LogiCORE IP Fast Fourier Transform v8.0

DS808 July 25, 2012 **Product Specification**

Introduction

The Xilinx LogiCORE™ IP Fast Fourier Transform (FFT) implements the Cooley-Tukey FFT algorithm, a computationally efficient method for calculating the Discrete Fourier Transform (DFT).

Features

- Drop-in module for Virtex®-7 and Kintex™-7, Virtex®-6 and Spartan®-6 FPGAs
- AXI4-Stream compliant interfaces.
- Forward and inverse complex FFT, run-time configurable
- Transform sizes $N = 2^m$, $m = 3 – 16$
- Data sample precision $b_x = 8 – 34$
- Phase factor precision $b_w = 8 – 34$
- Arithmetic types:
  - Unscaled (full-precision) fixed-point
  - Scaled fixed-point
  - Block floating-point
- Fixed-point or floating-point interface
- Rounding or truncation after the butterfly
- Block RAM or Distributed RAM for data and phase-factor storage
- Optional run-time configurable transform point size
- Run-time configurable scaling schedule for scaled fixed-point cores
- Bit/digit reversed or natural output order
- Optional cyclic prefix insertion for digital communications systems
- Four architectures offer a trade-off between core size and transform time
- Bit-accurate C model and MEX function for system modeling available for download
- For use with Xilinx CORE Generator™ software and Xilinx System Generator for DSP 13.1

LogiCORE IP Facts	
Core Specifics	
Supported Device Family(1)	Kintex-7, Virtex-7 Virtex-6, Spartan-6
Supported User Interfaces	AXI4-Stream
Configuration	See Tables 25 to 28
Provided with Core	
Documentation	Product Specification
Design Files	Netlist
Example Design	Not Provided
Test Bench	VHDL
Constraints File	N/A
Simulation Model	VHDL Verilog C Model
Tested Design Tools	
Design Entry Tools	CORE Generator 13.1 System Generator for DSP 13.1
Simulation	Mentor Graphics ModelSim 6.6d Cadence Incisive Enterprise Simulator (IES) 10.2 Synopsys VCS and VCS MX 2010.06 ISIM 13.1
Synthesis Tools	N/A
Support	
Provided by Xilinx, Inc.	

1. For the complete list of supported devices, see the release notes for this core.

Functional Description

Overview

The FFT core computes an *N*-point forward DFT or inverse DFT (IDFT) where *N* can be 2^m, m = 3–16.

For fixed-point inputs, the input data is a vector of *N* complex values represented as dual b_x-bit two's-complement numbers, that is, b_x bits for each of the real and imaginary components of the data sample, where b_x is in the range 8 to 34 bits inclusive. Similarly, the phase factors b_w can be 8 to 34 bits wide.

For single-precision floating-point inputs, the input data is a vector of *N* complex values represented as dual 32-bit floating-point numbers with the phase factors represented as 24- or 25-bit fixed-point numbers.

All memory is on-chip using either block RAM or distributed RAM. The *N* element output vector is represented using b_y bits for each of the real and imaginary components of the output data. Input data is presented in natural order and the output data can be in either natural or bit/digit reversed order. The complex nature of data input and output is intrinsic to the FFT algorithm, not the implementation.

Three arithmetic options are available for computing the FFT:

- Full-precision unscaled arithmetic
- Scaled fixed-point, where you provide the scaling schedule
- Block floating-point (run-time adjusted scaling)

The point size *N*, the choice of forward or inverse transform, the scaling schedule and the cyclic prefix length are run-time configurable. Transform type (forward or inverse), scaling schedule and cyclic prefix length can be changed on a frame-by-frame basis. Changing the point size resets the core.

Four architecture options are available: Pipelined Streaming I/O, Radix-4 Burst I/O, Radix-2 Burst I/O, and Radix-2 Lite Burst I/O. For detailed information about each architecture, see Architecture Options, page 14.

Theory of Operation

The FFT is a computationally efficient algorithm for computing a Discrete Fourier Transform (DFT) of sample sizes that are a positive integer power of 2. The DFT $X(k),\ k = 0,\ldots,N-1$ of a sequence $x(n),\ n = 0,\ldots,N-1$ is defined as

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-jnk2\pi/N} \quad k = 0,\ldots,N-1 \qquad \textit{Equation 1}$$

where *N* is the transform size and $j = \sqrt{-1}$. The inverse DFT (IDFT) is given by

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) e^{jnk2\pi/N} \quad n = 0,\ldots,N-1 \qquad \textit{Equation 2}$$

Algorithm

The FFT core uses the Radix-4 and Radix-2 decompositions for computing the DFT. For Burst I/O architectures, the decimation-in-time (DIT) method is used, while the decimation-in-frequency (DIF) method is used for the Pipelined Streaming I/O architecture. When using Radix-4 decomposition, the *N*-point FFT consists of $\log_4(N)$ stages, with each stage containing *N/4* Radix-4 butterflies. Point sizes that are not a power of 4 need an extra Radix-2 stage for combining data. An *N*-point FFT using Radix-2 decomposition has $\log_2(N)$ stages, with each stage containing *N/2* Radix-2 butterflies.

The inverse FFT (IFFT) is computed by conjugating the phase factors of the corresponding forward FFT.

Finite Word Length Considerations

The Burst I/O architectures process an array of data by successive passes over the input data array. On each pass, the algorithm performs Radix-4 or Radix-2 butterflies, where each butterfly picks up four or two complex numbers, respectively, and returns four or two complex numbers to the same memory. The numbers returned to memory by the core are potentially larger than the numbers picked up from memory. A strategy must be employed to accommodate this dynamic range expansion. A full explanation of scaling strategies and their implications is beyond the scope of this document; for more information about this topic; see [Ref 1] and [Ref 2].

For a Radix-4 DIT FFT, the values computed in a butterfly stage can experience growth by a factor of up to $1 + 3\sqrt{2} \approx 5.242$. This implies a bit growth of up to 3 bits.

For Radix-2, the growth is by a factor of up to $1+\sqrt{2} \approx 2.414$. This implies a bit growth of up to 2 bits. This bit growth can be handled in three ways:

- Performing the calculations with no scaling and carrying all significant integer bits to the end of the computation
- Scaling at each stage using a fixed-scaling schedule
- Scaling automatically using block floating-point

All significant integer bits are retained when using full-precision unscaled arithmetic. The width of the data path increases to accommodate the bit growth through the butterfly. The growth of the fractional bits created from the multiplication are truncated (or rounded) after the multiplication. The width of the output is (input width + $\log_2$(transform length) + 1). This accommodates the worst case scenario for bit growth.

Consider an unscaled Radix-2 DIT FFT: the data path in each stage must grow by 1 bit as the adder and subtractor in the butterfly may add/subtract two full-scale values and produce a sample which has grown in width by 1 bit. This yields the $\log_2$(transform length) part of the increase in the output width relative to the input width. The complex multiplier preserves the magnitude of an input (as it applies a rotation on the complex plane), but can theoretically produce bit-growth when the magnitude of the input is greater than 1 (for example, 1+j has a magnitude of 1.414). This means that the complex multiplier bit growth must only be considered once in the entire FFT process, yielding the additional +1 increase in the output width relative to the input width. For example, a 1024-point transform with an input of 16 bits consisting of 1 integer bit and 15 fractional bits has an output of 27 bits with 12 integer bits and 15 fractional bits. Note that the core does not have a specific location for the binary point. The output simply maintains the same binary point location as the input. For the preceding example, a 16 bit input with 3 integer bits and 13 fractional bits would have an unscaled output of 27 bits with 14 integer bits and 13 fractional bits.

When using scaling, a scaling schedule is used to divide by a factor of 1, 2, 4, or 8 in each stage. If scaling is insufficient, a butterfly output may grow beyond the dynamic range and cause an overflow. As a result of the scaling applied in the FFT implementation, the transform computed is a scaled transform. The scale factor *s* is defined as

$$s = 2^{\sum_{i=0}^{\log_2 N-1} b_i}$$ *Equation 3*

where b_i is the scaling (specified in bits) applied in stage *i*.

The scaling results in the final output sequence being modified by the factor *1/s*. For the forward FFT, the output sequence *X' (k), k = 0,...,N - 1* computed by the core is defined as

$$X'(k) = \frac{1}{s}X(k) = \frac{1}{s}\sum_{n=0}^{N-1} x(n)e^{-jnk2\pi/N} \quad k = 0,\ldots,N-1$$ *Equation 4*

For the inverse FFT, the output sequence is

$$x(n) = \frac{1}{s}\sum_{k=0}^{N-1} X(k)e^{jnk2\pi/N} \quad n = 0,\ldots,N-1$$ *Equation 5*

If a Radix-4 algorithm scales by a factor of 4 in each stage, the factor of *1/s* is equal to the factor of *1/N* in the inverse FFT equation (Equation 2). For Radix-2, scaling by a factor of 2 in each stage provides the factor of *1/N*.

With block floating-point, each stage applies sufficient scaling to keep numbers in range, and the scaling is tracked by a block exponent.

As with unscaled arithmetic, for scaled and block floating-point arithmetic, the core does not have a specific location for the binary point. The location of the binary point in the output data is inherited from the input data and then shifted by the scaling applied.

Floating Point Considerations

The FFT core optionally accepts data in IEEE-754 single-precision format with 32-bit words consisting of a 1-bit sign, 8-bit exponent, and 23-bit fraction. The construction of the word matches that of the Xilinx Floating-Point Operator core.

Implementing full floating-point on an FPGA can be expensive in terms of the resources required. The floating-point option in the Xilinx FFT core utilizes a higher precision fixed-point FFT internally to achieve similar noise performance to a full floating-point FFT, with significantly fewer resources. Figure 1 illustrates the two levels of noise performance possible by selecting either 24 bits or 25 bits for the phase factor width. By increasing the phase factor width to 25 bits, more resources may be required, depending on the target FPGA device.

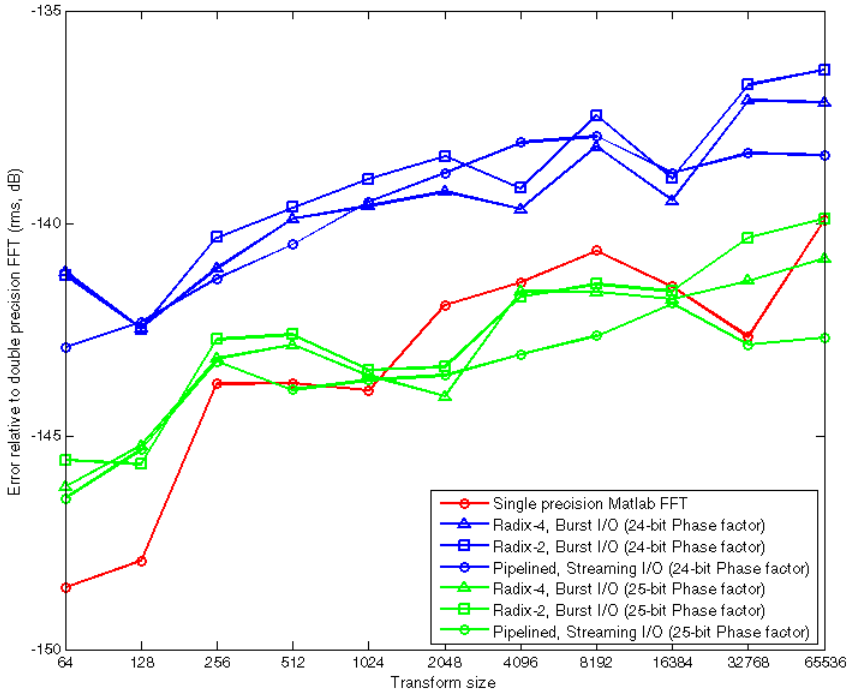

Figure 1: **Comparison of Two Levels of Noise Performance**

Figure 1 shows the ratio of the RMS difference between various models and the double-precision MATLAB® FFT to the data set peak amplitude. The models shown are the single-precision MATLAB FFT function (calculated by casting the input data to single-precision floating-point type), the Xilinx FFT core using a 24-bit phase factor width, and the Xilinx FFT core using a 25-bit phase factor width. To calculate the error signal, a randomized impulse (in magnitude and time) was used as the input signal, with the RMS error averaged over five simulation runs.

All optimization options (memory types and XtremeDSP™ slice optimization) remain available when floating-point input data is selected, allowing you to trade off resources with transform time.

Transform time for Burst I/O architectures is increased by approximately N, the number of points in the transform, due to the input normalization requirements. For the Pipelined Streaming I/O architecture, the initial latency to fill the pipeline is increased, but data still streams through the core with no gaps.

Denormalized Numbers

The floating-point interface to the FFT core does not support denormalized numbers. To match the behavior of the Xilinx Floating-Point Operator core, the core treats denormalized operands as zero, with a sign taken from the denormalized number.

NaNs and ± Infinity

If the core detects a NaN or ± Infinity value on the input, all output samples associated with the current input frame are set to NaN. The sign bit is set to zero and all exponent and fraction bits are set to 1.

Real-Valued Input Data

The FFT core accepts complex data samples, but can perform a transform on real-valued data by setting all imaginary input samples to zero.

Due to the finite wordlength effects described previously, noise is introduced during the transform, resulting in the output data not being perfectly symmetric. The DIT and DIF FFT algorithms have different noise effects due to the different calculation order.

For a thorough treatment of this topic, see [Ref 3] and [Ref 4].

The asymmetry between the two halves of the result is more noticeable at larger point sizes. In addition, the noise is more prominent in the lower frequency bins. Therefore, Xilinx recommends that the upper half (N/2+1 to N points) of the output data is used when performing a real-valued FFT.

Rounding Implementation

An option is available, in all architectures, to apply convergent rounding to the data after the butterfly stage. However, selecting this option does not apply convergent rounding to all points in the data path where wordlength reduction occurs.

In particular, the outputs of all complex multipliers in the FFT data path are truncated to reduce data path width (while still maintaining adequate precision) and a simple rounding constant added to the fractional bits. This constant implements non-symmetric, round-towards-minus-infinity rounding, and can introduce a small bias to the FFT results over a large number of samples.

Dynamic Range Characteristics

The dynamic range characteristics are shown by performing *slot noise* tests. First, a frame of complex Gaussian noise data samples is created. An FFT is taken to acquire the spectrum of the data. To create the slot, a range of frequencies in the spectra is set to zero. To create the input slot noise data frame, the inverse FFT is taken, then the data is quantized to use the full input dynamic range. Because of the quantization, if a perfect FFT is done on the frame, the noise floor on the bottom of the slot is non-zero. The Input Data figures, which basically represent the dynamic range of the input format, display this.

