
CoreFFT v5.0

Handbook



Table of Contents

Introduction	3
Core Overview	3
Key Features	3
Core Version	4
Supported Families	4
Utilization and Performance	4
Architecture Options	7
Natural Output Order	7
In-Place FFT	9
Verilog/VHDL Parameters	9
I/O Signals	10
In-Place FFT Architecture	11
Streaming FFT	17
Verilog/VHDL Parameters	17
I/O Signals	18
Streaming FFT Architecture	20
Handshake Signals	20
Finite Word Length Considerations	24
Tool Flows	25
Licensing	25
SmartDesign	25
Simulation Flows	25
Synthesis in Libero IDE	26
Place-and-Route in Libero IDE	26
Ordering Information	27
Ordering Codes	27
List of Changes	29
Product Support	31
Customer Service	31
Technical Support	31

Introduction

Core Overview

The Microsemi Fast Fourier transform (FFT) core implements the efficient Cooley-Tukey algorithm for computing the discrete Fourier transform. The FFT is used in a broad range of applications like digital communications, audio, measurements, control, and biomedical. CoreFFT provides highly parameterizable, area-efficient, and high performance MAC-based FFT. The core is available as an RTL code of the transform in Verilog and VHDL languages.

The N-point forward FFT (N is a power of 2) of a sequence $x(0), x(1), \dots, x(N-1)$ is defined in EQ 1:

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-jnk2\pi/N} \quad \text{EQ 1}$$

where $k = 0, 1 \dots N-1$

The N-point inverse FFT (N is a power of 2) of a sequence $X(0), X(1), \dots, X(N-1)$ is defined in EQ 2:

$$x(n) = \sum_{k=0}^{N-1} X(k)e^{jnk2\pi/N} \quad \text{EQ 2}$$

where $n = 0, 1 \dots N-1$

Note: While performing an inverse FFT, the core does not apply division by N of EQ 2 (as the division by a power of two is trivial).

An FFT-based system (Figure 1) consists of a data source, the FFT module, and a data sink which is the transformed data recipient.



Figure 1 FFT-Based System Example

Key Features

CoreFFT supports two transform implementations:

- Radix-2 Decimation-in-time in-place
- Radix² Decimation-in-frequency streaming FFT

The key features for every implementation are listed in Table 1.

Table 1 Key Features Support

Feature	In-Place	Streaming
Transform sizes	32-, 64-, 128-, 256-, 512-, 1024-, 2048-, 4096-, and 8192-point	16-, 32-, 64-, 128-, 256-, 512-, and 1024-point
Forward and Inverse FFT	Yes	Yes
Input data bit width	8 – 32	8 – 32
Twiddle factor bit width	8 – 32	8 – 32
Input/output data format	Two's complementary	Two's complementary
Natural output sample order	Yes	Optional
Conditional block floating point scaling	Yes	No
Pre-defined scaling schedule	No	Yes
Optional minimal or buffered memory configurations	Yes	No
Embedded RAM-block based twiddle look-up table	Yes	Yes
Support for refreshing twiddle look-up tables	Yes	Yes
Handshake signals to facilitate easy interface to the user circuitry	Yes	Yes
Run-time Forward/Inverse transform configuration	No	Yes

Core Version

This handbook applies to CoreFFT v5.0.

Supported Families

The CoreFFT supports the following devices in the RTAX family:

- RTAX2000D
- RTAX4000D

Utilization and Performance

CoreFFT has been implemented in the RTAX2000D device using the standard speed grade. A summary of the implementation data is provided in Table 2 through Table 4.

In-Place FFT

Table 2 and Table 3 show utilization and performance for variety of the in-place FFT sizes and data widths. The numbers were obtained from the configuration listed in Table 4 on page 6.

Table 2 In-place FFT Device Utilization and Performance (Minimal Memory Configuration)

Core Parameters		Tiles				Blocks		Performance	
POINTS	WIDTH	Sequential	Combinational	Total	%	RAM	MAC	Clock Rate	FFT Time (μs)
256	9	579	651	1230	4.2	3	4	124.2	8.9
256	18	933	1029	1962	6.6	3	4	112.4	9.8
256	24	1927	1316	3243	11.0	6	16	118.5	9.7
512	9	612	771	1383	4.7	3	4	103.6	23.1
512	18	967	1233	2200	7.4	6	4	107.5	22.3
512	24	1941	1562	3503	11.8	9	16	107.0	22.9
1024	9	641	891	1532	5.2	6	4	113.4	46.1
1024	18	977	1557	2534	8.6	12	4	97.7	53.5
1024	24	1965	1976	3941	13.3	18	16	100.8	52.4
2048	12	769	1367	2136	7.2	18	4	88.0	129.2
2048	18	1005	2097	3102	10.5	27	4	69.4	163.9
2048	24	1958	2827	4785	16.2	36	16	63.6	179.8
4096	12	811	1779	2590	8.8	36	4	73.6	335.7
4096	15	930	2285	3215	10.9	45	4	62.2	397.0
4096	18	1009	2884	2893	13.2	54	4	58.4	422.8
4096	20	1707	3419	5126	17.3	60	16	50.8	422.8
8192	10	742	1708	2450	8.3	60	4	59.7	893.7

Table 3 In-place FFT Device Utilization and Performance (Buffered Configuration)

Core Parameters		Tiles				Blocks		Performance	
POINTS	WIDTH	Sequential	Combinational	Total	%	RAM	MAC	Clock Rate	FFT Time (μs)
256	9	654	719	1373	4.6	7	4	111.1	10.0
256	18	1048	1129	2177	7.4	7	4	107.4	10.3
256	24	2067	1467	3534	12.0	14	16	107.9	10.7
512	9	690	824	1514	5.1	7	4	103.7	23.1
512	18	1092	1350	2442	8.3	14	4	102.7	23.3
512	24	2094	1705	3799	12.8	21	16	103.6	23.7
1024	9	723	953	1676	5.7	14	4	105.2	49.6
1024	18	1106	1669	2775	9.4	28	4	105.0	49.7
1024	24	2148	2129	4277	14.5	42	16	85.8	61.6
2048	12	904	1412	2316	7.8	42	4	88.3	128.8
2048	18	1178	2200	3378	11.4	63	4	66.1	172.2

Notes:

1. Data in Table 2 and Table 3 were achieved using typical synthesis settings. The Synplify® Frequency (MHz) was set to 150.
2. Layout settings were as follows:
 Designer block creation enabled
 Layout effort level = 5
 Use multiple passes with number of passes = 5
3. The FFT time shown reflects the transformation time only. It does not account for data downloading or result uploading times.

Table 4 In-Place FFT Utilization and Performance Configuration

Parameter	Value
INVERSE	0
SCALE	0
SCALE_EXP_ON	0
FPGA part	RTAX2000D
Speed grade	Std
HDL type	Verilog

Streaming FFT

Table 5 shows utilization and performance for a variety of Streaming FFT parameters. The numbers were obtained from the configuration listed in Table 6.

