Fast Fourier Transform IP Core v1.0
Block Floating-Point Streaming Radix-4 Architecture

IPC0006 | October 2014 **Data Sheet**

Introduction

The FFT/IFFT IP core is a highly configurable Fast Fourier Transform (FFT) and Inverse Fast Fourier Transform (IFFT) VHDL IP component. The core performs an N-point complex forward or inverse Discrete Fourier Transform (DFT), where N is any power-of-four value, up to 65,536. The constituent radix-4 processing stages utilize a decimation-in-frequency (DIF) decomposition and internal structure.

The FFT/IFFT core accepts a length N frame of complex data samples, represented as a pair of fixed-point two's complement numbers. The streaming architecture supports continuous loading of data into the core. After an initial transform latency, data can be continually unloaded from the core to support high throughput applications. The number of bits used to represent input data samples and twiddle factors are independently configurable at build-time. For this architecture, a block floating-point data path representation is used to account for bit growth at the output of each processing stage and to maintain a high degree of precision. The input frame of data samples is presented to the core in natural order. The output frame of data from the core in the transform-domain can be configured for natural or bit-reversed ordering.

The source code for the FFT/IFFT core was developed in a portable, vendor-agnostic manner. The underlying components of the core were developed to infer and take advantage of hardware features found in many FPGAs from various vendors, including Block RAM and hardware multipliers/DSP blocks. The degree of inference of these platform-specific features is controlled via vendor-specific tool settings during the synthesis process.

Additional architectures are also available from GIRD Systems providing other options for transform latency, core throughput, resource utilization, and arithmetic scaling.

Features

- Complex FFT/IFFT operation, run-time configurable on a per-frame basis
- Configurable transform sizes: $N = 4^q$, maximum 65,536; $q = 8$
- Configurable data sample precision
- Configurable twiddle factor precision
- Configurable bit-reversed or natural output ordering
- Fixed-point data interface
- Block floating-point data path for high precision
- Supports streaming processing (continuous data feed)
- Utilizes FPGA architecture features (multiplier/DSP blocks, Block RAM, etc.) via inference
- Bit-accurate MATLAB model and testbench
- VHDL testbench

Interface Description

The component interface is shown in Figure 1.

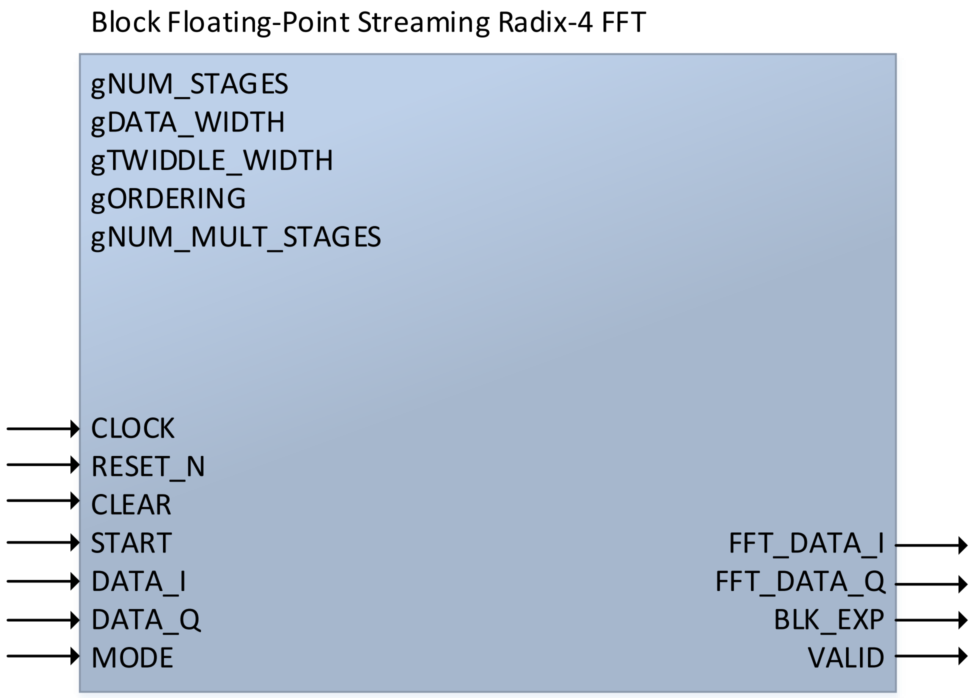

Figure 1: FFT/IFFT Core Top-Level Interface

Generics

The generic values can be modified at compile time to configure the FFT/IFFT core for the targeted application. Table 1 lists the generics and their purpose.