This slot noise input data frame is fed to the FFT core to see how shallow the slot becomes due to the finite precision arithmetic. The depth of the slot shows the dynamic range of the FFT.

Figure 2 through Figure 11 show the effect of input data width on the dynamic range. All FFTs have the same bit width for both data and phase factors. Block floating-point arithmetic is used with rounding after the butterfly. The figures show the input data slot and the output data slot for bit widths of 24, 20, 16, 12, and 8.

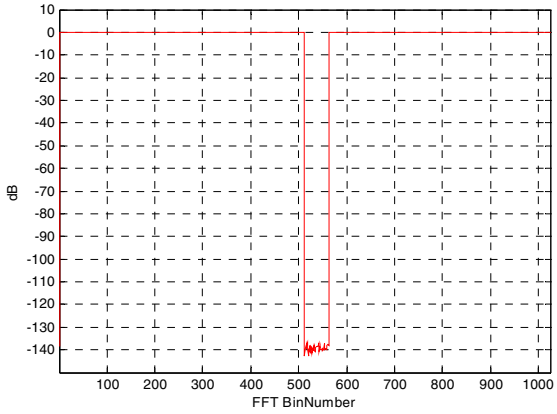

Figure 2: **Input Data: 24 Bits**

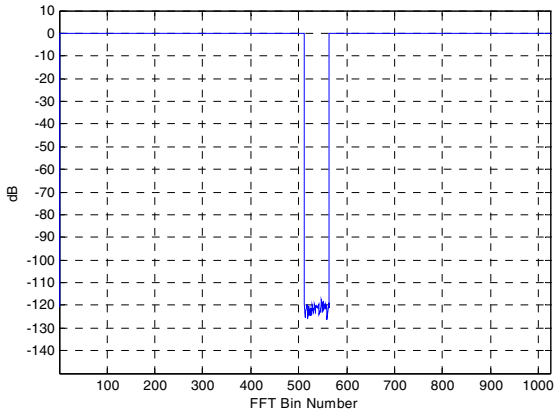

Figure 3: **FFT Core Results: 24 Bits**

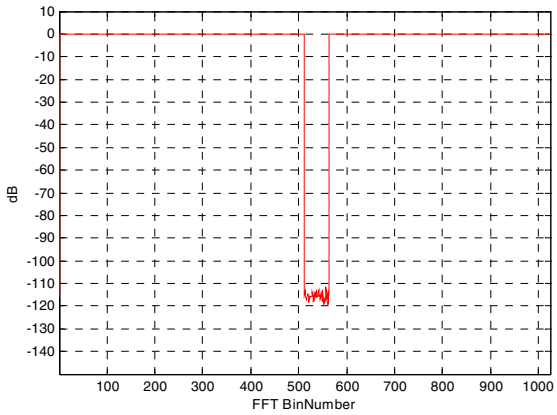

Figure 4: **Input Data: 20 Bits**

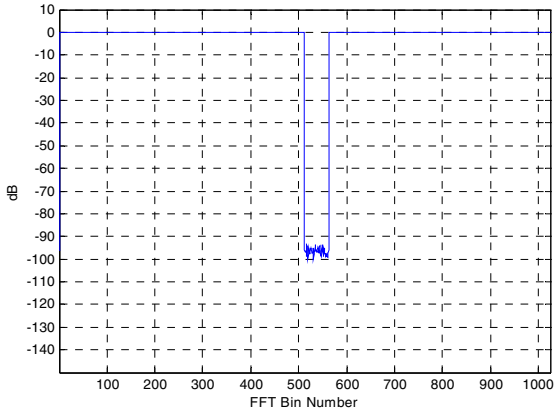

Figure 5: **FFT Core Results: 20 Bits**

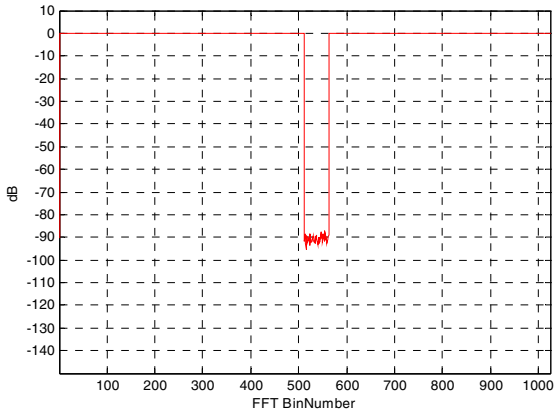

Figure 6: **Input Data: 16 Bits**

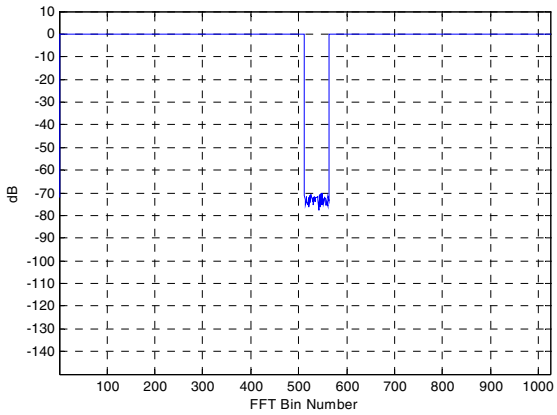

Figure 7: **FFT Core Results: 16 Bits**

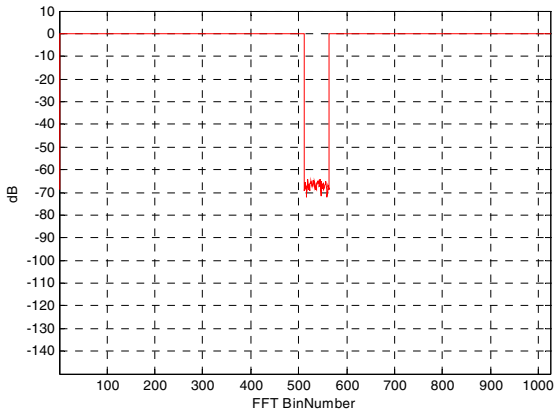

Figure 8: **Input Data: 12 Bits**

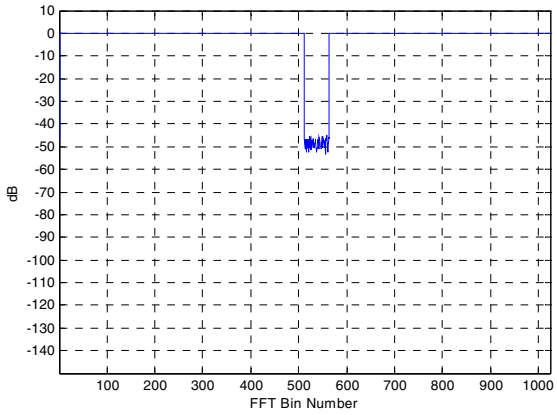

Figure 9: **FFT Core Results: 12 Bits**

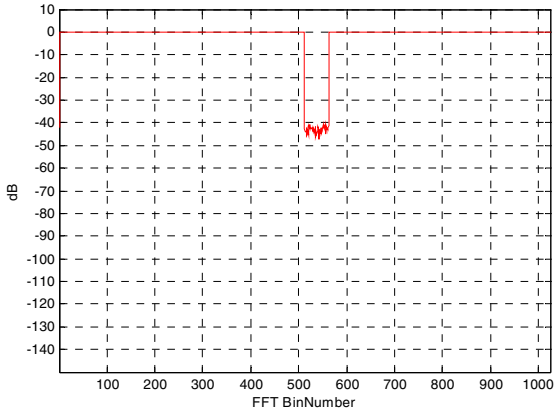

Figure 10: **Input Data: 8 Bits**

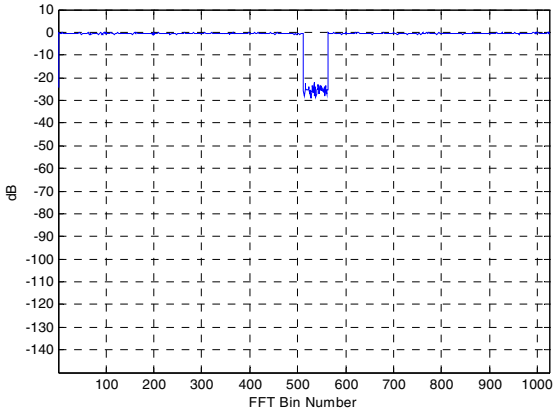

Figure 11: **FFT Core Results: 8 Bits**

There are several options available that also affect the dynamic range. Consider the arithmetic type used.

Figure 12, Figure 13, and Figure 14 display the results of using unscaled, scaled (scaling of 1/1024), and block floating-point. All three FFTs are 1024 point, Radix-4 Burst I/O transforms with 16-bit input, 16-bit phase factors, and convergent rounding.

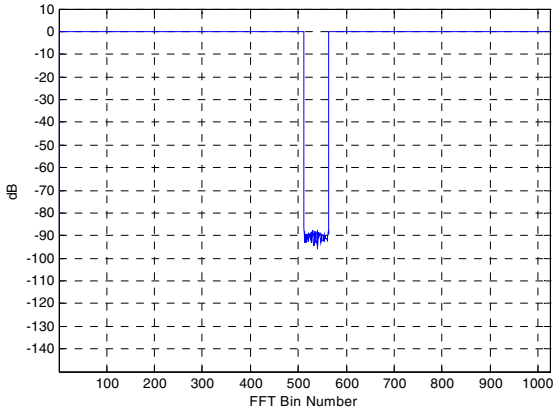

Figure 12: **Full-Precision Unscaled Arithmetic**

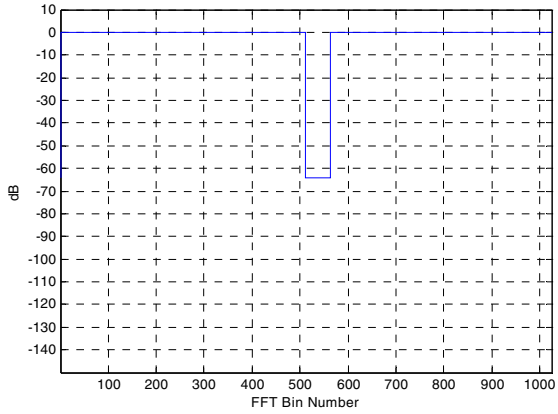

Figure 13: **Scaled (scaling of 1/N) Arithmetic**

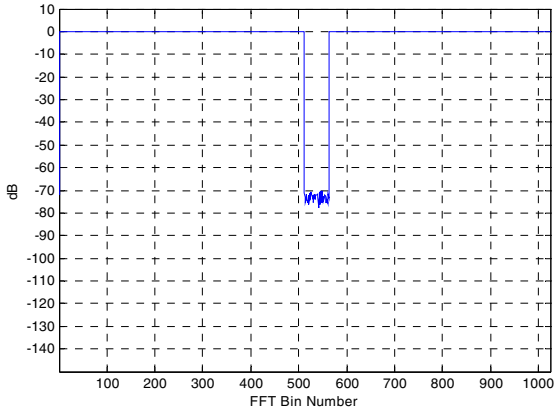

Figure 14: **Block Floating-Point Arithmetic**

After the butterfly computation, the LSBs of the data path can be truncated or rounded. The effects of these options are shown in Figure 15 and Figure 16. Both transforms are 1024 points with 16-bit data and phase factors using block floating-point arithmetic.

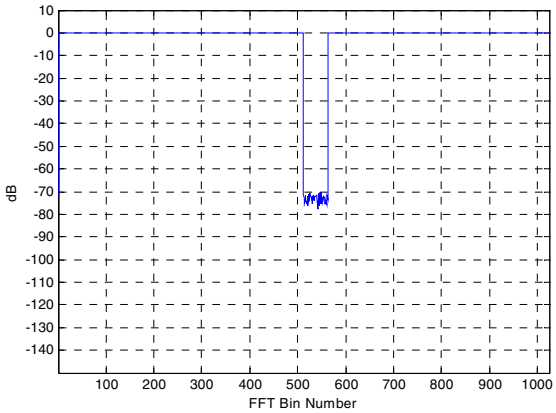

Figure 15: **Convergent Rounding**

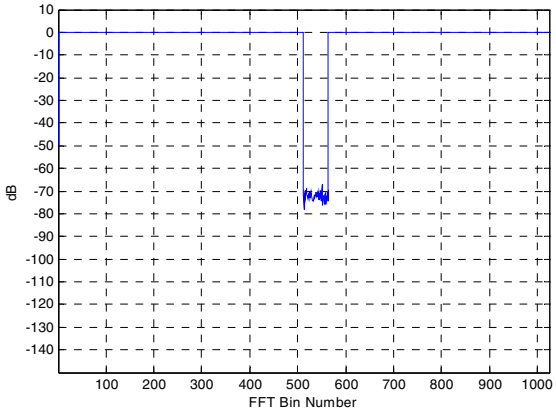

Figure 16: **Truncation**

For illustration purposes, the effect of point size on dynamic range is displayed Figure 17 through Figure 19. The FFTs in these figures use 16-bit input and phase factors along with convergent rounding and block floating-point arithmetic.

