CoreFFT Fast Fourier Transform

DirectCore

Product Summary

Intended Use

- Fast Fourier Transform (FFT) Function for Actel FPGAs

Key Features

- Forward and Inverse 32-, 64-, 128-, 256-, 512-, 1,024-, and 2,048-Point Complex FFT
- Decimation–In-Time (DIT) Radix-2 Implementation Optimized for Actel FPGAs
- Selection of Unconditional or Conditional Block Floating-Point Scaling
- Embedded RAM-Block-Based Twiddle Factor Generator
- 8- to 16-Bit Configurable Input/Output Data and Twiddle Coefficients Precision
- Naturally Ordered Input and Output Data
- Two's-Complement Fixed-Point Arithmetic
- Built-In Memory Buffers
- CoreFFT Provides Register Transfer Level (RTL) Code and a Behavioral Testbench

Targeted Devices

- ProASIC®3/E
- ProASIC PLUS®
- Axcelerator®
- RTAX-S

Core Deliverables

- Full Version
  - CoreFFT RTL Generator; Generates User-Defined FFT Model and Test Harness; Fully Supported in Actel Libero® Integrated Design Environment (IDE)
- Evaluation Version
  - Supports FFT Engine and Test Harness Generation with Limited Parameters; Fully Supported in Actel Libero IDE

Synthesis and Simulation Support

- Actel Libero IDE
- Synthesis: Synplicity,® Synopsys® (Design Compiler / FPGA Compiler), Exemplar™
- Simulation: OVI-Compliant Verilog Simulators and Vital-Compliant VHDL Simulators

Contents

General Description

CoreFFT is an RTL generator that produces an Actel FPGA–optimized FFT engine. The resulting module computes 32-, 64-, 128-, 256-, 512-, 1,024-, or 2,048-point complex forward or inverse decimation-in-time (DIT) FFTs. The input and output data is represented as *bb*-bit words comprising *b*-bit real and imaginary parts (*bb* = *b* + *b*; *b* = 8 to 16 bits). Both the real and imaginary parts of the input and output data are two's-complement numbers. The FFT module contains all the necessary memory buffers and butterfly and control logic, as well as a twiddle factor generator. A dual input buffer and a single output buffer support simultaneous input of the new data samples with FFT computation and result output. The module processes frames (bursts) of data with a frame size equal to the transform size of N words. The FFT computational process occurs in a

© 2007 Actel Corporation

Downloaded from Arrow.com.

sequence of stages with the final result obtained at the last computational stage. As soon as a final output vector is ready, the FFT module puts out an N-word frame of FFT results.

An FFT-based system (Figure 1) consists of the following:

- A host presenting data to the FFT module to be processed
- The FFT module
- A host accepting processed data

Note: Signals shown in parentheses are optional. The host may use/generate these optional signals if required by the application.

A negative ***nreset*** signal resets the FFT module. After reset (input ***nreset*** taken HIGH), the module enters an initialization state where internal RAM-based lookup tables (LUTs) are initialized. Once initialization is complete, the CoreFFT module automatically switches to a ready state, prepared to receive data samples to be processed. The module input ***start*** can be used to bring the module to the ready state at any time after initialization.

Note: The CoreFFT module will discard data collected in its input and output buffers when ***start*** is taken HIGH by the host.

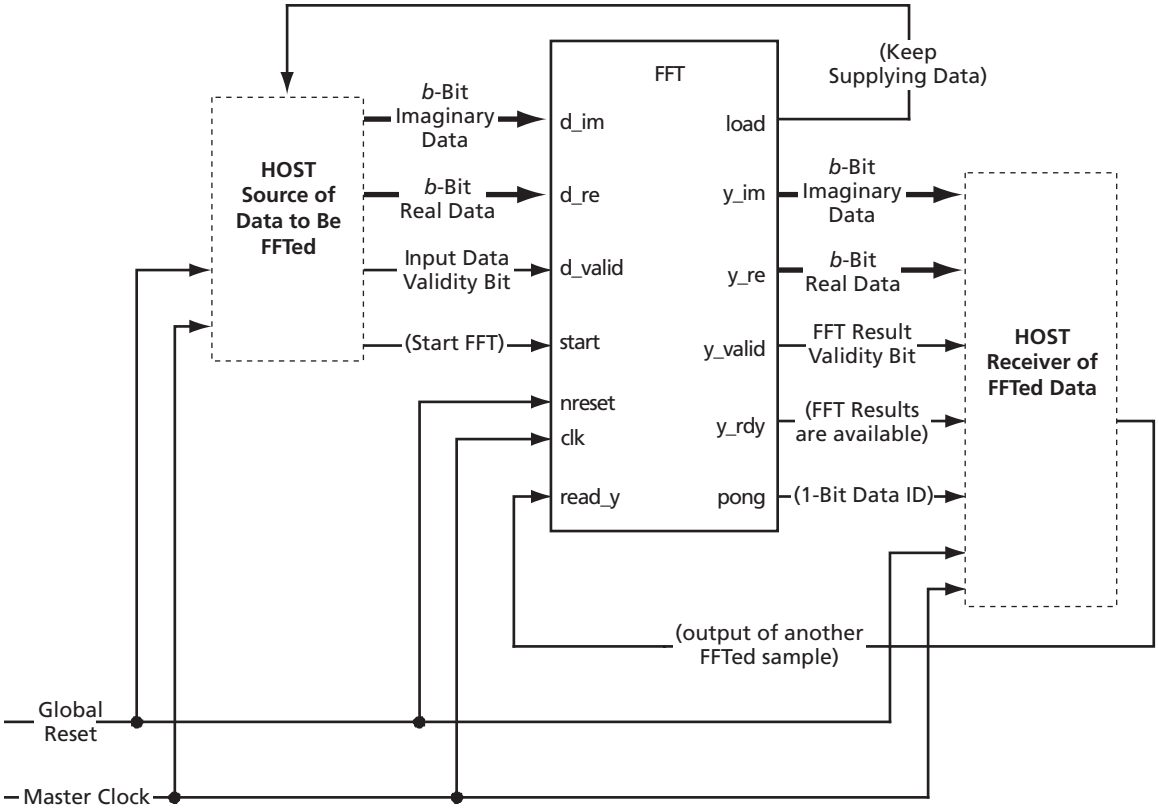

Figure 1 • **FFT Module System Block Diagram**

The data-source host supplies the FFT engine with the data to be transformed. Every complex input data sample (i.e., a pair of *b*-bit imaginary and real words) is accompanied with a validity bit. Upon receiving the validity bit, the module assumes a valid complex data sample is present on both *b*-bit input data busses.