Table 5 Streaming FFT Device Utilization and Performance

Core Parameters				Tiles				Blocks		Clock Rate
FFT_SIZE	DATA_BITS	TWID_BITS	Order	Sequential	Combinational	Total	Percentage (%)	RAM	MAC	
16	18	18	Reverse	1399	1344	2743	9.28	1	4	112
16	18	18	Normal	1422	1401	2823	9.55	3	4	113
32	18	18	Reverse	1781	1745	3526	11.93	3	8	112
64	18	18	Reverse	2123	2266	4389	14.84	4	8	110
128	18	18	Reverse	2546	2823	5369	18.16	6	12	109
256	18	18	Reverse	2935	3488	6423	21.72	8	12	109
256	18	18	Normal	2965	3587	6552	22.16	12	12	108
512	18	18	Reverse	3452	4319	7771	26.28	13	16	111
512	18	24	Reverse	3997	4882	8879	30.03	18	32	100
1024	18	18	Reverse	3985	5780	9765	33.03	23	16	90
1024	18	24	Reverse	4604	6651	11255	38.06	29	32	96
1024	24	24	Reverse	7025	7710	14735	49.83	36	64	93

Notes:

1. Data in Table 5 were achieved using typical synthesis settings. Synplify® Frequency (MHz) was set to 150.
2. Layout settings were set as follows:
 Designer block creation enabled
 Layout effort level = 5
 Use multiple passes with number of passes = 5

Table 6 Streaming FFT Utilization and Performance Configuration

Parameter	Value
SCALE	1
FPGA part	RTAX2000D
Speed grade	Std
HDL type	Verilog

Architecture Options

Depending on user configuration, CoreFFT generates one of the supported transformation implementations.

- Streaming FFT: The architecture supports continuous complex data processing, one input data sample per clock period.
- In-Place FFT: The transform loads a frame of N complex data samples in the in-place RAM, and processes them sequentially, stage by stage. It stores the results of every stage back in the in-place RAM. The in-place FFT takes fewer chip resources than the streaming FFT but the transformation time is longer.

Natural Output Order

The output results obtained from the Radix-2 and Radix-2² FFT algorithms are in the bit-reversed order. The in-place implementation however, internally performs the sample ordering. Therefore, the core outputs the results in a natural order. The Streaming FFT supports both bit-reversed and natural output orders. Some FFT applications can easily use the bit-reversed FFTed data. On the other hand, the bit-reversed option utilizes fewer chip resources and provides smaller latency.

In-Place FFT

Verilog/VHDL Parameters

CoreFFT has parameters (Verilog) or generics (VHDL) for configuring the RTL code. These parameters and generics are described in Table 7. All parameters and generics are integer types.

Table 7 CoreFFT In-Place Architecture Parameter Descriptions

Parameter	Valid Range	Default	Description
INVERSE	0 – 1	0	<ul style="list-style-type: none"> • 0 – Forward Fourier transform • 1 – Inverse Fourier transform
SCALE	0 – 1	0	<ul style="list-style-type: none"> • 0 – Conditional block floating point scaling • 1 – Unconditional block floating point scaling <p>To apply the input data scaling rather than conditional or unconditional block floating point techniques, set the SCALE parameter to 0 and prepend the proper number of guard bits to the input data. Then the conditional block floating point will take no effect.</p>
POINTS	32, 64, 128, 256, 512, 1024, 2048, 4096, 8192	256	Transform size
WIDTH	8 – 32	18	Data and twiddle factor bit width
MEMBUF	0 – 1	0	<ul style="list-style-type: none"> • 0 – Minimal (no buffer) configuration • 1 – Buffered configuration
SCALE_EXP_ON	0 – 1	0	<ul style="list-style-type: none"> • 0 – Do not build the Conditional block floating-point exponent calculator • 1 – Build the calculator

I/O Signals

The port signals for the in-place CoreFFT macro are illustrated in Figure 2 and defined in Table 8.

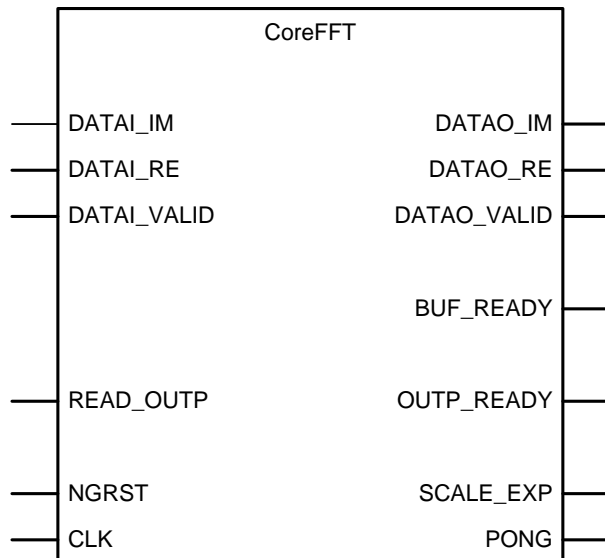


Figure 2 Core I/O Ports

Table 8 In-Place CoreFFT I/O Signal Descriptions

Port Name	Type	Description
DATAI_IM[WIDTH-1:0]	In	Imaginary input data to be transformed
DATAI_RE[WIDTH-1:0]	In	Real input data to be transformed
DATAI_VALID	In	Input complex word valid. The signal accompanies valid input complex words present on inputs DATAI_IM, DATAI_RE. When the signal is active, the input complex word is loaded into the core memory provided BUF_READY signal has been asserted.
READ_OUTP	In	Read transformed data. Normally the module outputs FFT results once they are ready in a single burst of N complex words. The transformed data recipient can insert arbitrary breaks in the burst by deasserting the READ_OUTP signal.
DATAO_IM[WIDTH-1:0]	Out	Imaginary output data
DATAO_RE[WIDTH-1:0]	Out	Real output data
DATAO_VALID	Out	Output complex word valid. The signal accompanies valid output complex words present on outputs DATAO_IM, DATAO_RE.
BUF_READY	Out	The FFT accepts fresh data. The core asserts the signal when it is ready to accept data. The signal stays active until the core memory is full. In other words, the signal stays active until POINTS complex input samples are loaded.

OUTP_READY	Out	FFT results ready. The core asserts the signal when the FFT results are ready for the transformed data recipient to read. The signal stays active while the transformed data frame is being read. Normally it lasts for POINTS clock intervals unless READ_OUTP signal is deasserted.
SCALE_EXP	Out	Conditional block floating-point scaling exponent. Enable this optional output by setting the parameter SCALE_EXP_ON. The output can be enabled when the core is in Conditional block floating-point scaling mode only (the parameter SCALE = 0). It is calculated from the expression $\lceil \log_2(\lceil \log_2(\text{POINTS}) \rceil) - 1:0 \rceil$.
PONG	Out	Pong bank of the input memory buffer is being used by the FFT engine as a working in-place memory. This optional signal is valid only in the Buffered configuration.
CLK	In	Clock. Rising edge active. The core master clock.
NGRST	In	Asynchronous reset. Active low.

Note: All signs are active high (logic 1) unless otherwise specified.

In-Place FFT Architecture

CoreFFT implements the area-efficient in-place Radix-2 Fast Fourier Transform. Figure 3 depicts a functional diagram of the transform.