Table 1: FFT/IFFT Core Generics

Generic Name	Type	Description
gNUM_STAGES	natural	The number of stages, q, needed to generate an N length power-of-four transform in the form $N = 4^q$.
gDATA_WIDTH	natural	Bit width of input/output data.
gTWIDDLE_WIDTH	natural	Bit width precision of twiddle factors.
gORDERING	std_logic	Specifies output data ordering: '0': bit-reversed ordering '1': natural ordering
gNUM_MULT_STAGES	natural	Number of pipeline stages used in multiplier operations.

Inputs

The signal inputs to the FFT/IFFT core are defined in Table 2.

Table 2: FFT/IFFT Core Input Signals

Input Name	Type	Description
CLOCK	std_logic	Core processing clock.
RESET_N[1]	std_logic	Active low, asynchronous reset.
CLEAR[1]	std_logic	Active high, synchronous reset.
START	std_logic	Specifies the first sample in an input data frame. $N-1$ input samples must follow this pulse on consecutive clock cycles.
DATA_I	signed(gDATA_WIDTH-1:0)	Real (in-phase) input data samples. The bit width is specified by gDATA_WIDTH.
DATA_Q	signed(gDATA_WIDTH-1:0)	Imaginary (quadrature) input data samples. The bit width is specified by gDATA_WIDTH.
MODE	std_logic	Specifies the mode of operation: '0': forward FFT '1': inverse FFT This value is accepted when START is asserted.

Outputs

The signal outputs from the FFT/IFFT core are defined in Table 3.

Table 3: FFT/IFFT Core Output Signals

Output Name	Type	Description
FFT_DATA_I	signed(gDATA_WIDTH-1:0)	Real (in-phase) output data samples in the transform domain. The bit width is specified by gDATA_WIDTH.
FFT_DATA_Q	signed(gDATA_WIDTH-1:0)	Imaginary (quadrature) output data samples in the transform domain. The bit width is specified by gDATA_WIDTH.
BLK_EXP	natural	Indicates the block exponent. The output data from each radix-4 stage can grow by up to 3 bits; BLK_EXP tracks the amount of bit shift scaling that was applied to prevent overflow. For example, a value of 7 indicates samples were shifted right by 7 bits, thus the magnitude of the output is: $FFT\_DATA \times 2^{BLK\_EXP}$.
VALID	std_logic	Indicates valid transformed samples are being output from the core.

---

[1] Both a synchronous and asynchronous reset are provided for portability. Only one of the reset circuits should be used depending on the target platform. The unused port should be tied to a logical constant ('1' for RESET_N, '0' for CLEAR).

Timing Diagram

An overall sample timing diagram for a 1,024-point FFT is shown in Figure 2. In this example, 3 consecutive frames of input data are streamed into the core on the DATA_I and DATA_Q ports. As noted previously, the two reset circuits are mutually exclusive, thus in this example the RESET_N signal is tied to logic-‘1’ and the CLEAR signal is pulsed high at the start of the test.

The START signal is pulsed high for a single clock cycle at the start of each frame of input data, coincident with the first sample. The next $N - 1$ consecutive input samples following a START pulse are considered part of the same frame. Figure 3 shows a magnified view of the start of frame #2 and also shows a MODE change coincident with the START pulse for frame #2 to perform a forward FFT.

After an initial transform calculation latency, data is streamed out on the FFT_DATA_I and FFT_DATA_Q ports and is indicated to be valid by the VALID port. The BLK_EXP output port indicates the total amount of scaling applied during the FFT operation. Figure 2 shows the 3 frames of output data being consecutively streamed out; for the 1,024-point example, the VALID signal is high for 3,072 clock cycles over all 3 frames of output data. In the example given in Figure 2, all 3 output frames have a block exponent of 7. Figure 4 shows a magnified view of the start of output frame #1.