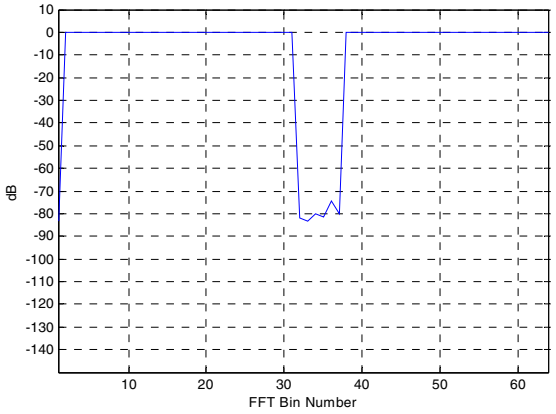

Figure 17: **64-point Transform**

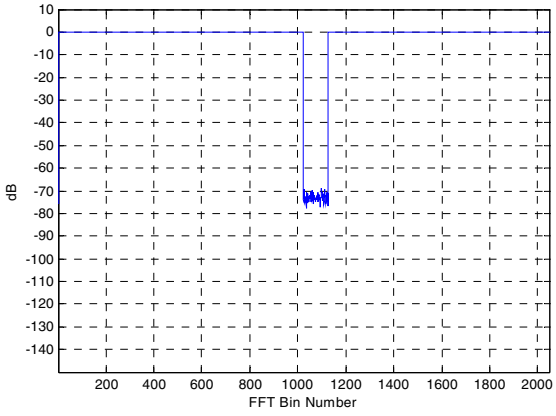

Figure 18: **2048-point Transform**

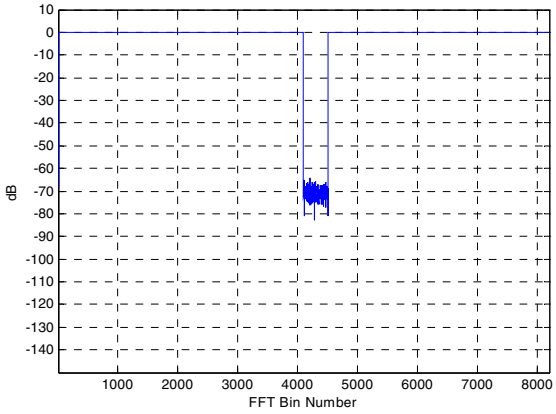

Figure 19: **8192-point Transform**

All of the preceding dynamic range plots show the results for the Radix-4 Burst I/O architecture. Figure 20 and Figure 21 show two plots for the Radix-2 Burst I/O architecture. Both use 16-bit input and phase factors along with convergent rounding and block floating-point.

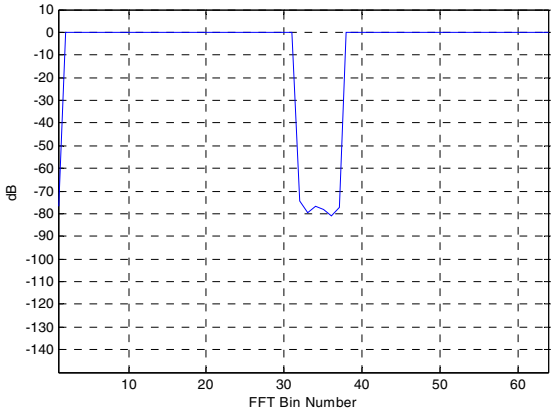

Figure 20: **64-point Radix-2 Transform**

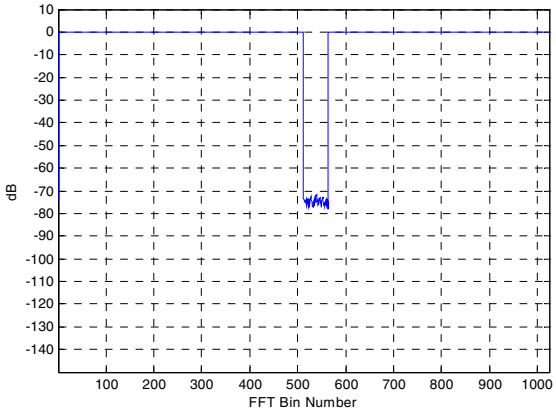

Figure 21: **1024-point Radix-2 Transform**

Architecture Options

The FFT core provides four architecture options to offer a trade-off between core size and transform time.

- **Pipelined Streaming I/O** – Allows continuous data processing.
- **Radix-4 Burst I/O** – Loads and processes data separately, using an iterative approach. It is smaller in size than the pipelined solution, but has a longer transform time.
- **Radix-2 Burst I/O** – Uses the same iterative approach as Radix-4, but the butterfly is smaller. This means it is smaller in size than the Radix-4 solution, but the transform time is longer.
- **Radix-2 Lite Burst I/O** – Based on the Radix-2 architecture, this variant uses a time-multiplexed approach to the butterfly for an even smaller core, at the cost of longer transform time.

Figure 22 illustrates the trade-off of throughput versus resource use for the four architectures. As a rule of thumb, each architecture offers a factor of 2 difference in resource from the next architecture. The example is for an even power of 2 point size. This does not require the Radix-4 architecture to have an additional Radix-2 stage.

All four architectures may be configured to use a fixed-point interface with one of three fixed-point arithmetic methods (unscaled, scaled or block floating-point) or may instead use a floating-point interface.

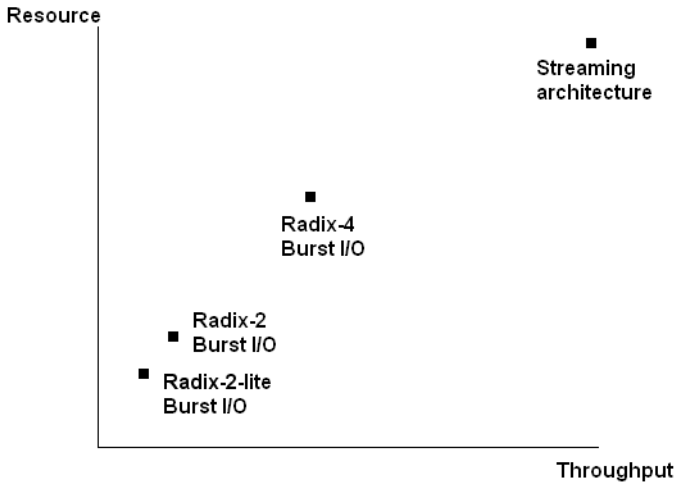

Figure 22: **Resource versus Throughput for Architecture Options**

Bit and Digit Reversal

Each architecture offers the option of natural or reversed ordering of output data, with data being input in natural order. The FFT algorithm reorders the samples during processing such that data input in natural order is output in reversed order. The core can optionally output the data in natural order. However, this imposes a cost on each architecture. For the Burst I/O architectures, this imposes a time penalty, because unloading the data cannot take place at the same time as loading input data for the next frame, so separate unload and load phases are required. In the pipelined architecture, it requires additional RAM storage to perform the reordering.

In the Radix-2 Burst I/O, Radix-2 Lite Burst I/O, and Pipelined Streaming I/O architectures, the Bit Reverse order is simple to calculate by taking the index of the data point, written in binary, and reversing the order of the digits. Hence, 0000, 0001, 0010, 0011, 0100,...(0, 1, 2, 3, 4,...) becomes 0000, 1000, 0100, 1100, 0010,...(0, 8, 4, 12, 2,...).

In the case of the Radix-4 Burst I/O architecture, the reversal applies to *digits* and, therefore, is called Digit Reversal. A digit in Radix-4 is two bits. Hence, 0000, 0001, 0010, 0011, 0100,...(0, 1, 2, 3, 4,...) becomes 0000, 0100, 1000, 1100, 0001,...(0, 4, 8, 12, 1,...), as the pairs of digits are reversed. Where the transform size requires an odd number of index

bits, the odd digit in the least significant place is moved to the most significant place, so 00000, 00001, 00010, 00011, 00100,... (0, 1, 2, 3, 4,...) becomes 00000, 10000, 00100, 10100, 01000,...(0, 16, 4, 20, 8,...)

Note: The core can optionally output a data point index along with the data. See XK Index for more information.

Pipelined Streaming I/O

The Pipelined Streaming I/O solution pipelines several Radix-2 butterfly processing engines to offer continuous data processing. Each processing engine has its own memory banks to store the input and intermediate data (Figure 23). The core has the ability to simultaneously perform transform calculations on the current frame of data, load input data for the next frame of data, and unload the results of the previous frame of data. You can continuously stream in data[1] and, after the calculation latency, can continuously unload the results. If preferred, this design can also calculate one frame by itself or frames with gaps in between.

In the scaled fixed-point mode, the data is scaled after every pair of Radix-2 stages. The block floating-point mode may use significantly more resources than the scaled mode, as it must maintain extra bits of precision to allow dynamic scaling without impacting performance. Therefore, if the input data is well understood and is unlikely to exhibit large amplitude fluctuation, using scaled arithmetic (with a suitable scaling schedule to avoid overflow in the known worst case) is sufficient, and resources may be saved.

The input data is presented in natural order. The unloaded output data can either be in bit reversed order or in natural order. When natural order output data is selected, additional memory resource is utilized.

This architecture covers point sizes from 8 to 65536. You have the flexibility to select the number of stages to use block RAM for data and phase factor storage. The remaining stages use distributed memory.

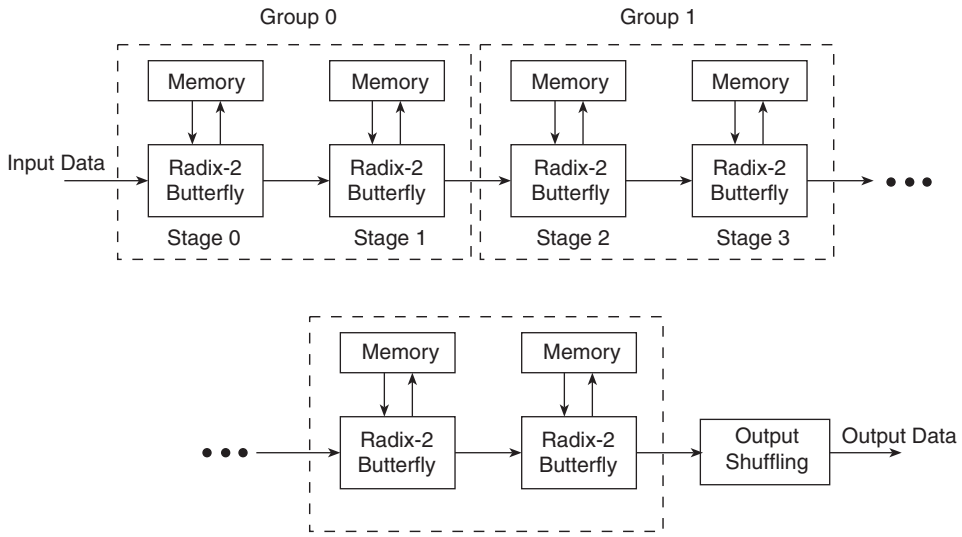

Figure 23: **Pipelined Streaming I/O**

Radix-4 Burst I/O

With the Radix-4 Burst I/O solution, the FFT core uses one Radix-4 butterfly processing engine (Figure 24). It loads and/or unloads data separately from calculating the transform. Data I/O and processing are not simultaneous. When the FFT is started, the data is loaded. After a full frame has been loaded, the core computes the transform.

1. Note that continually streaming data does not imply that AXI4-Stream waitstates from the FFT can be ignored. There are situations where the FFT core may have to insert waitstates to pause the incoming sample data.

When the computation has finished, the data can be unloaded, but cannot be loaded or unloaded during the calculation process. The data loading and unloading processes can be overlapped if the data is unloaded in digit reversed order.

This architecture has lower resource usage than the Pipelined Streaming I/O architecture, but a longer transform time, and supports point sizes from 64 to 65536. Data and phase factors can be stored in block RAM or in distributed RAM (the latter for point sizes less than or equal to 1024).

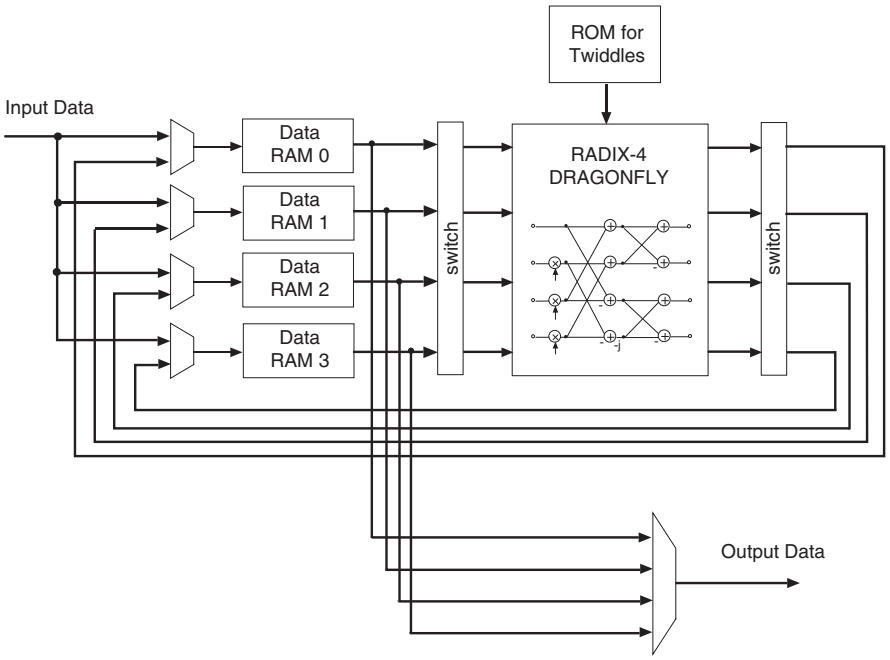

Figure 24: **Radix-4 Burst I/O**

Radix-2 Burst I/O

The Radix-2 Burst I/O architecture uses one Radix-2 butterfly processing engine (Figure 25). After a frame of data is loaded, the input data stream must halt until the transform calculation is completed. Then, the data can be unloaded. As with the Radix-4 Burst I/O architecture, data can be simultaneously loaded and unloaded when the output samples are in bit reversed order. This solution supports point sizes from 8 to 65536. Both the data memories and phase factor memories can be in either block RAM or distributed RAM (the latter for point sizes less than or equal to 1024).

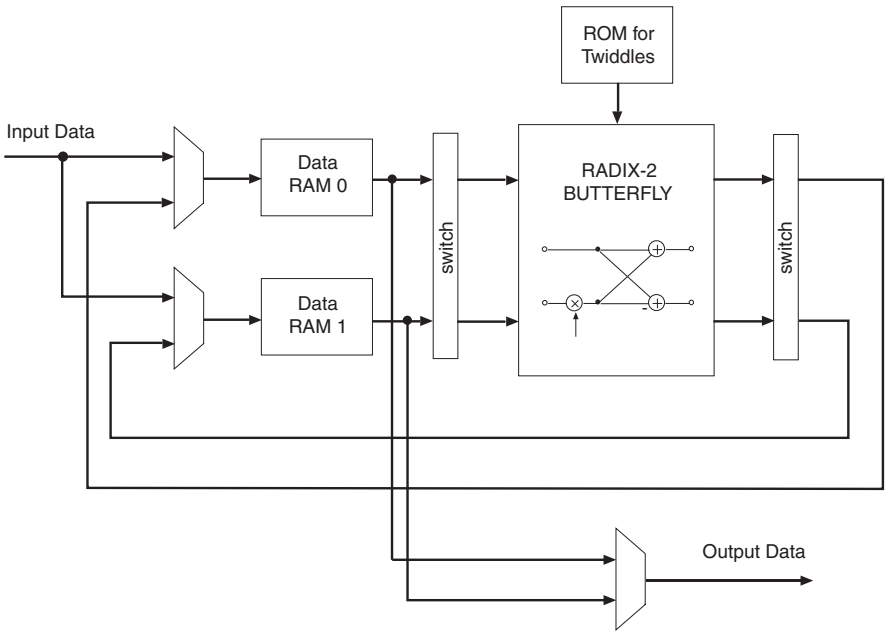

Figure 25: **Radix-2 Burst I/O**

Radix-2 Lite Burst I/O

This architecture differs from the Radix-2 Burst I/O in that the butterfly processing engine uses one shared adder/subtractor, hence reducing resources at the expense of an additional delay per butterfly calculation. Again, as with the Radix-4 and Radix-2 Burst I/O architectures, data can be simultaneously loaded and unloaded only if the output samples are in bit reversed order. This solution supports point sizes from 8 to 65536. See Figure 26.