Once the input data buffer is full, CoreFFT automatically starts processing data stored in the buffer. At this time, the host source should stop supplying the data to CoreFFT, thus ending a current burst of input data. The data-source host can do so either by counting the number of input samples transferred or by monitoring the state of the module signal, ***load***. The CoreFFT module drives ***load*** LOW once N samples (N is a transform size) of the current burst are received into the input buffer. As soon as the dual input buffer is ready for the next data burst, ***load*** is asserted again. The module is then ready to receive the next data burst to be processed.

The data-source host can supply data at a maximum of every clock cycle, or it may skip an arbitrary number of "empty" clock periods. The host signals to the module that no data is being transferred for a given clock cycle by taking the validity bit ***d_valid*** LOW.

Downloaded from Arrow.com.

Once the CoreFFT module has computed the FFT, the results are written to the module's output memory buffer, and signal ***y_rdy*** is taken HIGH. Every output sample is accompanied by a validity bit, ***y_valid***, to indicate to the receiving host that a valid output sample is ready to be read from both *b*-bit output busses. The receiving host can control the output sample rate using the ***read_y*** input of the module. Asserting ***read_y*** indicates to the FFT module that the receiving host is ready to read samples. Deasserting ***read_y*** informs the module that the host is not ready. Any unread samples are held in the module's output buffer until the host is ready.

CoreFFT Device Requirements

Table 1 and Table 2 on page 4 provide typical utilization and performance data for CoreFFT, which is implemented in various Actel devices with the configurations listed in Table 1 and Table 2 on page 4. Device utilization and performance will vary depending upon the FFT parameters used. The transform size parameter N primarily impacts the number of RAM blocks and the time required for transformation. "FFT Computation" on page 5 provides more details on how the FFT time depends on the transform size.

Table 1 • **CoreFFT Device Utilization and Performance (bit width *b* = 16)**

FPGA Family and Device	FFT Points	Cells or Tiles			Utilization %	RAM Blocks	Device Speed Grade	Clock Rate, MHz	FFT Time, µsec
		Comb.	Seq.	Total					
ProASIC3/E A3P1000	256	5,325	2,039	7,364	29.96%	14	–2	100	11
	512	6,105	2,062	8,167	33.23%	14	–2	92	26
	1,024	6,904	2,126	9,030	36.74%	28	–2	90	58
ProASIC[PLUS] APA1000	256	6,904	2,026	8,930	15.90%	28	Std	59	19
	512	6,901	2,019	8,920	15.80%	28	Std	62	39
	1,024	9,091	2,431	11,522	20.50%	56	Std	62	85
Axcelerator AX1000	256	3,398	2,373	5,771	31.81%	7	–2	130	9
	512	3,601	2,380	5,981	32.96%	14	–2	120	20
	1,024	3,863	2,404	6,267	34.54%	28	–2	105	50
RTAX-S RTAX1000S	256	3,407	2,370	5,777	31.84%	7	–1	107	10
	512	3,596	2,381	5,977	32.94%	14	–1	90	27
	1,024	3,881	2,397	6,278	34.60%	28	–1	76	69

Notes:

1. Auto-scaling (block floating point) is enabled in all cases.
2. The above data were obtained by typical synthesis and place-and-route methods. Other core parameter settings can result in different utilization and performance values.
3. All memory buffers are RAM-block-based.
4. Timing constraints supplied with CoreFFT were used.
5. Timing-driven layout options were used, effort level 3, with no multiple passes.

Downloaded from Arrow.com.

Table 2 • **CoreFFT Device Utilization and Performance (bit width *b* = 8)**

FPGA Family and Device	FFT Points	Cells or Tiles			Utilization %	RAM Blocks	Device Speed Grade	Clock Rate, MHz	FFT Time, µsec
		Comb	Seq	Total					
ProASIC3/E A3P1000	256	2,126	968	3,094	12.6%	7	–2	114	10
	512	2,283	989	3,272	13.3%	7	–2	122	20
	1,024	2,509	1,026	3,535	14.4%	14	–2	118	44
ProASIC^{PLUS} APA1000	256	2,386	949	3,335	5.9%	14	Std	87	13
	512	2,569	974	3,543	6.3%	14	Std	82	29
	1,024	2,759	1,124	3,883	6.9%	28	Std	82	64
Axcelerator AX1000	256	1,317	958	2,275	12.5%	7	–2	170	7
	512	1,435	986	2,421	13.3%	7	–2	159	15
	1,024	1,620	1,016	2,636	14.5%	14	–2	137	38
RTAX-S RTAX1000S	256	1,317	958	2,275	12.5%	7	–1	132	8
	512	1,435	986	2,421	13.3%	7	–1	116	21
	1,024	1,611	1,027	2,638	14.5%	14	–1	101	52

Notes:

1. Auto-scaling (block floating point) is enabled in all cases.
2. The above data were obtained by typical synthesis and place-and-route methods. Other core parameter settings can result in different utilization and performance values.
3. All memory buffers are RAM-block-based.
4. Timing constraints supplied with CoreFFT were used.
5. Timing-driven layout options were used, effort level 3, with no multiple passes.