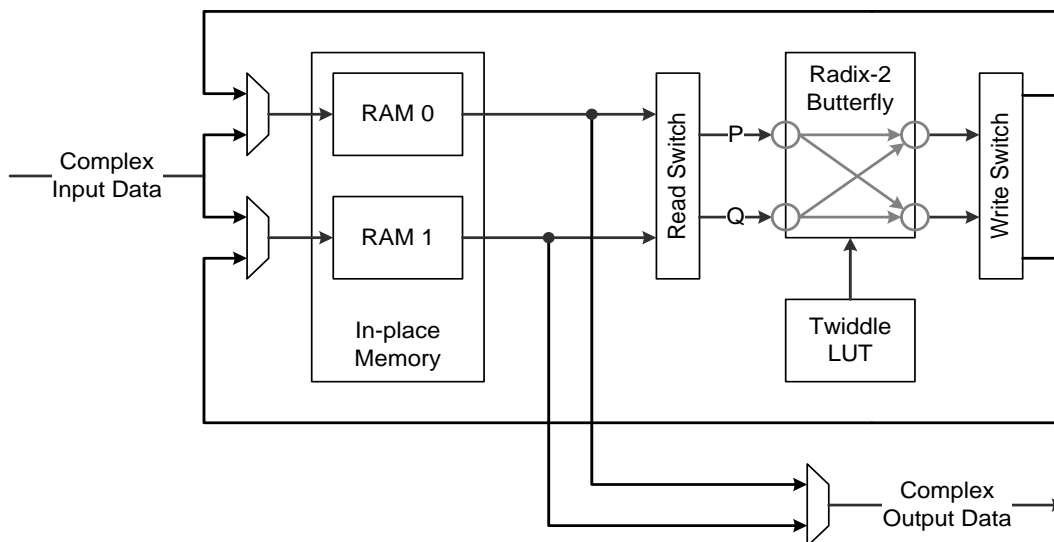


Figure 3 In-Place Radix-2 FFT Functional Block Diagram (Minimal Configuration)

The input and output data are represented as $2 \cdot \text{WIDTH}$ -bit words comprised of real and imaginary parts. Both parts are two's complementary numbers of WIDTH bits each. The module processes frames (bursts) of data with a frame size of N complex words. The frame to be processed is loaded in an in-place memory. The memory contains two identical RAM modules, each capable of storing $N/2$ complex words. This is how the in-place memory supports double bandwidth; i.e., it can read and write two complex words at the same time. Once the N complex data samples are loaded in the memory, the FFT computation starts automatically and the in-place memory is used for computations.

The in-place FFT computational process occurs in a sequence of stages with the number of stages equal to $\log_2 N$. At every stage of the FFT data processing engine, the Radix-2 butterfly reads all the data stored in the in-place memory, two complex words at a time. The Read Switch along with a Read Address Generator (not shown in Figure 3) helps the butterfly to obtain stored data in the order required by the FFT algorithm. In

In addition to the data, the butterfly obtains twiddle factors (sine/cosine coefficients) from the twiddle look-up table. The butterfly writes intermediate results back to the in-place memory via the Write Switch.

After the last computational stage, the in-place memory stores the fully transformed data. The module outputs an N-word transformed data frame, one word at a time, provided the signal READ_OUTP is active.

CoreFFT calculates the twiddle factors required by the FFT algorithm and writes them to the twiddle LUT. This happens automatically on power-on when asynchronous global reset is asserted.

In-Place Memory Buffers

Minimal Configuration

The minimal configuration shown on Figure 3 is sufficient to accomplish the FFT because it has the in-place RAM required by the FFT algorithm. But the minimal configuration does not utilize the processing engine all the time. On the contrary, when data is loaded in the in-place Memory, or the transformed data are read out, the butterfly stays idle. Figure 4 shows the FFT cycle timeline. The cycle consists of the three following phases:

- Download a fresh input data frame in the in-place RAM.
- Perform the actual transformation, which was loaded in the first phase after computing the FFT of the frame.
- Upload the transformed frame to where it needs to be.

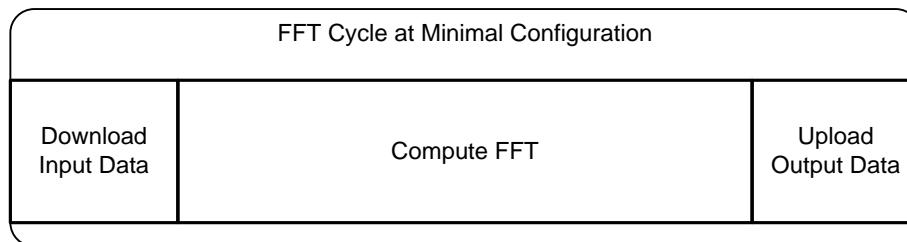


Figure 4 Minimal Configuration In-Place FFT Cycle

In the minimal configuration, the butterfly runs only during the computation phase. When the data burst rate permits, the minimal configuration provides the best device resource utilization. In particular, it saves a significant number of RAM blocks.

Buffered Configuration

In order to improve the butterfly utilization and consequently reduce average transformation time, additional memory buffers are optionally used. Figure 5 depicts the buffered FFT block diagram.

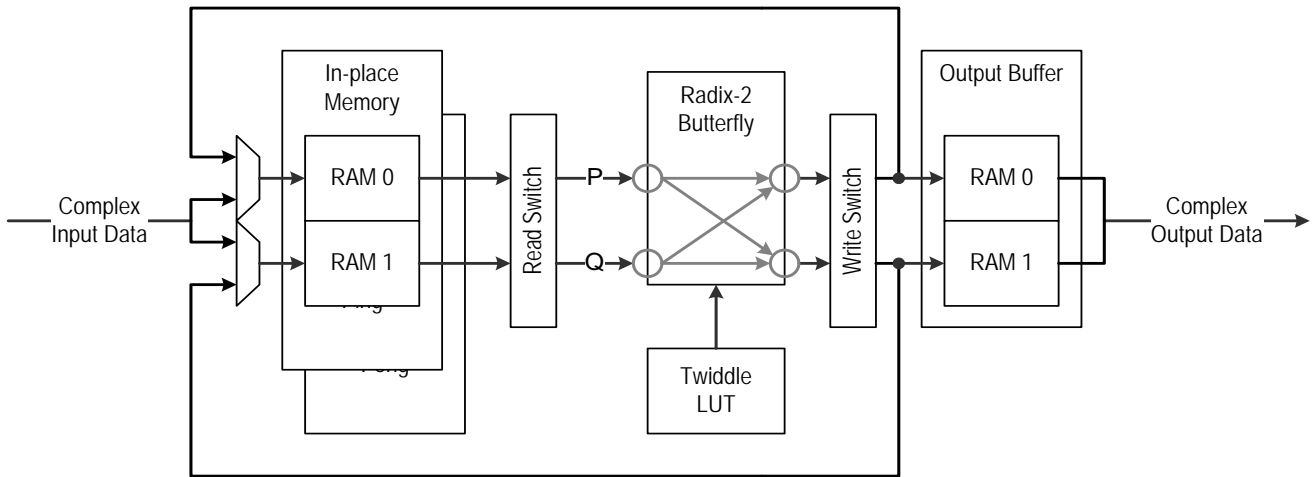


Figure 5 Buffered FFT Block Diagram

The buffered option has two identical in-place memory banks implementing a ping-pong buffer and one output buffer. Each bank is capable of storing N complex words and reading two complex words at a time. The core state machine controls the ping-pong switching so that a data source sees only a buffer that is ready to accept new data. The buffer not accepting the new data is used as an in-place RAM by the FFT engine.

The ping-pong buffering architecture increases the efficiency of the FFT engine. While one of the two input banks is involved in current FFT computation, the other is available for downloading the next input data frame. As a result, the FFT engine does not sit idle waiting for fresh data to fill the input buffer. From the data source perspective, the core can receive a data burst anywhere within the FFT computation period. When the engine has finished processing the current data frame and the input buffer bank has been filled with another data frame, the state machine swaps the ping-pong banks, 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 engine reads intermediate data from the in-place memory but writes the final result in the output data buffer. The final results remain in the output buffer until the FFT engine is ready to put there the results of the next data frame. From the data recipient perspective, the output data are available for reading any time, except for the last FFT stage.

The buffered configuration FFT cycle is shown in Figure 6.

