Published in final edited form as:

J Magn Reson. 2015 June ; 255: 51–58. doi:10.1016/j.jmr.2015.03.007.

Design of an MR image processing module on an FPGA chip

Limin Li¹ and Alice M. Wyrwicz^{1,2}

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

²Department of Biomedical Engineering, Northwestern University, Evanston, IL, USA

Abstract

We describe the design and implementation of an image processing module on a single-chip Field-Programmable Gate Array (FPGA) for real-time image processing. We also demonstrate that through graphical coding the design work can be greatly simplified. The processing module is based on a 2D FFT core. Our design is distinguished from previously reported designs in two respects. No off-chip hardware resources are required, which increases portability of the core. Direct matrix transposition usually required for execution of 2D FFT is completely avoided using our newly-designed address generation unit, which saves considerable on-chip block RAMs and clock cycles. The image processing module was tested by reconstructing multi-slice MR images from both phantom and animal data. The tests on static data show that the processing module is capable of reconstructing 128×128 images at speed of 400 frames/second. The tests on simulated real-time streaming data demonstrate that the module works properly under the timing conditions necessary for MRI experiments.

Keywords

FPGA; real time; image processing; 2D FFT; graphical coding

1 Introduction

Speed of MR imaging is primarily determined by the rates of data acquisition and image processing. MRI systems equipped with a multi-channel data-acquisition module, particularly high-channel-count, 64- [1], 96- [2] and 128-channel [3] systems, allow for high-speed data acquisition, collecting a huge amount of data within a short time. In order to obtain the full benefit of these data-acquisition capabilities, it is desirable to have more sophisticated hardware capable of running increasingly complicated image processing algorithms at comparable speeds or in real time. Therefore much effort has been devoted to

© 2015 Published by Elsevier Inc.

Correspondence should be addressed to: Limin Li, Ph.D, Center for Basic MR Research, NorthShore University HealthSystem Research Institute, 1033 University Place, Suite 100, Evanston, IL 60201, Tel: (224) 364-1405, FAX: (847)492-0731, LLi@northshore.org.

Publisher's Disclaimer: This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final citable form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.

developing hardware-accelerated processing for parallel MRI. In some of these methods, computation-intensive image processing is realized by a host computer that is equipped with multi-core general-purpose central processing units (CPUs) or graphics processing units (GPUs) [4–5]. In others, special image processing boards are built upon digital signal processing (DSP) chips [6–8] that include application-specific integrated circuit (ASIC) or Field Programmable Gate Array (FPGA) chips [10–19]. Among these hardware devices, an FPGA offers the characteristics that are most desirable and suitable for real-time data processing.

An FPGA is a semiconductor integrated-circuit (IC) device consisting of a large number of generic logic cells along with input/output (I/O) blocks and interconnections [9]. The interconnections between the logic cells can be configured to form digital circuits based on user-defined functionalities. An FPGA has a number of unique characteristics. It provides precise timing and enables time deterministic processes. Multiple processing tasks can be carried out simultaneously and independently and the digital circuits on an FPGA are reconfigurable and reusable. Thus, FPGA technology is very attractive for real-time parallel MRI in terms of processing speed, timing jitter, flexibility, development time and cost. In fact, FPGA technology has been gradually incorporated in digital receiver/transmitter modules of modern research MRI systems. Initially, FPGA functions in most of these modules typically were confined to timing and synchronization control while other specialized DSP chips were responsible for computationally-intensive data processing [4–8]. Recently, with increasing capacities and decreasing cost of FPGAs, researchers have demonstrated that hardware-intensive computations such as image reconstruction and filtering can be carried out inside a single FPGA chip [13,20,21]. When a single-chip FPGA has enough circuit capacity to accommodate digital receiver and image processing modules, FPGA technology offers the possibility for processing digital signals at all stages, from RF inputs to final images, in real time.

As the first step towards real-time image processing for parallel MRI, we have designed and implemented a 2D image processing module on a single-chip FPGA using the LabView platform (LabView 2013, National Instruments, Inc., Austin, TX USA.), which provides a graphical programming environment. Our design method is distinguished from the previously reported work [13] in two main respects. By using graphical programming, we are able to view more intuitively the details of concurrent processing operations inside an FPGA, and to optimize relatively easily the desired circuits under strict timing constraints and tight hardware margins. In addition, we have designed a new address generation unit (AGU) which helps minimize on-chip memory usage without a performance penalty so that the processing module can run more efficiently.