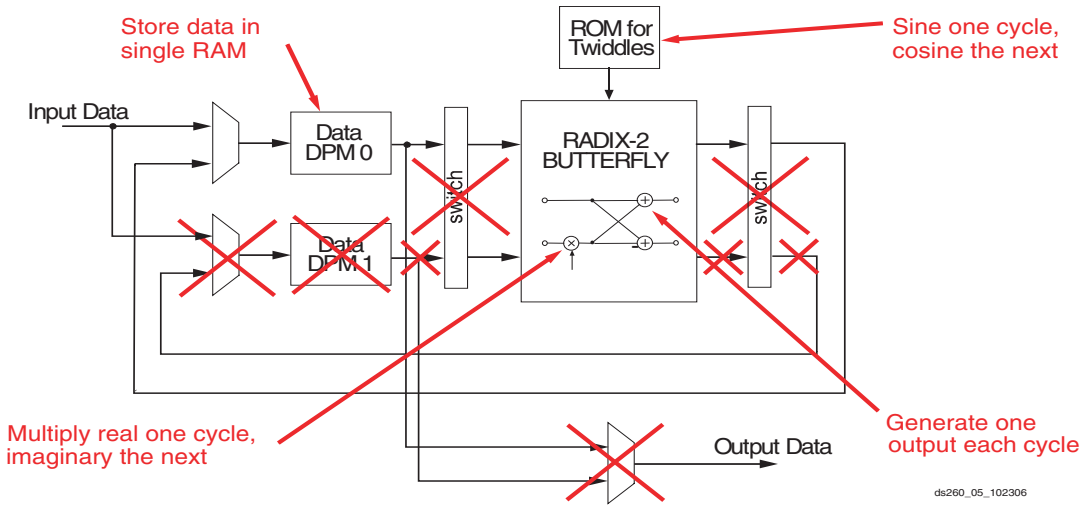

Figure 26: **Radix-2 Lite Burst I/O**

Run-Time Transfer Configuration

All run-time configuration options discussed in this section are programed using the Configuration channel. Please see section Configuration Channel for more information.

Transform Size

The transform point size can be set through the NFFT field in the Configuration Channel if the run-time configurable transform length option is selected. Valid settings and the corresponding transform sizes are provided in Table 1. If the NFFT value entered is too large, the core sets itself to the largest available point size (selected in the GUI). If the value is too small, the core sets itself to the smallest available point size: 64 for the Radix-4 Burst I/O architecture and 8 for the other architectures.

Table 1: **Valid NFFT Settings**

NFFT[4:0]	Transform size (*N*)
00011	8
00100	16
00101	32
00110	64
00111	128
01000	256
01001	512
01010	1024
01011	2048
01100	4096
01101	8192
01110	16384
01111	32768
10000	65536

Forward/Inverse and Scaling Schedule

The transform type (forward or inverse) and the scaling schedule can be set frame-by-frame without interrupting frame processing. Both the transform type and the scaling schedule can be set independently for each FFT channel in a multichannel core. Each FFT data channel has an assigned `FWD_INV` field and `SCALE_SCH` field in the Configuration channel.

Setting the FWD_INV field to 0 produces an inverse FFT, and setting the `FWD_INV` field to 1 creates the forward transform.

Burst I/O Architectures

The scaling performed during successive stages can be set via the appropriate `SCALE_SCH` field in the Configuration channel. For the Radix-4, Burst I/O and Radix-2 architectures, the value of the `SCALE_SCH` field is used as pairs of bits [... N4, N3, N2, N1, N0], each pair representing the scaling value for the corresponding stage. Stages are computed starting with stage 0 as the two LSBs. There are $\log_4$(point size) stages for Radix-4 and $\log_2$(point size) stages for Radix-2. In each stage, the data can be shifted by 0, 1, 2, or 3 bits, which corresponds to `SCALE_SCH` values of 00, 01, 10, and 11. For example, for Radix-4, when $N = 1024$, [01 10 00 11 10] translates to a right shift by 2 for stage 0, shift by 3 for stage 1, no shift for stage 3, a shift of 2 in stage 3, and a shift of 1 for stage 4 (there are $\log_4(1024) = 5$ Radix-4 stages). This scaling schedule scales by a total of 8 bits which gives a scaling factor of 1/256. The conservative schedule `SCALE_SCH` = [10 10 10 10 11] completely avoids overflows in the Radix-4, Burst I/O architecture. For the Radix-2, Burst I/O and Radix-2 Lite, Burst I/O architectures, the conservative scaling schedule of [01 01 01 01 01 01 01 01 01 10] prevents overflow for $N = 1024$ (there are $\log_2(1024) = 10$ Radix-2 stages).

Pipelined Streaming I/O Architecture

For the Pipelined Streaming I/O architecture, consider every pair of adjacent Radix-2 stages as a group. That is, group 0 contains stage 0 and 1, group 1 contains stage 2 and 3, and so forth. The value of the SCALE_SCH field is also used as pairs of bits [... N4, N3, N2, N1, N0]. Each pair represents the scaling value for the corresponding group of two stages. Groups are computed starting with group 0 as the two LSBs. In each group, the data can be shifted by 0, 1, 2, or 3 bits which corresponds to SCALE_SCH values of 00, 01, 10, and 11. For example, when *N* = 1024, [10 10 00 01 11] translates to a right shift by 3 for group 0 (stages 0 and 1), shift by 1 for group 1 (stages 2 and 3), no shift for group 3 (stages 4 and 5), a shift of 2 in group 3 (stages 6 and 7), and a shift of 2 for group 4 (stages 8 and 9). The conservative schedule SCALE_SCH = [10 10 10 10 11] completely avoids overflows in the Pipelined Streaming I/O architecture. When the point size is not a power of 4, the last group only contains one stage, and the maximum bit growth for the last group is one bit. Therefore, the two MSBs of the scaling schedule can only be 00 or 01. A conservative scaling schedule for N = 512 is SCALE_SCH = [01 10 10 10 11].

The initial value and reset value of the FWD_INV field is forward = 1. The scaling schedule is set to *1/N*. That translates to [10 10 10 10... 10] for the Radix-4, Burst I/O and Pipelined Streaming I/O architectures, and [01 01... 01] for the Radix-2 architectures. The core uses the (2*number of stages) LSBs for the scaling schedule. So, when the point size decreases, the leftover MSBs are ignored. However, all bits are programmed into the core and are used in later transforms if the point size increases.

Cyclic Prefix Insertion

Cyclic prefix insertion takes a section of the output of the FFT and prefixes it to the beginning of the transform. The resultant output data consists of the cyclic prefix (a copy of the end of the output data) followed by the complete output data, all in natural order. Cyclic prefix insertion is only available when output ordering is Natural Order.

When cyclic prefix insertion is used, the length of the cyclic prefix can be set frame-by-frame without interrupting frame processing. The cyclic prefix length can be any number of samples from zero to one less than the point size. The cyclic prefix length is set by the CP_LEN field in the Configuration channel. For example, when *N* = 1024, the cyclic prefix length can be from 0 to 1023 samples, and a CP_LEN value of 0010010110 produces a cyclic prefix consisting of the last 150 samples of the output data.

The initial value and reset value of CP_LEN is 0 (no cyclic prefix). The core uses the $\log_2$(point size) MSBs of CP_LEN for the cyclic prefix length. So, when the point size decreases, the leftover LSBs are ignored. This effectively scales the cyclic prefix length with the point size, keeping them in approximately constant proportion. However, all bits of CP_LEN are programmed into the core and are used in later transforms if the point size increases.

Transfer Status

Overflow

Fixed-Point Data

The Overflow (OVFLO) field in the Data Output and Status channels is only available when the Scaled arithmetic is used. OVFLO is driven high during unloading if any point in the data frame overflowed. For a multichannel core, there is a separate OVFLO field for each channel.

When an overflow occurs in the core, the data is wrapped rather than saturated, resulting in the transformed data becoming unusable for most applications.

Floating-Point Data

The Overflow field is used to indicate an exponent overflow when the FFT is processing floating-point data. The output sample which overflowed is set to +/- Infinity, depending on the sign of the internal result. The Overflow

field is not asserted when a NaN value is present on the output. NaN values can only occur at the FFT output when the input data frame contains NaN or +/- Infinity samples.

Block Exponent

The Block Exponent (BLK_EXP) field in the Data Output and the Status channels (used only with the block floating-point option) contains the block exponent. For a multichannel core, there is a separate BLK_EXP field for each channel. The value present in the field represents the total number of bits the data was scaled during the transform. For example, if BLK_EXP has a value of 00101 = 5, this means the associated output data (XK_RE, XK_IM) was scaled by 5 bits (shifted right by 5 bits), or in other words, was divided by 32, to fully utilize the available dynamic range of the output data path without overflowing.

XK Index

The XK_INDEX field (if present in the Data Output channel) gives the sample number of the XK_RE/XK_IM data being presented at the same time. In the case of natural order outputs, XK_INDEX increments from 0 to (point size) -1. When bit reversed outputs are used, XK_INDEX covers the same range of numbers, but in a bit (or digit) reversed manner.

For example, when you have an 8 point FFT, XK_INDEX takes on the following values:

Table 2: **XK _INDEX values for 8 point FFT**

XK_INDEX with Natural Outputs	XK_INDEX with Bit Reversed Outputs
0 ('b000)	0 ('b000)
1 ('b001)	4 ('b100)
2 ('b010)	2 ('b010)
3 ('b011)	6 ('b110)
4 ('b100)	1 ('b001)
5 ('b101)	5 ('b101)
6 ('b110)	3 ('b011)
7 ('b111)	7 ('b111)

If cyclic prefix insertion is used, the cyclic prefix is unloaded first and XK_INDEX counts from (point_size) - (cyclic prefix length) up to (point size) -1. After the cyclic prefix has been unloaded, or if the cyclic prefix length is zero, the whole frame of output data is unloaded. XK_INDEX counts from 0 up to (point size) -1 as before. Cyclic Prefix Insertion is only possible with natural order outputs.

Controlling the FFT

Symbol data to be processed is loaded into the FFT core using the Data Input channel. Processed symbol data is unloaded using the Data Output channel. Both of these use the AXI4-Stream protocol. Figure 27 shows the basics of this protocol.

TVALID is driven by the Master component to show that it has data to transfer, and TREADY is driven by the Slave component to show that it is ready to accept data. When both TVALID and TREADY are high, a transfer takes place. Points A in the diagram show clock cycles where no data is transferred because neither the Master or the Slave is ready. Point B shows two clock cycles where data isn't transferred because the Master doesn't have any data to transfer. This is known as a Master Waitstate. Point C shows a clock cycle where no data is transferred because the

Slave isn't ready to accept data. This is known as a Slave Waitstate. Master and Slave waitstates can extend for any number of clock cycles.

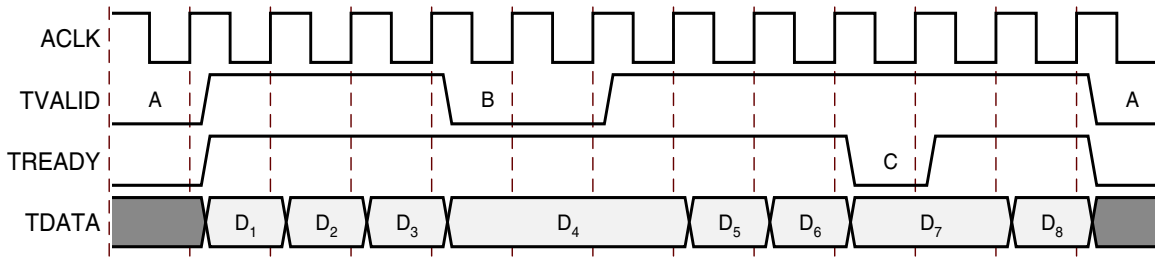

Figure 27: AXI Transfers and Terminology

Once the master asserts TVALID high, it must remain asserted (and the associated data remain stable) until the slave asserts TREADY high.

To load a frame into the FFT, the upstream master supplying the `XN_RE` and `XN_IM` data simply has to send it when it is ready. If the FFT core can accept it (which is when `s_axis_data_tready` = 1) then it is buffered by the FFT core until it can be processed. If the FFT core cannot accept it (which is when `s_axis_data_tready` = 0), a slave waitstate exists in the AXI channel and the master is stalled. Figure 27 shows the loading of the sample data for an 8 point FFT. The upstream master drives TVALID and the FFT drives TREADY. In this case, both the master and the FFT insert waitstates.

Unloading a frame works in a similar manner, except that the FFT core is the master in this case. When it has `XK_RE` and `XK_IM` data to unload, it asserts its TVALID signal (`m_axis_data_tvalid` = 1). The downstream slave that consumes the processed sample data can then accept the data (`m_axis_data_tready` = 1) or not (`m_axis_data_tready` = 0). Figure 27 also shows the unloading of the sample data for an 8 point FFT (with no cyclic prefix). The FFT drives TVALID and the downstream slave drives TREADY. In this case, both the FFT and the slave insert waitstates.

The previous description only applies when the core is configured to use Non-Realtime mode. The situation is different in Realtime mode, which is used to create a smaller and faster design at the expense of flexibility in loading and unloading data. When the core is configured to use Realtime mode, the following occurs:

1. The TREADY signal on the Data Output channel (`m_axis_data_tready`) is removed
2. The TREADY signal on the Status channel (`m_axis_status_tready`) is removed
3. The TVALID signal on the Data Input channel is ignored once the loading of a frame has begun

The first two points mean that neither the downstream slave that consumes processed data, or the downstream slave that consumes status information, can insert waitstates using TREADY (`m_axis_data_tready` and `m_axis_status_tready` respectively) as the pins are not present on the core. Both slaves must be able to respond immediately on every clock cycle where the FFT is producing data (`m_axis_data_tvalid` asserted high or `m_axis_status_tvalid` asserted high). If the slave cannot respond immediately, then data will be lost.

