated phase along the trajectory always depends linearly on k_x or k_y over the entire k space. Accordingly, the data structure of the signal samples encoded under a regular k-space trajectory is considered to be regular because the data can directly pass through the Fourier-decoding process to generate a 2D image. However, a variety of other imaging sequences are characterized by irregular k-space trajectories. For example, in an echo planar imaging (EPI) sequence,²¹ the trajectory advances in the same direction for all even index k_y values and in the opposite direction for all odd index values, creating a zigzag trajectory (Fig. 1(b)). For a Rapid Acquisition with Relaxation Enhancement (RARE) sequence,²² the trajectory proceeds in one direction along the k_x axis but increments several lines along k_y axis, (i.e., $\Delta k_y > 1$) (Fig. 1(c)). For signal samples encoded under an irregular k-space trajectory, the data elements must be reordered correctly prior to passing through the Fourier-decoding process in order to generate a correct image.

In slower software processing implemented on a host computer, data reordering is a trivial process. However for real-time processing implemented on an FPGA, data reordering presents a number of unique challenges. Data reordering requires a large storage buffer on the FPGA chip and many additional clock cycles. An FPGA allocates almost all hardware resources for computation-orientated tasks, and thus it is more constrained in devices such as on-chip memory as compared to other types of microprocessors. Furthermore, the input data are continuously loaded to a FFT processor and must be processed immediately without any skipped clock cycles. However the efficient usage of hardware resources and clock cycles can be realized by using one or more AGUs that support irregularly structured data accesses.

Modern MRI scanners are equipped with a rich library of imaging sequences with new sequences being developed and added to the library almost daily. In theory, different AGUs would be required to support the data encoded with different imaging schemes. Due to the diversity of encoding schemes, developing individual AGUs to cover the wide range of commonly used imaging sequences would be tedious and time consuming. On the other hand, developing a mode-based AGU that can accommodate different imaging sequences under different mode settings is a major technical challenge. Here we propose a modular methodology to facilitate the design work.

II. THEORETICAL FOUNDATION

A. Empirical observations

In general, MR data structures are multi-dimensional (MD) arrays. The access to the data is realized via nested loops. It might appear convenient to address the structured data in multi-dimensional representation due to the familiarity of array properties and operations at programming level. However, due to memory physical structures, the peak performance of many types of memory is achieved only when memory accesses are sequential.²³ In order to generate 1D address patterns for accessing multi-dimensional data, we have to establish a relation that maps a MD address representation to a 1D address representation. The data structures are defined by the imaging sequence and known at compile time, and so are the index computations to the array elements via the nested loops. Therefore a logic unit is required to generate a 1D address pattern based on an address mapping equation.

An AGU is a 1D addressing device which generates a sequence of addresses. The address patterns are determined by a set of parameters. According to the description above, the main function of the AGU is to map MD address representation to 1D address representation. As the first step towards designing the architecture of the AGU, we observe the behavior of the indices of the MD array in nested loops.

Let us consider a set of data stored in a 3D array defined with the three dimension lengths of N_1 , N_2 , and N_3 . Any of these lengths is not necessarily a power of 2. At the same time, this dataset is held in a 1D array with a length of $N_1 \times N_2 \times N_3$, whose index values will be considered equivalent to the linear addresses of a memory throughout the following discussion. The 3D array indices can be converted to the 1D array index through Eq. (2), a mapping equation,

$$l = i + jN_1 + kN_1N_2, \quad (2)$$

with $0 \leq i < N_1$, $0 \leq j < N_2$, $0 \leq k < N_3$, and $0 \leq l < N_1N_2N_3$.

The form of this mapping equation (Eq. (2)) illustrates that the 1D addressing device functions as a counting system. In a decimal (base 10) counting system, if a number x is denoted as $x = kji$, its true value can be calculated by the following formula:

$$x = k \cdot 10^2 + j \cdot 10^1 + i \cdot 10^0, \quad (3)$$

where $i, j, k \in [0, 9]$. If we compare Eq. (2) with Eq. (3) term by term, the structures of these two systems are essentially the same with the exception that one has a fixed base (base 10) and the other has varying bases because any two out of N_1 , N_2 , and N_3 integers are not necessarily the same. Thus, constructing the 1D addressing device is analogous to building a counting system with varying bases. This conclusion is also true for the accesses to the higher dimensional data arrays. Our fundamental design concept was primarily motivated by these observations.