Architecture

The CoreFFT module input and output data are stored in on-chip RAM blocks. The input memory buffer is also used by the FFT processor as working memory where the FFT engine stores results obtained at any intermediate FFT stage. This dual memory usage is possible due to the in-place FFT algorithm implemented by the core.

The twiddle factors (algorithm coefficients) used by the FFT processor are generated by CoreFFT and stored in a RAM-based LUT.

In addition to the FFT processor, the resulting module also contains control logic and a host interface used for entering data and reading the FFT results. See Figure 2.

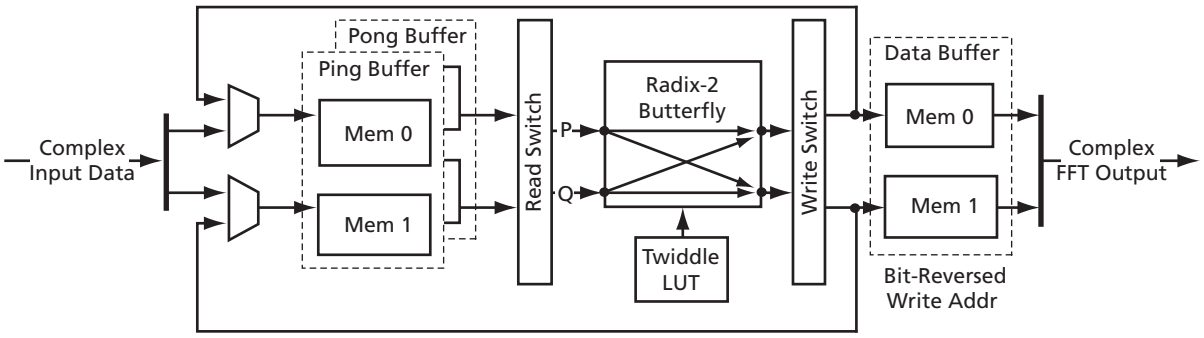

Figure 2 • **CoreFFT Architecture**

Downloaded from Arrow.com.

Buffering Scheme

The CoreFFT module has one radix-2 butterfly, two input memory banks implementing a ping-pong input buffer, and one output buffer (Figure 2 on page 4).

Both of the identical memory banks can store N complex samples. Each bank consists of two memory blocks, each of N / 2 complex words, so it can read/write two complex samples per clock. Thus, the memory bandwidth is (4 × *b*) bits per clock cycle. Internal logic controls the ping-pong switching between the banks so a data-source host only sees the buffer ready to accept new data. The buffer not accepting data is used by the in-place FFT engine as working memory.

This ping-pong buffering architecture increases the efficiency of the FFT engine. While one of the two identical ping-pong input banks is involved in current FFT computation, the other is available for the downloading of the next frame of input data. As a result, the FFT engine does not sit idle waiting for fresh data to fill the input buffer. From the data-source host standpoint, the core is capable of receiving a data burst anywhere within the FFT computational period. When the module has finished processing the current data frame *and* the input buffer bank has been filled with another data frame, the memory ping-pong banks are swapped, and the data load and computation continues on the alternate memory banks.

The last stage of the FFT computation uses an out-of-place scheme—the FFT final results are routed to the output data buffer. The results appear at the FFT engine output in bit-reversed order. A bit-reversed write address is used when writing the results to the output buffer to restore the data to normal read address order. The last stage results remain valid until the FFT engine is ready to store the results of the next data frame.

The CoreFFT generator also calculates the twiddle factors required by the FFT algorithm. At power-up, the twiddle factors generated are written to the twiddle factor lookup table (Twiddle LUT).

FFT Computation

An FFT computational cycle starts when input data is stored in the active ping-pong buffer bank and the FFT engine has finished processing the previous N data samples. Each memory bank comprises two *bb*-bit-wide memories (Mem 0 and Mem 1), supplying a data bandwidth of (4 × *b*) bits per clock cycle (two complex samples per clock cycle). Even input samples D_i (i = 0, 2, 4, …, N –) are stored in Mem 0, odd samples in Mem 1.

The Read Switch function is used to rearrange the two sample pairs read from the input bank to match the input sample order required by the DIT FFT algorithm tree. The switch output samples are then routed to the butterfly P and Q inputs. Table 3 shows a 16-point FFT example of how Read Switch rearranges sample indices coming from the Mem 0 and Mem 1 of the input memory bank.

Table 3 • **Sample Indices before and after Read Switch for 16-Point FFT Example**

Input From		Output	
Mem 0	**Mem 1**	**P**	**Q**
Stage 1			
14	15	7	15
6	7	6	14
12	13	5	13
4	5	4	12
10	11	3	11
2	3	2	10
8	9	1	9
0	1	0	8
Stage 2			
14	15	11	15
10	11	10	14
12	13	9	13
8	9	8	12
6	7	3	7
2	3	2	6
4	5	1	5
0	1	0	4
Stage 3			
14	15	13	15
12	13	12	14
10	11	9	11
8	9	8	10
6	7	5	7
4	5	4	6
2	3	1	3
0	1	0	2
Stage 4			
14	15	14	15
12	13	12	13
10	11	10	11
8	9	8	9
6	7	6	7
4	5	4	5
2	3	2	3
0	1	0	1

Downloaded from Arrow.com.

The radix-2 butterfly processes the data according to the DIT algorithm,[1] one pair of samples per clock. The Write Switch works similarly to the Read Switch, rearranging the butterfly results prior to being written back to the in-place memory bank.

On the last stage of every FFT computational cycle, the results are written into the output memory buffer rather than back to the in-place memory bank. Figure 3 shows the FFT computational sequence.

