HB0267

CoreFFT v7.0 Handbook

12 2016

Microsemi
Power Matters.™

About Microsemi

Microsemi Corporation (Nasdaq: MSCC) offers a comprehensive portfolio of semiconductor and system solutions for aerospace & defense, communications, data center and industrial markets. Products include high-performance and radiation-hardened analog mixed-signal integrated circuits, FPGAs, SoCs and ASICs; power management products; timing and synchronization devices and precise time solutions, setting the world's standard for time; voice processing devices; RF solutions; discrete components; enterprise storage and communication solutions; security technologies and scalable anti-tamper products; Ethernet solutions; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Microsemi is headquartered in Aliso Viejo, Calif., and has approximately 4,800 employees globally. Learn more at www.microsemi.com.

Microsemi Corporate Headquarters
One Enterprise, Aliso Viejo,
CA 92656 USA
Within the USA: +1 (800) 713-4113
Outside the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996
E-mail: sales.support@microsemi.com
www.microsemi.com

1 Revision History

The revision history describes the changes that were implemented in the document. The changes are listed by revision, starting with the most current publication.

1.1 Revision 9.0

Removed Product Support section.

1.2 Revision 8.0

Updated changes related to CoreFFT v7.0 release.

1.3 Revision 7.0

Updated changes related to CoreFFT v6.4 release.

1.4 Revision 6.0

Updated changes related to CoreFFT v6.3 release.

1.5 Revision 5.0

Updated changes related to Supported Families section (SAR 47942).

1.6 Revision 4.0

Updated changes related to CoreFFT v6.1 release.

1.7 Revision 3.0

Updated changes related to CoreFFT v6.0 release. The release adds support for SmartFusion2 family (In-Place architecture only).

1.8 Revision 2.0

Updated changes related to CoreFFT v5.0 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.

1.9 Revision 1.0

Revision 1.0 was the first publication of this document. Created for CoreFFT v4.0.

Contents

List of Figures

List of Tables

2 Introduction

2.1 Overview

The Microsemi® fast Fourier transform (FFT) core implements the efficient Cooley-Tukey algorithm for computing the discrete Fourier transform. CoreFFT is used in a broad range of applications such as digital communications, audio, measurements, control, and biomedical. CoreFFT provides highly parameterizable, area-efficient, and high performance MACC-based FFT. The core is available as a register transfer level (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),…, x(N-1) is defined in the following equation:

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

where k = 0, 1… N-1

The N-point inverse FFT (N is a power of 2) of a sequence X(0), X(1),…, X(N-1) is defined in the following equation:

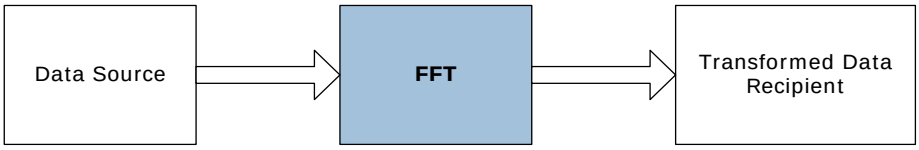
$$x(n) = \sum_{k=0}^{N-1} X(k) e^{jnk2\pi/N}$$

where n = 0, 1… 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



2.2 Features

CoreFFT supports the following two transform implementations:

- Radix-2 decimation-in-time in-place FFT
- Radix-2^2 decimation-in-frequency streaming FFT

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

Table 1 Key Features Support

Feature	In-Place	Streaming
Transform sizes, points	32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, and 16384. **Note:** The 16384-pt FFT is supported on RTG4 and PolarFire parts only	16, 32, 64, 128, 256, 512, and 1024
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 complement	Two's complement
Natural output sample order	Yes	Optional
Conditional block floating point scaling	Yes	No
Pre-defined scaling schedule or no scaling	No	Yes
Optional minimal or buffered memory configurations	Yes	No
Embedded RAM-block based twiddle look-up table (LUT)	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

2.3 Core Version

This handbook is for CoreFFT version 7.0.

2.4 Supported Families

- PolarFire
- RTG4™
- IGLOO®2
- SmartFusion®2

2.5 Device Utilization and Performance

CoreFFT has been implemented in the SmartFusion2 M2S050 device using speed grade –1 and PolarFire MPF300 using speed grade -1. A summary of the implementation data is provided in Table 2 through Table 8.

2.5.1 In-Place FFT

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

Table 2 In-Place FFT SmartFusion2 M2S050 Device Utilization and Performance (Minimal Memory Configuration)

Core Parameters		Fabric Resource Usage			Blocks		Performance	
POINTS	WIDTH	DFF	4LUT	Total	LSRAM	MACC	Clock Rate	FFT Time (µs)
256	18	972	938	1,910	3	4	352	3.1
512	18	1,007	1,239	2,246	3	4	352	6.8
1024	18	1,035	1,758	2,793	3	4	336	15.5
4096	18	1,118	3,468	4,586	12	4	336	73.5

Table 3 In-Place FFT SmartFusion2 M2S050 Device Utilization and Performance (Buffered Configuration)

Core Parameters		Fabric Resource Usage			Blocks		Performance	
POINTS	WIDTH	DFF	4LUT	Total	LSRAM	MACC	Clock Rate	FFT Time (µs)
256	18	1,096	1,098	2,194	7	4	327	3.4
512	18	1,137	1,411	2,548	7	4	311	7.7
1024	18	1,171	1,916	3,087	7	4	315	16.6
4096	18	1,260	3,666	4,926	28	4	309	79.9

Notes:

- *Data in Table 2and Table 3 were obtained using typical synthesis settings. The Synplify frequency (MHz) was set to 500.*
- *Layout settings were as follows:*
  Designer block creation enabled
  High Effort Layout enabled
- *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 PolarFire MPF300 Device Utilization and Performance (Minimal Memory Configuration)

Core Parameters			Fabric Resource Usage					Max Clock Frequency	Transform Time (uS)
POINTS	WIDTH	uRAM Depth	4LUT	DFF	uRAM	LSRAM	MACC		
64	18	512	1072	1278	9	0	4	428	0.6
128	18	512	1162	1310	9	0	4	428	1.2
256	18	512	1562	1524	18	0	4	428	2.6
512	18	0	1613	1321	0	3	4	383	6.3
512	25	0	2542	2908	0	6	16	376	6.5
1024	25	0	3116	2949	0	6	16	371	14.2
4096	18	0	4192	1806	0	12	4	363	68.0
4096	25	0	6312	3406	0	15	16	362	68.4
16384	18	0	9506	3550	0	54	4	328	350.1
16384	25	0	16869	5820	0	75	16	284	404.6

Table 5 In-Place FFT PolarFire MPF300 Device Utilization and Performance (Buffered Configuration)

Core Parameters			Fabric Resource Usage					Max Clock Frequency	Transform Time (uS)
POINTS	WIDTH	uRAM Depth	4LUT	DFF	uRAM	LSRAM	MACC		
64	18	512	1371	1591	21	0	4	428	0.6
256	18	512	2195	2101	42	0	4	428	2.6
512	18	512	3008	2946	84	0	4	396	6.0
1024	18	512	5092	4571	168	0	4	329	15.9
16384	18	0	12269	6301	0	126	4	290	396.0

Notes:

- *Data in Table 4 and Table 5 were obtained using typical Libero SoC tool settings. The Timing constraint was set to 400 MHz.*
- *Place and Route was set for Timing-driven High Effort Layout*
- *The FFT time shown reflects the transformation time only. It does not account for data downloading or result uploading times.*

Table 6 In-Place FFT Utilization and Performance Configuration

Parameter	Value
INVERSE	0
SCALE	0
SCALE_EXP_ON	0
HDL type	Verilog

2.5.2 Streaming FFT

Table 7 and Table 8 show the utilization and performance for a variety of streaming FFT configurations.