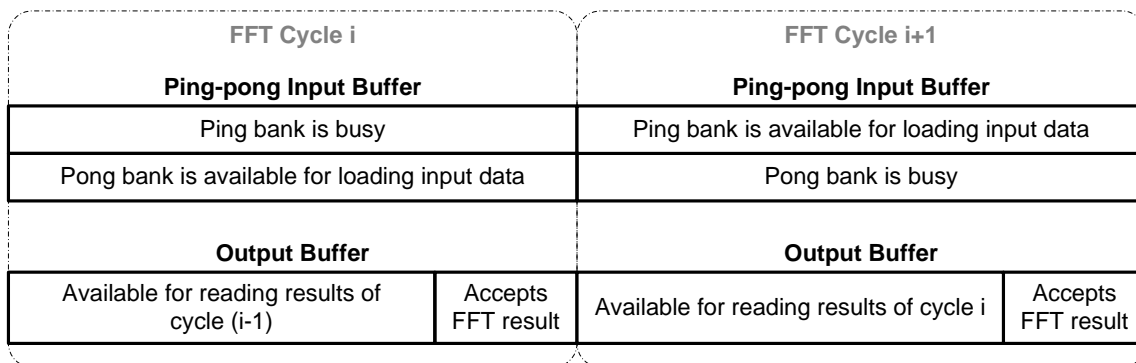


Figure 6 Buffered Configuration FFT Cycles

Example of Using In-Place FFT

Figure 7 presents an example of using the core. When the in-place FFT asserts the BUF_READY signal, a data source starts supplying the data samples to be transformed. Imaginary and real halves of the input data sample should be supplied simultaneously and accompanied with validity bit DATAI_VALID. The data source can supply the sample at every clock cycle or at an arbitrary slower rate (see Figure 8 on page 14). Once the FFT module receives N input samples, it lowers the BUF_READY signal.

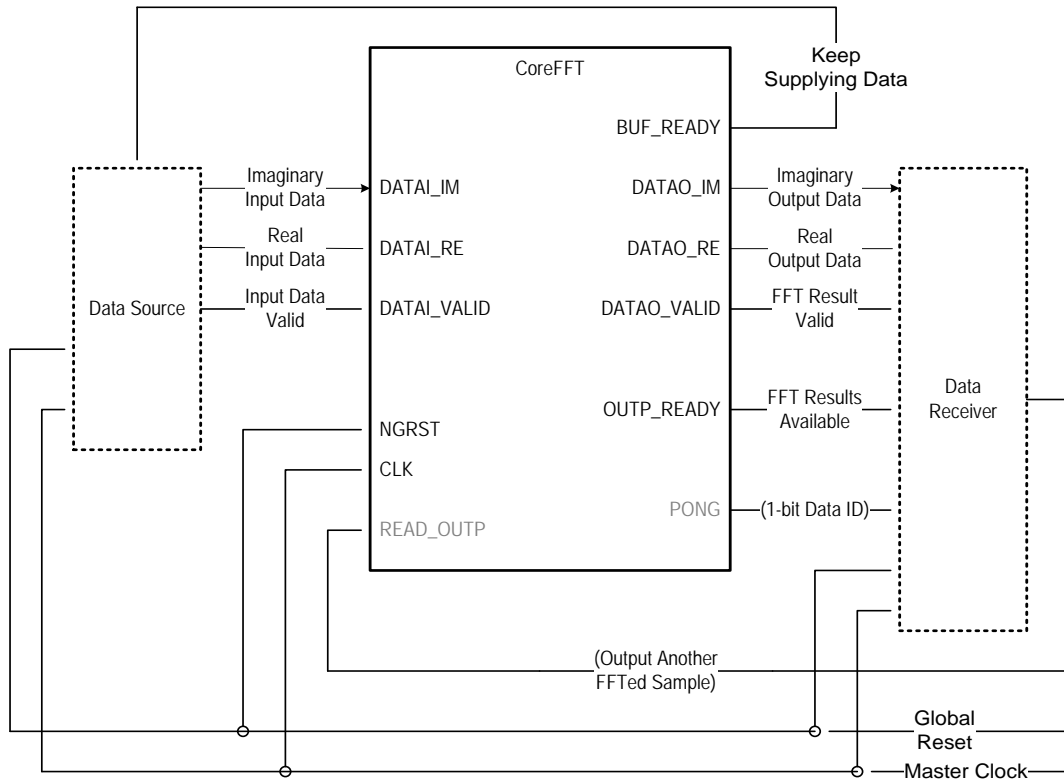


Figure 7 Example of the FFT System

The FFT engine starts processing the data automatically once it is ready. In the minimal memory configuration, the processing phase starts immediately after data loading is complete. In the Buffered configuration, the FFT engine may wait until a previous data burst is processed. Then the engine starts automatically.

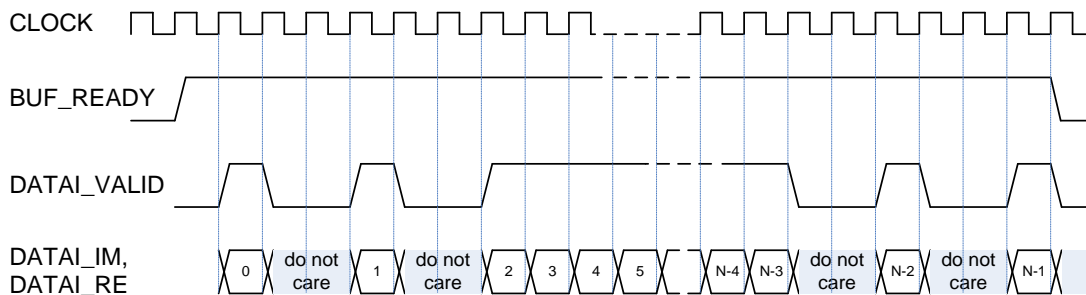


Figure 8 Loading Data to be Transformed

Upon completing the transformation, the FFT module asserts the OUP_READY signal and starts generating the FFT results. The imaginary and real halves of the output samples appear simultaneously on DATAO_IM and DATAO_RE multi-bit outputs. Every output sample is accompanied by the DATAO_VALID bit. The data receiver can accept the transformed data either at every clock cycle or at an arbitrary slower

rate. The FFT module will keep providing data output while the READ_OUTP signal is asserted. To control the output sample rate, the receiver should deassert the READ_OUTP signal as and when needed (Figure 9).

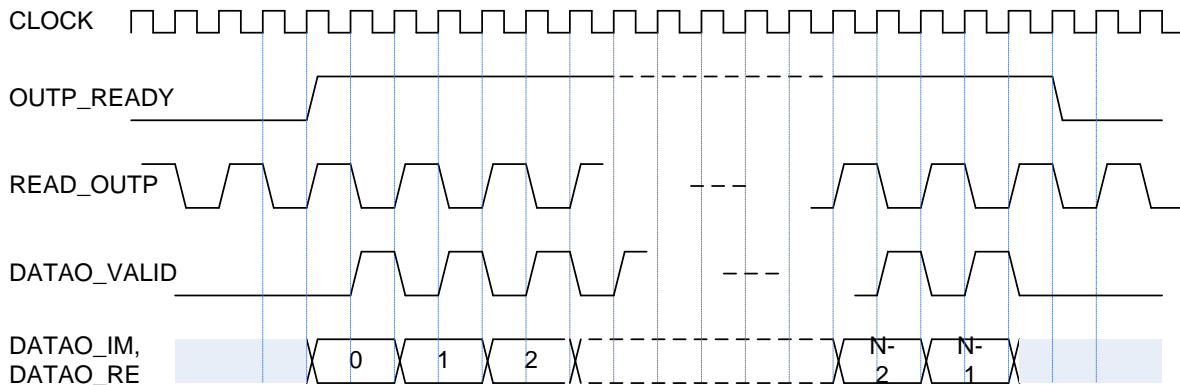


Figure 9 Receiving Transformed Data

When using the READ_OUTP signal to control reading rate, you have to consider possible FFT cycle growth. In the Minimal memory configuration, any prolongation of the read (upload) time extends the FFT cycle (see Figure 4). In the Buffered configuration, the FFT cycle grows when the actual upload time exceeds the dedicated interval shown in Figure 6 on page 13 as 'Available for reading results of cycle i'. Also, in the Buffered configuration, the output buffer starts accepting the fresh FFT results even if the older results have not been read out, thus overwriting the older ones. In this case the core deasserts the OUTP_READY and DATAO_VALID signals when they are no longer valid.