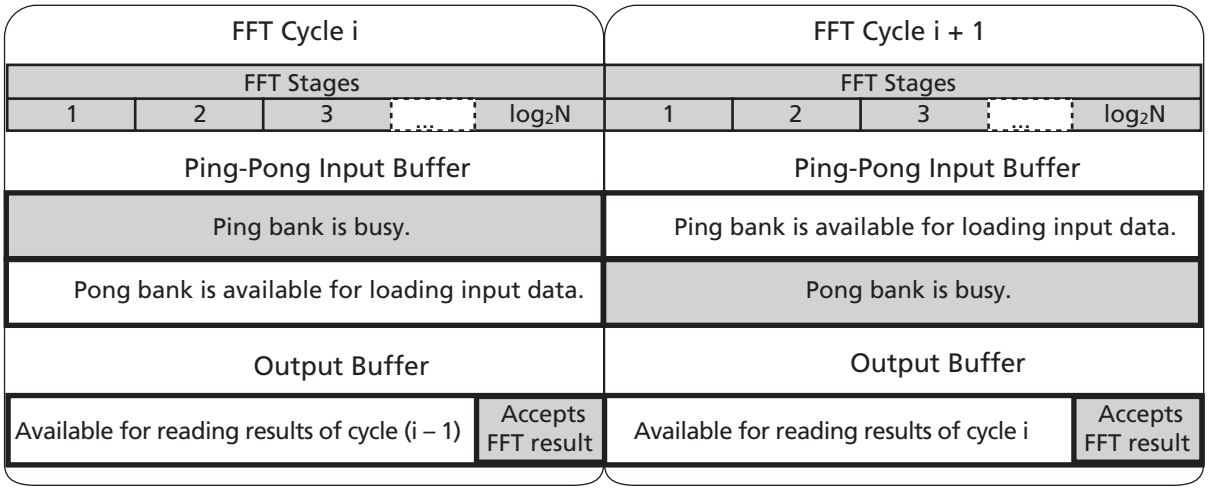

Figure 3 • **FFT Computational Sequence**

Every FFT stage takes

$$(N / 2 + L) \text{ clock cycles} \qquad EQ\ 1$$

to complete, where

- N / 2 = the number of butterflies to be performed within a stage
- L = an implementation-specific parameter representing the aggregate latency of the memory bank, switches, and butterfly. L is much less than the number of butterflies required (N / 2) and does not depend on transform size N.

The full FFT cycle takes

$$(N / 2 + L) \log_2 N \text{ clock cycles.} \qquad EQ\ 2$$

This time is available for the new frame of N data samples to be loaded into the memory bank not involved in the current FFT computation. To provide maximum FFT engine utilization (no idle time, FFT engine full loading), the minimal input sample rate that the host should provide is

$$((N / 2 + L) \log_2 N) / N \approx (\log_2 N) / 2 \text{ clock cycles} \qquad EQ\ 3$$

The host can read the output buffer during the first $\log_2 N - 1$ stages of the next FFT computational cycle (the last stage is used to write fresh FFT results). Therefore, the minimal read sample rate to avoid FFT engine idle time is

$$(N / 2 + L) (\log_2 N - 1) / N \approx (\log_2 N - 1) / 2 \text{ clock cycles} \qquad EQ\ 4$$

As a result, the minimal input and output sample rates required to avoid FFT engine idle time depend on the transform size N (Table 4).

Table 4 • **Minimal Input and Output Sample Rates**

Transform Size N, Points	Input Sample Rate, Clock Cycles	Output Sample Rate, Clock Cycles
256	4	3
512	4	4
1,024	5	4

Finite Word Length Considerations

The butterfly calculation involves complex multiplication, addition, and subtraction. These operations can potentially cause the butterfly data width to grow by two bits from input to output.[1, 2] At every stage of the in-place FFT algorithm, the butterfly takes two samples out of the input buffer and returns two processed samples to the same buffer location. Potentially, returning samples may have a larger data width than the

Downloaded from Arrow.com.

samples picked from the memory. Precautions must be taken to ensure that there are no data overflows.

To avoid risk of overflow, one of three methods can be employed:

- Input data scaling
- Unconditional block floating-point scaling
- Conditional block floating-point scaling

One way to ensure that overflow never occurs is to include enough extra sign bits, called guard bits, in the FFT input data. Data can grow by a maximum factor of 2.4 from butterfly input to output (two bits of growth). However, it is not possible for the data value to grow by this maximum amount in two consecutive stages.

The number of guard bits necessary to compensate for the maximum possible bit growth for an N-point FFT is

$$\log_2 N + 1$$

EQ 5

For example, each of the input samples of a 256-point FFT should contain nine guard bits, leaving only seven bits for actual data. Obviously, the data bit resolution is greatly limited when using the input data scaling technique.

Another way to compensate for bit growth is to scale the butterfly outputs down by a factor of two after each stage. Consequently, the final FFT results are scaled down by a factor of 1 / N. This approach is called unconditional block floating-point scaling. Initially, two guard bits are included in the input data to accommodate the maximum bit growth at the very first stage. In each successive butterfly calculation, the data can grow into these guard bits. To prevent overflow in successive stages, the guard bits are replaced before the next stage is executed by shifting the entire block of data (all results of the current stage) one bit to the right. The input data of an unconditional block floating-point FFT can have at most 14 bits (1 sign bit and 13 magnitude bits). EQ 6 shows the total number of bits the data loses because of bit shifting in the FFT calculation.

$$\log_2 N - 1 \text{ bits}$$

EQ 6

Unconditional block floating-point scaling results in the same number of bits lost as in input data scaling. However, it produces more precise results, as the FFT engine starts with more precise input data.