The third point is slightly more complex as TVALID (`s_axis_data_tvalid`) cannot be removed. The upstream master still controls the start of a frame with TVALID. The FFT does not try to load a frame until the upstream master has asserted TVALID to provide the first symbol and there is no requirement for the master to supply the first sample of a frame at any particular time. However, once this has occurred, TVALID is then ignored by the FFT and it assumes that the master provides symbol data immediately on every clock cycle where TREADY is high. If the master does not provide data when requested, the data from the last provided symbol is reused and the `event_data_in_channel_halt` is asserted to show that the timing requirements have been violated. Please

note that the FFT can still insert waitstates when in Realtime mode. It is only the response to externally induced waitstates that changes.

Figure 28 shows the upstream master inserting waitstates while loading an 8 point frame in Realtime mode. At point A, the master has sent one sample to the Data Input Channel. The FFT then inserts a waitstate while it waits for the FFT processing core to start the transform. This is shown as one cycle here, but it could be longer in certain cases. At point B, the master inserts two waitstates using TVALID. However, the FFT ignores them and uses the previous data (D_3) for the missing data. It is likely that the processed frame will be corrupted.

At point C, the master starts supplying the last samples of the frame (D_7 and later D_8) but the FFT has already started processing the frame and inserts a waitstate. The Master and the FFT are now out of synchronisation. When the FFT finishes processing the frame and is ready for a new frame, it sees D_7 as the first symbol of the new frame and starts to consume another 8 samples.

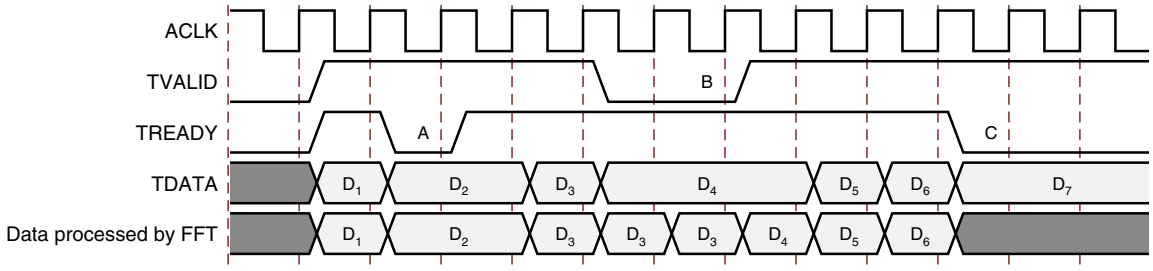

Figure 28: Incorrect transfer in Realtime mode

It is important that Realtime mode is only selected when the appropriate external masters and slaves can meet the timing requirements on supplying and consuming data.

Transfer Timing

The FFT starts to process a frame as soon as a) the upstream master asks it to by supplying data to process, and b) when it is able to. The chosen architecture and cyclic prefix insertion are the major configuration options that affect when the FFT is able to process a new frame.

The following timing diagrams are generalisations of actual behaviour used to show the broad phases the FFT moves through when processing frames, and how these phases can (or cannot) overlap. The lengths of the various phases are not to scale, and the processing time may be much longer than the time required to input or output a frame.

In particular, the behaviour of TREADY on the input data channel is not fully accurate as the Data Input channel buffers the data (16 symbols in Non-Realtime mode and 1 symbol in Realtime mode). However, this data waits in the buffer until the FFT processing core is ready for it. The Data Input channel's TREADY in these diagrams is used as an indication of when the FFT processing core wants data rather than when the AXI channel (with its buffer) wants data.

Pipelined Streaming I/O with no Cyclic Prefix Insertion

When Pipelined Streaming I/O is selected and no cyclic prefix is used, the FFT can overlap the loading of a frame with the processing and unloading of earlier frames. If the upstream master supplies the first symbol for a new frame immediately after the last symbol for the previous frame, the FFT starts loading it immediately.

Figure 29 shows the general timing for back-to-back frames in the Pipelined Streaming architecture.

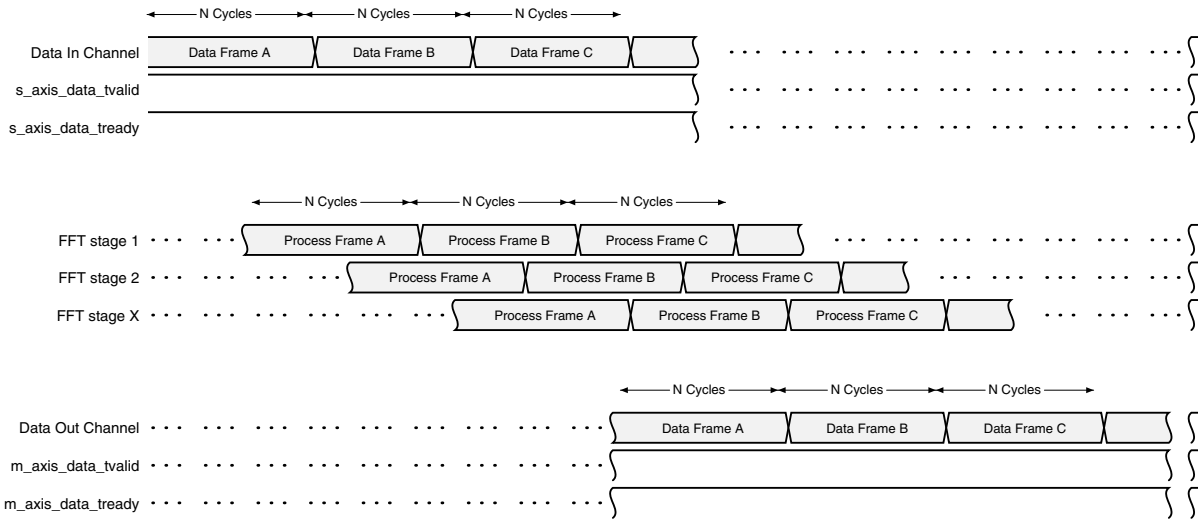

Figure 29: Transfer timing for entire frames in Pipelined Streaming I/O with no Cyclic Prefix Insertion

Note that there is a latency between a frame being loaded and the processed data for that frame being available. This latency depends on the options chosen in the GUI to parameterise the core. However, once that latency has passed, processed frames appear back-to-back.

Pipelined Streaming I/O with Cyclic Prefix Insertion

If cyclic prefix insertion is used, more samples are unloaded from the core than are loaded. Therefore, the core cannot continuously stream frames, but must insert a gap of cyclic prefix length clock cycles in between each frame of input data to accommodate the additional clock cycles required to unload the cyclic prefix (see Figure 30). This is indicated by the TREADY signal on the Data Input channel. This goes low to allow the FFT time to unload the cyclic prefix

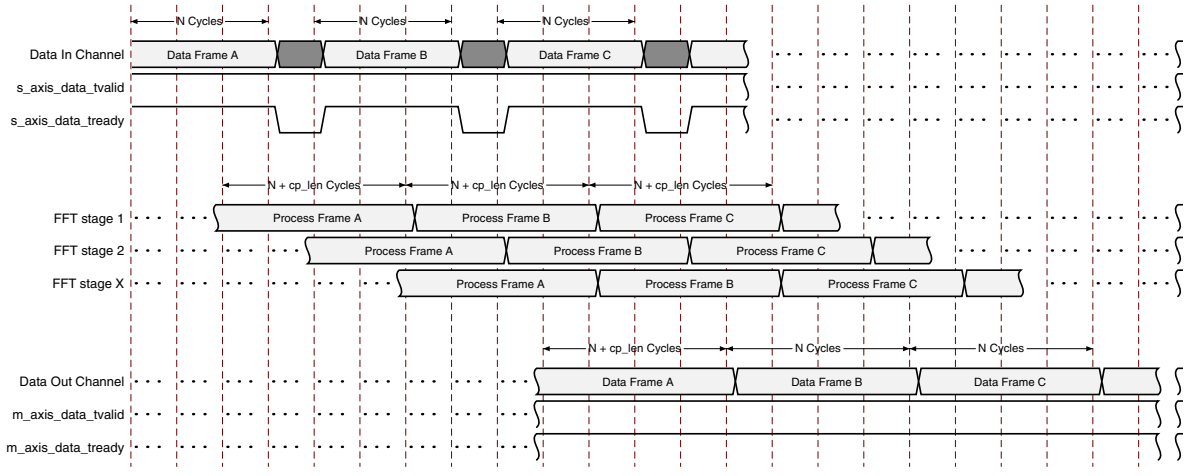

Figure 30: Transfer timing for entire frames in Pipelined Streaming I/O with Cyclic Prefix Insertion

Burst I/O Architectures

The Burst I/O architectures do not allow frame overlapping to the same degree as the Pipelined Streaming I/O architecture. When natural ordered outputs are used, a frame has to be processed and unloaded before the FFT can start to load the following frame[1]. When bit reversed outputs are used, the FFT only unloads data when a new frame is loaded. This means that the loading of frame N+1 overlaps with (and actually causes) the unloading of frame N. However, if the upstream master does not supply data to the FFT when it is ready to start unloading a frame, the FFT will flush the frame out manually. If this occurs, the loading and unloading phases do not overlap.

Figure 31 shows the general transform timing for a Burst I/O architecture with natural ordered outputs. This requires distinct load, process and unload phases. The upstream master is constantly attempting to stream data as is the downstream slave. These examples do not show the effect of a cyclic prefix, which is to extend the unloading phase.

The Upstream Master loads all of the data for Frame A into the Data Input Channel of the FFT. As the FFT is loading this data to process it, the buffer in the channel never fills. However, the master immediately starts sending data for Frame B. At point A in the waveform, the buffer in the Data Input channel fills, because the FFT is processing frame A and no longer draining the buffer. This can be seen externally as `s_axis_data_tready` going low. The Data Input Channel will remain in a slave waitstate situation, where the FFT cannot accept data from the upstream Master, until point B. At this point the FFT has unloaded frame A and started loading Frame B into the processing core. This drains the buffer in the Data Input Channel, which unblocks the Upstream Master and allows it to send the remaining data for Frame B. The situation then repeats itself with Frame C.

The important points here are:

1. Activity on the AXI interface to the Data Input channel does not necessarily correlate to the activity inside the FFT. For example, just before point A, the channel loads sample data for frame B yet the FFT is internally processing Frame A.
2. The Upstream Master cannot always stream frame data without reference to `s_axis_data_tready`.
3. The FFT unloads a frame before loading the subsequent frame.

Figure 32 is similar to Figure 31, except that the FFT is configured to have bit reversed outputs. As the upstream master is always supplying data, the loading and unloading of frames can overlap.

Figure 33 is similar to Figure 32, except that the upstream master does not supply data for Frame B until the FFT has started flushing out Frame A. As the FFT has already started flushing Frame A, it will complete this before loading Frame B. The loading and unloading of frames do not overlap.

In this example, `s_axis_data_tready` remains high at Point A. Loading Frame A into the FFT drained the buffer in the Data Input Channel, and as the Upstream Master didn't send any new data, the buffer is empty. The FFT is ready to accept new frame data at point A although it isn't able to do anything with it at this point. At point B the Upstream Master starts to send data from Frame B. This fills the buffer in the Data Input Channel, but as the FFT is committed to flushing Frame A, the buffer fills and the FFT stalls the Upstream Master with waitstates. At point C, the FFT has started loading Frame B to process it, so the buffer drains and more data can be accepted to finish off Frame B.

The key difference between the situation in Figure 32 and Figure 33 is that the master in Figure 32 has provided new frame data during the processing phase of the previous frame. As a result, the FFT knows there is a new frame

1. This refers to the FFT processing core. As the Data In channel has a 16 element deep buffer on its input, it can start to pre-buffer a frame while a frame is still being processed. In the case of 8 and 16 point FFTs, it can pre-buffer entire frames. However, this buffered data waits in the buffer until the FFT engine has finished dealing with the current frame.

coming so when processing finishes, it starts to load the new frame as this will flush the old frame out. In Figure 33, the master did not provide data (and therefore did not tell the FFT that there would be a new frame) during the processing phase, so when the FFT finishes processing the frame, it moves to a flushing phase where it is no longer possible to load a new frame. Even if the master provides a sample for the new frame a cycle after unloading has begin, that sample will not be loaded until the FFT is finished unloading the old frame.