Table 7 Streaming FFT SmartFusion2 M2S050T Speed grade -1

Core Parameters				Resource Usage			Blocks			Clock Rate
FFT_SIZE	DATA_BITS	TWID_BITS	Order	DFF	4LUT	Total	LSRAM	uRAM	MACC	
16	18	18	Reverse	1,576	1,446	3,022	-	4	4	290
16	18	18	Normal	1,740	1,621	3,361	-	8	4	290
32	18	18	Reverse	2,276	2,036	4,312	-	8	8	297
64	18	18	Reverse	2,725	2,560	5,285	-	10	8	286
128	18	18	Reverse	3,407	3,302	6,709	-	12	12	292
256	18	18	Reverse	4,059	3,848	7,907	2	12	12	284
256	18	18	Normal	4,219	3,956	8,175	4	12	12	287
256	24	25	Reverse	7,663	6,551	14,214	4	18	48	281
512	18	18	Reverse	4,513	5,251	9,764	4	12	16	292
512	18	24	Reverse	5,687	6,554	12,241	6	14	32	296
1,024	24	24	Reverse	10,017	10,828	20,845	11	18	64	260
1,024	24	25	Reverse	10,053	10,961	21,014	11	18	64	260

Notes:

- *uRAM max depth was set at 64*
- *Data in Table 7 were achieved using typical synthesis settings. The Synplify frequency (MHz) was set to 500.*
- *Layout High effort mode was set*

Table 8 Streaming FFT PolarFire MPF300 Speed grade -1

Core Parameters						Resource Usage					Clock Rate
FFT_SIZE	DATA_BITS	TWID_BITS	SCALE	uRAM Depth	Order	4LUT	DFF	uRAM	LSRAM	MACC	
16	16	18	On	256	Reverse	1229	1614	6	0	4	410
16	16	18	On	256	Normal	1350	1721	12	0	4	428
32	16	18	On	256	Reverse	1795	2176	12	0	4	428
64	16	18	On	256	Reverse	2325	2769	15	0	8	413
128	20	18	On	256	Reverse	4003	4687	28	0	24	415
256	22	18	Off	256	Reverse	5441	6063	46	0	24	428
256	24	25	Off	256	Reverse	6980	8641	60	0	48	428
512	24	25	On	256	Reverse	8750	9780	15	14	64	414
1024	24	25	On	0	Reverse	11283	10826	0	21	64	394
1024	24	25	Off	64	Reverse	11378	11106	32	11	64	417

Notes:

- *Data in Table 8 were obtained using typical Libero SoC tool settings. The Timing constraint was set to 400 MHz.*
- *Place and Route was set for Timing-driven High Effort Layout*

3 Functional Description

3.1 Architecture Options

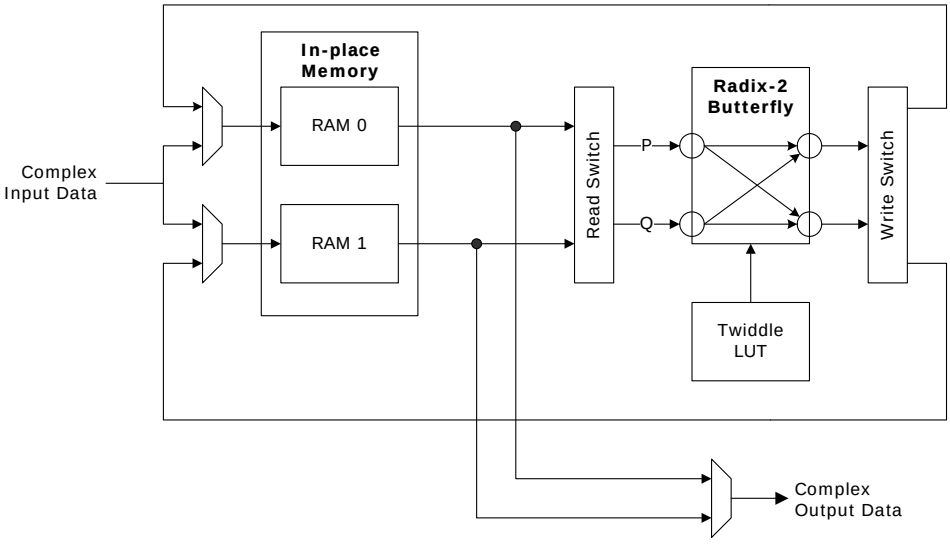
Depending on user configuration, CoreFFT generates one of the following transformation implementations:

- In-Place FFT
- Streaming FFT

3.2 In-Place FFT

The architecture option loads a frame of N complex data samples in its in-place RAM, and processes them sequentially, stage by stage using a single Radix-2 processor. 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. Figure 2 shows a functional diagram of the in-ilace transform.

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



The input and output data are represented as 2*WIDTH-bit words comprised of real and imaginary parts. Both parts are two's complement 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 the in-place memory. The memory contains two identical RAM blocks, each is capable of storing N/2 complex words. The in-place memory supports double bandwidth, it can read and write two complex words at the same time. Once the N complex data samples are loaded in the memory, FFT computation starts automatically and the in-place memory is used for the computations.

The in-place FFT computational process occurs in a sequence of stages with the number of stages equal to log2N. At every stage of the FFT data processing, 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 2) helps the butterfly to obtain stored data in the order required by the FFT algorithm. In addition to the data, the butterfly obtains twiddle factors (sine/cosine coefficients) from the twiddle LUT. The butterfly writes intermediate results back to the in-place memory through the write switch.

After the last computational stage, the in-place memory stores the fully transformed data. The module puts out 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 NGRST is asserted.