Finite Word Length Considerations

The butterfly calculation involves complex multiplication, addition, and subtraction. These operations can cause the butterfly data width to grow by two bits from input to output. At every stage of the in-place FFT algorithm, the butterfly takes two samples out of the in-place memory and returns two processed samples to the same memory locations. Potentially, returning samples may have a larger data width than the samples picked from the memory. Precautions must be taken to ensure that there are no data overflows.

To avoid risk of overflow, the core employs one of the following three methods:

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

You can select one of the three options.

The input data scaling requires pre-pending the input data samples with enough extra sign bits, called guard bits. The number of guard bits necessary to compensate for the maximum possible bit growth for an N-point FFT is $\log_2 N + 1$. For example, every input sample of a 256-point FFT must contain nine guard bits. Such a technique greatly reduces the effective FFT bit resolution.

The second way to compensate for the FFT bit growth is to scale the data down by a factor of two at every 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, the input data are scaled down by the factor of two to prevent overflow at the first stage. To prevent overflow in successive stages, the results of every previous stage are scaled down by the factor of two by shifting the entire block of data (all results of the current stage) one bit to the right. The total number of bits the data loses because of the bit shifting in the FFT calculation is $\log_2 N$.

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

In the 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 (quantization noise) caused by finite word length.

In the conditional block floating-point mode, the core can optionally calculate the actual scaling factor. It does so if the parameter `SCALE_EXP_ON` is set to be 1. Then the `SCALE_EXP` signal represents the number of right shifts the FFT engine applied to the results. For example, the `SCALE_EXP` value of 4 (100) means the FFT results were shifted right (downscaled) by 4 bits; i.e. divided by $2^{\text{SCALE_EXP}} = 16$. The signal accompanies the FFT results and is valid while `OUTP_READY` is asserted.

Note: The scale exponent calculator can be enabled in the Conditional block floating-point mode only.

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

Transformation Time

The FFT computation takes $(N/2 + L) \times \log_2 N + 2$ clock cycles, where L is an implementation specific parameter representing the aggregate latency of a memory bank, switches and the butterfly. L does not depend on transform size N . It only depends on the FFT bit resolution: $L = 10$ at bit resolutions 8 to 18, and $L = 16$ at bit resolutions from 19 to 32. E.g. For a 256-point 16-bit FFT the computation time = $(256/2 + 10) \times \log_2 256 + 2 = 1,106$ clock periods. For a 4096-point 24-bit FFT, the computation time will be $(4096/2 + 16) \times \log_2 4096 + 2 = 24,770$ clock periods.

Streaming FFT

Verilog/VHDL Parameters

CoreFFT has parameters (Verilog) or generics (VHDL) for configuring the RTL code, described in Table 9. All parameters and generics are integer types.

Table 9 CoreFFT Streaming Architecture Parameter Descriptions

Parameter Name	Valid Range	Default	Description
FFT_SIZE	16, 32, 64, 128, 256, 512, 1024	256	Transform size, points. The core processes frames of complex data with every frame containing FFT_SIZE complex samples. The transformed data frames are of the same size.
SCALE_ON	0 – 1	1	<ul style="list-style-type: none"> • 1 – Enable configurable scale schedule. When the option is enabled, the core applies the configurable scale factors defined by SCALE_SCH. • 0 – Disable scaling at all stages. The risk of overflow exists.
SCALE_SCH	0x0 – 0x3FF	0xFF	Scale schedule. If the SCALE_ON parameter = 1, the SCALE_SCH is used to indicate scaling factor for every processing stage. Every bit of the SCALE_SCH value defines the scaling factor for a corresponding FFT calculation stage: the least significant bit (LSB) applies to the first FFT stage, the next bit to the next stage and so on.
DATA_BITS	8 – 32	18	Data bit width
TWID_BITS	8 – 32	18	Twiddle factor bit width
ORDER	0 – 1	0	<ul style="list-style-type: none"> • 0 – Output data in bit-reversed order • 1 – Output data in normal order

I/O Signals

The port signals for the Streaming CoreFFT macro are illustrated in Figure 10 and defined in Table 10.

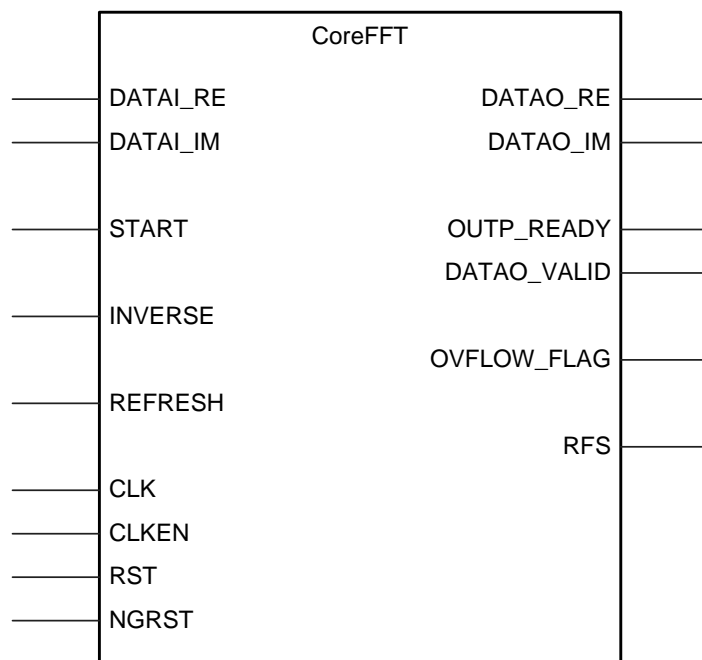


Figure 10 Streaming FFT I/O Ports

Table 10 Streaming FFT I/O Signal Descriptions

Port Name	In/Out	Port Width, bits	Description
DATAI_IM	In	DATA_BITS	Imaginary input data to be transformed
DATAI_RE	In	DATA_BITS	Real input data to be transformed
START	In	1	Transformation start signal. Signifies the moment the first sample of an input data frame of N complex samples enters the core. If the START comes when the previous input data frame has not been completed, the signal shall be ignored
INVERSE	In	1	Inverse transformation. When the signal is asserted, the core computes Inverse FFT of the following data frame, otherwise Forward FFT.
REFRESH	In	1	Reload the twiddle coefficient LUTs in the corresponding RAM blocks.
DATAO_IM	Out	DATA_BITS	Imaginary output data
DATAO_RE	Out	DATA_BITS	Real output data
OUTP_READY	Out	1	FFT results are ready. The core asserts the signal when it is about to output a frame of N FFT'ed data. The width of the signal = one clock interval.
DATAO_VALID	Out	1	Output frame is valid. The signal accompanies valid output data frame. Once started, the signal lasts N clock cycles. If the input data are coming continuously with no gaps in between frames, the DATAO_VALID once started will last indefinitely.
OVFLOW_FLAG	Out	1	Arithmetic overflow flag. CoreFFT asserts the flag if the FFT/IFFT computation overflows. The flag starts as soon as the core detects overflow. The flag ends when the current output data frame ends.
RFS	Out	1	Request for start. The core asserts the signal when it is ready for the next input data frame. The signal starts as soon as the core is ready for the next frame. The signal ends when the core gets the requested START signal.
CLK	In	1	Rising-edge clock signal.
CLKEN	In	1	Optional clock enable signal. After deasserting the signal, the core stops generating valid results.
NGRST	In	1	Asynchronous reset signal. Active low.
RST	In	1	Optional synchronous reset signal. Active high.