Based on these observations, we can design an AGU by a modular approach which consists of two steps: (1) design a basic building block device, called an Address Generation Core (AGC); (2) integrate AGCs to construct a composite AGU.

B. Functions of a generic AGC

A generic AGC can be considered to function as a single indexing unit. To minimize the overheads on hardware resources, the functions of the indexing unit must be further simplified and generalized. Therefore we define the functions of an AGC device by the following:

$$\begin{aligned} A(n+1) &= (A(n) + C_{in}(n) \cdot S) \bmod B, \\ C_{out}(n+1) &= 1, \text{ if } (A(n) + C_{in}(n) \cdot S) \geq B, \\ C_{out}(n+1) &= 0, \text{ else.} \end{aligned} \quad (4)$$

The AGC device has one input port C_{in} , as well as two output ports, A and C_{out} , which output an address and a carry signal, respectively. The carry signal has only two possible values: 0 or 1. The n denotes the n th clock cycle. B is base, or modulus. S is the **Stride**, which defines the address distance between two consecutive outputs from one cycle to next and can be any integer in $[1, B-1]$. B and S are the parameters which determine a pattern of the AGC's output numbers. We have implicitly assumed that in practice a number of AGCs will be cascaded together to form an AGU. Each AGC can have either one prior stage, one post stage, or both prior and post stages. Therefore, the input port C_{in} is actually connected to the output port C_{out} from the prior stage. The C_{out} output

delivers a carry signal to the post stage. As described in Eq. (4), there are two possible actions for an AGC at any given clock cycle: (1) if no carry signal is generated from the prior stage ($C_{in} = 0$), the address output A is unchanged; (2) otherwise ($C_{in} = 1$), the output A advances S from the value at previous clock cycle. In addition, if the output advancing S causes an overflow, the output value will be replaced by that from modulo operation, and at the same time $C_{out} = 1$ will be set. If an AGC is in the first stage, $C_{in}(n) \equiv 1$.

C. Architecture of an AGU

The architecture of the AGU designed to support an M -dimensional array data can be described by the following:

$$AGU(n) = \sum_{m=1}^M AGC_m(n) = \sum_{m=1}^M A_m(n), \quad (5)$$

where $AGU(n)$ represents the AGU's output at the n th clock cycle and $A_m(n)$ should be substituted by the expression of A in Eq. (4). This equation implies that the circuitry of an AGU can be constructed by M cascaded AGCs and that the AGU address output is the sum of all the individual AGC outputs (Fig. 2). The top AGC is assigned to be the first stage ($m = 0$), and thus its C_{in} is always equal to 1. For any other AGC, its C_{in} is connected to the port C_{out} of the prior AGC.

By combining Eqs. (4) and (5), the AGU does not contain as input variables any indices of the multi-dimensional array. This feature distinguishes an AGU from other address calculating devices,^{23,24} as it enables the AGU to generate a sequence of addresses with a predefined pattern automatically without the requirement of input variables. Another important feature of the architecture is that one AGC stage can be assigned to correspond to one dimension of a multi-dimension

FIG. 2. Block diagram of a composite AGU which consists of M AGCs. The AGC at the top is assigned to the first stage where $C_{in} = 1$ at any cycle.

FIG. 3. LabView code of the developed generic AGC and its graphical interface.

array which enables the AGU to handle irregular data with a relative ease, as will be demonstrated below.

III. APPLICATION EXAMPLES

We now demonstrate how this modular method is applied in designing AGUs capable of supporting 2D- and 3D-structured data, and the detailed implementation of these AGUs suitable for 2D and 3D FFT executions. The design was developed in a graphical coding environment (LabView 2014) and all the AGUs were built around an FPGA chip XC7K410T (Xilinx, Inc., San Jose, CA USA.) which sits on a NI USRP-2940R board (National Instruments, Inc., Austin, TX, USA). The chip contains 406 720 logic cells, 795 36-Kbit BRAMs, and 1540 DSP slices.