2 Design

2.1 FPGA platform

Our image processing module is designed around an FPGA IC chip XC6SLX45 (Xilinx Inc., San Jose, CA USA). The FPGA contains 43,661 logic cells, 58 DSP48A1 slices that provide high-performance arithmetic and signal processing, 116 18-Kbit block RAMs and 358 I/O pins. It has five first-in first-out (FIFO) memory channels for facilitating data transfers

between the FPGA chip and external devices. The FPGA chip runs at a basic clock rate of 40MHz, but could run at higher clock rates. It sits on a NI sbRIO-9636 board (National Instruments, Inc., Austin, TX USA). The board also provides a real-time processor PowerPC running at 400MHz. Onboard ADC and DAC chips are provided for facilitating analog inputs to and outputs from the FPGA. There are 16 single-ended or 8 differential 16-bit analog input channels, four 16-bit analog output channels and 28 digital I/O lines. Analog signals can be sampled at a rate up to 200 KS/s. The board has a built-in 10/100 Mbps/s Ethernet transceiver that can be used for fast transfer of commands or data between the FPGA and remote devices such as a host computer. Other communication channels can also be established via RS232/RS485 serial and USB ports.

2.2 Functions of a 2D image processing module

Inside the processing module computation tasks are divided into two general stages: tasks performed prior to image reconstruction (i.e., pre-processing) and the image reconstruction itself. The pre-processing tasks consist of baseline correction, low-pass filtering and digital decimation (Fig. 1). At the heart of the processing module is a 2D-FFT core, which performs the basic algorithm used for MR image processing. 2D and 3D images can be reconstructed in a variety of ways by repeated applications of this core. Thus, we will primarily describe the design and implementation of 2D-FFT algorithm processing inside an FPGA chip.

2.3 Basic considerations

An FPGA is intended to perform algorithm computations at high speeds. It allocates almost all hardware resources for this purpose, and thus it is more constrained in other devices such as on-chip memory as compared with other types of microprocessors. In this design we considered the 2D FFT processing core as a basic block upon which the digital circuits for processing more complicated algorithms will be built. This requires the core to be compact and highly portable. Given the limited 2Mbits block RAMs available in our FPGA chip, we chose to design the processing core to handle a 128×128 image in a single pass. Although it is possible to scale up the design for processing an image with higher pixel resolution (256×256 pixels for example), in our case, we would have to utilize off-chip memory and so we limit our resolution to 128×128 in this work. Basic considerations in the design are to minimize usage of hardware resources while maintaining data integrity and data coherence.

2.4 2D FFT Algorithm Computations

2D FFT computation tasks actually consist of three sequential steps: first 1D FFT on the column of a 2D data matrix, row-column transposition on the result matrix and second 1D FFT on the column of the transposed data matrix. In order to minimize the number of memory accesses the 2D data matrix must be transposed before the second 1D FFT starts. Usually, the matrix transpose is accomplished by directly exchanging columns with rows. Hardware resource usages required for execution of a complete 2D FFT on an FPGA chip are illustrated in Fig. 2. Besides the hardware resources for processing functions, four memory blocks are allocated for temporary data storage. Two FIFO buffers are used for storing input and output data respectively, one SRAM block, for storing intermediate results after the first FFT, and the second SRAM block, for storing results after matrix transposition. Because the clock rates of an FPGA chip can be set much faster than the rates

of the data transfer between the chip and external devices, the FIFO buffer sizes can be set small. However, if an $N_x \times N_y$ image is reconstructed, the two SRAM blocks must be capable of holding $N_x \times N_y$ data points each. In the cases of $N_x = N_y = 128$, the occupied SRAMs are significant given the limited on-chip memory. Thus direct transposition of a matrix not only occupies large on-chip hardware resources but also uses many clock cycles, directly lowering the throughput of the image processing.

Several methods [13, 22–25] have been proposed to speed up the matrix transposition. In some of these methods, proper addresses are calculated for read/write access to the SRAM storing the intermediate results. However, the address patterns must be generated in advance and saved in a separate buffer before the 2D FFT starts, which requires one additional memory buffer. In others, matrix elements must be swapped many times, which involves significant numbers of clock cycles. In addition, 2D data arrays are not allowed in FPGA programs in the LabView platform. Therefore we must use a 1D array to hold the 2D image data during any processing task within the FPGA chip. Conventional 2D matrix transposition must be accomplished through appropriate addressing the on-chip memory that holds the intermediate results between the two 1D FFT computations. In this work, we have designed an address generation subVI, named Address Generation Unit (AGU), to avoid direct matrix transposition.