Note: All signs are active high (logic 1) unless otherwise specified.

Streaming FFT Architecture

CoreFFT implements the area-efficient Streaming Radix-2² Fast Fourier transform. Figure 11 depicts a functional diagram of the 256-point transform.

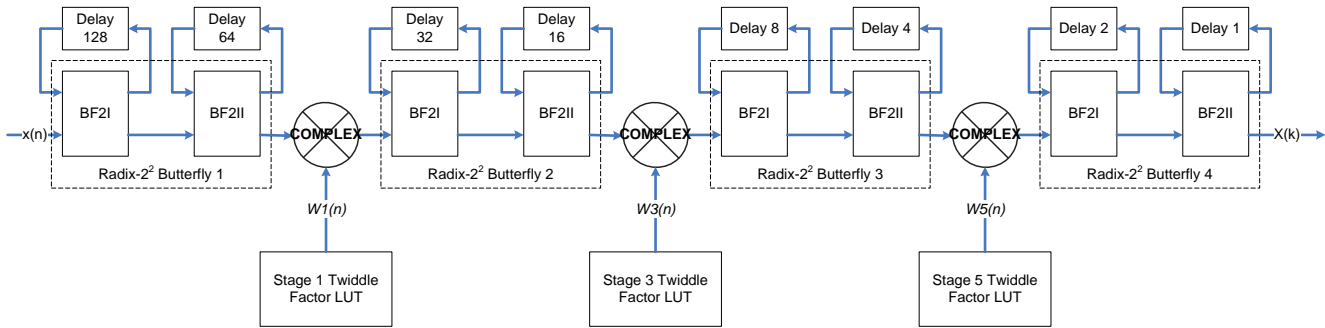


Figure 11 Streaming Radix-2² 256-pt FFT Functional Block Diagram

The input and output data are represented as (2 x DATA_BITS)-bit words comprised of real and imaginary parts. Both parts are two's complementary numbers of DATA_BITS bits each. The module processes frames of data with a frame size equal to the transform size of N complex words. The frame to be processed comes to the x(n) input as a sequence of the complex data words, one (2 x DATA_BITS)-bit word per clock interval. The next frame can start immediately after the last data word of a current frame or at any time later on. Figure 12 on page 20 depicts an example of the frame *i+1* immediately following the frame *i*, and the frame *i+2* coming after an arbitrary gap. The input data samples within a frame should come at every clock interval, thus a frame lasts exactly N clock intervals. There is a substantial latency associated with the streaming algorithm. The output data frames appear at the same order, clock rate, and with the same gaps (if any) between the output frames, as those between the input frames.

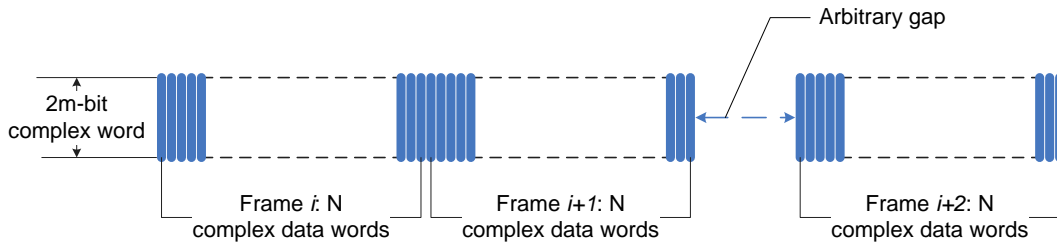


Figure 12 Streaming FFT Input Data Frames

The number of FFT butterflies equals $\log_2(N)$, thus every stage is processed by a separate butterfly. As a result, all stages are processed in parallel.

CoreFFT calculates the twiddle factors required by the FFT algorithm. At power-up, the core automatically uploads the twiddle factors in on-chip RAMs that become the twiddle LUTs. No user action is required to make it happen. Upon completion of the uploading, the core activates the RFS signal, letting a data source know the core is ready to start FFT processing. A user can refresh the LUT contents at any time by issuing a one clock wide signal, REFRESH.

Handshake Signals

RFS and START

The core generates the RFS signal to let a data source know it is ready for the next frame of the input data samples. Once asserted, the RFS will stay active until the data source responds with the START signal. Figure 13 shows a common case, when the FFT engine waits for the data source to supply the START signal. Once the core gets the START, it deasserts the RFS signal and starts receiving the input data frame. After N clock intervals the data frame reception is completed, and the RFS signal goes active again.

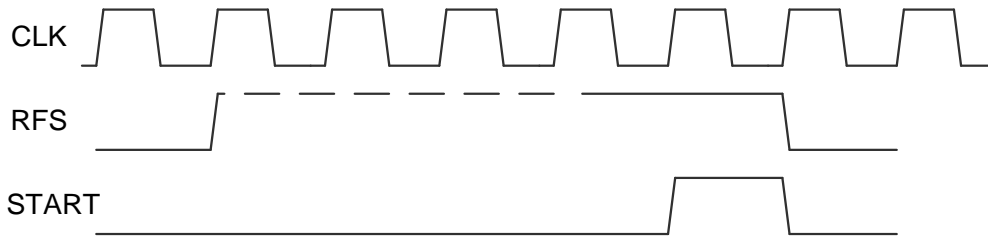


Figure 13 RFS Waits for START

Figure 14 depicts another common case example where the input data are coming indefinitely without gaps between the frames. There the START signal has a permanent active value, and the core starts receiving another input frame right after the end of a previous frame.

It is optional for the data source to watch for the RFS signal. It can assert the START signal at any time, and the core will start accepting another input frame as soon as it can. In the situation of Figure 13 a new frame loading starts immediately after the START signal. If the START signal comes when a previous input frame is being loaded, the core waits until the frame ends and then starts loading another frame.

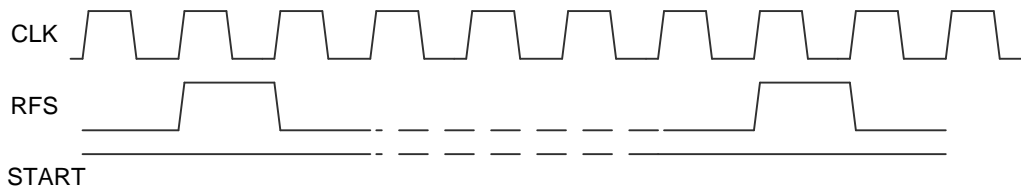


Figure 14 FFTing Streaming Data

The START signal leads the actual input frame by one clock interval (Figure 15).