In implementing the theoretical AGC design on an FPGA, we must consider the issue of reducing the overheads on hardware resources and operation latencies. Fig. 3 shows the LabView code of the developed generic AGC and its graphical interface. The function of C_{in} in Eq. (4) is performed by a Boolean control “Enable” in order to avoid a multiplication operation. On FPGAs a multiplier requires significantly more resources than an adder or a subtractor does. The carry signal has only two logic states: TRUE or FALSE. A Reset Boolean control is also added to the AGC for clearing the residual states, which in our experience is necessary if the AGC is reused. Another parameter, Offset, is added to the AGC in order to gain flexibility of operations. Registers B and S hold parameters S and B (see Eq. (4)).

A. An AGU supporting 2D data

Only two AGCs are required to construct a 2DAGU that supports 2D data. Fig. 4(a) shows the LabView code for a workable 2DAGU. Note the two pairs of parameters, S_1 and

FIG. 4. LabView code of the 2DAGU (a) and 3DAGU (b).

TABLE I. Settings of the 2DAGU embedded into a 2D FFT processor.

FFT Stage	S_1	B_1	S_2	B_2
1st FFT	1	N_1	N_1	$N_1 \cdot N_2$
2nd FFT	N_1	$N_1 \cdot N_2$	1	N_1

B_1 , S_2 and B_2 , determine the pattern of output addresses. In considering the setup of the 2DAGU to support the execution of a 2D FFT algorithm, let us first assume that a data set is held in a 1D array with a length of $N_1 \times N_2$. The data set is partitioned into N_2 blocks, where each block contains one row with N_1 elements. The partitioned data are loaded block-by-block into a N_1 -point FFT processor. The first-stage AGC always corresponds to the dimension of the fastest-running index.

Table I lists the parameter settings for the 2DAGU to properly control data flows for the first and second 1D FFT computations. If $S_1 = 1$, the output addresses are always linear with clock cycle, i.e., $AGU(n) = n$. Note that by using 2DAGU the matrix transposition conventionally required between the first and the second FFTs is realized automatically prior to the execution of the second FFT¹⁷ and thus no additional clock cycles are wasted.

B. An AGU supporting 3D data

Three AGCs are needed to form a composite 3DAGU (Fig. 4(b)) that supports 3D-structured data. Since a higher-dimension FFT can be decomposed into multiple lower-dimension FFTs, there are many possible ways that a 3D FFT can be executed and accordingly, several possible settings of the 3DAGU. In a conventional approach, a 3D FFT on the data in an $N_1 \times N_2 \times N_3$ array can be computed by executing N_1 -, N_2 -, and N_3 -point 1D FFTs consecutively along the three dimensions. Although this execution approach may not be the most efficient, it provides a simple demonstration of how the 3DAGU can be used for controlling data flow during the execution of the 3D FFT algorithm. Unlike the case of 2D FFT computations, we must not only choose the fast-running index but also determine the slower- and the slowest-running indices. We assign the fast-running index to the first AGC, slower-running index to the second AGC, and the slowest-running index to the third AGC. After the fast-running index is chosen, there are two options for the assignments of the remaining two indices. Table II lists the settings corresponding to one of the options chosen for demonstration.

C. An AGU supporting irregular 2D data

As examples to demonstrate how the proposed method can be applied to designing an AGU that handles the irregularly structured 2D data, we designed two AGUs (2DAGU(EPI) and 2DAGU(RARE)) suitable for the 2D data acquired under an EPI and RARE encoding trajectories (Figs. 1(b) and 1(c)), respectively.

1. Design of 2DAGU(EPI)

In the case of EPI data, since the data irregularity occurs along the k_x dimension, which was assigned to be the first

TABLE II. Settings of the 3DAGU embedded into a 3D FFT processor.

FFT Stage	S_1	B_1	S_2	B_2	S_3	B_3
1st FFT	1	N_1	N_1	$N_1 \cdot N_2$	$N_1 \cdot N_2$	$N_1 \cdot N_2 \cdot N_3$
2nd FFT	N_1	$N_1 \cdot N_2$	1	N_1	$N_1 \cdot N_2$	$N_1 \cdot N_2 \cdot N_3$
3rd FFT	$N_1 \cdot N_2$	$N_1 \cdot N_2 \cdot N_3$	N_1	$N_1 \cdot N_2$	1	N_1

dimension, it was unnecessary to design a whole AGU, but instead it was sufficient to add a simple circuit to the first-stage AGC. Based on the data acquisition under EPI, the equation mapping a 2D address representation to a 1D address representation can be written by the following:

$$l = \begin{cases} i + jN_1, & \text{if } j = \text{even integers,} \\ N_1 - 1 - i + jN_1, & \text{if } j = \text{odd integers.} \end{cases} \quad (6)$$