2.5 Address Generation Unit

The working principle of AGU can be explained by comparing the 2D FFT computations on the data held in 1D and 2D arrays (Fig. 3). In case of the 2D array, let us assume that the array is $N_x \times N_y$ with indices $i \in [0, N_x - 1]$ and $j \in [0, N_y - 1]$. The top portion of Fig. 3 illustrates the execution of 2D FFT by two consecutive steps: the first 1D FFT on rows, followed by the second 1D FFT on columns of the 2D data array, where $N_x = N_y = 8$ is assumed. The bottom of the figure illustrates the implementation of 2D FFT on a 1D data array presenting an $N_x \times N_y$ image. Here 1D-array indices can be considered as addresses of an on-chip memory buffer. In the first step, the FFT is performed repeatedly on the 1D-array data block by block, each block having N_x points. The resulting data points from the first FFT are written back to the same 1D array with linear array indices. If the data in the 2D array are held in a 1D array with index l , we have a relation that maps 2D-array indices to 1D-array indices:

$$l = i + jN_x \text{ for } i \in [0, N_x - 1] \text{ and } j \in [0, N_y - 1]. \quad [1]$$

Thus, when the second FFT is performed on the updated 1D array, a block of N_y points are read in with each point located at indices l that must be advanced at a step of N_x , as indicated by blue arrows Fig. 3 (bottom). Then the second FFT is performed again on second block of N_y points in a similar manner. All points in the second block correspond to the right-shifted points in the first block. Fig. 3(bottom) illustrates this process, where all points with the same color represent a single N_y -point block. This process is repeated for index i from 0 to $N_x - 1$.

It would be straightforward to design a small circuit on the FPGA to perform the operations in Eq. [1]. However in order to minimize the usage of hardware resources, the AGU is designed without direct operations of Eq. [1], but based on the recursive process depicted in Fig. 3. Fig. 4 shows the LabView code and interface for the AGU.

The patterns of output numbers are determined by setting five controls: **Samples**, **D**, **Offset**, **Reset** and **Enable**. The output number is held in indicator **Count**. All numeric controls, indicators and constants are in 32-bit unsigned integer type. Normally **Samples** is set equal to the total output numbers, $N_x \times N_y$, for example, in 2D FFT computations; but in some other applications it may be set to be smaller. **D** is set to the index distance between adjacent output numbers. Boolean control **Enable** can be set On/Off to determine linear or non-linear patterns of output numbers. Another Boolean control, **Reset**, can be set to be True to force output number zero.

The AGU has several important features. After optimization, it generates one address index per clock cycle. Addressing schemes can be changed at run time. It is also capable of generating correct addresses even if the total number is not a power of 2. In addition, the AGU can perform other functions that are useful for organizing the acquired data for some MRI experiments.

There are several operation modes for general applications. Examples of two different modes relevant to this work are listed in Table 1. With mode 0 AGU generates a linear output pattern. With mode 1 it generates the output pattern that can be used for the addresses on an intermediate buffer in 2D FFT computations, which is explained in greater detail below. This output pattern is the same as the bit-reversed address pattern that is often used in FFT hardware [24]. However it should be noted that the bit-reversed addressing requires that the buffer size be equal to a power of two, whereas the AGU here works without such a requirement.

Table 2 provides examples of two selected patterns of the output numbers generated by choosing two different settings. The numbers on the first row indicate the order in which the output numbers are generated. Any series of output numbers are generated by placing AGU in a **For** loop with 20 iterations. The output numbers shown with settings 0 and 1 actually represent the address indices used for an intermediate buffer in execution of 16×16 2D FFT.

2.6 Performing 2D FFT

We chose National Instruments' (NI) fixed-point 1D FFT subroutine (subVI) for the 2D FFT processing operations. This point-by-point FFT subVI is suitable for processing streaming data. One important feature of this subVI is that it has been optimized internally so that it can be placed in a single-cycle timed loop (SCTL). Fig. 5 shows LabView graphical code, an FPGA program, for performing 2D FFT computations on a 1D array of data. The FPGA program consists of two SCTLs. The first 1D FFT is carried out in the top loop (First FFT Loop in Fig. 5); while the second 1D FFT is carried out in the bottom loop (Second FFT Loop in Fig. 5). All operations inside a loop are completed in a single clock cycle (one tick) which is equivalent to 16.67ns for our FPGA chip.