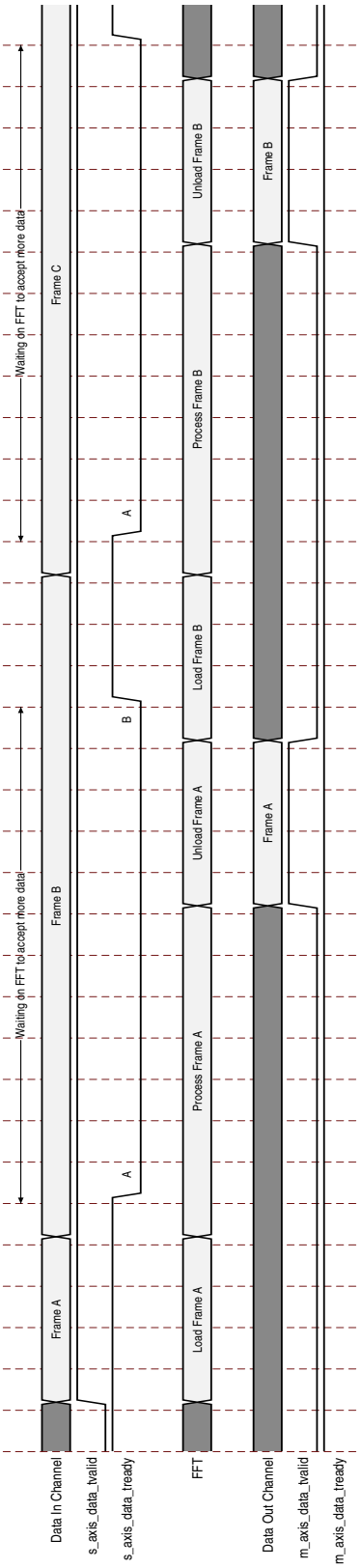

Figure 31: Transfer timing for entire frames in Burst I/O mode with natural ordered outputs

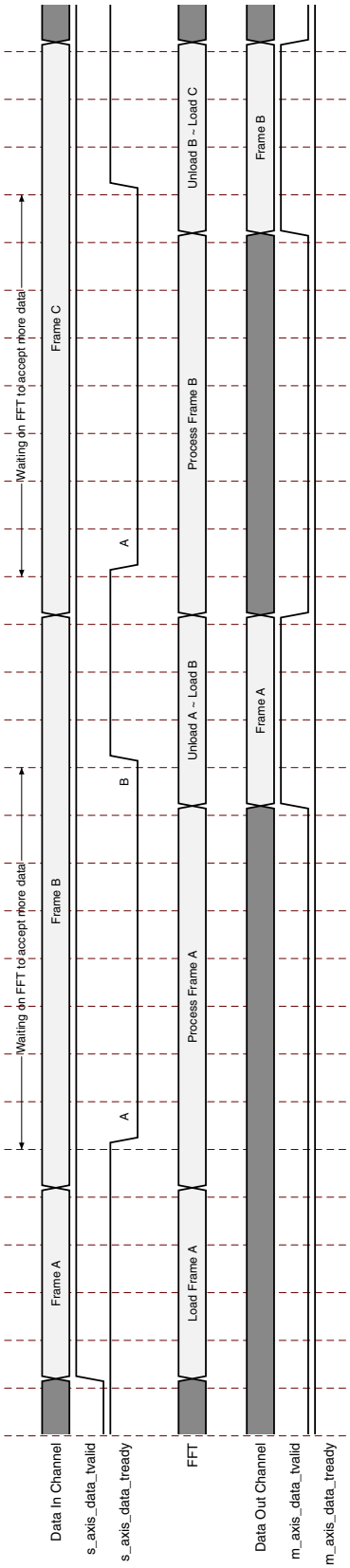

Figure 32: Transfer timing for entire frames in Burst I/O mode with bit reversed outputs

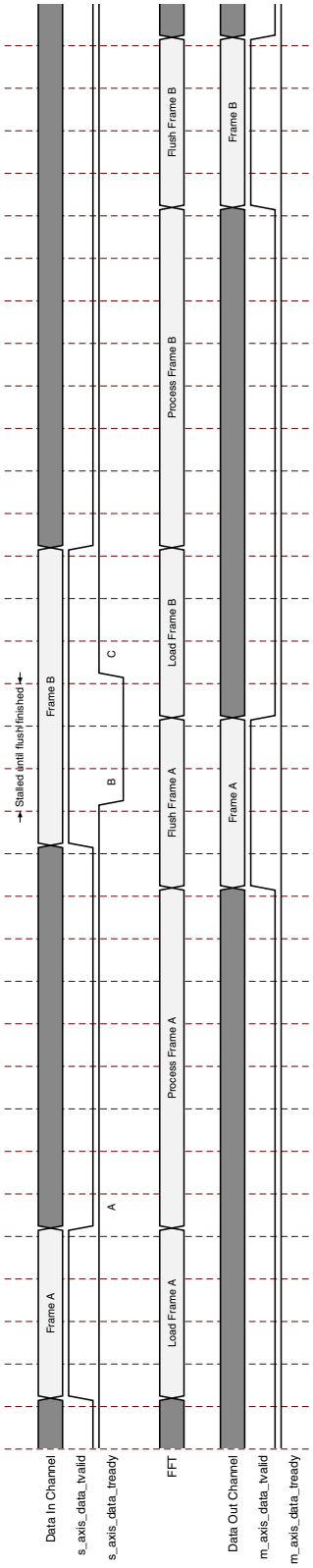

Figure 33: Transfer timing for entire frames in Burst I/O mode with bit reversed outputs when the FFT has to flush the frame itself

Configuring the FFT

FFT transforms are configured using the Configuration channel. Details about the configuration information carried in this channel, and how it is packed, is discussed in more detail in section Configuration Channel, page 48. When the FFT is ready to load a new frame for processing, it checks to see if a new configuration has been supplied on the Configuration channel. If it has, the FFT processing core is configured using that information before the frame is loaded. If no new configuration information has been supplied then the FFT processes the frame using the last configuration it had. If no configuration has ever been supplied, then the core defaults described in aresetn (Synchronous Clear), page 44 are used.

The process of applying configuration data to a particular frame depends on the current status of the FFT:

1. To apply a configuration to the very first frame after power on or after an idle period
2. To apply the configuration to the next frame in a sequence of frames

Applying a New Configuration While Idle

If the FFT core is idle (that is, it isn't loading, processing or unloading any frames), it waits for either frame data or configuration data to decide what action to take next. If new frame data is seen by the FFT control module without new configuration information being seen, then the FFT starts to process a frame using the existing configuration. If configuration information is seen before frame data, or on the same clock edge as frame data, then the configuration is applied to that frame.

To ensure that the configuration data is applied before the frame is processed, the configuration information should be written to the Configuration channel with the following timing:

- Realtime Mode: the write of configuration data to the Configuration channel must complete at least 1 clock cycle before the write of the first data to the Data Input channel. Failure to do so results in the frame being processed with the previous configuration options in use.
- Non-Realtime Mode: the write of configuration data to the Configuration channel can happen before or with the write of the first data to the Data Input channel.

Perhaps the easiest way to satisfy this in a system context is to configure the FFT before enabling the upstream data master.

Applying a New Configuration While Streaming Frames

Once the upstream master is active and sending frame data to the FFT core, it becomes difficult to use the previous technique to synchronise configuration with particular frames as data for a new frame may have already been loaded into the Data Input channel. The recommended way of synchronising configuration to frames is to use the `event_frame_started` signal.

This signal is asserted high when the FFT starts to load data for a frame into the FFT processing core. This is a known safe point to send configuration information for the next frame. Configuration data sent after this may or may not be applied to the subsequent frame, depending on the frame size and the latency between `event_frame_started` asserting and the configuration write occurring.

How Changing the Configuration can Change Transfer Timing

There are two situations where changing the configuration can temporarily reduce the throughput of the FFT core:

1. A Pipelined Streaming FFT is processing frames and the transform size (NFFT) is changed.
2. A Burst I/O FFT with bit reversed outputs is processing a frame, and the master supplies frame data in time to avoid the FFT automatically flushing the frame, and the transform size (NFFT) is changed.

Both the Pipelined Streaming architecture and the Burst I/O architectures (when bit reversed outputs are used) implement pipelining to achieve better throughput. In the case of the Pipelined Streaming architecture, it pipelines the loading, processing and unloading of entire frames (see Figure 29). In Burst I/O architectures when bit reversed outputs are used, the FFT implements a partial pipeline to overlap the loading on one frame with the unloading of another (see Figure 32).

However, a change to the transform size can only be applied when the pipeline is empty. Changing the transform size when the pipeline is not empty would result in data loss, so the FFT prevents this. When new configuration information is sent to the Configuration channel, and that information contains a change in transform size, the FFT will not load any more frames until all frames already in the pipeline are processed and unloaded.

This is all handled automatically by the FFT core, allowing the user to send the configuration information at any time they desire. However, throughput will drop until the pipeline is fully flushed. This behaviour only occurs if the transform size is to change. All other configuration options can be applied without waiting for the FFT's pipeline to empty.

Pinout

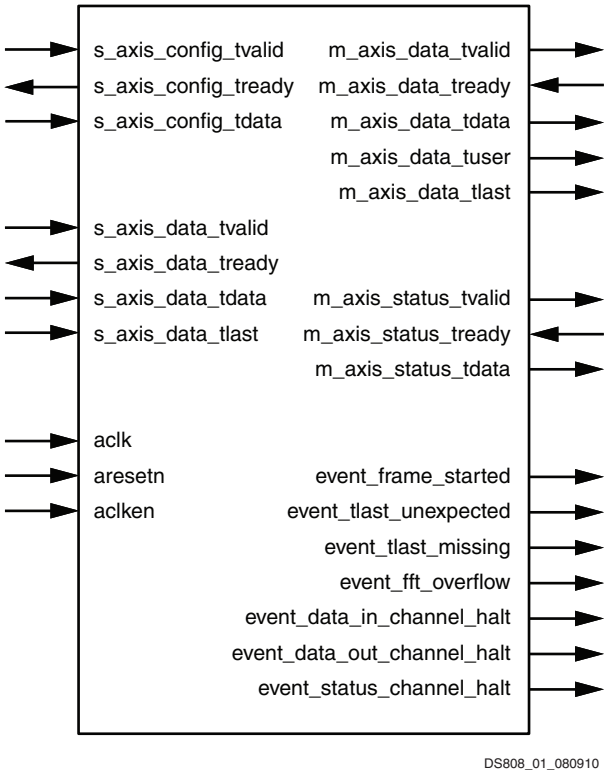

Figure 34: **Core Schematic Symbol**

This section describes the core ports as shown in Figure 34 and described in Table 3.

Table 3: **Core Signal Pinout**

Name	Direction	Optional	Description
aclk	Input	No	Rising-edge clock.
aclken	Input	Yes	Active-high clock enable (optional).
aresetn	Input	Yes	Active-low synchronous clear (optional, always take priority over aclken). A minimum aresetn active pulse of two cycles is required.
s_axis_config_tvalid	Input	No	TVALID for the Configuration channel. Asserted by the external master to signal that it is able to provide data.
s_axis_config_tready	Output	No	TREADY for the Configuration channel. Asserted by the FFT to signal that it is ready to accept data.
s_axis_config_tdata	Input	No	TDATA for the Configuration channel. Carries the configuration information: `CP_LEN`, `FWD/INV`, `NFFT` and `SCALE_SCH`. See Section Run-Time Transfer Configuration.
s_axis_data_tvalid	Input	No	TVALID for the Data Input channel. Used by the external master to signal that it is able to provide data.
s_axis_data_tready	Output	No	TREADY for the Data Input channel. Used by the FFT to signal that it is ready to accept data.

Table 3: **Core Signal Pinout *(Cont'd)***

Name	Direction	Optional	Description
s_axis_data_tdata	Input	No	TDATA for the Data Input channel. Carries the unprocessed sample data: XN_RE and XN_IM. See section Data Input Channel.
s_axis_data_tlast	Input	No	TLAST for the Data Input channel. Asserted by the external master on the last sample of the frame. This is not used by the FFT except to generate the events event_tlast_unexpected and event_tlast_missing events
m_axis_data_tvalid	Output	No	TVALID for the Data Output channel. Asserted by the FFT to signal that it is able to provide sample data.
m_axis_data_tready	Input	No	TREADY for the Data Output channel. Asserted by the external slave to signal that it is ready to accept data. Only present in "Non-Realtime" mode.
m_axis_data_tdata	Output	No	TDATA for the Data Output channel. Carries the processed sample data XK_RE and XK_IM. See section Data Output Channel.
m_axis_data_tuser	Output	No	TUSER for the Data Output channel. Carries additional per-sample information, such as XK_INDEX, OVFLO and BLK_EXP. See section Data Output Channel.
m_axis_data_tlast	Output	No	TLAST for the Data Output channel. Asserted by the FFT on the last sample of the frame.
m_axis_status_tvalid	Output	No	TVALID for the Status channel. Asserted by the FFT to signal that it is able to provide status data.
m_axis_status_tready	Input	No	TREADY for the Status channel. Asserted by the external slave to signal that it is ready to accept data. Only present in "Non-Realtime" mode
m_axis_status_tdata	Output	No	TDATA for the Status channel. Carries the status data: BLK_EXP or OVFLO. See section Status Channel.
event_frame_started	Output	No	Asserted when the FFT starts to process a new frame. See section event_frame_started.
event_tlast_unexpected	Output	No	Asserted when the FFT sees s_axis_data_tlast high on a data sample that isn't the last one in a frame. See section event_tlast_unexpected.
event_tlast_missing	Output	No	Asserted when s_axis_data_tlast is low on the last data sample of a frame. See section event_tlast_missing.
event_fft_overflow	Output	No	Asserted when an overflow is seen in the data samples being unloaded from the Data Output channel. Only present when overflow is a valid option. See section event_fft_overflow.
event_data_in_channel_halt	Output	No	Asserted when the FFT requests data from the Data Input channel and none is available. See section event_data_in_channel_halt.

Table 3: **Core Signal Pinout *(Cont'd)***

Name	Direction	Optional	Description
event_data_out_channel_halt	Output	No	Asserted when the FFT tries to write data to the Data Output channel and it is unable to do so. Only present in "Non-Realtime" mode. See section event_data_out_channel_halt.
event_status_channel_halt	Output	No	Asserted when the FFT tries to write data to the Status channel and it is unable to do so. Only present in "Non-Realtime" mode. See section event_status_channel_halt.

Note that all AXI4-Stream port names are lower case, but for ease of visualization, upper case is used in this document when referring to port name suffixes, such as TDATA or TLAST.

CORE Generator Graphical User Interface

The FFT core graphical user interface (GUI) provides several screens with fields to set the parameter values for the particular instantiation required. A description of each CORE Generator GUI field follows:

Page 1

- **Component Name**: The name of the core component to be instantiated. The name must begin with a letter and be composed of the following characters: a to z, A to Z, 0 to 9, and "_".
- **Channels**: Select the number of channels from 1 to 12. Multichannel operation is available for the three Burst I/O architectures.
- **Transform Length**: Select the desired point size. All powers of two from 8 to 65536 are available.
- **Implementation Options**: Select an implementation option, as described in Architecture Options, page 14.
  - The Pipelined Streaming I/O, Radix-2 Burst I/O, and Radix-2 Lite Burst I/O architectures support point sizes 8 to 65536.
  - The Radix-4 Burst I/O architecture supports point sizes 64 to 65536.
  - Check Automatically Select to choose the smallest implementation that meets the specified Target Data Throughput, provided the specified Target Clock Frequency is achieved when the FFT core is implemented on an FPGA device.
  - Target Clock Frequency and Target Data Throughput are only used to automatically select an implementation and to calculate latency. The core is not guaranteed to run at the specified target clock frequency or target data throughput.
- **Transform Length Options**: Select the transform length to be run-time configurable or not. The core uses fewer logic resources and has a faster maximum clock speed when the transform length is not run-time configurable.

Page 2

- **Data Format**: Select whether the input and output data samples are in Fixed Point format, or in IEEE-754 single precision (32-bit) Floating-Point format. Floating-Point format is not available when the core is in a multichannel configuration.
- **Precision Options**: Input data and phase factors can be independently configured to widths from 8 to 34 bits, inclusive. When the Data Format is Floating-Point, the input data width is fixed at 32 bits and the phase factor width can be set to 24 or 25 bits depending on the noise performance required and available resources.
- **Scaling Options**: Three options are available, for all architectures:
  - Unscaled
    - All integer bit growth is carried to the output. This can use more FPGA resources.