Here indices i and j correspond to the k_x and k_y , respectively. Based on this mapping equation, we designed the architecture of 2DAGU(EPI) as shown in Fig. 5. Compared with the 2DAGU shown in Fig. 4(a), one can see that an additional minor circuit is attached to the first-stage AGC while the second-stage AGC remains unmodified. The top and bottom AGCs are denoted by AGC₁ and AGC₂, respectively. With the added circuit the first stage generates two address components, A_{11} and A_{12} . Either one of these two components takes two possible values corresponding to the parity of j . The two possible values are selected by two **Select** nodes solely controlled by the logic signal from the output of **Exclusive OR** node. With the combination of **Exclusive OR** and **Feedback** nodes, the logic signal changes whenever the parity of index j changes such that when $j = \text{even number}$, $A_{11} = i$ and $A_{12} = 0$ and that when $j = \text{odd number}$, $A_{11} = -i$ and $A_{12} = N_1 - 1$. As can be seen from Fig. 5 (top), $A = A_{11}$

+ $A_{12} + A_2$. The 2DAGU(EPI) gives the address pattern following the description of Eq. (6). When the 2DAGU(EPI) is embedded into a 2D FFT processor, the settings listed in Table I are still applied.

2. Design of 2DAGU(RARE)

In the case of RARE data, we have an address mapping equation,

$$l = i + (j \bmod R) \cdot N_1 \cdot N_{2R} + [j/R] \cdot N_1, \quad (7)$$

where R is a RARE factor and $N_{2R} = N_2/R$. Function $[x]$ gives the integer part of x . The composite AGU, 2DAGU(RARE), consists of two AGCs as shown in Fig. 5. Again the top and bottom AGCs are denoted by AGC₁ and AGC₂, respectively. Since the structural irregularity occurs in the k_y dimension which is designed to be handled with the second stage, only the output of AGC₂ has to be modified. It follows from Eq. (7) that the output A of the 2DAGU(RARE) is the sum of three address components: $A = A_1 + A_{21} + A_{22}$. A_1 is the output of AGC₁ while A_{21} is the output of AGC₂. A_{22} is generated by an additional circuit which consists primarily of **Select Node** and **Feedback Node** (Fig. 5). The circuit acts as an accumulator and is solely controlled by the carry signal from AGC₂. Whenever this carry signal is logic high, the accumulator adds to itself the value stored in register B_1 ; otherwise it adds zero (does nothing). Another **Select Node** is added and controlled by **Reset** register. The function of this second **Select** node is to clear the accumulator whenever 2DAGU(RARE) is started in a new loop. The output A_{22} of the added circuit follows the behavior described by the third term, $[j/R] \cdot N_1$, on the right side of Eq. (7). When the three address components are combined, the 2DAGU(RARE) generates the address pattern as described in Eq. (7). When 2DAGU(RARE) is embedded into a 2D FFT processor, the settings summarized in Table I are still applied except that in the first FFT, $S_2 = N_1 \cdot N_{2R}$.

IV. EXPERIMENTAL VERIFICATIONS

We tested and validated the functions and capabilities of the 2DAGU, 2DAGU(EPI), 2DAGU(RARE), and 3DAGU through a series of experiments. Specifically, the experiments consisted of three parts: computer simulations on the host PC; address generations of these AGUs when all the individual AGUs were built and operated on the FPGA chip; and image reconstruction using a 2D FFT processor with the embedded AGUs. In part 1, we placed an AGU into a loop with a number of iterations equal to or larger than $N_1 \cdot N_2$ in cases of 2DAGUs, or $N_1 \cdot N_2 \cdot N_3$ in the case of 3DAGU. For each of these AGUs, we observed and recorded the address patterns

FIG. 5. LabView code of the 2DAGU(EPI) (top) and 2DAGU(RARE) (bottom) designs. The top and bottom AGCs are denoted by AGC₁ and AGC₂, respectively.

with various N_1 , N_2 , and N_3 values. For the purpose of testing, these numbers were not necessarily a power of 2, and any two numbers were not necessarily equal to each other. The results verified that these AGUs were able to generate the address patterns that followed the descriptions given in Eqs. (2), (6), and (7). In addition, it was verified that when the number of iterations was set to a multiple integer of $N_1 \cdot N_2$, the total address patterns generated from the 2DAGUs were merely multiple copies of the pattern in $[0, N_1 \cdot N_2 - 1]$. Furthermore, the results verified that the reset functioned as expected and an AGU was clear whenever entering into a new loop. These two features are required in the cases of multi-slice image reconstruction.