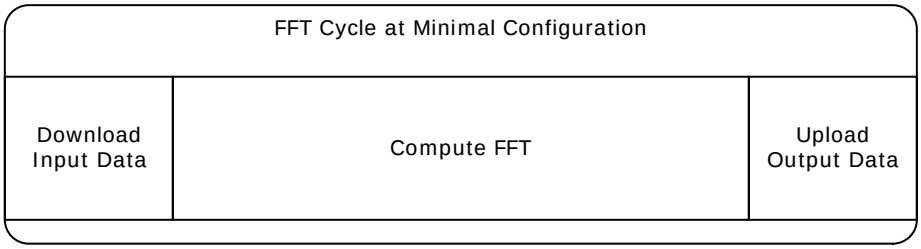
In-Place Memory Buffers

Minimal Configuration

The minimal configuration, as shown in Figure 2, 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 3 shows the FFT cycle timeline. The cycle consists of the following three phases:

1. Download a fresh input data frame in the in-place RAM
2. Perform the actual transformation
3. Upload the transformation result to free up the in-ilace RAM

Figure 3 Minimal Configuration In-Place FFT Cycle

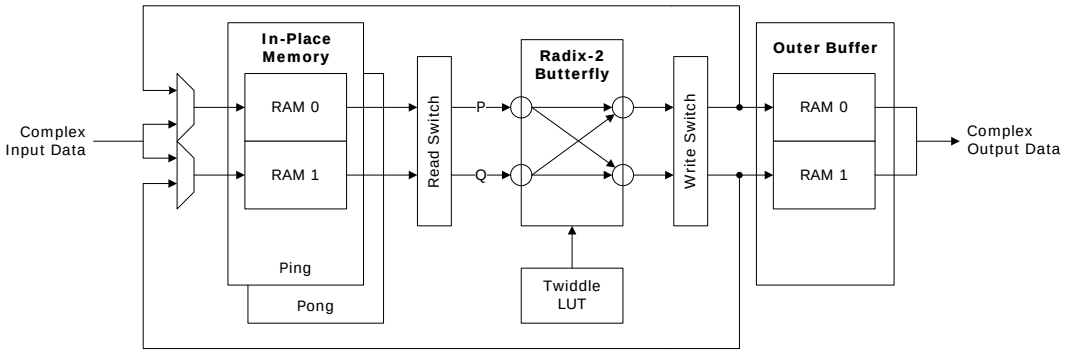


In the minimal configuration, the butterfly runs only during the computation phase. When the data burst rate permits, the minimal configuration provides the best device resource utilization. In particular, it saves a significant number of RAM blocks.

Buffered Configuration

In order to improve the butterfly utilization and consequently reduce the average transformation time, additional memory buffers can be used. Figure 4 shows the buffered FFT block diagram.

Figure 4 Buffered FFT Block Diagram



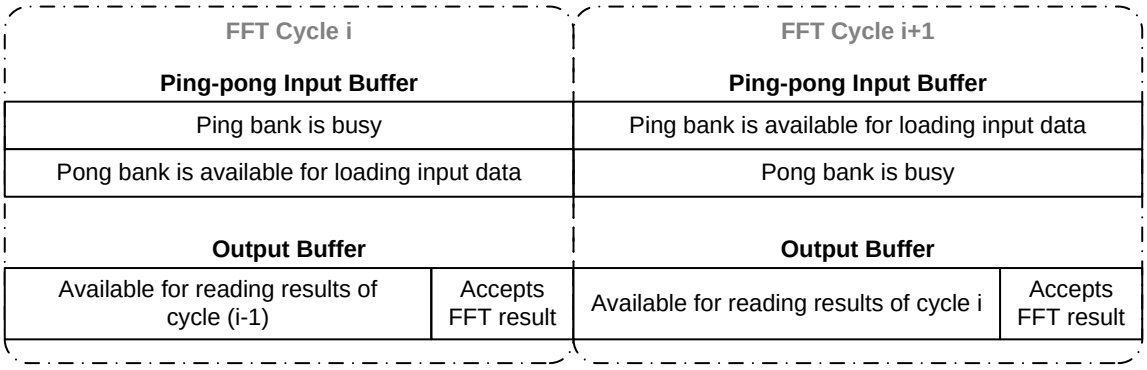
The buffered option has two identical in-place memory banks implementing a ping-pong buffer and one output buffer. Each bank is capable of storing N complex words and reading two complex words at a time. The core state machine controls the ping-pong switching so that a data source sees only a buffer that is ready to accept new data. The buffer that does not accepts 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 the 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 replaces them with 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 5.

Figure 5 Buffered Configuration FFT Cycles



Finite Word Length Considerations

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. The butterfly calculation involves complex multiplication, addition, and subtraction. The 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

Input Data Scaling: 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.

Unconditional Block Floating-Point Scaling: 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.

The input data need to be scaled down by a factor of two to prevent overflow at the first stage. To prevent the overflow in successive stages, the core scales down the results of every previous stage 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 lost bits as in the input data scaling. However, it produces more precise results, as the FFT engine starts with more precise input data.

Conditional Block Floating-Point Scaling: 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 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 calculated actual factor appears on the SCALE_EXP port. The factor 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; that is, divided by 2^{SCALE_EXP} = 16. The signal accompanies the FFT results and is valid while OUTP_READY is asserted. To scale back the actual CoreFFT results, that is, to make them comparable to floating point transformed bins, every FFT output sample needs to be multiplied by 2^{SCALE_EXP}:

$$\mathbf{FFT\ Result\ (Real) = DATAO\_RE*2^{SCALE\_EXP}}$$
$$\mathbf{FFT\ Result\ (Imaginary) = DATAO\_IM*2^{SCALE\_EXP}}$$

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

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

Transformation Time

The FFT computation takes (N/2 + L) x $\log_2$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 is equal to 10 at bit resolutions 8 to 18, and L is equal to 16 at bit resolutions from 19 to 32.
For example, for a 256-point 16-bit FFT, the computation time = (256/2 + 10) x $\log_2$256 + 2 = 1,106 clock periods. For a 4,096-point 24-bit FFT, the computation time = (4096/2 + 16) x $\log_2$4096 + 2 = 24,770 clock periods.

Memory Implementation

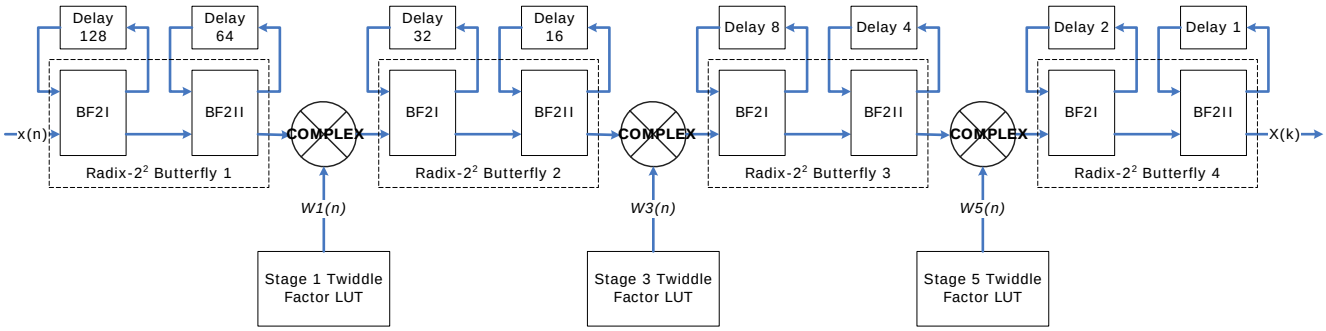
The core uses hard RAM blocks to implement the in-place memory, other memory buffers, and a twiddle LUT. The FPGAs carry two hard RAM types: large (LSRAM) and micro-RAMs. The memory implementation can be controlled by setting the URAM_MAXDEPTH parameter. CoreFFT uses micro-RAMs if the required depth does not exceed the parameter value. For example, the URAM_MAXDEPTH set to 64 utilizes micro-RAMs in any FFT size up to 128 points, as the required depth is POINTS/2. Setting the parameter value to 0 prevents the core from using the micro-RAMs at all, so that they can be used elsewhere.