  - Scaled
    - A user-defined scaling schedule determines how data is scaled between FFT stages.
  - Block Floating-Point
    - The core determines how much scaling is necessary to make best use of available dynamic range, and reports the scaling factor as a block exponent.
- **Control Signals**: Clock Enable (`aclken`) and Synchronous Clear (`aresetn`) are optional pins. Synchronous Clear overrides Clock Enable if both are selected. If an option is not selected, some logic resources may be saved and a higher clock frequency may be attainable.
- **Optional Output Fields**: `XK_INDEX` is an optional field in the Data Output Channel. `OVFLO` is an optional field in both the Data Output channel and Status Channel.
- **Throttle Schemes**: Select trade off between performance and data timing requirements. Realtime mode typically gives a smaller and faster design, but has strict constraints on when data must be provided and consumed. Non-Realtime mode has no such constraints, but the design may be larger and slower. See Controlling the FFT for more details.
- **Rounding Modes**: At the output of the butterfly, the LSBs in the data path need to be trimmed. These bits can be truncated or rounded using convergent rounding, which is an unbiased rounding scheme. When the fractional part of a number is equal to exactly one-half, convergent rounding rounds up if the number is odd, and rounds down if the number is even. Convergent rounding can be used to avoid the DC bias that would otherwise be introduced by truncation after the butterfly stages. Selecting this option increases slice usage and yields a small increase in transform time due to additional latency.
- **Output Ordering**: Output data selections are either Bit/Digit Reversed Order or Natural Order. The Radix-2 based architectures (Pipelined Streaming I/O, Radix-2 Burst I/O and Radix-2 Lite Burst I/O) offer bit-reversed ordering, and the Radix-4 based architecture (Radix-4 Burst I/O) offers digit-reversed ordering. For the Pipelined Streaming I/O architecture, selecting natural order output ordering results in an increase in memory used by the core. For Burst I/O architectures, selecting natural order output increases the overall transform time because a separate unloading phase is required.
  - Cyclic Prefix Insertion can be selected if the output ordering is Natural Order. Cyclic Prefix Insertion is available for all architectures, and is typically used in OFDM wireless communications systems.

Page 3

- **Memory Options**:
  - **Data And Phase Factors (Burst I/O architectures)**: For Burst I/O architectures, either block RAM or distributed RAM can be used for data and phase factor storage. Data and phase factor storage can be in distributed RAM for all point sizes up to and including 1024 points.
  - **Data And Phase Factors (Pipelined Streaming I/O)**: In the Pipelined Streaming I/O solution, the data can be stored partially in block RAM and partially in distributed RAM. Each pipeline stage, counting from the input side, uses smaller data and phase factor memories than preceding stages. You can select the number of pipeline stages that use block RAM for data and phase factor storage. Later stages use distributed RAM. The default displayed on the GUI offers a good balance between both. If output ordering is Natural Order, the memory used for the reorder buffer can be either block RAM or distributed RAM. The reorder buffer can use distributed RAM for point sizes less than or equal to 1024.
    - When block floating-point is selected for the Pipelined Streaming I/O architecture, a RAM buffer is required for natural order *and* bit reversed order output data. In this case, the reorder buffer options remain available and distributed RAM may be selected for all point sizes below 2048.
  - **Hybrid Memories**: Where data, phase factor, or reorder buffer memories are stored in block RAM, if the size of the memory is greater than one block RAM, the memory can be constructed from a hybrid of block RAMs and distributed RAM, where the majority of the data is stored in block RAMs and a few bits that are left over are stored in distributed RAM. This Hybrid Memory is an alternative to constructing the memory entirely from multiple block RAMs. It provides a reduction in the block RAM count, at the cost of an

increase in the number of slices used. Hybrid Memories are only available when block RAM is used for one or more memories and the number of slices required for a Hybrid Memory implementation is below an internal threshold of 256 LUTs per memory. If these conditions are met, Hybrid Memories are made available and can be selected.

- **Optimize Options**:
  - **Complex Multipliers**: Three options are available for customization of the complex multiplier implementation:
    - **Use CLB logic**: All complex multipliers are constructed using slice logic. This is appropriate for target applications that have low performance requirements, or target devices that have few XtremeDSP slices.
    - **Use 3-multiplier structure (resource optimization)**: All complex multipliers use a three real multiply, five add/subtract structure, where the multipliers use XtremeDSP slices. This reduces the XtremeDSP slice count, but uses some slice logic. This structure can make use of the XtremeDSP slice pre-adder to reduce or remove the need for extra slice logic, and improve performance.
    - **Use 4-multiplier structure (performance optimization)**: All complex multipliers use a four real multiply, two add/subtract structure, utilizing XtremeDSP slices. This structure yields the highest clock performance at the expense of more dedicated multipliers. In devices with XtremeDSP slices, the add/subtract operations are implemented within the XtremeDSP slices.

    Note: The core may override the complex multiplier implementation internally to ensure the fewest number of XtremeDSP slices are used, without impacting performance. For this reason, some core configurations may show no difference in XtremeDSP slice usage when toggling between the 3-multiplier and 4-multiplier options. If “Use CLB logic” is selected, however, slice logic is always utilized.
  - **Butterfly Arithmetic**: Two options are available for customization of the butterfly implementation:
    - **Use CLB logic**: All butterfly stages are constructed using slice logic.
    - **Use XtremeDSP Slices:** For devices with XtremeDSP slices, this option forces all butterfly stages to be implemented using the adder/subtracters in XtremeDSP slices.

Information Tabs

- Implementation Details:
  - **Implementation**: This field displays the currently selected architecture. This is useful to see the result of automatic architecture selection.
  - **Transform Size**: When the transform length is run-time configurable, the core has the ability to reprogram the point size while the core is running; that is, the core can support the selected point size and any smaller point size. This field displays the supported point sizes based on the Transform Length, Transform Length Options, and the Implementation Options selected.
  - **Output Data Width**: The output data width equals the input data width for scaled arithmetic and block floating-point arithmetic. With unscaled arithmetic, the output data width equals (input data width + $\log_2$(point size) + 1).
  - **Resource Estimates**: Based on the options selected, this field displays the XtremeDSP slice count and 18K block RAM numbers (9K block RAM numbers for Spartan-6 devices). The resource numbers are just an estimate. For exact resource usage, and slice/LUT-FlipFlop pair information, a MAP report should be consulted.
  - **AXI4-Stream Port Structure:** This section shows how the FFT's fields are mapped to the AXI channels. This information can be copied to the Clipboard and pasted as plain text into other applications.
- **Latency:**
  - This tab shows the latency of the FFT core in clock cycles and microseconds (µs) for each point size supported. The latency is from the Upstream Master supplying the first sample of a frame to the last sample of output data coming out of the core, assuming that the FFT core was idle and neither the

Upstream Master or the Downstream Slave inserted wait states. This is not the minimum number of cycles between starting consecutive frames, as frames may overlap in some cases. The latency in microseconds is based on the target clock frequency. The latency figures can be copied to the Clipboard and pasted as plain text into other applications.

- **C Model**:
  - This tab provides a link to the Xilinx LogiCORE IP FFT web page where the core C model can be downloaded. For details of the C model, see Bit Accurate C Model, page 38.

Using the FFT IP Core

Simulation Models

When the core is generated using the CORE Generator software, a UniSim-based simulation model is created. The FFT core does not have a VHDL or Verilog functional behavioral model. For this reason, the core overrides the CORE Generator Project Options and always delivers a Structural model type.

Xilinx recommends that the designer run simulations using a resolution of 1 ps. Some Xilinx library components require a 1 ps resolution to work properly in either functional or timing simulation. The FFT core UniSim-based structural model may produce incorrect results if simulated with a resolution other than 1 ps. See the "Register Transfer Level (RTL) Simulation Using Xilinx Libraries" section in Chapter 6 of the *Synthesis and Simulation Design Guide* for more information. This document is part of the ISE® Design Suite Manual set available at www.xilinx.com/support/software_manuals.htm.

XCO Parameters

Table 4 defines valid entries for the XCO parameters. Parameters are not case sensitive. Default values are displayed in bold. Xilinx strongly recommends that XCO parameters are not manually edited in the XCO file; instead, use the CORE Generator GUI to configure the core and perform range and parameter value checking.

Table 4: **XCO Parameters**

XCO Parameter	Valid Values
component_name	Name must begin with a letter and be composed of the following characters: a to z, A to Z, 0 to 9, and "_".
channels	1 - 12 (default value is **1**)
transform_length	8, 16, 32, 64, 128, 256, 512, **1024**, 2048, 4096, 8192, 16384, 32768, 65536
implementation_options	**automatically_select** pipelined_streaming_io radix_4_burst_io radix_2_burst_io radix_2_lite_burst_io
target_clock_frequency	0 - 550 (default is **250**)
target_data_throughput	0 - 550 (default is **50**)
run_time_configurable_transform_length	**false** true
data_format	**fixed_point** floating_point
input_width	8 - 34 (default value is **16**)
phase_factor_width	8 - 34 (default value is **16**)
scaling_options	**scaled** unscaled block_floating_point
rounding_modes	**truncation** convergent_rounding
aclken	**false** true
aresetn	**false** true
ovflo	**false** true
xk_index	**false** true
throttle_scheme	**nonrealtime** realtime
output_ordering	**bit_reversed_order** natural_order
cyclic_prefix_insertion	**false** true

Table 4: **XCO Parameters *(Cont'd)***

XCO Parameter	Valid Values
memory_options_data	**block_ram** distributed_ram
memory_options_phase_factors	**block_ram** distributed_ram
memory_options_reorder	**block_ram** distributed_ram
number_of_stages_using_block_ram_for_data_and_phase_factors	0 - 11 (default value depends on transform length)
memory_options_hybrid	**false** true
complex_mult_type	use_luts **use_mults_resources** use_mults_performance
butterfly_type	**use_luts** use_xtremedsp_slices

Bit Accurate C Model

The FFT core has a bit-accurate C model designed for system modeling and selecting parameters before generating an FFT core. The model is bit-accurate but not cycle-accurate, so it produces exactly the same output data as the core on a frame-by-frame basis. However, it does not model the core latency or its interface signals.

The C model is generally required before generating an FFT core, so it is not delivered as an output of CORE Generator software. Instead it is available for download on the Xilinx LogiCORE IP FFT web page at www.xilinx.com/products/ipcenter/FFT.htm. The C model is available as a dynamically-linked library for 32-bit and 64-bit Windows platforms, and 32-bit and 64-bit Linux platforms. The C model may also be compiled into a MATLAB software MEX function. Download a zip file and unzip it to install the C model. A README.txt file describes the contents of the installed directory structure, and any further platform-specific installation instructions.

C Model Interface

The C model is used through three functions, declared in the header file `xfft_v8_0_bitacc_cmodel.h`:

```
struct xilinx_ip_xfft_v8_0_state*
xilinx_ip_xfft_v8_0_create_state(struct xilinx_ip_xfft_v8_0_generics generics);

int xilinx_ip_xfft_v8_0_bitacc_simulate
(
 struct xilinx_ip_xfft_v8_0_state*   state,
 struct xilinx_ip_xfft_v8_0_inputs   inputs,
 struct xilinx_ip_xfft_v8_0_outputs* outputs
 );

void xilinx_ip_xfft_v8_0_destroy_state(struct xilinx_ip_xfft_v8_0_state* state);
```

The first function, `xilinx_ip_xfft_v8_0_create_state`, creates a new state structure for the FFT C model, allocating memory to store the state as required, and returns a pointer to that state structure. The state structure contains all information required to define the FFT being modeled. The function is called with a structure

containing the core generics: these are all of the parameters that define the bit-accurate numerical performance of the core, represented as integers, and are derived from the XCO parameters that are the result of selections in the CORE Generator GUI. The generics required for the C model and their mappings from XCO parameters are shown in Table 5.

Table 5: **C Model Generics**

Generic	Description	Range	XCO parameter and mapping
C_NFFT_MAX	$\log_2$(maximum point size)	3-16	transform_length: take $\log_2$
C_ARCH	Architecture	1-4	implementation_options: radix_4_burst_io => 1, radix_2_burst_io => 2, pipelined_streaming_io => 3, radix_2_lite_burst_io => 4
C_HAS_NFFT	Run-time configurable transform length	0,1	run_time_configurable_transform_length: false => 0, true => 1
C_INPUT_WIDTH	Input data width (bits)	8-34	input_width
C_TWIDDLE_WIDTH	Phase factor width (bits)	8-34	phase_factor_width
C_USE_FLT_PT	Input/output data format	0,1	data_format: fixed_point => 0, floating_point => 1
C_HAS_SCALING	Scaling option: unscaled or not. Ignored when C_USE_FLT_PT = 1	0,1	scaling_options: unscaled => 0, scaled / block_floating_point => 1
C_HAS_BFP	Scaling option: if not unscaled, scaled or block floating-point. Ignored when C_USE_FLT_PT = 1	0,1	scaling_options: unscaled / scaled => 0, block_floating_point => 1
C_HAS_ROUNDING	Rounding mode. Ignored when C_USE_FLT_PT = 1	0,1	rounding_modes: truncation => 0, convergent_rounding => 1

After a state structure has been created, it can be used as many times as required to simulate the FFT core. A simulation is run using the second function, `xilinx_ip_xfft_v8_0_bitacc_simulate`. Call this function with the pointer to the existing state structure, and structures to hold the inputs and outputs of the C model. These input and output structures are fully defined and described in the C model header file. Note that memory for all input and output data arrays must be allocated by the calling program before simulating the C model.

Finally, the state structure must be destroyed to free up any memory used to store the state, using the third function, `xilinx_ip_xfft_v8_0_destroy_state`, called with the pointer to the existing state structure.

If the generics of the core need to be changed, destroy the existing state structure and create a new state structure using the new generics. There is no way to change the generics of an existing state structure.

An example C++ file, `run_bitacc_cmodel.c`, is included in the C model zip file. This shows all of the stages required to run the C model.

Due to differences between the FFT core and the C model in the order of operations within the processing phase, when using the Pipelined Streaming I/O architecture, if fixed-point data is being processed, the scaling option is Scaled and overflow occurs, the `xk_re` and `xk_im` data outputs of the C model may not match the `XK_RE` and `XK_IM` data outputs of the core. The overflow output of the C model and the `OVFLO` output of the core (if present) do match in all cases. The overflow output of the C model is always set correctly when the scaling option is Scaled (when the C model generics `C_HAS_SCALING` = 1 and `C_HAS_BFP` = 0).