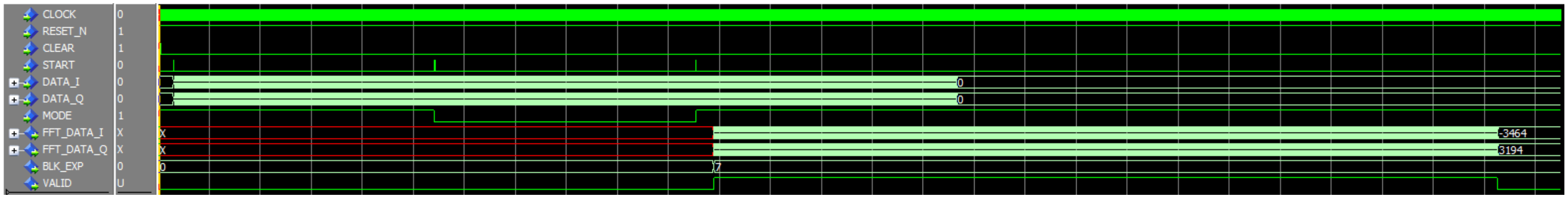

Figure 2: Overall Timing Diagram

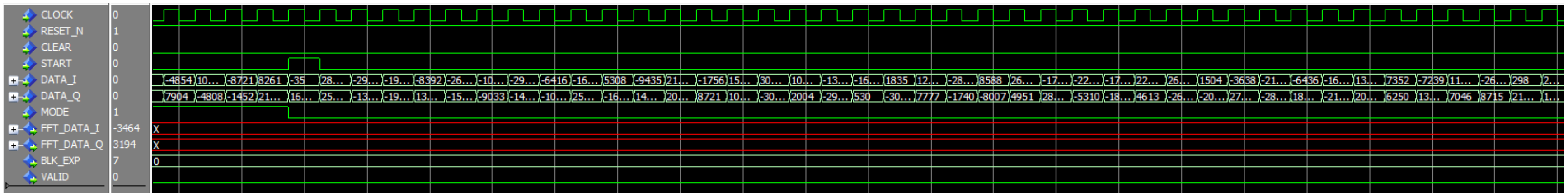

Figure 3: Start of Input Frame #2 Timing Diagram

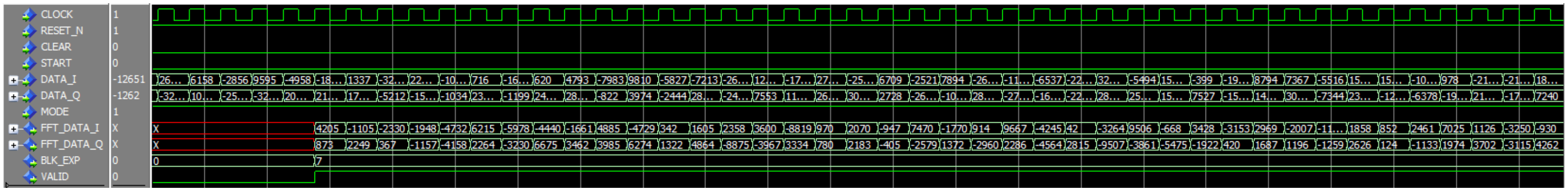

Figure 4: Start of Output Frame #1 Timing Diagram

File List

Table 4 lists the VHDL source files included with the project. Source files are also included for both block floating-point and floating-point MATLAB models as listed in Table 5. Additionally, testbenches for VHDL and block floating-point and floating-point MATLAB models are included as listed in Table 6 and Table 7.