The parameter URAM_MAXDEPTH is accessible through the core user interface.

3.3 Streaming FFT

Streaming FFT supports continuous complex data processing, one complex input data sample per clock period. The streaming architecture has as many Radix-2^2 processors, RAM blocks, and LUT's as necessary to support streaming data transformation. Figure 6 shows a functional diagram of the 256-point streaming transform.

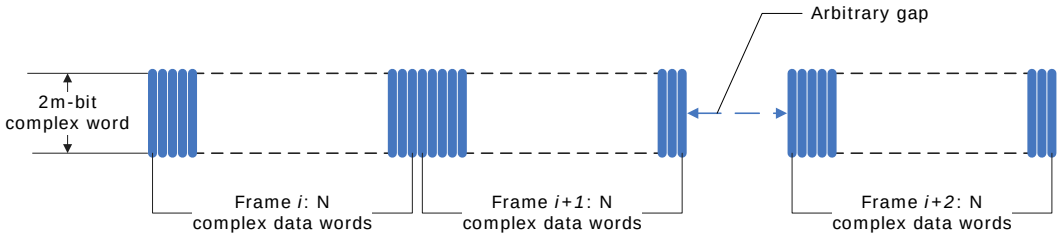
Figure 6 Streaming Radix-22 256-pt FFT Functional Block Diagram



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 7 shows an example of frame *i+1* immediately following frame *i*, and the frame *i+2* coming after an arbitrary gap. The input data samples within a frame must come at every clock interval, thus a frame lasting 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 7 Streaming FFT Input Data Frames



The number of FFT butterflies equals $\log_2(N)$, thus every stage being 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. User action is not required to make it happen. Upon completion of the uploading, the core activates the

RFS signal, letting a data source know that the core is ready to start FFT processing. The LUT contents can be refreshed at any time by issuing a one clock wide signal, REFRESH.

Streaming FFT Latency

The streaming FFT latency is primarily defined by the transform size, N. The implementation adds up a number of pipeline delays that depend on the FFT size and data path bit width. In other words the FFT results are delayed regarding the input data by not less than N data intervals for the bit-reversed outputs. The ordered output latency is about two times larger.

Streaming FFT Memory Implementation

Similarly to the in-place architecture, the streaming FFT uses hard RAM blocks to implement the required memories, LUTs and delay lines. The memory implementation can be controlled by setting the URAM_MAXDEPTH parameter. CoreFFT uses micro RAMs if the depth of the memory does not exceed the parameter value. For example, the URAM_MAXDEPTH parameter set to 128 utilizes micro-RAMs to create memories of depth of 128 and less. Setting the parameter value to 0 prevents the core from using the micro RAMs at all, so that they can be used elsewhere.

Finite Word Length Considerations

Unscaled and Scale Schedule Modes

The butterfly calculation involves addition and subtraction. These operations can cause the butterfly data width to grow from input to output. Every butterfly, BF2I or BF2II (see Figure 6), can introduce an extra bit to the data width. In addition, the multiplications can add one bit to the result. The overall potential bit growth = $\log_2(N)+1$ bits. Precautions must be taken to ensure that there are no data overflows.

To avoid or reduce a risk of overflow, the core employs one of two techniques:

- Unscaled mode builds data path wide enough to accommodate the bit growth. The data path width grows from stage to stage to fully accommodate the algorithm bit growth so that data overflow never happens. The real or imaginary output bit width is $\log_2(N)+1$ bits wider than the input one. The design is entirely safe from the overflow point of view.
- Configurable scale schedule. This technique provides a user with control over scaling down (truncation of) every intermediate result that can cause overflow. The output bit width equals the input bit width. The technique is overflow-safe only when the scaling schedule matches the actual bit growth, which is not easy to achieve. Cautious approach to the configurable scaling often leads to extra down scaling. But if the nature of the transformed signal is known to be overflow-safe with some or all stages omitting the extensive downscaling, the technique is beneficial both from signal-to-noise ratio and chip resource utilization standpoints. When configured for the scale schedule mode, the core generates an overflow flag if the overflow happened.

The Radix-2^2 butterfly can introduce 3-bit growth: butterflies BF2I, BF2II and a multiplier each can add a bit. But only one multiplication out of all FFT stages can add the bit. Since it is unknown upfront, at which stage the multiplier induces the extra bit if any, the FFT engine in the unscaled mode extends the data path by the bit starting at the first stage.

In the scale schedule mode, again every Radix-2^2 stage can introduce 3-bit growth. The data path within the stage grows accordingly, that is, the stage output is three bits wider than the stage input. The engine cuts out the three extra bits after the stage result is calculated, that is, the stage output gets truncated by three bits before it goes to the next stage. Such approach eliminates the need of guessing the sub-stage at which downscaling needs to be applied.

Table 9 explains the three bits that get cut out in the scale schedule mode depending on the 2-bit schedule value for a particular stage.

Table 9 Cutting Out Three Extra Bits in Scale Schedule Mode

Scale Schedule for a Given Radix-2^2 Stage	Bits the Core Cuts Out
00	Cut out three MSB's
01	Cut out two MSB's and round one LSB
10	Cut out one MSB and round two LSB's
11	Round three LSB's

The FFT/IFFT of the sizes 32, 128, or 512 that are not a power-of-four, in addition to the Radix-2^2 butterflies, utilize a single Radix-2 butterfly. The one applies to the last processing stage and cuts out a single extra bit.

The core automatically invokes overflow detection in the scale schedule mode. The 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.

Unscaled Mode Input Bit Width Limitations

The unscaled mode limits the maximal input sample bit width handled by the core. Table 10 lists the max bit widths for every FFT size.

Table 10 Streaming Unscaled FFT Max Input Data Bit Width

FFT Size	Max Input Width
16	32
32	30
64	30
128	28
256	28
512	26
1024	26

Entering Scale Schedule

The scale schedule identifies the downscaling factor for every streaming FFT stage. Every Radix-2^2 stage scaling factor is controlled by dedicated two bits of the scale schedule, and the Radix-2 stage used in the non-power-of-four FFTs is controlled by a single bit. Figure 8 depicts an example of a scale schedule user interface for 1024-pt FFT. A pair of checkboxes corresponds to a specific Radix-2^2 stage and presents two bits of the downscaling factor. The actual downscaling factor at a particular stage is calculated as $2^{2*Bit1+Bit0}$ and takes one of the following values: 1, 2, 4, 8. The checkboxes shown in the Figure 8 correspond to the binary scale schedule value of 10 10 10 10 11. This value presents a conservative scale schedule that does not cause the overflow.