In part 2, we built the AGUs on the FPGA chip to generate several sequences of addresses with various N_1 , N_2 , and N_3 values. Compared with the simulation tests on the host PC, there was one important difference in the testing setup: each of the AGUs was placed and operated in a single-cycle timed loop (SCTL) on the FPGA in which an AGU generated one address per clock cycle. At first glance, an AGU placed inside the SCTL may seem unable to generate correct addresses because the state of one AGC may be controlled by the trigger signal from the AGC in the prior stage. Let us take 2DAGU (Fig. 4) as an example. The correct output of the bottom AGC relies on the trigger (carry) signal from the top AGC. For conventional execution, the bottom AGC is seemingly unable to generate the correct output at current clock cycle. In fact, to satisfy the requirement of one iteration latency, the compilation removes flow-control logic in such way that all connected components inside the SCTL are synthesized to function like a synchronous circuit.²⁶ Data are not latched while following from one AGC to next within one clock cycle. Because of this unconventional logic synthesis, although the successful operations of the AGUs verified on the CPU-based tests, it was necessary to verify that the behavior of the developed AGUs followed the mapping equations when the AGUs were placed in a SCTL.

Each AGU placed in the SCTL was compiled successfully with clock rates up to 80 MHz. With a 32-bit data format, these AGUs offer memory access bandwidth up to 320 MB/s. Table III lists the hardware usage for the 2DAGU, 2DAGU (EPI), 2DAGU (RARE), and 3DAGU. For each AGU, the hardware usage represents less than 1.5% hardware resources of the FPGA chip.

Fig. 6 shows the output of the 2DAGU versus clock cycle n with $N_1 = 8$ and $N_2 = 6$. Small numbers of N_1 and N_2 were chosen for clarity of illustration. Figure 7 shows the output addresses of the 3DAGU versus clock cycle n with $N_1 = 12$, $N_2 = 8$, and $N_3 = 4$. The figure displays only two sequences of addresses generated by the 3DAGU with S and B settings corresponding to those listed in the 2nd FFT and 3rd FFT of

FIG. 6. The output addresses, $AGU(n)$, of the 2DAGU versus clock cycle n . The vertical axis represents address and the horizontal axis represents clock cycle n . The two sequences of the output addresses were generated from the 2DAGU, respectively, with the settings listed in Table I. The linear pattern of the addresses (red squares) was generated when index i was chosen as the fast-running index. The other pattern (yellow circles) was generated when index j was chosen as the fast-running index.

Table II. Fig. 7(a) shows the addresses from the 3DAGU when index j is chosen as the fast-running index. Fig. 7(b) shows the output addresses when index k is chosen as the fast-running index.

Fig. 8(a) shows the addresses of the 2DAGU(EPI) versus clock cycle n with $N_1 = 12$ and $N_2 = 8$. The S and B settings were chosen to correspond to those listed in Table I. As expected, the sequential addresses show the increase in the address values in first address block and the decrease in the address values in the next block. Fig. 8(b) shows the addresses of 2DAGU(RARE) versus clock cycle n with $N_1 = 8$ and $N_2 = 16$ and rare factor $R = 8$. All settings of S and B were obtained by means of Table I except that $S_2 = N_1 \cdot N_2 R = 16$.

In part 3, the functions of the developed AGUs were further validated by employing them to support 2D and 3D image reconstruction. Since an FPGA-based 3D processing

FIG. 7. The output addresses, $AGU(n)$, of the 3DAGU versus clock cycle n . The vertical axis represents output address and the horizontal axis represents clock cycle n . The addresses are displayed respectively when index j is chosen as the fast-running index (a) and when index k is chosen as the fast-running index (b).

TABLE III. The usage of the hardware resources of the developed AGUs on the FPGA.