Table 4: FFT/IFFT Core VHDL Source File List

File Name	Description
radix4_bfly.vhd	Radix-4 butterfly operations
radix4_dft.vhd	FFT signal processing top level
radix4_stage.vhd	Radix-4 stage processing and control
radix4_stage0.vhd	Radix-4 processing and control for the first stage
radix4_dft_pkg.vhd	Package file for useful types and functions
clamp_pkg.vhd	Package file for saturation functions
complex_multiplier_growth.vhd	Complex multiplier with data path growth support
dp_ram.vhd	Dual port RAM
math_pkg.vhd	Package file for math convenience functions
multiplier_s.vhd	Signed multiplier
shift_pkg.vhd	Package file for bit shifting functions
shift_ram.vhd	RAM-based shift register
types_pkg.vhd	Package file for commonly used user-defined types

Table 5 : FFT/IFFT Core MATLAB Model Source File List

File Name	Description
radix4_bfly.m	Radix-4 butterfly operations
radix4_dft.m	FFT signal processing top level
radix4_stage.m	Radix-4 stage processing and control
clamp.m	Saturates input data
shift_down.m	Performs bit shifting operations on input data

Table 6: FFT/IFFT Core VHDL Testbench File List

File Name	Description
radix4_dft_tb.vhd	Testbench associated with top level VHDL source
run_tb.do	Modelsim-compatible script to compile source and testbench
testbench_pkg.vhd	Package file for testbench convenience functions
textio_pkg.vhd	Package file for text input/output; overloads/extends std.textio

Table 7: FFT/IFFT Core MATLAB Testbench File List

File Name	Description
radix4_dft_tb.m	Testbench associated with top level MATLAB model

Functional Description

The Discrete Fourier Transform, of length N, takes N complex time domain samples and expresses them in the frequency domain. The equation for the forward DFT is given as:

$$X[k] = \sum_{n=0}^{N-1} x_n \cdot e^{-j2\pi kn/N}$$

The inverse DFT reverses this process, transforming frequency domain signals into the time domain. The equation for the inverse DFT is given as:

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k \cdot e^{j2\pi kn/N}$$

The implementation of this core uses a modified Sande-Tukey decimation-in-frequency (DIF) algorithm to reduce computational time and complexity over a brute force DFT implementation. The core supports power-of-four transform lengths formed from radix-4 stages up to 65,536 points. A high-level block diagram of the FFT/IFFT core architecture is shown in Figure 5.

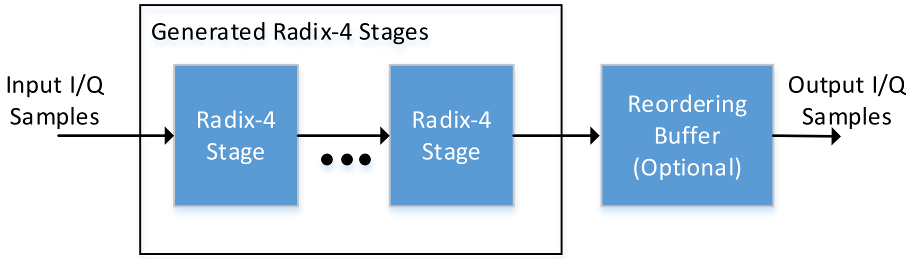

Figure 5 : Top Level Radix-4 FFT Block Diagram

The inverse operation is supported by swapping the real and imaginary components of the input samples, performing the forward FFT, then swapping the real and imaginary components of the output samples. Note that this core differs slightly from the classical IFFT definition in that it does not perform the $\frac{1}{N}$ scaling as part of the internal implementation. The final reordering buffer shown in Figure 5 is optional and is used to reorder the bit-reversed output samples to natural ordering.

For illustrative purposes, assume $N = 1{,}024$. Referring to Figure 5, 5 radix-4 stages would automatically be generated to perform the 1,024 length transform. Each stage produces N intermediate samples, which are forwarded to the next stage. Finally, an optional reordering buffer changes the output ordering into natural order.

The key to the FFT processing in this architecture is the radix-4 stage and butterfly engine as shown in Figure 6. This implementation uses a quad output radix-4 engine to minimize processing latency. Data from the previous stage is reordered via reordering buffers (dual-port RAMs). As data from the previous stage is being written to the reordering buffers, the bit growth detection block calculates the amount of bit growth caused by butterfly/twiddle operations in the previous stage. Bit growth is calculated based on the maximum/minimum value generated in an output data set from a given stage. Data read out of the reordering buffer is scaled down based on the calculated bit growth to prevent overflow from bit growth in previous stages and maximize precision.