In First FFT Loop, input data via ToFPGA FIFO first pass through a window function and then to the FFT. The FFT subVI accepts incoming data at one point per clock cycle until enough input points have been accepted, at which time it starts generating output data at one point per clock cycle. The resulting data are stored in buffer Mem0. The window filtering function is inserted prior to the FFT in order to improve signal-to-noise ratio (SNR) of the processed data and to minimize spectral leakage. Unlike the example seen in Fig. 3, here the output data of the FFT are stored in buffer Mem0 at non-linear addresses generated by AGU with mode 1 (see Table 1).

In general, when data are transferred from an external device to the FPGA, a handshaking scheme must be set up to avoid possible data losses due to a difference between data transfer and processing rates. The FFT and window filter subVIs have built-in terminals for implementing a handshaking through **Case** structure. The FFT subVI does not generate valid output every clock cycle. Thus another **Case** structure is set up to ensure that the buffer Mem0 can be accessed only when a valid output point is available. After all valid output points are written to buffer Mem0, the first FFT computations are completed and this processing loop is forced to stop.

In Second FFT loop, the data points of the intermediate results are first read in sequentially from buffer Mem0 at linear addresses and then passed to the FFT subVI. The output of the FFT subVI is connected to ToHost FIFO buffer through which the processed data are transferred to a host computer. Whether or not the FFT is operated depends on the availability of space in ToHost FIFO. The status of ToHost FIFO is checked constantly by “Get Number of Elements to Write” node. Only when a vacated spot is available is the FFT performed. The first **Case** structure ensures that the data stored in buffer Mem0 can be delivered to the input of the FFT when the FFT subVI is ready while the second **Case** structure ensures that only valid output data from the FFT can be written to ToHost FIFO. After all valid data points are written to ToHost FIFO, the computations of the second FFT are completed and the second FFT loop is forced to stop. Fig. 5 demonstrates one way AGU is used for 2D FFT algorithm computations, although if we swap the modes of AGU between these two loops, we will still get correct execution of 2D FFT.

To efficiently move data between processing functions, real(I)/imaginary(Q) components of the data are packed and moved together. In particular, the I/Q components, represented by two 16-bit fixed-point (FXP) numbers, are combined into a single 32-bit unsigned-integer number before the data are moved. Prior to any arithmetic operations the 32-bit unsigned-integer number is split into two 16-bit FXP numbers. The dynamic changes in data format are accomplished by using two functions, “2FXP16To1U32” and “1U32To2FXP16”, respectively.

Special attention must be paid to proper timing in the read/write access to the same memory buffer Mem0. Because two processing loops run simultaneously, the latest data in the memory must be read in before next updating. For this purpose, we have used two switch controls (not shown in Fig. 5) for each loop in the FPGA program. The switch control states can be scheduled in such a way that while the first 1D FFT is being executed in First FFT Loop, Second FFT Loop is locked to an idle state, and vice versa.

We chose 16-bit FXP data format for all processing operations on the FPGA chip. Compared with 32-bit floating-point data format, the FPGA circuits running with FXP data format utilize less hardware resources. When using FXP data format, certain measures must be implemented to avoid overflow errors. Some of these measures are built into the basic operations on the LabView platform. However, the intermediate results from more complicated processing functions should be monitored and scaled properly. For processing previously-stored data or real-time data from MRI experiments on the same subject, the proper scalar numbers can be obtained by trial runs. This simple measure was taken in our design. However if it is impossible to estimate and measure a dynamic range of the data sets being processed, more effective measures may be implemented [13].

2.7 Pre-processing portion

In the pre-processing portion, I/Q digital components flow in dual channels. At the digital filtering stage (Fig. 1), we chose the 4th order Butterworth lowpass filters for each channel. At the decimation stage, down sampling converts the bandwidth of the digital signals to the bandwidth of interest, which, in MRI, corresponds to the frequency range covered in a chosen FOV in the readout direction. The decimation function is carried out by using LabView accumulators. The input data from the digital filter stage are stored in two on-chip memory buffers. The output data of this stage should be passed to the input of the 2D FFT core. In practice, the pre-processing functions runs in a separate loop and their effective processing rates, limited by signal sampling rates in many MRI applications, are much slower than the FFT processing loop rates. Thus a very small FIFO buffer is sufficient to serve as a transfer channel of the output data to the 2D FFT core.