	2DAGU	2DAGU(EPI)	2DAGU(RARE)	3DAGU
Slice registers	3836	4005	3770	4275
Slice LUTs	2727	2863	2683	3012
Block RAMs	5	5	5	5

FIG. 8. The output addresses, $AGU(n)$, of the 2DAGU(EPI) (a) and 2DAGU(RARE) (b) versus clock cycle n . The vertical axis represents output address and the horizontal axis represents clock cycle n . For each of these AGUs, the output $AGU(n)$ varies non-linearly with clock cycle n .

module was not available, the application of 3DAGU for 3D image reconstruction was validated by means of computer simulations performed on the host PC. In particular, the 3D image reconstruction was performed on a 3D image dataset with matrix of $128 \times 128 \times 32$ acquired from spin-echo (SE) experiments.

To verify the function of the 2DAGUs when they are embedded into a 2D FFT processor for image reconstruction, we modified the design of a previous 2D processing module.¹⁷ A functional block diagram of the modified module, 2D image generator, is displayed in Fig. 9. The circuit inside the dashed line is the basic core of processing a 2D FFT algorithm. A complete process of generating a single $N_1 \times N_2$ image can be carried out in two main steps. In the first step, which is denoted by the first FFT, input data stream to the processor via the input FIFO buffer block by block of N_1 elements. After passing through the 1D FFT processing unit, the blocks of data are stored as intermediate results in the on-chip memory SRAM. The data-storing order is solely controlled by the first address generation unit, AGU_1 . At the moment when

FIG. 9. Functional block diagram of an FPGA-based 2D image generator. The 2D FFT core inside the dashed line is common for both regular and irregular image data. AGU_1 and AGU_2 are the address generation units that are used to reorder data so that to process different data acquired under different imaging sequences, only these AGUs are changed while the 2D FFT core remains the same.

all the data elements are stored in the SRAM, the first step is completed and the second step, which is denoted by the second FFT, is started. The data elements are read in from the SRAM and enter the second 1D FFT processing unit in the order solely controlled by the second address generation unit, AGU_2 . At the moment when all the data elements pass through the second 1D FFT unit, the second step is completed, and thus a 2D image is generated. To understand the roles of AGU_1 and AGU_2 , it is important to examine the difference between the first and second FFT. In the first FFT, not all input data elements are available at any moment prior to the 1D FFT. The data elements are processed as soon as they arrive, which implies that the data order can only be modified after the FFT by means of AGU_1 when they are stored in the SRAM. However, in the second FFT, all data elements that have been stored in the SRAM are available at any moment, and thus random access to the data is possible. The data order can be adjusted prior to the second 1D FFT. Furthermore, writing data to and reading in data from the SRAM are often concurrent with the progress of other processes. By properly embedding AGU_1 and AGU_2 separately into the data writing and reading processes, we are able to re-regulate the data order with AGU_1 and accomplish data matrix transpose operations¹⁷ with AGU_2 without wasting additional clock cycles.

The modified FPGA module was used for reconstructing spin-echo (SE), EPI, and RARE images. Only AGU_1 was substituted by the 2DAGU, 2DAGU(EPI) and 2DAGU(RARE), respectively, according to the data structures; while AGU_2 was always the same which was 2DAGU. The image reconstruction was performed on the multi-slice image datasets with matrix of $128 \times 128 \times 8$ and $128 \times 128 \times 32$ which were acquired from a structural phantom consisting of a water-filled plastic vial (25 mm in diameter) with multiple glass tubes inside, under SE, EPI, and RARE (rare factor = 8) sequences. The settings used in the first FFT were as follows. For 2DAGU and 2DAGU(EPI), $N_1 = N_2 = 128$, $S_1 = 1$, $B_1 = S_2 = 128$, and $B_2 = 16\,384$. For 2DAGU(RARE), all these settings except S_2 were the same and $S_2 = 2048$. In the second FFT, $S_1 = 128$, $B_1 = 16\,384$, $S_2 = 1$, and $B_2 = 128$. Fig. 10 illustrates single-slice SE (a), EPI (b), and RARE (c) images selected from the reconstructed multi-slice image sets. These images are visually identical, except the image in Fig. 10(b) in which ghosting artifacts are visible. This type of image artifacts often observed in EPI images is caused by the misalignment of the even/odd echo centers, and is not an indication of 2DAGU(EPI) malfunction. The artifacts could be removed by using a program to realign the even/odd echo centers. In this work, we did not implement such a program in our processing module. We also measured the processing rate of the module with the various 2DAGUs. The average time required for processing a single-slice 128×128 image was $\sim 955\ \mu\text{s}$ at a clock rate of 80 MHz. This time is equivalent to a processing rate of ~ 1000 frames/s.