In conditional block floating-point scaling, data is shifted only if bit growth actually occurs. If one or more butterfly outputs grow, the entire block of data is shifted to the right. The conditional block floating-point monitor checks every butterfly output for growth. If shifting is necessary, it is performed after the entire stage is complete (at the input of the next stage butterfly). This technique provides the least amount of distortion (noise) caused by finite word length.

The CoreFFT module is configured to apply conditional block floating-point scaling by default. In this mode, the input data is checked as well and, if necessary, downscaled by a factor of two prior to the first stage.

The user can optionally select one of the other two scaling modes. To apply unconditional block floating-point scaling, the CoreFFT configuration parameter **scale** needs to be set to 1. To apply input data scaling, the **scale** configuration parameter has to take the default value of 0, and the FFT input data has to contain the proper number of guard bits. Then the conditional block floating-point scaling will take no effect.

CoreFFT Generator Parameters

CoreFFT generates RTL code for a few selectable FFT cores that vary depending on parameters set by the user when generating the module. The core generator supports the variations specified in Table 5.

Table 5 • **Core Generator Parameters**

Parameter Name	Description	Recommended Selection
inv	Forward/inverse FFT	**0** (FFT) / **1** (IFFT)
scale	Unconditional block floating-point scaling	**0** (conditional block floating-point) / **1** (unconditional block floating-point)
points	Transform size	**32**, **64**,**128**, **256**, **512**, **1024**, **2048**
bits	FFT engine bit width	**8** to **16**
fpga_family	FPGA family	**ax** (Axcelerator, RTAX-S), **apa** (ProASIC^{PLUS}), **pa3** (ProASIC3)
lang	RTL code language	**vhdl**, **verilog**

Downloaded from Arrow.com.

I/O Signal Description

Figure 4 shows the CoreFFT module pinout.

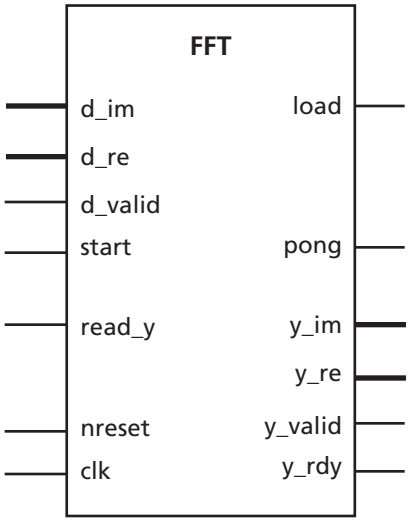

Figure 4 • **CoreFFT I/O Signals**

The CoreFFT module I/O signal functionality is listed in Table 6. It is assumed that the module has been configured to compute an N-point FFT/IFFT.

Table 6 • **I/O Signal Description**

Signal Name	Direction	Description
clk	Input	System clock. Active rising edge.
nreset	Input	System asynchronous reset. Active low.
d_im[*b* – 1:0]	Input	Input imaginary data bus. The imaginary part of the input complex data should be placed on this bus. Bit *b* – 1 is the MSB. Data are assumed to be presented in two's-complement format. The imaginary and real parts should be supplied simultaneously.
d_re[*b* – 1:0]	Input	Input real data bus. The real part of the input complex data should be placed on this bus. Bit *b* – 1 is the MSB. Data are assumed to be presented in two's-complement format. The imaginary and real parts should be supplied simultaneously.
d_valid	Input	Input complex word valid. Active high. The bit accompanies valid input samples coming to input busses ***d_im*** and ***d_re***. At any system clock interval where ***d_valid*** is active, input busses ***d_im*** and ***d_re*** are considered to present another input complex sample.
load	Output	The FFT module input buffer accepts data. Active high. The signal is active when the input buffer (either of two banks) is ready to accept data. The signal stays active until the buffer is full.
start	Input	FFT start signal. Active high. ***start*** is asserted to begin the transform processing or to return the module to the initial ready state.
y_rdy	Output	FFT results ready. Active high. The signal goes active when the FFT results are ready for the host to read. It stays HIGH during host read.
y_im[*b* – 1:0]	Output	Output imaginary data bus. The imaginary part of the output complex data appears on this bus. Bit *b* – 1 is the MSB. Data are presented in two's-complement format. The imaginary and real parts appear simultaneously.
y_re[*b* – 1:0]	Output	Output real data bus. The real part of the output complex data appears on this bus. Bit *b* – 1 is the MSB. Data are presented in two's-complement format. The imaginary and real parts appear simultaneously.

Downloaded from Arrow.com.

Table 6 • **I/O Signal Description (continued)**

Signal Name	Direction	Description
y_valid	Output	Output complex word valid. Active high. The bit accompanies valid output samples on output busses ***y_im*** and ***y_re***. At any system clock interval while ***y_valid*** is active, a complex sample is available on output busses ***d_im*** and ***d_re.***
read_y	Input	Read FFT output. Active high. If the signal is active, the module puts out the FFT results in a single burst, one complex word per clock cycle. The host can insert arbitrary breaks into the burst by deactivating the signal any time during the burst.
pong	Output	Pong bank of the input buffer is being used by the FFT engine as a working memory.

I/O Interface and Timing

Resetting the Module

Upon reset, the module returns to its initial state with input and output buffer pointers reset to zero. The input buffer is now ready to accept a new data frame; signal ***load*** is asserted, and signals ***y_rdy*** and ***y_valid*** are deasserted. Both ***nreset*** and ***start*** reset the module.