3 Implementation

We tested the image processing module by reconstructing multi-slice MR images. The tests were carried out separately on streaming data and static data.

3.1 A host computer and control software

The FPGA board was connected by an Ethernet cable to a local network. The host personal computer (PC) (HP HPE-380t) is equipped with an i7 CPU 960 with 12GB DRAM. The PC runs at 3.2GHz under Windows 7. The PC was also connected to the same local network.

We developed a graphical user interface (GUI) program in the LabView platform for running the tests. The GUI program consists three sections. In Initialization section, the initial states of the FPGA are set through FPGA interface components. A configuration bitfile that defines the hardware circuitry for performing the 2D FFT is downloaded from the PC to the FPGA. The host sides of ToFPGA and ToHost FIFOs are configured to be deep enough to hold all data points of a single 128×128 image to prevent possible data losses during transfer. Parameters of raw data sets such as image pixel resolution and the number of slices are set on the FPGA. The FPGA is then set to start running but in an idle state. In Main section, there are two loops, the sending data and receiving data loops, running simultaneously. In the sending data loop, raw data of a multi-slice image are transferred block by block to the FPGA through ToFPGA FIFO. Each block of data holds a single

128×128 image. Because data processing on the FPGA is much faster than the data transfer between the PC and the FPGA, the 2D FFT processing is not initialized until the FPGA receives a trigger signal from this loop when the host PC is ready. The receiving data loop is responsible for receiving, storing and displaying the processed data from the FPGA board. During the 2D FFT processing, the PC is locked in an idle state in which data are neither sent nor received. Valid output data points are constantly accumulated in ToHost FIFO. Whenever the accumulated points reach the total points of one frame, the FPGA notifies the PC that the 2D FFT processing is finished and results are ready for transfer. Then the whole block of the processed data is burst-read into the PC. At this point, the PC sends a signal to the FPGA to restart the 2D FFT processing for next block of raw data. The Close section simply closes the FPGA program and provides an option to save the processed data on a local hard drive. All raw data previously stored on the main memory of the PC are in 32-bit signed-integer type. Prior to transferring to the FPGA, the I/Q components of each data point are packed in the same way as described above. Similarly all processed data points received at the PC are either split into two 16-bit signed integer numbers for storage in a hard drive, or two 32-bit floating-point numbers prior to any further processing such as magnitude calculations for display, or SNR calculations.

3.2 Generation of simulated real-time signals

To demonstrate that the image processing module is capable of processing streaming data, we set up the FPGA board to generate simulated real-time MR signals. On the side of the host computer, the raw data are transferred to FPGA block by block. Each block contains only N_x points. The transfer rate is set to be block/TR, which is controlled by trigger signals from the FPGA. This data transfer process repeats N_y times to complete one $N_x \times N_y$ image. On the FPGA side, unlike for the static data described above, the received data are routed to the analog output channels on the FPGA board to generate analog output signals via on-board DACs. When observed in an oscilloscope, these simulated signals show the same main features as demodulated MR signals. We have designed a simple timing-event-sequence generation unit on the FPGA to control all events in such a way that simulated real-time baseband signals are created with the same timing as a typical spin-echo signal observed in an MRI experiment. After the analog output connectors are hardwired to the analog input connectors on the FPGA board, the analog signals are sampled with 16-bit ADCs at rate of 100KS/s to generate streaming data. The streaming data are then passed through the stages of the pre-processing portion. Because the sampling rate represents 10 times oversampling, at the decimation stage the digital signals are down-sampled by a factor of 10 so the effective sampling rate matches the bandwidth at which the original signals were acquired. All the setup and experimental procedures described here enable us to generate simulated baseband MR signals and to demonstrate the module capabilities of performing real-time image processing under timing conditions of MRI experiments.