In summary, these experiments verify that the address generation units developed with our method are able to produce the address patterns that exactly follow the mapping equations (Eqs. (2), (6), and (7)). When integrated with 2D FFT processing, the units are able to operate properly to support real-time image reconstruction.

FIG. 10. Examples of images generated using the FPGA-based processing module from the data acquired from a structural phantom under SE (a), EPI (b), and RARE (c) sequences, respectively. These single-slice images were selected from the multi-slice image sets.

V. DISCUSSION

There are several advantages of the modular design methodology as compared to other existing designs.^{23–25} The architecture of an AGU is simple and intuitive. It can be easily expanded to support data with higher-dimensional structures simply by cascading more AGCs. As can be seen from the architecture of an AGU, one AGC is allocated to one dimension of a multi-dimensional array. This relative independence between the AGCs enables us to make minor modifications to a specific dimension without affecting the others. This feature is especially useful for handling irregular MR data because in practice the irregular pattern of a structured data is usually confined to one particular dimension. As we have demonstrated, the data encoded under the irregular trajectories (Figs. 1(b) and 1(c)) defined with the EPI and RARE sequences can be re-regularized by using the 2DAGU(EPI) and 2DAGU(RARE). More complex encoding trajectories can be supported with the proposed AGU architecture with only minor modifications. In summary, with our modular design method, the structural irregularities of one dimension can be specifically handled by the AGC allocated for that dimension without affecting the remaining AGCs. Thus a change in the internal structure of a target data would not require redesigning the entire AGU. Instead it would be sufficient to modify one of the AGCs, thus saving considerable designer development time. Although we have used simple examples to demonstrate how an AGU is developed to perform memory access schemes that support irregular data, we expect that the proposed modular method will be effective for designing an AGU capable of performing more complex memory access schemes.

Throughout our discussion, we have assumed that either 2D or 3D FFT algorithms are executed sequentially with 1D FFT processors. However, the 2D/3D FFT can be decomposed into several 1D/2D FFTs. By taking advantage of the internal parallel processing capabilities of an FPGA, the FFT algorithms are usually executed by simultaneously employing multiple 1D/2D FFT processing elements (PEs). In this case, multiple memory modules are employed and accordingly, parallel memory accesses are controlled by multiple AGUs. There are two schemes to address memory: centralized and distributed. In the centralized addressing scheme, local addresses of all memory modules are parts of one central address space. An address routing circuit is

required to distribute these calculated local addresses to the corresponding modules. A synchronic signal is needed to trigger the AGUs for inter-PE coordination. In the distributed addressing scheme, the local addresses of each memory module are generated independently. The address routing and trigger signal generation circuits can be avoided. As we have shown, our AGUs are not controlled with a trigger signal (Figs. 4 and 5) and are therefore only suitable for the distributed addressing schemes. The independent AGUs allow designers flexibility of routing data between a processing unit and its corresponding memory module.

The architecture of an AGU designed with this modular method supports hardware modules for processing streaming data. In cases of streaming data as the input data to the multi-dimensional FFT processors, initially not all necessary data elements may be ready at any moment in the stage of execution of the first FFT algorithm. Thus, the memory access can be performed only in a single read/write operation. In later stages, the intermediate results are stored in memory so that all data elements may be ready for different processors. Then the memory access can be performed in single access mode or in burst mode. Depending upon memory hardware structures, in general, the burst-mode access would significantly boost the memory access throughput. The architectures of our AGUs, as in their current form, support only the single memory access. However with slight modification to their architecture, these AGUs would be capable of supporting the burst-mode memory access.

VI. CONCLUSION

A modular method has been proposed to design address generation units capable of supporting irregularly structured data. As demonstrated, the AGUs supporting 2D/3D-structured data have been developed and implemented on a Kintex-7 FPGA. They can run at clock rates up to 80 MHz and are able to output an address in a single clock cycle. Without significant modifications, the architecture of these AGUs may be easily expanded to potentially support the data with more complex structures. Although the design method was initially motivated by solving the issue associated with the irregular MR data, we expect that the method would have much wider applications.

ACKNOWLEDGMENTS

This work was supported by NIH Grant Nos. R01 NS44617 and 1S10RR15685.