Loading Input Data

Input data can be loaded once the signal ***load*** is asserted; otherwise, the module ignores any activity on the data loading pins. When the host detects an active ***load*** signal, it may begin writing data via the *b*-bit busses ***d_im*** and ***d_re***. Every valid complex sample must be accompanied by an active ***d_valid*** signal (Figure 5 on page 10). The module samples ***d_valid*** at each rising edge of the system clock. Once an active ***d_valid*** signal is detected, the core assumes a new complex sample has been written to the input busses. The module then writes the new sample to the input buffer. By the next system clock edge, the module is ready to accept another input sample. The host can control the input sample rate via ***d_valid***. Once the module has received N complex samples, the input buffer is now full, and the module deasserts the signal ***load***.

Reading Output Data

Once the FFT engine completes another FFT computational cycle, it asserts the ***y_rdy*** signal. The FFT results are now available for the host in the output buffer. The CoreFFT module puts out the post-processed complex samples on the two *b*-bit busses, ***y_im*** and ***y_re***. Every valid complex sample is accompanied by a ***y_valid*** bit. In the basic mode where the host does not control the FFT output data rate (signal ***read_y*** remains permanently active), all N post-processed complex samples from a single burst are available consecutively at each rising system clock edge (Figure 6 on page 10). During this mode, ***y_valid*** remains valid for N system clock cycles.

The host can control the FFT output sample rate via the ***read_y*** signal. The input ***read_y*** acts similarly to an output clock enable signal: when held HIGH, the module will continue generating FFT results at each clock edge; when held LOW, the module will pause in generating.

An example of a controlled output rate mode is depicted in Figure 7 on page 10. Every new output sample is valid for two system clock cycles, as ***read_y*** is asserted only every other clock cycle. (Note that there is latency of one clock cycle between the signal ***read_y*** and a valid sample output).

The CoreFFT module design does not place any restrictions on the duty cycle of the ***read_y*** signal. However, for the FFT engine to operate at maximum efficiency (i.e., no idle time), the post-processed results must be read out of the output buffer before the engine needs to write the results of the next data frame (this time is marked as Accept FFT Result in Figure 3 on page 6). Table 4 on page 6 shows the minimal output reading rate that does not impact the efficient use of the FFT engine.

Downloaded from Arrow.com.