Figure 8 Scale Schedule User Interface

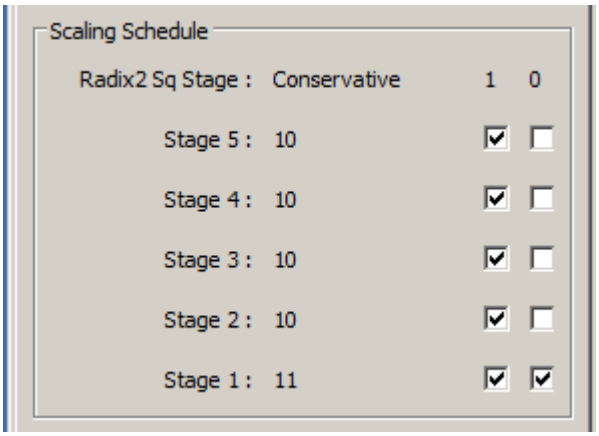


Table 11 shows the conservative scale schedules for every FFT size that is completely overflow safe.

Table 11 Conservative Scale Schedules for Various FFT Sizes

FFT Size	Radix-2^2 Stage									
	4		3		2		1		0	
1024	1	0	1	0	1	0	1	0	1	1
512	x	1	1	0	1	0	1	0	1	1
256	x	x	1	0	1	0	1	0	1	1
128	x	x	x	1	1	0	1	0	1	1
64	x	x	x	x	1	0	1	0	1	1
32	x	x	x	x	x	1	1	0	1	1
16	x	x	x	x	x	x	1	0	1	1

3.4 Natural Output Order

The output results obtained from the Radix-2 and Radix-2^2 FFT algorithms are in the bit-reversed order. The in-place implementation however, internally performs the sample ordering. Therefore, the core puts the results out in a natural order. The Streaming FFT supports both bit-reversed and natural output orders. The bit-reversed option utilizes fewer chip resources and provides smaller latency.

4 Interface

4.1 In-Place FFT

4.1.1 Configuration Parameters

CoreFFT has parameters (Verilog) or generics (VHDL) for configuring the RTL code. These parameters and generics are described in Table 12. All parameters and generics are integer types.

Table 12 In-Place CoreFFT Parameter Descriptions

Parameter	Valid Range	Default	Description
INVERSE	0 – 1	0	0: Forward Fourier transform 1: Inverse Fourier transform
SCALE	0 – 1	0	0: Conditional block floating point scaling 1: Unconditional block floating point scaling To apply the input data scaling, 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 have no effect.
POINTS	32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384	256	Transform size. **Note:** the 16384-pt FFT is supported on RTG4 and PolarFire parts only
WIDTH	8 – 32	18	Data and twiddle factor bit width.
MEMBUF	0 – 1	0	0: Minimal (no buffer) configuration 1: Buffered configuration
SCALE_EXP_ON	0 – 1	0	0: Does not build the conditional block floating-point exponent calculator 1: Builds the calculator
URAM_MAXDEPTH	0, 4, 8, 16, 32, 64, 128, 256, 512		The largest RAM depth to be implemented with the micro-RAM available on the SmartFusion2, IGLOO2, RTG4 and PolarFire parts. When the RAM depth required for a user-selected transform size POINTS exceeds the URAM_MAXDEPTH, large LSRAM blocks are used.

4.1.2 Ports

Figure 9 shows the port signals for the in-place CoreFFT architecture. Table 13 describes the port signals.

Figure 9 CoreFFT I/O Ports

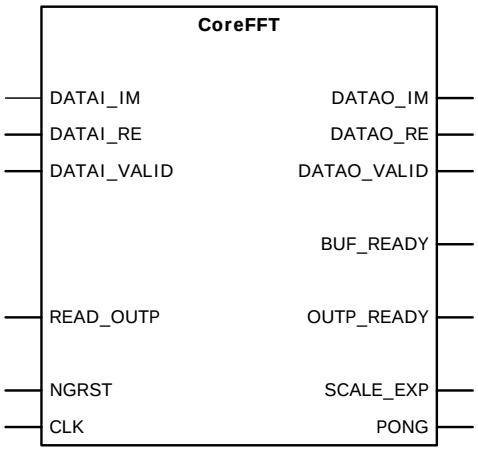


Table 13 In-Place CoreFFT Port Descriptions

Port Name	In/Out	Port Width Bits	Description
DATAI_IM	In	WIDTH	Imaginary input data to be transformed
DATAI_RE	In	WIDTH	Real input data to be transformed
DATAI_VALID	In	1	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 the BUF_READY signal has been asserted.
READ_OUTP	In	1	Read transformed data. Normally the module puts out 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	Out	WIDTH	Imaginary output data
DATAO_RE	Out	WIDTH	Real output data
DATAO_VALID	Out	1	Output complex word valid. The signal accompanies valid output complex words present on DATAO_IM and DATAO_RE outputs.
BUF_READY	Out	1	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	1	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 the READ_OUTP signal is deasserted.
SCALE_EXP	Out	1	Conditional block floating-point scaling exponent. This optional output can be enabled by setting the SCALE_EXP_ON parameter. The output can be enabled when the core is in conditional block floating-point scaling mode only (the parameter SCALE = 0).

Port Name	In/Out	Port Width Bits	Description
PONG	Out	1	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	1	Clock. Rising edge active. The core master clock.
NGRST	In	1	Asynchronous reset. Active low.

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

4.2 Streaming FFT

4.2.1 Configuration Parameters

CoreFFT has parameters (Verilog) or generics (VHDL) for configuring the RTL code. These parameters and generics are described in Table 14. All parameters and generics are integer types.

Table 14 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	1 – Enable configurable scale schedule. When the option is enabled, the core applies the configurable scale factor, SCALE_SCH after every butterfly. 0 – Unscaled mode
SCALE_SCH		0	Scale schedule. If the SCALE_ON parameter equals 1, SCALE_SCH is used to define the scaling factor for every processing stage.
DATA_BITS	8 – 32	18	Input data bit width of real or imaginary parts.
TWID_BITS	8 – 32	18	Twiddle factor bit width of its real or imaginary parts.
ORDER	0 – 1	0	0: Output data in bit-reversed order 1: Output data in normal order
URAM_MAXDEPTH	0, 4, 8, 16, 32, 64, 128, 256, 512	0	The largest RAM depth to be implemented with micro-RAM available on the SmartFusion2, IGLOO2, RTG4 or PolarFire parts. When the RAM depth required for a user-selected transform size POINTS exceeds the URAM_MAXDEPTH, large LSRAM blocks will be used

4.2.2 Ports

The port signals for the Streaming CoreFFT macro are shown in Figure 10 and described in Table 15.

Figure 10 Streaming FFT I/O Ports

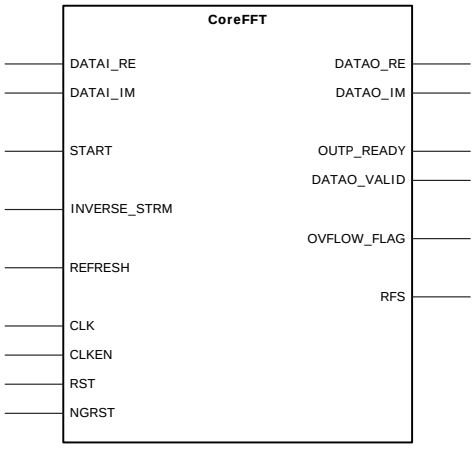


Table 15 Streaming FFT I/O Signal Descriptions

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 it comes when the previous input data frame has not been completed, the signal will 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 FFTed data. The width of the signal is one clock interval.
DATAO_VALID	Out	1	Output frame is valid. Accompanies valid output data frame. After it is started, the signal lasts N clock cycles. If the input data are coming continuously with no gaps in between frames, the DATAO_VALID 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 signals are active high (logic 1) unless otherwise specified.