3.3 FPGA program compilation

Due to long compilation time, the functions of an FPGA program are intensively tested prior to program compilation by running the simulated FPGA on the PC. After the design is finalized, the FPGA program is compiled in the LabView FPGA module package (National Instruments, Inc., Austin, TX USA.). At the end of the compilation process, a compilation

report and a binary file, or called bitfile, are generated. The bitfile is used to configure the logic circuits on the FPGA chip to ensure correct functionalities. The report reveals how hardware resources are utilized and timing constraints are satisfied, which aids in optimizing the design. For the design of the whole image processing module shown in Fig. 1, compilation was completed in approximately 33 minutes. The final resource utilization for FPGA chip XC6SLX45 is shown in Table 3.

4 Testing and results

As proof-of-concept of the design, we tested the 2D image processing module by reconstructing multi-slice MR images from both simulated real-time signals and static data. Prior to testing the 2D image processing, we tested the 1D-version module by disabling the second FFT processing loop. Real-time single-frequency test signals were created and digitized to generate streaming data as described above. The data flowed through the pre-processing portion and the first FFT loop. Analyses of these known signals and their spectra enable us to ensure the fidelity of the simulated real-time signals, and to make fine adjustments of the FXP numbers (fraction word length versus integer word length) so that the SNR of the simulated signals is close to that of the original data. In a similar manner, we tested the image processing module by applying simulated real-time echo signals that were generated from the raw data of real multi-slice MR images of an animal brain. Fig. 6 shows an example of an echo signal and its FFT (1D profile) at the 64th step of phase encoding.

To test the 2D image processing module, we applied various simulated 2D phase-encoding echo signals that were created from the raw data originally acquired from a phantom and animal brain in previous MRI experiments. Fig. 7A shows a single-slice 128×128 axial image selected from a 4-slice image of a “structured phantom” which consisted of a water-filled plastic container with multiple glass tubes inside. Fig. 7D shows a single-slice 128×128 axial image selected from a 8-slice image of a rabbit brain. The testing procedures were similar for static data. The only difference is that the data flow directly to the 2D FFT core without passing the pre-processing portion. Fig. 7B and E show single-slice axial images of the phantom and animal brain respectively reconstructed from the static data on the FPGA. For comparison, Fig. 7C and F show 2D images reconstructed on the PC from the same raw data sets using a conventional 2D FFT. Overall, the images of the phantom (Fig. 7A-C), or the animal brain (Fig. 7D-F), are nearly identical in terms of image geometry, although some subtle differences in intensity can be seen in Fig. 7A and D, in which the image intensities are slightly decreased toward the two sides of the FOV in readout direction. These images verify that the 2D FFT core works properly. The variation in image intensity in Fig. 7A and D is due to the effect of the decimation function, which can be viewed as a single-stage CIC filter [26]. The frequency response of the filter is characterized by a drooping passband gain and a wide transition region.

We also measured the processing rates of the 2D image processing module. For this purpose, we modified the design shown in Fig. 5 by replacing two FIFO buffers with on-chip block RAMs. The raw data were sent to the FPGA during Initialization prior to 2D FFT processing in order to eliminate the possibility of delays during data transfer between the

FPGA and the host PC. Typically, for a 128×128 image, the processing rate was measured to be 2.5ms, equivalent to 400 frames/second.

5 Discussion

Despite the attractiveness of FPGA technology, it has not yet been widely applied in research MRI systems mainly due to the difficulty of programming FPGAs. FPGAs are usually designed by using high-level Hardware Description Languages (HDLs) such as VHDL and Verilog which currently are standardized. However, directly writing HDL programs is time consuming and requires considerable experience. The difficulties in HDL programming may be ameliorated through Xilinx System Generator (XSG) with a high-level graphical interface [9], or graphical coding on the Labview platform. Graphical FPGA coding in LabView essentially provides a translation layer so that designers can completely avoid HDL coding. As a result, development time can be cut dramatically. In this work, we have demonstrated that graphical coding enables us to explicitly route individual digital signals. One of the advantages of using graphical coding approach is that designers can view more intuitively the timing relations of data flowing between processing tasks, which is often lacking in most previously reported designs [12,13]. There are some disadvantages to using graphical coding alone, however. When a design is very complex, it may become a challenge to manage a large number of signal routings. Another disadvantage is that currently LabView source codes are portable within the LabView platform, but not across different development platforms.