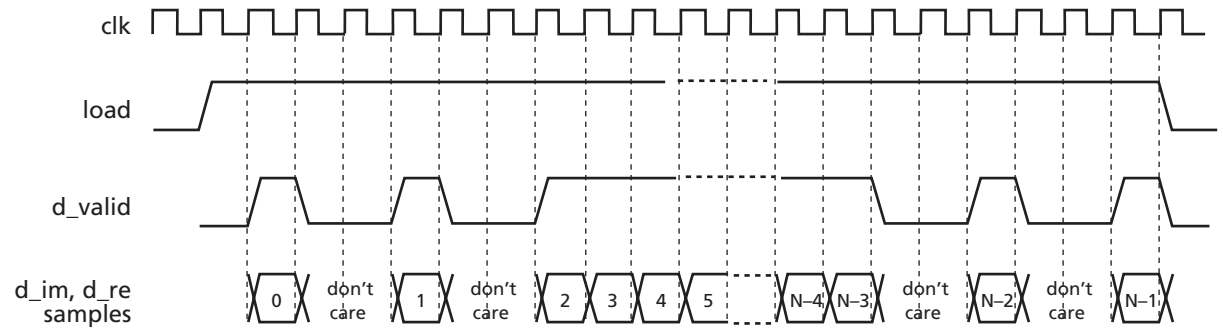

Figure 5 • **Data Load Timing**

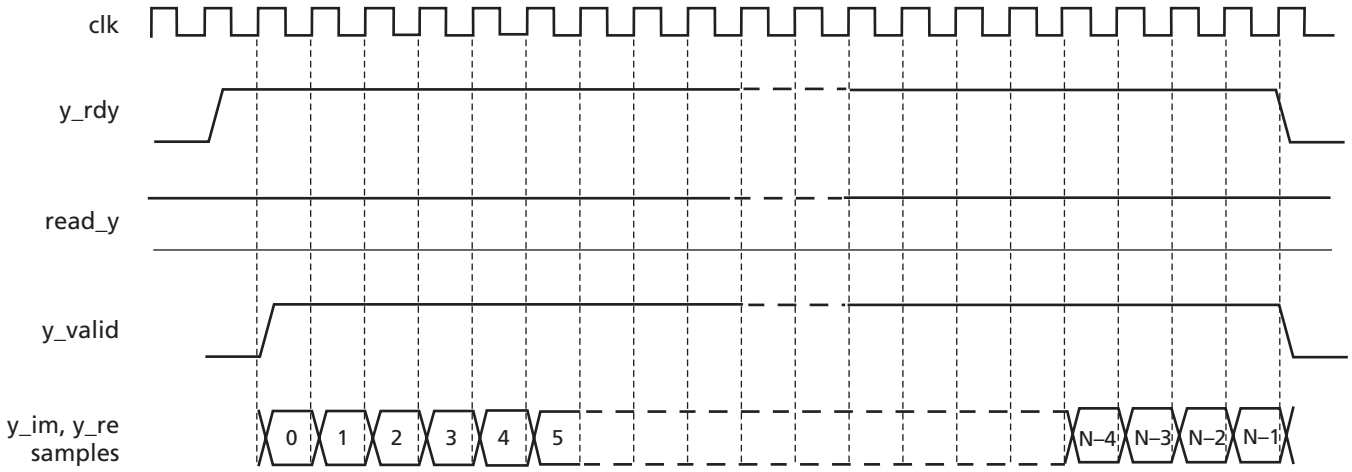

Figure 6 • **FFT Results Output**

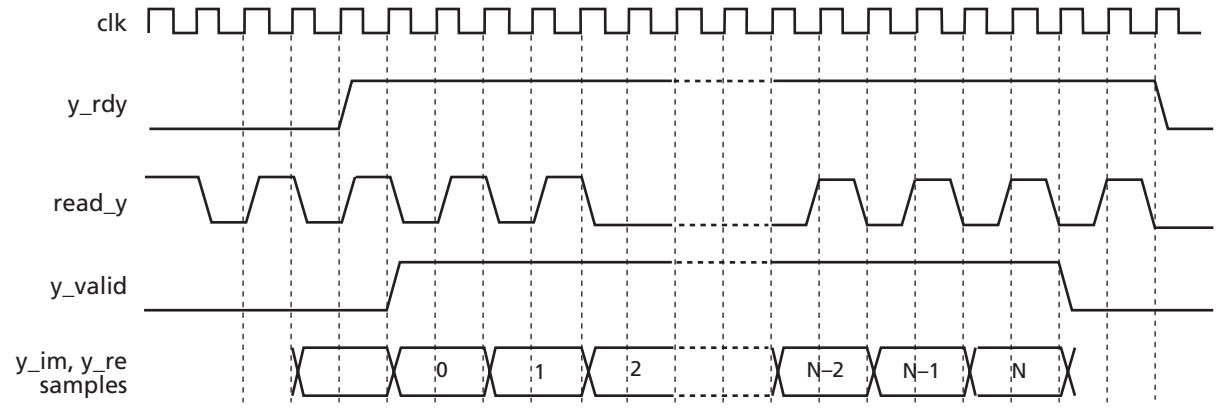

Figure 7 • **Host Controls the Output Sample Rate**

Downloaded from Arrow.com.

References

1. L. R. Rabiner and B. Gold, *Theory and Application of Digital Signal Processing,* Prentice Hall, 1975.

2. Alan V. Oppenheim and Ronald W. Schafer with John R. Buck, *Discrete-Time Signal Processing, Second Edition*, Prentice Hall, 1998.

A Sample Configuration File

The following is an example configuration file:

```
inv             0
scale           0
points          256
bits            16
fpga_family     pa3
lang            verilog
```

Ordering Information

Order CoreFFT through your local Actel sales representative. Use the following numbering convention when ordering: CoreFFT-XX, where XX is listed in Table 7.

Table 7 • **Ordering Codes**

XX	Description
EV	Evaluation version
AR	RTL for unlimited use on Actel devices
UR	RTL for unlimited use and not restricted to Actel devices

Downloaded from Arrow.com.

Appendix I: Fast Fourier Transform

The FFT is a computationally efficient algorithm for computing a discrete Fourier transform (DFT). The N-point DFT is defined as

$$X(k) \equiv \sum_{n=0}^{N-1} x[n] e^{(-jnk2\pi)/N} \qquad k = 0, 1, 2, \ldots, N-1$$

EQ 7

where N is the transform size, or number of points. The inverse N-point DFT is defined as

$$x(n) = \left(\frac{1}{N}\right) \sum_{k=0}^{N-1} X[k] e^{(jnk2\pi)/N} \qquad k = 1, 2, 3, \ldots, N-1$$

EQ 8

It is common practice to call the exponential vector rotating factors above the "twiddle factors." Every twiddle factor contains real and imaginary parts

$$W = W_r - jW_i = e^{(-jnk2\pi)/N}$$

EQ 9

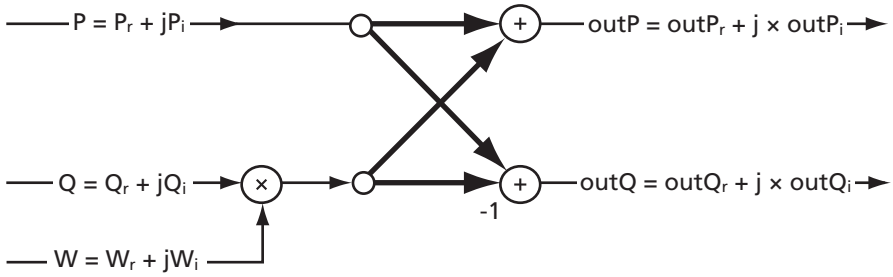

Figure 8 • **Radix-2 DIT Butterfly**

The butterfly performs the basic FFT computation according to the following equations:

$$outP = P + Q \times W$$

EQ 10

$$outP = P - Q \times W$$

EQ 11

The twiddle factor can be expressed as

$$W = \cos X - j \sin X$$

EQ 12

By substituting the values from EQ 12 and expressing P and Q in terms of their real and imaginary parts, EQ 10 and EQ 11 become

$$\begin{aligned} outP &= P_r + jP_i + (Q_r \cos X + Q_i \sin X) + j(Q_i \cos X - Q_r \sin X) \\ &= (P_r + Q_r \cos X + Q_i \sin X) + j(P_i + Q_i \cos X - Q_r \sin X) \end{aligned}$$

EQ 13

$$\begin{aligned} outQ &= P_r + jP_i - (Q_r \cos X + Q_i \sin X) - j(Q_i \cos X - Q_r \sin X) \\ &= (P_r - Q_r \cos X - Q_i \sin X) + j(P_i - Q_i \cos X + Q_r \sin X) \end{aligned}$$

EQ 14

Downloaded from Arrow.com.

Figure 9 depicts an example of an 8-point FFT algorithm. The tree contains $\log_2 8 = 3$ stages with $8 / 2 = 4$ butterflies calculated at every stage.