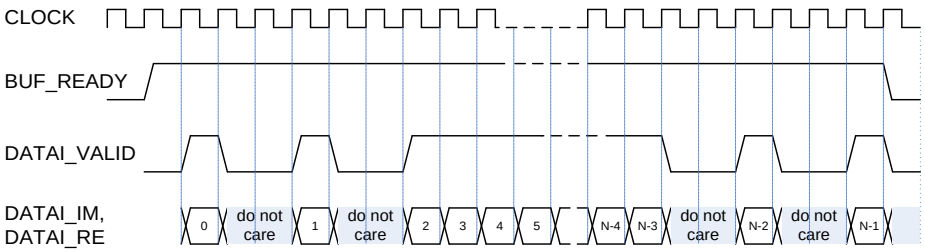
5 Timing Diagrams

5.1 In-Place FFT

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 must be supplied simultaneously and accompanied with the validity bit DATAI_VALID. The data source can supply the sample at every clock cycle or at an arbitrary slower rate (refer to Figure 11). Once the FFT module receives N input samples, it lowers the BUF_READY signal.

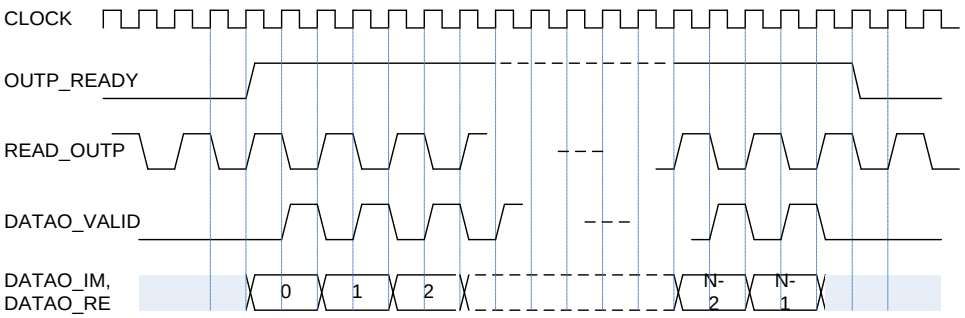
The FFT engine starts processing the data automatically after it is ready. In the minimal memory configuration, the processing phase starts immediately after data loading is complete. In the buffered configuration, the FFT engine can wait until a previous data burst is processed. Then the engine starts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

Figure 11 Loading Input Data



Upon completing the transformation, the FFT module asserts the OUTP_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 keeps providing data output while the READ_OUTP signal is asserted. To control the output sample rate, the receiver must deassert the READ_OUTP signal as and when needed (as shown in Figure 12).

Figure 12 Receiving Transformed Data



When using the READ_OUTP signal to control reading rate, possible FFT cycle growth needs to be considered. In the minimal memory configuration, any prolongation of the read (upload) time extends the FFT cycle as shown in Figure 3. In the buffered configuration, the FFT cycle grows when the actual upload time exceeds the dedicated interval shown in Figure 4 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.

5.2 Streaming FFT

5.2.1 RFS and START

The core generates the RFS signal to let a data source know that it is ready for the next frame of the input data samples. After it is asserted, the RFS stays active until the data source responds with the START signal.

Figure 13 shows an example 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

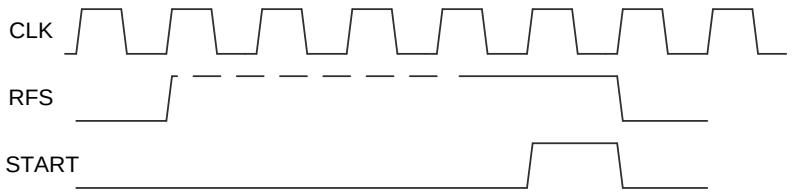
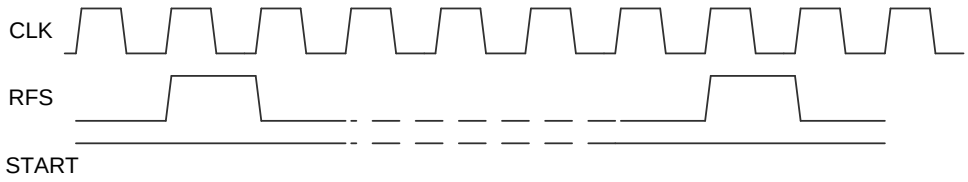


Figure 14 shows another example where the input data come indefinitely without gaps between the frames. The START signal has a permanent active value, and the core starts receiving another input frame right after the end of a previous frame.

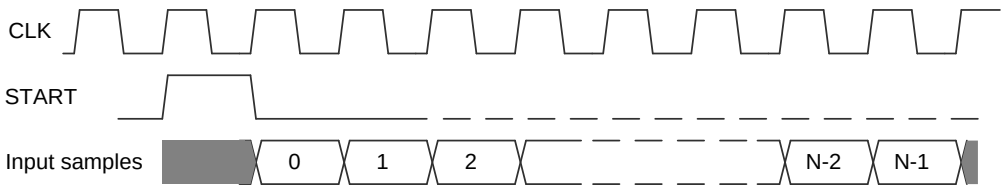
It is optional for the data source to watch for the RFS signal. It can assert the START signal at any time, and the core starts accepting another input frame as soon as it can. In the situation of the Figure 13, a new frame loading starts immediately after the START signal. If the START signal comes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

Figure 14 Transforming Streaming Data



The START signal leads the actual input frame by one clock interval, as shown in Figure 15.