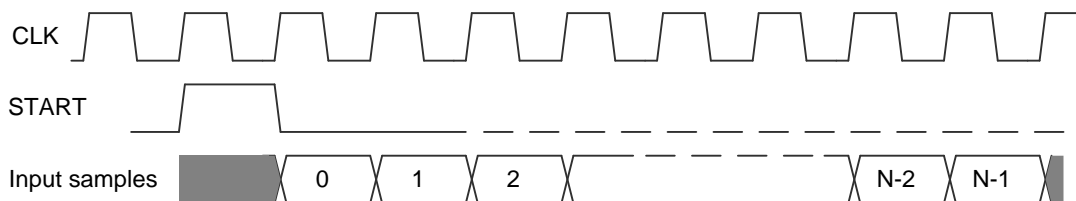


Figure 15 START Leads the Data

OUTP_READY and DATAO_VALID

These two signals serve to notify a data receiver when the FFT results are ready. The OUTP_READY is a clock-wide pulse the core asserts when the output data frame is about to output. The core asserts the DATAO_VALID while generating the output frame. The DATAO_VALID trails the OUTP_READY by one clock interval. Figure 16 depicts the timing relations between the two signals and the FFTed data frame.

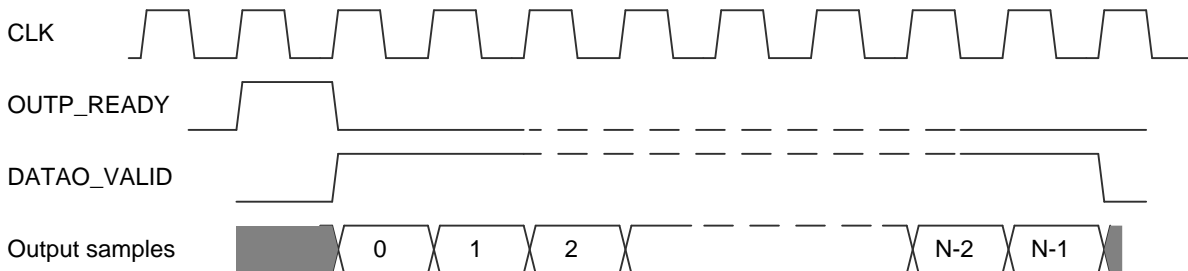


Figure 16 Output Data and Handshake Signals

In the case of streaming data with no gaps between the frames, the DATAO_VALID is permanently active (Figure 17).

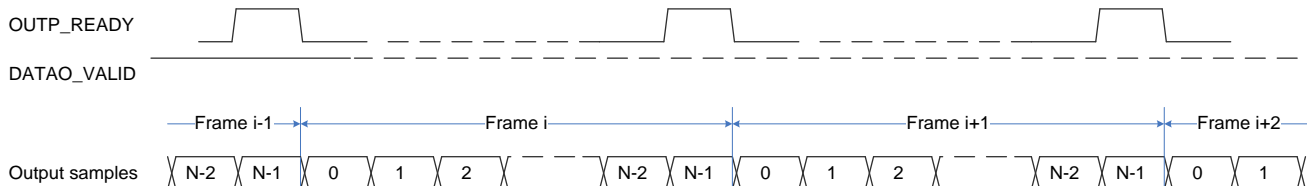


Figure 17 Streaming Output Data without Gaps

FFT Latency

The FFT latency is primarily defined by the transform size, N. The implementation adds up a number of pipeline delays depends somewhat on the datapath bit width. The ordered output latency is substantially longer than the one for the bit-reversed result.

Table 11 presents the output latencies expressed in number of clock cycles.

Table 11 Streaming FFT Latencies

FFT_SIZE	DATA_BITS	TWID_BITS	Bit-Reversed Output Latency	Ordered Output Latency
16	<=18	<=18	28	46
	>18	>18	34	52
	>18	<=18	30	48
	<=18	>18		
32	<=18	<=18	50	84
	>18	>18	62	96
	>18	<=18	54	88
	<=18	>18		
64	<=18	<=18	85	151
	>18	>18	97	163
	>18	<=18	89	155
	<=18	>18		
128	<=18	<=18	155	285
	>18	>18	173	303
	>18	<=18	161	291
	<=18	>18		
256	<=18	<=18	286	544
	>18	>18	304	562
	>18	<=18	292	550
	<=18	>18		
512	<=18	<=18	548	1,062
	>18	>18	572	1,086
	>18	<=18	556	1,070
	<=18	>18		
1024	<=18	<=18	1,063	2,089
	>18	>18	1,087	2,113
	>18	<=18	1,071	2,097
	<=18	>18		

Finite Word Length Considerations

The butterfly calculation involves addition and subtraction. These operations can cause the butterfly data width to grow from input to output. Every butterfly can introduce an extra bit to the data width. Precautions must be taken to ensure that there are no data overflows.

To control risk of overflow, one of two methods or their combination can be employed:

- Set the bit width DATA_BITS large enough to accommodate the bit growth. When using this technique, the input data need to be sign extended to match the DATA_BITS set. This can be achieved by adding the necessary number of leading sign bits to the data. To prevent the FFT engine entirely from scaling the results, set the SCALE parameter to 0 or set all scale schedule bits to 0.
- Use a configurable scale schedule. The technique scales a full butterfly result by shifting it down by one bit if the scale schedule directs it to do so. The technique is fully overflow-safe only when the schedule downscales every butterfly result. But if the nature of the transformed signal is known to be overflow-safe with some or all stages omitting the downscaling, the technique is beneficial both from signal-to-noise ratio and chip resource utilization standpoints.

Every bit of the configurable scale schedule SCALE_SCH controls the corresponding stage butterfly: the least significant schedule bit controls the first stage butterfly, the next bit controls the next stage, and so on. Figure 18 illustrates a difference in the butterfly results when the appropriate scale schedule bit is 0 or 1. An example of 5-bit processing is shown.

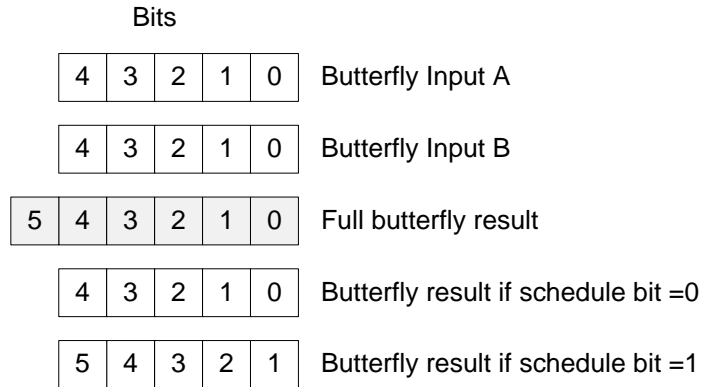


Figure 18 Non-Scaled vs Scaled Down Butterfly Results

The butterfly adds or subtracts two 5-bit numbers. The internal full result is 6-bit wide. If overflow is not expected, the significant bits of the full result fit the 5 bits. Then the SCALE_SCH bit can be set to 0, and the next FFT stage uses the five rightmost bits from 0 to 4. The most significant bit 5 of the full result is discarded.

In the case where the SCALE_SCH bit is set to 1, the core rounds the full result and truncates the LSB. The next FFT stage will use the five leftmost bits from 1 to 5. CoreFFT implements the simple round-half-up rounding algorithm where the rounding is achieved by adding 1 to the bit that is going to be truncated. For the Figure 18 example, the core adds 000001 to the full butterfly result, and then truncates the LSB by one bit, right shifting the rounded result.

The core automatically invokes overflow detection at the stages where the overflow can theoretically take place – the stages not scheduled for scaling down. The overall overflow flag OVFLOW_FLAG appears as soon as the core detects the actual overflow. The flag stays active until the end of an output frame where the overflow was detected.

Tool Flows

Licensing

CoreFFT requires an RTL license to be used and instantiated. Complete source code and a testbench are provided for the core.