- ¹S. Stone, J. Haldar, S. Tsao, W. Hwu, B. Sutton, and Z. Liang, *J. Parallel Distrib. Comput.* **68**, 1307 (2008).
- ²R. Ning, G. Yang, and G. Li, *Rev. Sci. Instrum.* **82**, 054301 (2011).
- ³L. Xiao and W. Wang, *Rev. Sci. Instrum.* **80**, 124703 (2009).
- ⁴L. Xiao, B. Sun, Y. Ma, and R. Zhao, *Rev. Sci. Instrum.* **84**, 054702 (2013).
- ⁵W. Tang and W. Wang, *Meas. Sci. Technol.* **22**, 015902 (2011).
- ⁶I. L. Dalal, A. L. Kirpalani, and F. L. Fontaine, in *Proceedings of the 28th IEEE EMBS Annual International Conference, New York City, 30 August–3 September 2006* (IEEE, 2006), pp. 1897–1900.
- ⁷I. L. Dalal and F. L. Fontaine, in *Proceedings of the 40th Asilomar Conference on Signals, Systems and Computers, Pacific Grove, CA, 29 October–1 November 2006* (IEEE, 2006), pp. 1860–1864.
- ⁸A. F. Al-allaf and S. A. Dawwd, *Al-Rafidain Eng.* **19**(3), 76 (2011).
- ⁹H. Kee, S. S. Bhattacharyya, N. Petersen, and J. Kornerup, in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, Las Vegas, Nevada, March 2008* (IEEE, 2008), pp. 1413–1416.
- ¹⁰H. Kee, S. S. Bhattacharyya, N. Petersen, and J. Kornerup, in *Proceedings of the International Conference on Distributed Smart Cameras, Como, Italy, August 2009* (IEEE, 2009), pp. 1–8.
- ¹¹S. A. Christe, M. Vignesh, and A. Kandaswamy, *Int. J. VLSI Commun. Syst. (VLSICS)* **2**(4), 95 (2011).
- ¹²S. Hasan, S. Boussakta, and A. Yakovlev, *Am. J. Eng. Appl. Sci.* **5**(1), 25 (2012).
- ¹³C. L. Yu, K. Irick, C. Chakrabarti, and V. Narayanan, *IEEE Trans. Circuits Syst.* **58**, 755 (2011).
- ¹⁴C. L. Yu, J. S. Kim, L. Deng, S. Kestur, V. Narayanan, and C. Chakrabarti, *J. Signal Process. Syst.* **64**, 109 (2011).
- ¹⁵Xilinx, LogiCORE IP Fast Fourier Transform v9.0, Product Guide for Vivado Design Suite, 2015, http://www.xilinx.com/support/documentation/ip_documentation/.
- ¹⁶See <http://www.ti.com/lit/an/sprabb6b/sprabb6b.pdf> for TI Application Report: SPRABB6B-June 2010-Revised, January 2013.
- ¹⁷L. Li and A. M. Wyrwicz, *J. Magn. Reson.* **255**, 51 (2015).
- ¹⁸S. Ljunggren, *J. Magn. Reson.* **54**, 338 (1983).
- ¹⁹T. B. Donald, *Med. Phys.* **10**(5), 610 (1983).
- ²⁰W. A. Edelstein, J. M. S. Hutchison, G. Johnson, and T. Redpath, *Phys. Med. Biol.* **25**, 751 (1980).
- ²¹M. K. Stehling, R. Turner, and P. Mansfield, *Science* **254**, 43 (1991).
- ²²J. Hennieg, A. Nauerth, and H. Friedburg, *Magn. Reson. Med.* **3**, 823 (1986).
- ²³S. D. Kaushik, C. H. Huang, J. R. Johnson, R. W. Johnson, and P. Sadayappan, in *Proceedings of the 1993 ACM/IEEE Conference on Supercomputing, Portland, OR, USA, 15–19 November 1993* (IEEE, 1993), pp. 656–665.
- ²⁴K. E. Vistnes and O. Soeraasen, in *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2005), Denver, CO, 3–8 April 2005*, Paper 164a.
- ²⁵C. Galuzzi, C. Gou, H. Calderon, G. N. Gaydadjiev, and S. Vassiliadis, *J. Signal Process. Syst.* **57**(1), 33 (2009).
- ²⁶See <http://www.ni.com> for NI LabVIEW High-Performance FPGA Developer's Guide.