Beyond design optimization of design, the processing speed is primarily determined by two factors in the hardware: clock rate and capacity. Specifically, higher clock rates enable an FPGA to complete each computation in a shorter time; and higher capacities allow more parallel processing circuits per unit area within the FPGA chip. Compared to the FPGA chip used for our trial, FPGAs with much higher clock rates and capacities are now available. With advances in technology, the clock rates and capacities of the newer FPGA chips are several times higher than those of our chip [9]. Therefore it is possible to significantly increase image processing speed with newer FPGA chips. In addition, we have restricted our design to process a single 128×128 image in a single pass of the 2D image processing module. The current design can be easily scaled up for a higher-pixel-resolution image using higher-capacity FPGAs. Future work will expand the current module to 3D image processing, and explore parallel processing to further improve the processing speeds. In addition, it is also possible to integrate the module with an FPGA-based digital receiver for MRI.

Acknowledgments

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

References

1. McDougall MP, Wright SM. 64-channel array coil for single echo acquisition magnetic resonance imaging. *Magn Reson Med*. 2005; 54:386–392. [PubMed: 16032696]

2. Wiggins GC, Polimeni JR, Pollhast A, Schmitt M, Alagappan V, Wald LL. 96-channel receiver-only head coil for 3Tesla: design optimization and evaluation. *Magn Reson Med*. 2009; 62:754–762. [PubMed: 19623621]
3. Schmitt M, Pollhast A, Sosnovik DE, Polimeni JR, Wiggins GC, Triantafyllou C, Wald LL. A 128-channel receiver-only cardiac coil for highly accelerated cardiac MRI at 3 Tesla. *Magn Reson Med*. 2008; 59:1431–1439. [PubMed: 18506789]
4. Stone, SS.; Haldar, JP.; Tsao, SC.; Hwu, WW.; Liang, Z.; Sutton, BP. Accelerating advanced MRI reconstruction on GPUs. *Proceedings of the 5th International Conference on Computing Frontiers*; May 5–7; 2008.
5. Stone S, Haldar J, Tsao S, Hwu W, Sutton B, Liang Z. Accelerating advanced MRI reconstructions on GPUs. *J Parallel Distrib Cmpu*. 2008; 68:1307–1318.
6. Shen J, Xu Q, Liu Y, et al. Home-built magnetic resonance imaging system (0.3T) with a complete digital spectrometer. *Rev Sci Instrum*. 2005; 76:105101.
7. Ning R, Yang G, Li G. A versatile pulse programmer for magnetic resonance imaging. *Rev Sci Instrum*. 2011; 82:054301. [PubMed: 21639523]
8. Xiao L, Wang W. A radio-frequency source using direct digital synthesis and field programmable gate array for nuclear magnetic resonance. *Rev Sci Instrum*. 2009; 80:124703. [PubMed: 20059160]
9. <http://www.xilinx.com>
10. Xiao L, Sun B, Ma Y, Zhao R. A digital magnetic resonance imaging spectrometer using digital signal processor and field programmable gate array. *Rev Sci Instrum*. 2013; 84:054702. [PubMed: 23742570]
11. Tang W, Wang W. A single-board NMR spectrometer based on a software defined radio architecture. *Meas Sci Technol*. 2011; 22:015902.
12. Dalal, IL.; Kirpalani, AL.; Fontaine, FL. A low-cost scalable multichannel digital receiver for magnetic resonance imaging. *Proceedings of the 28th IEEE EMBS Annual International Conference*; New York City, USA. 2006.
13. Dalal, IL.; Fontaine, FL. A Reconfigurable FPGA-based 16-Channel Front-End for MRI, at the 40th Asilomar Conference on Signals, Systems and Computers; Pacific Grove, CA, USA. 2007.
14. Al-allaf AF, Dawwd SA. FPGA based reconfigurable 2D FFT system. *Al-Rafidain Engineering*. 2011; 19(3):76–88.
15. Takeda K. A highly integrated FPGA-based nuclear magnetic resonance spectrometer. *Rev Sci Instrum*. 2007; 78:0233103.
16. Takeda K. OPENCORE NMR: Open-source core modules for implementing an integrated FPGA-based NMR spectrometer. *Journal of Magnetic Resonance*. 2008; 192:218. [PubMed: 18374613]
17. Pikuski M, Shiroka T, Ott HR, Mesot J. A firmware-defined digital direct-sampling NMR spectrometer for condensed matter physics. *Rev Sci Instrum*. 2014; 85(9):093906. [PubMed: 25273738]
18. Liang X, Tang X, Tang W, Gao JH. A high-field magnetic resonance imaging spectrometer using an oven-controlled crystal oscillator as the local oscillator of its radio frequency transceiver. *Rev Sci Instrum*. 2014; 85(9):094705. [PubMed: 25273752]
19. Tang W, Sun H, Wang W. A digital receiver module with direct data acquisition for magnetic resonance imaging systems. *Rev Sci Instrum*. 2012; 83(10):104701. [PubMed: 23126784]
20. Christe SA, Vignesh M, Kandaswamy. An efficient FPGA implementation of MRI image filtering and tumor characterization using Xilinx system generator. *International J of VLSI & Communication Systems (VLSICS)*. 2011; 2(4):95–109.
21. Hasan S, Boussaka S, Yakovle A. FPGA-based architecture for a generalized parallel 2-D MRI filtering algorithm. *American J of Engineering and Applied Sciences*. 2012; 5(1):25–34.
22. Kjetil, E. Vistnes and Oddvar Søråsen, Reconfigurable address generators for stream-based computation implemented on FPGAs. *Parallel and Distributed Processing Symposium*; Proceedings. 19th IEEE International; 2005.
23. Galuzza C, Gou C, Calderon H, Gaydadjiev GN, Vassiliadis S. High-bandwidth address generator unit. *J Sign Process Syst*. 2009; 57(1):33–44.