Scaled data is then processed by the quad output radix-4 butterfly engine. The bit width of the data path through the quad output radix-4 butterfly engine and the reordering buffers grows by up to 3 bits (i.e., a total bit width of gDATA_WIDTH + 3) to prevent overflow for the current stage. Twiddle factors are applied via a complex multiplier. The wider, processed samples are forwarded on to the next stage.

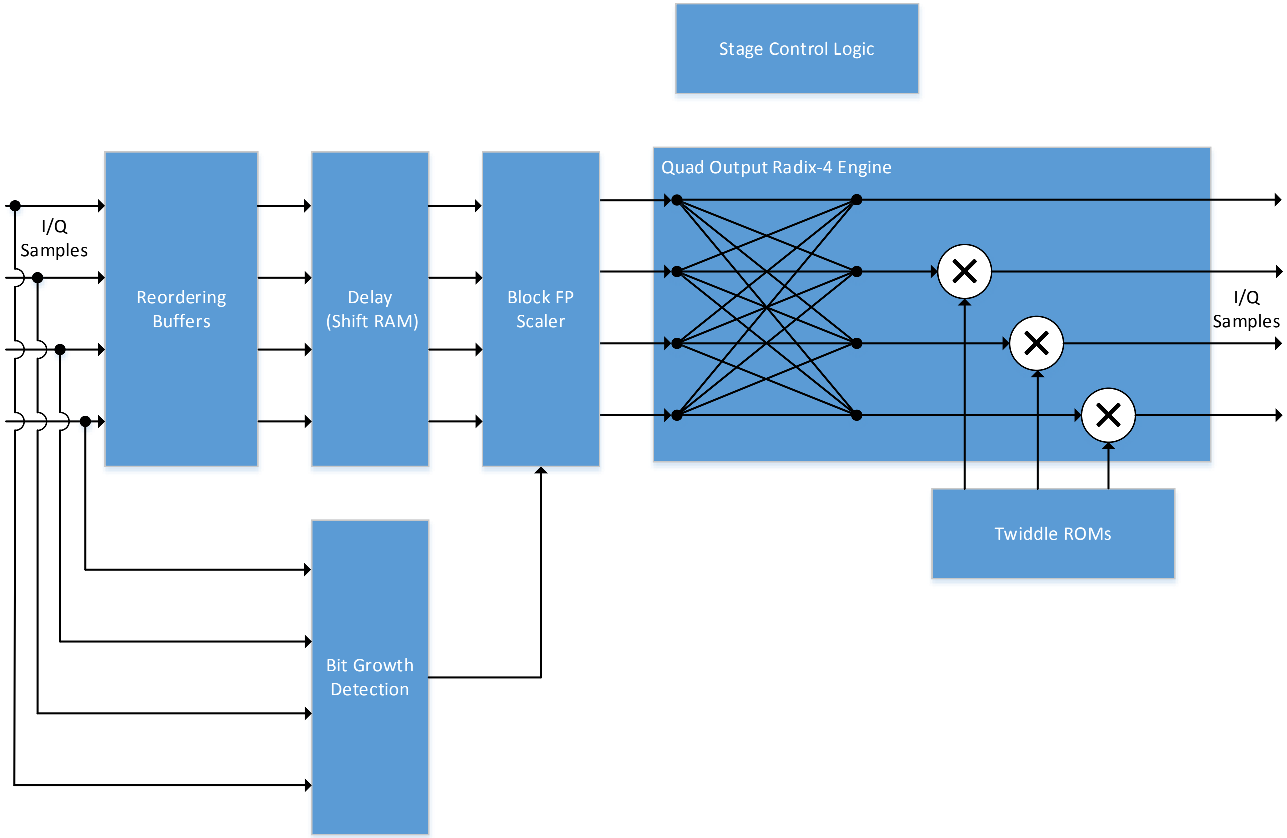

Figure 6: Radix-4 Stage Block Diagram

Resource Utilization

To generate resource utilization estimates, build synthesis and implementation was performed using generic inputs shown in Table 8, producing a 1,024-point FFT/IFFT with 16-bit I/Q data.