SmartDesign

CoreFFT is available for download to the Libero[®] Integrated Design Environment IP Catalog via the web repository. Once it is listed on the catalog, the core can be instantiated using SmartDesign flow. You can configure the core using the configuration GUI within SmartDesign.

For information on using SmartDesign to instantiate and generate cores, refer to the [Using DirectCore in Libero IDE User's Guide](#).

Simulation Flows

The User Testbench for CoreFFT is included in the release.

To run simulations, select the User Testbench flow within SmartDesign and click **Save & Generate** on the Generate pane. The User Testbench is selected through the Core Configuration GUI.

When SmartDesign generates the Libero IDE project, it will install the user testbench files.

To run the user testbench, set the design root to the CoreFFT instantiation in the Libero IDE design hierarchy pane and click the Simulation icon in the Libero IDE Design Flow window. This will invoke ModelSim[®] and automatically run the simulation.

User Testbench

Figure 19 depicts the testbench block diagram. The Golden Behavioral FFT implements the finite precision calculations shown in EQ 1 or EQ 2. Both the Golden FFT and CoreFFT are configured identically and receive the same test signal. The testbench compares the output signals of the Golden module and the actual CoreFFT.

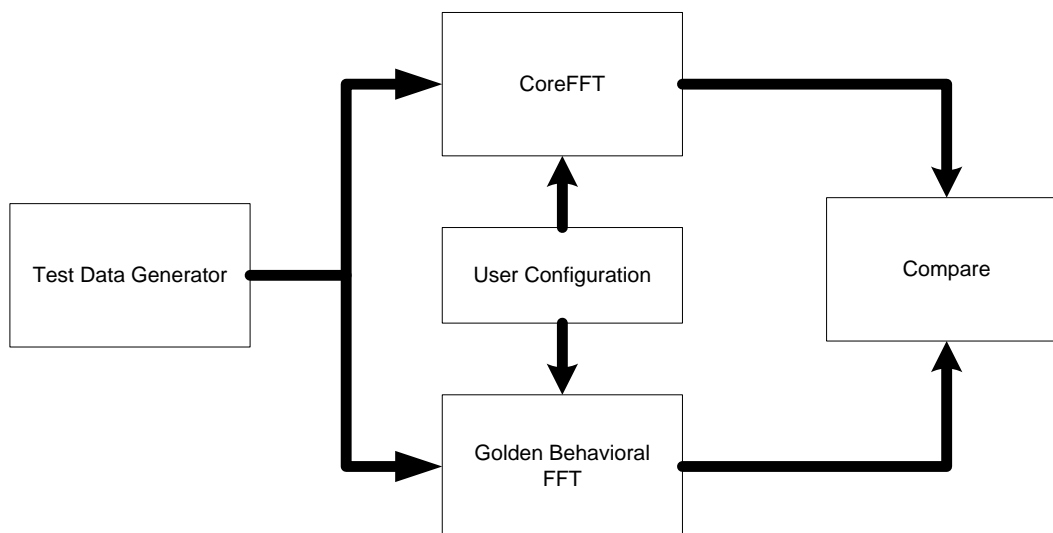


Figure 19 CoreFFT User Testbench

The testbench provides examples of how to use a generated FFT module. You can modify the testbench to suit your needs.

Synthesis in Libero IDE

Having set the design route appropriately, click the **Synthesis** icon in Libero IDE. The Synthesis window appears, displaying the Synplicity[®] project. Set Synplicity to use the Verilog 2001 standard if Verilog is being used. To run Synthesis, select the **Run** icon.

Place-and-Route in Libero IDE

Having set the design route appropriately and run Synthesis, click the **Layout** icon in the Libero IDE to invoke Designer. CoreFFT requires no special place-and-route settings.

Ordering Information

Ordering Codes

CoreFFT can be ordered through your local Sales Representative. It should be ordered using the following number scheme: Core FFT-XX, where XX is listed in Table 12.

Table 12 Ordering Codes

XX	Description
OM	RTL for Obfuscated RTL—multiple use license
RM	RTL for RTL source — multiple-use license

List of Changes

The following table lists critical changes that were made in each revision of the document.

Date	Change
May 2011	CoreFFT v5.0 first release. This release adds a new architecture to the existing In-place CoreFFT v4.0. The new architecture supports Streaming Forward and Inverse FFT that transforms high speed stream of data.
May 2010	CoreFFT v4.0 first release

Product Support

Microsemi backs its products with various support services including Customer Service, a Customer Technical Support Center, a web site, an FTP site, electronic mail, and worldwide sales offices. This appendix contains information about contacting Microsemi SoC Products Group (formerly Actel) and using these support services.

Customer Service

Contact Customer Service for non-technical product support, such as product pricing, product upgrades, update information, order status, and authorization.

From Northeast and North Central U.S.A., call **650.318.4480**
From Southeast and Southwest U.S.A., call **650.318.4480**
From South Central U.S.A., call **650.318.4434**
From Northwest U.S.A., call **650.318.4434**
From Canada, call **650.318.4480**
From Europe, call **650.318.4252** or **+44 (0) 1276 401 500**
From Japan, call **650.318.4743**
From the rest of the world, call **650.318.4743**
Fax, from anywhere in the world **650.318.8044**

Customer Technical Support Center

Microsemi staffs its Customer Technical Support Center with highly skilled engineers who can help answer your hardware, software, and design questions. The Customer Technical Support Center spends a great deal of time creating application notes and answers to FAQs. So, before you contact us, please visit our online resources. It is very likely we have already answered your questions.

Technical Support

Visit the Customer Support website (<http://www.actel.com/support/search/default.aspx>) for more information and support. Many answers available on the searchable web resource include diagrams, illustrations, and links to other resources on website.

Website

You can browse a variety of technical and non-technical information on the SoC home page, at <http://www.actel.com/>.

Contacting the Customer Technical Support Center

Highly skilled engineers staff the Technical Support Center from 7:00 A.M. to 6:00 P.M., Pacific Time, Monday through Friday. Several ways of contacting the Center follow:

Email

You can communicate your technical questions to our email address and receive answers back by email, fax, or phone. Also, if you have design problems, you can email your design files to receive assistance. We constantly monitor the email account throughout the day. When sending your request to us, please be sure to include your full name, company name, and your contact information for efficient processing of your request.

The technical support email address is soc_tech@microsemi.com.

Phone

Our Technical Support Center answers all calls. The center retrieves information, such as your name, company name, phone number and your question, and then issues a case number. The Center then forwards the information to a queue where the first available application engineer receives the data and returns your call. The phone hours are from 7:00 A.M. to 6:00 P.M., Pacific Time, Monday through Friday. The Technical Support numbers are:

650.318.4460

800.262.1060

Customers needing assistance outside the US time zones can either contact technical support via email (soc_tech@microsemi.com) or contact a local sales office. [Sales office listings](#) can be found at www.actel.com/company/contact/default.aspx.



Microsemi Corporate Headquarters
2381 Morse Avenue, Irvine, CA 92614
Phone: 949.221.7100 · Fax: 949.756.0308
www.microsemi.com

Microsemi Corporation (NASDAQ: MSCC) offers the industry's most comprehensive portfolio of semiconductor technology. Committed to solving the most critical system challenges, Microsemi's products include high-performance, high-reliability analog and RF devices, mixed signal integrated circuits, FPGAs and customizable SoCs, and complete subsystems. Microsemi serves leading system manufacturers around the world in the defense, security, aerospace, enterprise, commercial, and industrial markets. Learn more at www.microsemi.com.

© 2011 Microsemi Corporation. All rights reserved. Microsemi and the Microsemi logo are trademarks of Microsemi Corporation. All other trademarks and service marks are the property of their respective owners.