24. Hendrix, HD. Implementing circular buffers with bit-reversed addressing, Texas Instruments application report: SPRA292. 1997.
25. Kaushik SD, Huang CH, Johnson JR, Johnson RW, Sadayappan P. Efficient transposition algorithms for large matrices. Proceedings of Supercomputing '93. 1993:656–665.
26. Hogenauer EB. An economical class of digital filters for decimation and interpolation. IEEE Transactions on Acoustics, Speech and Signal Processing ASSP-29. 1981:155–162.

Author Manuscript

Author Manuscript

Author Manuscript

Author Manuscript

Highlights

- Image processing module designed by using graphical coding;
- Design implemented on an FPGA;
- Module capable of reconstructing 2D MR images at high speed;
- Module ready for integration with FPGA-based digital receivers for MRI.

Figure 1.

Architecture of an image processing module inside an FPGA chip. The module can process either real-time streaming data or existing data stored in the main memory of a host computer. Real-time (RT) samples from the RF front end undergo pre-processing, which consists of baseline correction (BC), low-pass filtering (Filter) and digital decimation (Deci), followed by image reconstruction by 2D FFT.

Figure 2.

Hardware resources required for execution of a complete 2D FFT on an FPGA.

Figure 3.

Comparison of 2D FFT performed on 1D and 2D arrays. Processing of image data in a typical 2D array is shown in (Top). For processing the data held in a 1D array (Bottom), the first FFT runs on one block of N_x -point data at a time until N_y blocks have been used. The second FFT runs on N_y -point blocks indicated by blue arrows. All points with the same color correspond to one N_y -point block.

Figure 4.
LabView code and interface for Address Generation Unit.

Figure 5.

LabView code for performing 2D FFT. The algorithm computations are carried out in two SCTLs. The first 1D FFT is carried out in the top loop; while the second 1D FFT is carried out in the bottom loop. All operations within a loop are completed in one clock cycle, equivalent to 16.67ns at clock rate of 60MHz.

Figure 6.

An example of a simulated real-time signal generated from multi-slice MR images of an animal brain. The echo signal (top) and its FFT (bottom) are shown. The FFT represents a 1D profile at the 64th step of phase encoding.

Figure 7.
Examples of images reconstructed on the FPGA chip from simulated streaming data (A, D), the static data (B, E), and on the PC from the same raw data (C, F).

Table 1

Control settings of two modes of AGU

Mode	Samples	D	Enable
0	any	1	False
1	$N_x \times N_y$	N_x	True

Table 2

Demonstration of output numbers generated by AGU with two settings

setting	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	
0	0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	1	2	3
1	0	0	4	8	12	1	5	9	13	2	6	10	14	3	7	11	15	0	4	8	12

Remarks:

- Setting 0: Mode 0, Samples=16, Offset = 0;
- Setting 1: Mode 1, Samples=16, D=4, Offset=0.

Table 3

FPGA hardware resource utilization

Resource	Used	Available	Percent
Total Slices	1943	6822	28.5
Slice Registers	3046	54576	5.6
Slice LUTs	4869	27288	17.8
DSP48s	34	58	58.6
Block RAMs	51	116	44.0