Table 8: Generic Values Used for Resource Utilization

Generic	Type	Value
gNUM_STAGES	natural	5
gDATA_WIDTH	natural	16
gTWIDDLE_WIDTH	natural	16
gORDERING	std_logic	'1'
gNUM_MULT_STAGES	natural	3

Resource estimates for several candidate Xilinx devices are shown in Table 9. Resource estimates for several candidate Altera devices are shown in Table 10. The asynchronous reset circuit for all Altera estimates and the synchronous reset circuit was used for all Xilinx estimates. For Xilinx devices, Vivado 2014.1 was used to generate resource estimates. For Altera devices, Quartus II 11.1 SP2 was used in estimate generation. Both vendors include several synthesis and implementation options that will impact resource utilization and maximum achievable circuit frequency (Fmax). The estimates reported in this document use the default values provided by each vendor for all synthesis and implementation options. The user can adjust vendor-specific settings to impact the performance/resource utilization tradeoff for their application. Contact GIRD Systems for assistance in tuning build parameters and constraints for specific application needs. Note that several other factors may cause variation in resource utilization and circuit performance estimates, such as overall device utilization, routing congestion, place-and-route/fitter seed, etc.

Table 9: Xilinx Device Utilization Estimates

Device	Logic		Block RAMs		Multipliers	Fmax
	Slice LUTs	Slice Registers	RAMB36s	RAMB18s	DSP48E1s	MHz
Artix 7 XC7A200T -1	8,224	12,994	35	20	60	167.48 MHz
Kintex 7 XC7K325T -1	9,401	13,680	29	6	60	268.53 MHz
Virtex 7 XC7VX485T -1	9,402	13,680	29	6	60	253.61 MHz

Table 10: Altera Device Utilization Estimates

Device	Logic		Block RAMs			Multipliers	Fmax
	LEs/ALUTs	Registers	M9Ks	M144Ks	M20Ks	DSP 9x9 / Blocks	MHz
Cyclone III EP3C40 C8	9,555	12,507	92	-	-	96	104.54 MHz
Cyclone IV EP4CGX30 C8	9,581	12,507	92	-	-	96	103.46 MHz
Stratix IV EP4SE230 C4	7,904	11,640	81	0	-	96	244.74 MHz
Stratix V 5SGSED6K C4	7,079	12,422	-	-	70	24	238.04 MHz

Simulation

Most VHDL simulators should be capable of simulating the FFT/IFFT IP core. During development and testing of the core, ModelSim DE 10.1d and MATLAB 2009a were used for simulation. No additional external libraries or toolboxes are required. There are 3 provided testbenches: a block floating-point VHDL testbench, a bit-accurate block floating-point MATLAB testbench, and a floating-point MATLAB testbench. The VHDL testbench depends on stimulus data generated by the MATLAB block floating-point testbench. Thus, the MATLAB block floating-point testbench must be run prior to running the VHDL testbench.

For the block floating-point MATLAB testbench, several simulation and core parameters must be set to configure the test. The ‘gen_vectors’ flag configures if new stimulus data should be generated or existing data should be used. The ‘num_frames’ variable defines the number of frames to process in the current test. Note that if ‘num_frames’ is modified, new stimulus data should be generated so that the correct number of input samples are available. Each entry in the ‘modes’ vector defines the forward/reverse operation for each frame. The length of the ‘modes’ vector should be equal to ‘num_frames’. The core-specific parameters, like transform length, data width, schedule, etc., are defined in the ‘params’ structure.

In the VHDL testbench, there are several simulation and core configuration parameters that have corresponding parameters in the MATLAB block floating-point testbench. To achieve a bit-true comparison with the MATLAB model, these parameters must have the same values as the corresponding MATLAB testbench parameters. Table 11 lists the relevant MATLAB and corresponding VHDL simulation parameters. A ModelSim-compatible DO script is also included to compile the VHDL source and testbench.

Table 11 : Simulation Parameters

MATLAB Parameter	VHDL Parameter	Description
num_frames	cFRAMES	Number of frames to test
params.data_width	cDATA_WIDTH	Bit width of the data samples
params.twiddle_width	cTWIDDLE_WIDTH	Bit width of twiddle factors
params.num_stages	cNUM_STAGES	Number of transform stages
modes	cMODE	Forward or inverse operation
params.ordering	cORDERING	Bit-reversed or natural output ordering