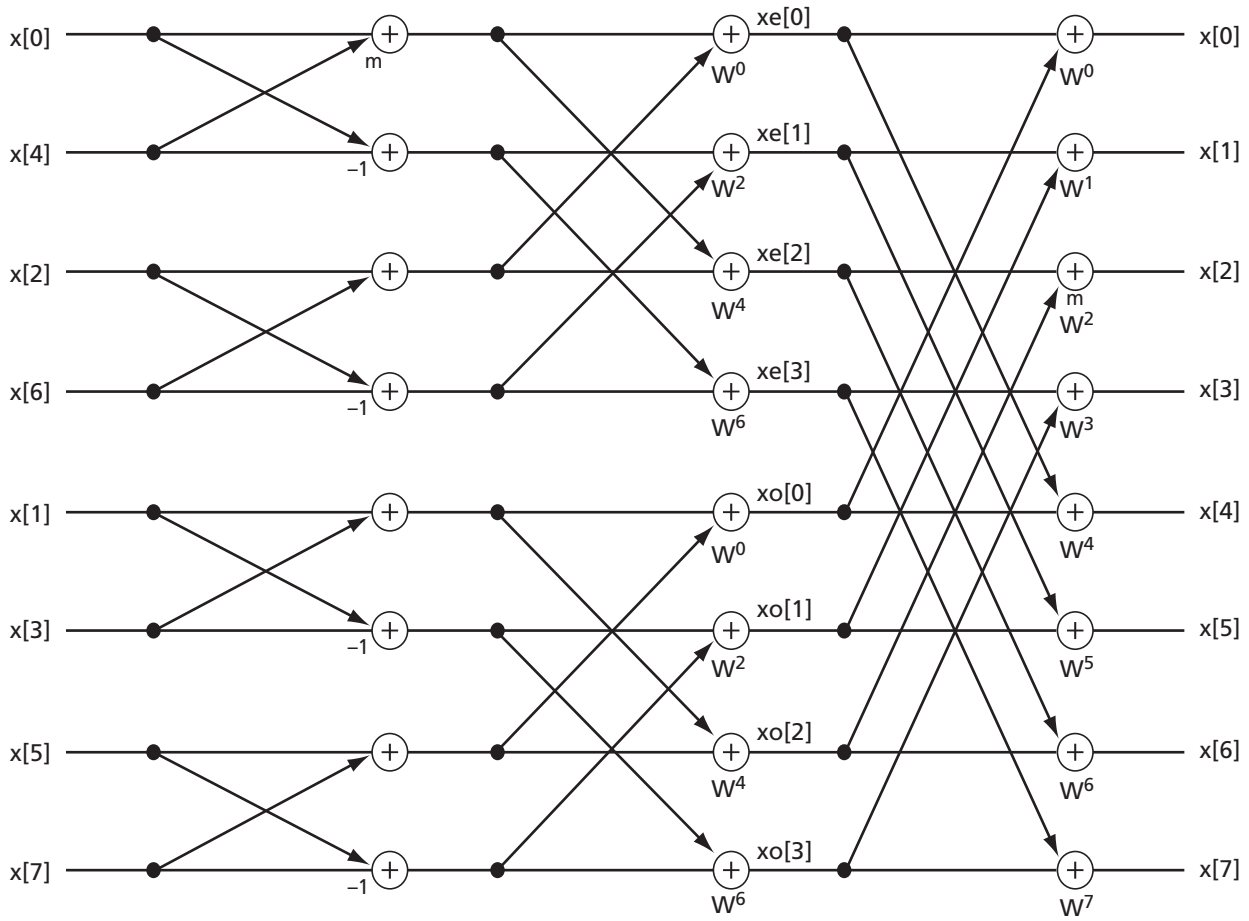

Figure 9 • **8-Point FFT Using Decimation-in-Time Algorithm**

Downloaded from Arrow.com.

List of Changes

The following table lists critical changes that were made in the current version of the document.

Previous Version	Changes in Current Version (v4.0)	Page
v3.0	Fusion was removed from the "Targeted Devices" section.	1
	The "Synthesis and Simulation Support" section was updated.	1
	FFT size expanded to include 32-, 64-, 128-, and 2,048-point transforms.	1
	Input/output data width changed to 8- to 16-bit (configurable).	1
	All input and output data widths throughout document now expressed in terms of configurable bit width *b* = 8 to 16 bits.	N/A
	Table 1 replaced with Table 1 and Table 2.	3–4
	EQ 4 was revised.	6
	Table 5 updated: removed module_name parameter, added bits parameter, updated points and fpga_family parameters.	7
	Updated the sample in the "A Sample Configuration File" section.	11
v2.0	The "Targeted Devices" section was updated to include Fusion.	1
	Table 1 (now Table 1 and Table 2) was updated to include Fusion data.	3

Datasheet Categories

In order to provide the latest information to designers, some datasheets are published before the data has been fully characterized. Datasheets are designated as "Product Brief," "Advanced," and "Production." The definitions of these categories are as follows:

Product Brief

The product brief is a summarized version of an advanced or production datasheet containing general product information. This brief summarizes specific device and family information for unreleased products.

Advanced

This datasheet version contains initial estimated information based on simulation, other products, devices, or speed grades. This information can be used as estimates, but not for production.

Unmarked (production)

This datasheet version contains information that is considered to be final.

Downloaded from Arrow.com.

Actel and the Actel logo are registered trademarks of Actel Corporation.
All other trademarks are the property of their owners.

www.actel.com

Actel Corporation

2061 Stierlin Court
Mountain View, CA
94043-4655 USA
Phone 650.318.4200
Fax 650.318.4600

Actel Europe Ltd.

Dunlop House, Riverside Way
Camberley, Surrey GU15 3YL
United Kingdom
Phone +44 (0) 1276 401 450
Fax +44 (0) 1276 401 490

Actel Japan
www.jp.actel.com

EXOS Ebisu Bldg. 4F
1-24-14 Ebisu Shibuya-ku
Tokyo 150 Japan
Phone +81.03.3445.7671
Fax +81.03.3445.7668

Actel Hong Kong
www.actel.com.cn

Suite 2114, Two Pacific Place
88 Queensway, Admiralty
Hong Kong
Phone +852 2185 6460
Fax +852 2185 6488

51700058-2/5.07

Downloaded from Arrow.com.