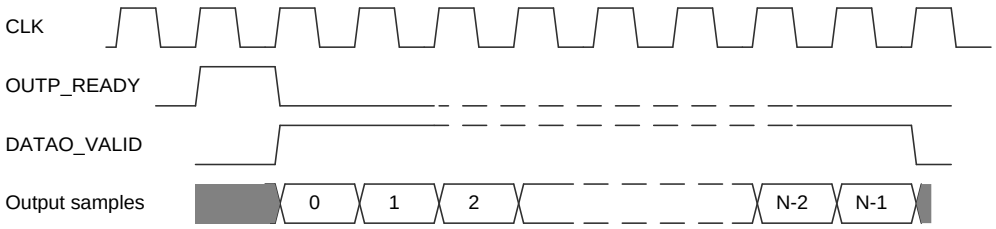
Figure 15 START Leads the Data



5.2.2 OUTP_READY and DATAO_VALID

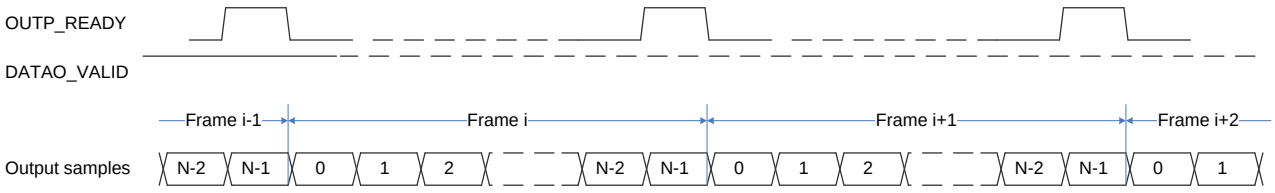
These two signals serve to notify a data receiver when the FFT results are ready. The OUTP_READY is a clock-wide pulse. The core asserts when the output data frame is about to output. The core asserts the DATAO_VALID signal while generating the output frame. The DATAO_VALID signal trails the OUTP_READY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

Figure 16 Output Data and Handshake Signals



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

Figure 17 Streaming Output Data without Gaps



6 Tool Flow

6.1 License

CoreFFT is distributed along with Libero. Source code and a testbench are provided for the core.

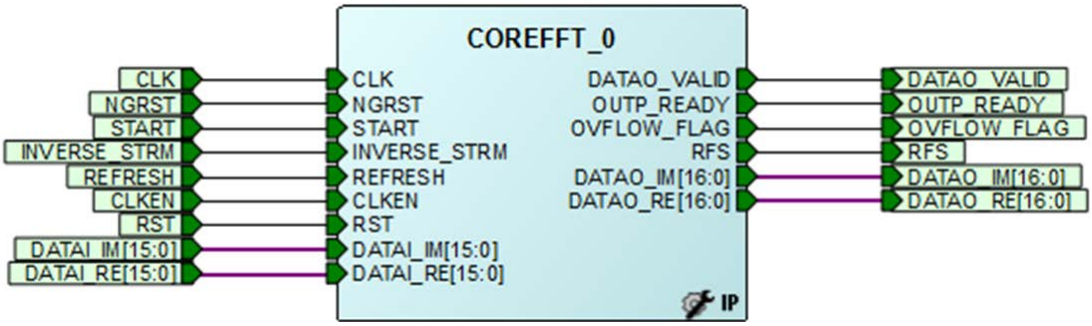
6.2 SmartDesign

CoreFFT is available for download in the Libero IP catalog through the web repository. After it is listed in the catalog, the core can be instantiated using the SmartDesign flow. For information on using SmartDesign to configure, connect, and generate cores, refer to the Libero Online Help.

After configuring and generating the core instance, the basic functionality can be simulated using the test-bench supplied with the CoreFFT. The testbench parameters automatically adjust to the CoreFFT configuration. The CoreFFT can be instantiated as a component of a larger design.

Note: CoreFFT is compatible with both Libero integrated design environment (IDE) and Libero SoC. Unless specified otherwise, this document uses the name Libero to identify both Libero IDE and Libero SoC.

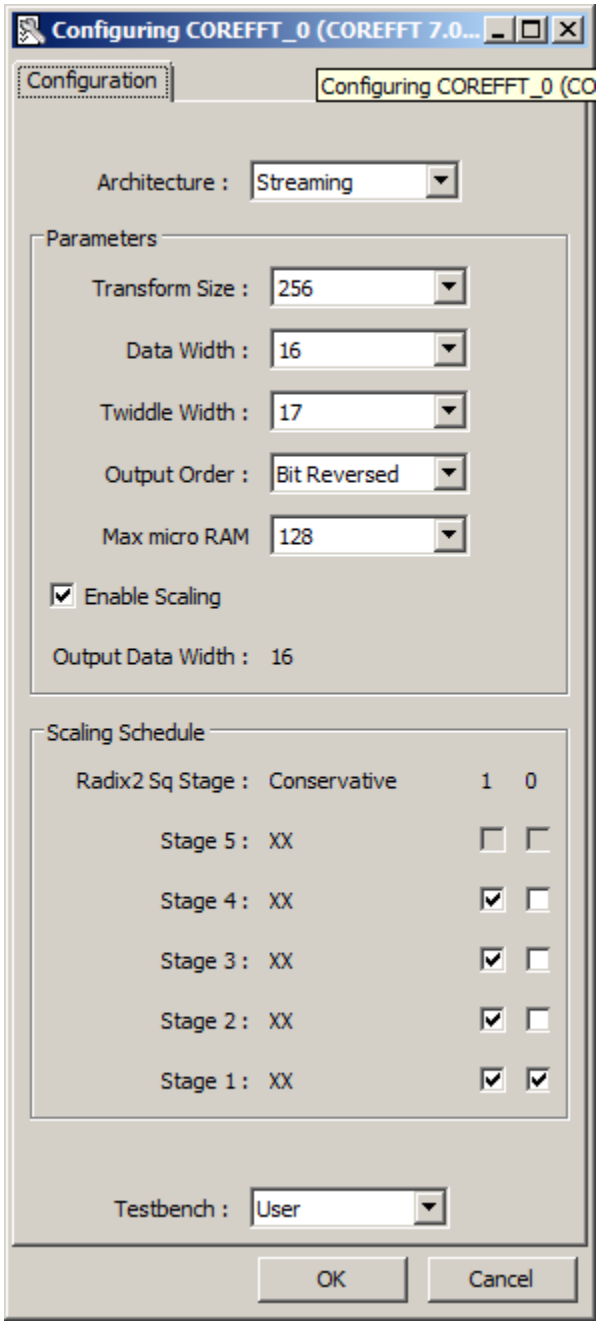
Figure 18 SmartDesign CoreFFT Instance View



6.3 Configuring CoreFFT in SmartDesign

The core can be configured using the configuration GUI within SmartDesign. An example of the GUI for the SmartFusion2 family is shown in Figure 19.

Figure 19 Configuring CoreFFT in SmartDesign



6.4 Simulation Flows

The user testbench for CoreFFT is included in the release.

To run simulations, select the User Testbench flow within SmartDesign. The User Testbench is selected using the Core Configuration window.

When SmartDesign generates the core, it installs the user testbench files.

To run the user testbench, set the design root to the CoreFFT instantiation in the Libero design hierarchy pane and run pre-synthesis design simulation.

Note: When simulating the VHDL version of the core you might want to get rid of the IEEE.NUMERIC_STD library warnings. To do so, add the following two lines to the automatically generated `run.do` file:

- set NumericStdNoWarnings 1
- set StdArithNoWarnings 1

6.5 Synthesis in Libero

To run synthesis on the CoreFFT, set the design root to the IP component instance, and run the synthesis tool from the Libero design flow pane.

6.6 Place-and-Route in Libero

After the design is synthesized, run the compilation and then place-and-route the tools. CoreFFT requires no special place-and-route settings.

7 Testbench

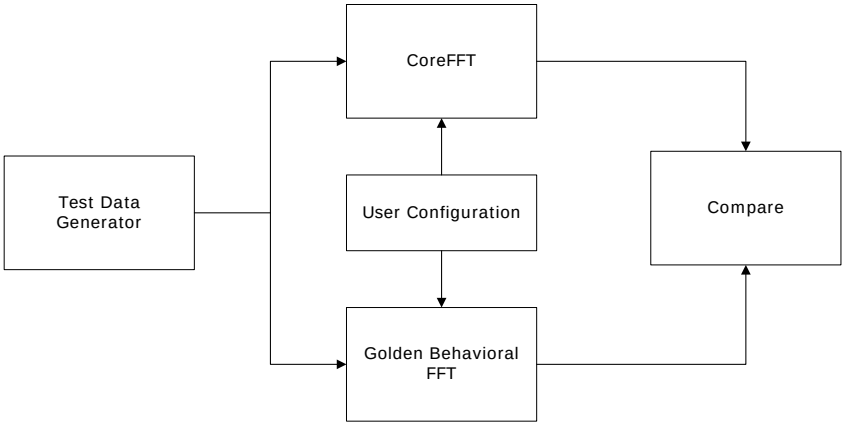
A unified test-bench is used to verify and test CoreFFT called as user test-bench.

7.1 User Test-bench

Figure 20 shows the testbench block diagram. The golden behavioral FFT implements the finite precision calculations shown in $x(k) = n = 0N\text{-}1X(n)e-jnk2\pi/N$

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 20 CoreFFT User Test-bench



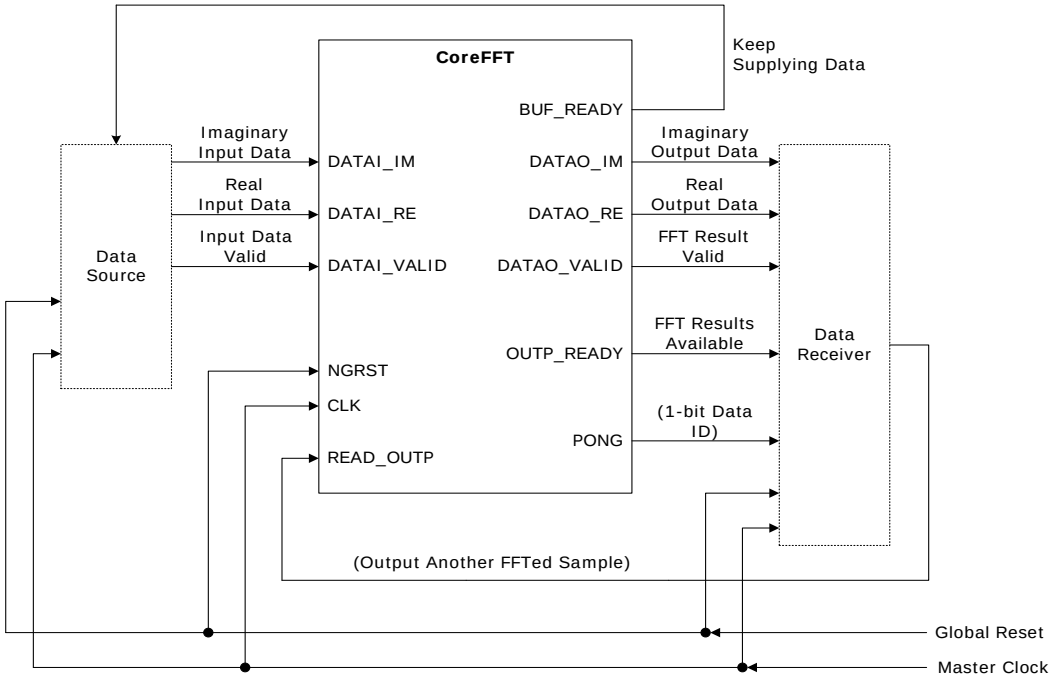
The testbench provides examples of how to use a generated FFT module. The testbench can be modified according to the requirements.

8 System Integration

8.1 In-Place FFT

Figure 21 shows 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 must be supplied simultaneously and accompanied with the validity bit-DATAI_VALID. The data source can supply the sample at every clock cycle or at an arbitrary slower rate (refer to Figure 11). After the FFT module receives N input samples, it lowers the BUF_READY signal.

Figure 21 Example of the In-Place FFT System

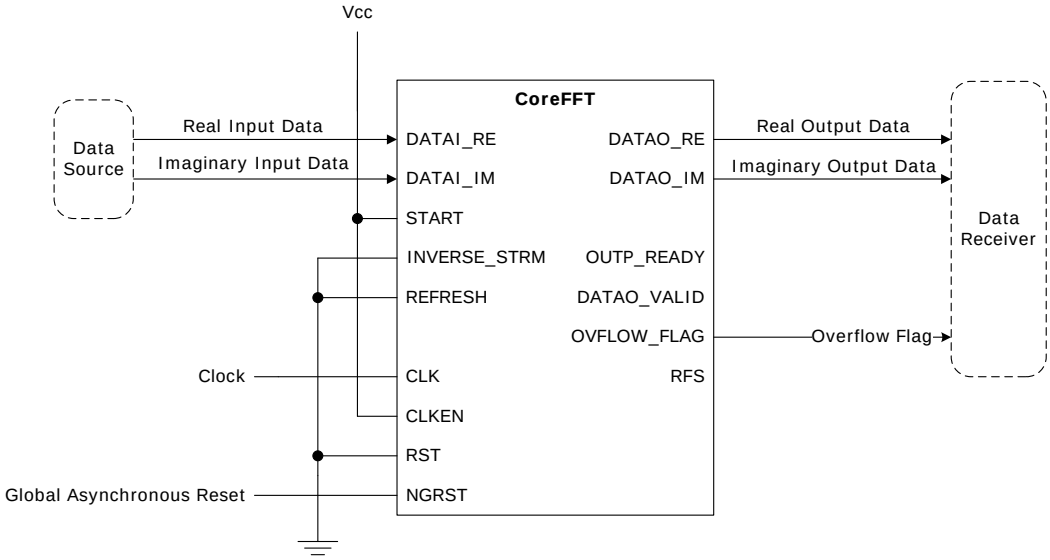


The FFT engine starts processing the data automatically after it is ready. In the minimal memory configuration, the processing phase starts immediately after data loading is complete. In the buffered configuration, the FFT engine can wait until a previous data burst is processed. Then the engine starts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

8.2 Streaming FFT

Streaming CoreFFT can process infinite complex data streams, as shown in Figure 22. The core performs forward FFT over the data coming at every clock cycle. The data source keeps supplying the data while the data receiver permanently receives the FFT-ed results and monitors the overflow flag if necessary. The optional input START signal and the output RFS signal can be used if processing of data frames is required. The data source generates the START signal to mark the beginning of another frame, and the data receiver uses the RFS signal to mark the beginning of the output frame.

Figure 22 Example of a Streaming FFT System



9 Ordering Information

9.1 Ordering Codes

CoreFFT can be ordered through your local Microsemi sales representative. It should be ordered using the following number scheme: CoreFFT -XX, where XX is listed in the following table.

Table 16 Ordering Codes

XX	Description
RM	RTL for RTL source - multiple-use license.