Therefore, Xilinx recommends that the overflow output of the C model is always checked when the scaling option is Scaled and the architecture is Pipelined Streaming I/O, and if overflow has occurred (overflow output = 1), the `xk_re` and `xk_im` outputs of the C model are ignored. This is the only case where the C model is not entirely bit-accurate to the core.

Using the C Model to Select a Scaling Schedule

When the scaling option for the FFT core is Scaled, you have great flexibility to set the scaling schedule that determines by how much to scale data values at each stage of the FFT processing phase. See Forward/Inverse and Scaling Schedule, page 18. It can be difficult to choose the best scaling schedule that avoids overflow in a sufficiently large proportion of frames for a particular type of input data. The C model is a tool that can help with the selection of a scaling schedule. A process for this is as follows:

1. Create a set of frames of typical FFT input data for the intended application.
2. Create a state structure using the required generics. Set the scaling option to Scaled by setting the C model generics `C_HAS_SCALING` = 1 and `C_HAS_BFP` = 0.
3. Set the scaling schedule in the structure of inputs to some initial scaling schedule, such as the reset value of 1 in each stage for Radix-2 Burst I/O and Radix-2 Lite Burst I/O architectures, or 2 in each stage for Radix-4 Burst I/O, and Pipelined Streaming I/O architectures.
4. Simulate the C model with each frame of typical input data in turn. Count the number of frames in which overflow occurred (overflow output was 1).
5. If the percentage of frames in which overflow occurred is lower than the acceptable overflow rate, reduce the scaling value in one or more stages in the scaling schedule. If the percentage of frames in which overflow occurred is higher than the acceptable overflow rate, increase the scaling value in one or more stages in the scaling schedule.
6. Repeat stages 4 and 5 until the percentage of frames in which overflow occurred matches the acceptable overflow rate.

This process produces a scaling schedule that is tailored to the typical FFT input data for the intended application.

Demonstration Testbench

When the core is generated using CORE Generator, a demonstration testbench is created. This is a simple VHDL testbench that exercises the core.

The demonstration testbench source code is one VHDL file: `demo_tb/tb_<component_name>.vhd` in the CORE Generator output directory. The source code is comprehensively commented.

Using the Demonstration Testbench

The demonstration testbench instantiates the generated FFT core. If the CORE Generator project options were set to generate a structural model, a VHDL or Verilog netlist named `<component_name>.vhd` or `<component_name>.v` was generated. If this file is not present, generate it using the netgen program, for example in Unix:

```
netgen -sim -ofmt vhdl <component_name>.ngc <component_name>.vhd
```

Compile the netlist and the demonstration testbench into the work library (see your simulator documentation for more information on how to do this). Then simulate the demonstration testbench. View the testbench's signals in your simulator's waveform viewer to see the operations of the testbench.

The Demonstration Testbench in Detail

The demonstration testbench performs the following tasks:

- Instantiates the core
- Generates an input data frame consisting of one or the sum of two complex sinusoids
- Generates a clock signal
- Drives the core's input signals to demonstrate core features (see below for details)
- Checks that the core's output signals obey AXI protocol rules (data values are not checked in order to keep the testbench simple)
- Provides signals showing the separate fields of AXI TDATA and TUSER signals

The demonstration testbench drives the core's input signals to demonstrate the features and modes of operation of the core. This includes performing an FFT on a pre-generated input data frame. The input data frame consists of a complex sinusoid with a frequency of 2.6 times the frame size. The FFT of this input frame is a peak centred between output samples 2 and 3. For FFTs with a maximum point size of 64 or greater, the input data is modified by adding a second complex sinusoid with a frequency of 23.2 times the frame size and a quarter of the magnitude of the first sinusoid. This modifies the FFT by adding a smaller peak centred between output samples 23 and 24. The testbench captures this output frame and uses it as the input frame for an inverse transform. The output of this inverse transform is therefore the same as the original input frame (modified by the scaling and finite precision effects of the FFT core).

The operations performed by the demonstration testbench are appropriate for the configuration of the generated core, and are a subset of the following operations:

- Frame 1: drive a frame of pre-generated input data
- Frame 2: configure an inverse transform; drive the output of frame 1 as a frame of input data
- Configure frame 3: a forward transform while the previous transform is running
- Frame 3: drive the output of frame 2 as a frame of input data; de-assert AXI TVALID (and TREADY if present) signals occasionally to demonstrate AXI handshaking
- If ARESETn present: start another frame but reset the core before it completes

- Frames 4-7: run these back-to-back, as quickly as possible:
  - Queue up configurations for a forward transform (frame 4) followed by a reverse transform (frame 5), both with a smaller point size (if point size is configurable) and a short cyclic prefix (if available)
  - Frame 4: drive a frame of pre-generated input data
  - Frame 5: drive the output of frame 1 as a frame of input data; simultaneously configure frame 6: a forward transform with maximum point size, a longer cyclic prefix (if available) and a zero scaling schedule (if fixed scaling is used)
  - Frame 6: drive a frame of pre-generated input data; simultaneously configure frame 7: an inverse transform with maximum point size, no cyclic prefix and default scaling schedule (if fixed scaling is used)
  - Frame 7: drive the output of frame 1 as a frame of input data
- Wait until all frames are complete

Customizing the Demonstration Testbench

It is possible to modify the demonstration testbench to drive the core's inputs with different data or to perform different operations.

Input data is pre-generated in the `create_ip_table` function and stored in the `IP_DATA` constant. New input data frames can be added by defining new functions and constants. Make sure that each input data frame is of the `T_IP_TABLE` array type.

All operations performed by the demonstration testbench to drive the core's inputs are done in the `data_stimuli` process. This process also contains procedures to simplify driving a frame of input data. Configuration is requested in this process by setting `cfg_*` signals to the desired configuration and setting the `do_config` shared variable to either `IMMEDIATE` or `AFTER_START`. The configuration signals are actually driven by the `config_stimuli` process.

The `data_stimuli` process is comprehensively commented, to explain clearly what is being done. New configuration and data operations can be added by copying and modifying sections of this process.

The clock frequency of the core can be modified by changing the `CLOCK_PERIOD` constant.

System Generator For DSP Graphical User Interface

This section describes each tab of the System Generator GUI and details the parameters that differ from the CORE Generator GUI. See CORE Generator Graphical User Interface, page 33 for more detailed information about all other parameters.

Tab 1: Basic

The Basic tab is used to specify the transform configuration and architecture in a similar way to page 1 of the CORE Generator GUI.

Implementation Options: Select an implementation option as described in Architecture Options, page 14.

- The Pipelined Streaming I/O, Radix-2 Burst I/O, and Radix-2 Lite Burst I/O architectures support point sizes 8 to 65536.
- The Radix-4 Burst I/O architecture supports point sizes 64 to 65536.

System Generator supports only single-channel implementation of the FFT and, hence, Channels is not available as a GUI option.

Tab 2: Advanced

The Advanced tab is used to specify phase factor precision, scaling, rounding, optional output fields, throttle scheme, and optional port options in a similar way to page 2 of the CORE Generator GUI.

System Generator can optionally shorten the AXI4-Stream signal names on the symbol by removing the m_axis_ or s_axis_ prefixes.

System Generator automatically sets the Input Data Width parameter based on the signal properties of the `XN_RE` and `XN_IM` ports. System Generator supports only fixed-point data types and, hence, Data Format is not available as an option on the GUI.

Tab 3: Implementation

The Implementation tab is used to specify memory and optimization options in a similar way to page 3 of the CORE Generator GUI.

- **Number of stages using block RAM**: Specifies the number of stages for the Pipelined Streaming I/O architecture that uses block RAM for data and phase factor storage. As dynamic list boxes are not offered with the System Generator GUI, this option displays the full range (0 to 11) selection, but allows you to select only valid values as visible in the CORE Generator GUI.
- **FPGA Area Estimation**: See the System Generator documentation for detailed information about this option.

Control Signals

aclken (Clock Enable)

If the Clock Enable (`aclken`) pin is present on the core, driving the pin low pauses the core in its current state. All logic within the core is paused. Driving the `aclken` pin high allows the core to continue processing.

Note that `aclken` can reduce the maximum frequency that the core can run at.

aresetn (Synchronous Clear)

If the `aresetn` pin is present on the core, driving the pin low results in all output pins, internal counters, and state variables being reset to their initial values. All pending load processes, transform calculations, and unload processes stop and are re-initialized. NFFT is set to the largest FFT point size permitted (the Transform Length value set in the GUI). The scaling schedule is set to 1/N. For the Radix-4 Burst I/O and Pipelined Streaming I/O architectures with a non-power-of-four point size, the last stage has a scaling of 1, and the rest have a scaling of 2. See Table 6.

Table 6: **Synchronous Clear Reset Values**

Signal	Initial / Reset Value
NFFT	maximum point size = *N*
FWD_INV	Forward = 1
SCALE_SCH	1/*N* [10 10... 10] for Radix-4 Burst I/O or Pipelined Streaming I/O architectures when *N* is a power of 4. [01 10... 10] for Radix-4 Burst I/O or Pipelined Streaming I/O architectures when *N* is not a power of 4. [01 01... 01] for Radix-2 Burst I/O or Radix-2 Lite Burst I/O architectures

The `aresetn` pin takes priority over `aclken`. If `aresetn` is asserted, reset occurs regardless of the value of `aclken`. A minimum `aresetn` active pulse of two cycles is required, since the signal is internally registered for performance. A pulse of one cycle resets the core, but the response to the pulse is not in the cycle immediately following.

Event Signals

The FFT core provides some real-time non-AXI signals to report information about the core's status. These event signals are updated on a clock cycle by clock cycle basis, and are intended for use by reactive components such as interrupt controllers. These signals are not optionally configurable from the GUI, but are removed by synthesis tools if left unconnected.

event_frame_started

This event signal is asserted for a single clock cycle when the FFT starts to process a new frame. This signal is provided to allow users to count frames and to synchronise the configuration of the core to a particular frame if required.

event_tlast_missing

This event signal is asserted for a single clock cycle when `s_axis_data_tlast` is low on a frame's last incoming data sample. This is intended to show a configuration mismatch between the FFT and the upstream data source with regard to the frame size, and indicates that the upstream data source is configured to a larger point size than the FFT is.

This is only calculated when the FFT starts processing a frame, so the event can lag the missing `s_axis_data_tlast` by a large number of clock cycles.

event_tlast_unexpected

This event signal is asserted for a single clock cycle when the FFT sees `s_axis_data_tlast` high on any incoming data sample that isn't the last one in a frame.

This is intended to show a configuration mismatch between the FFT and the upstream data source with regard to the frame size, and indicates that the upstream data source is configured to a smaller point size than the FFT is.

This is only calculated when the FFT starts processing a frame, so the event can lag the unexpected high on `s_axis_data_tlast` by a large number of clock cycles.

If there are multiple unexpected highs on `s_axis_data_tlast` for a frame, then this is asserted for each of them.

event_fft_overflow

This event signal is asserted on every clock cycle when an overflow is seen in the data samples being transferred on `m_axis_data_tdata`.

It is only possible to get FFT overflows when scaled arithmetic or single-precision floating-point I/O is used. In all other configurations the pin is removed from the core.

event_data_in_channel_halt

This event is asserted on every cycle where the FFT needs data from the Data Input channel and no data is available.

- In Realtime Mode the FFT continues processing the frame even though it is unrecoverably corrupted.
- In Non-Realtime Mode, FFT processing halts and only continues when data is written to the Data Input channel. The frame is not corrupted.

In both modes the event remains asserted until data is available in the Data Input Channel.

event_data_out_channel_halt

This event is asserted on every cycle where the FFT needs to write data to the Data Output channel but cannot because the buffers in the channel are full. When this occurs, the FFT core is halted and all activity stops until space is available in the channel's buffers. The frame is not corrupted.

The event pin is only available in Non-Realtime mode.

event_status_channel_halt

This event is asserted on every cycle where the FFT needs to write data to the Status channel but cannot because the buffers on the channel are full. When this occurs, the FFT core is halted, and all activity stops until space is available in the channel's buffers. The frame is not corrupted

The event pin is only available in Non-Realtime mode.

AXI4-Stream Considerations

The conversion to AXI4-Stream interfaces brings standardization and enhances interoperability of Xilinx IP LogiCORE solutions. Other than general control signals such as `aclk`, `aclken` and `aresetn`, and event signals, all inputs and outputs to the FFT are conveyed via AXI4-Stream channels. A channel always consists of TVALID and TDATA plus additional ports (such as TREADY, TUSER and TLAST) when required and optional fields. Together, TVALID and TREADY perform a handshake to transfer a message, where the payload is TDATA, TUSER and TLAST. The FFT operates on the operands contained in the TDATA fields and outputs the result in the TDATA field of the output channel.

For further details on AXI4-Stream Interfaces see the Xilinx AXI Design Reference Guide (UG761) and the AMBA 4 AXI4-Stream Protocol Version: 1.0 Specification.

Basic Handshake

Figure 35 shows the transfer of data in an AXI4-Stream channel. TVALID is driven by the source (master) side of the channel and TREADY is driven by the receiver (slave). TVALID indicates that the value in the payload fields (TDATA, TUSER and TLAST) is valid. TREADY indicates that the slave is ready to receive data. When both TVALID and TREADY are true in a cycle, a transfer occurs. The master and slave will set TVALID and TREADY respectively for the next transfer appropriately.

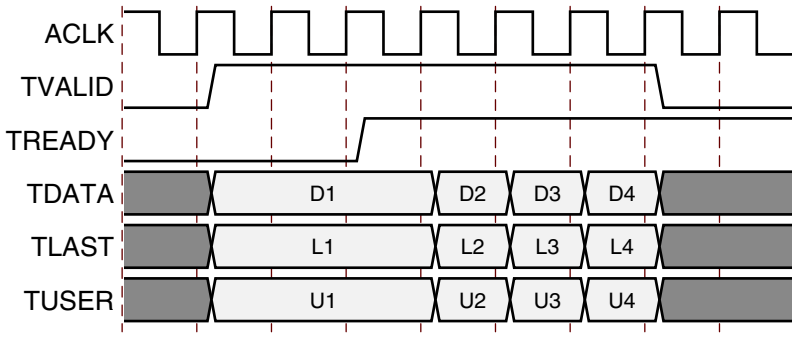

Figure 35: **Data Transfer in an AXI-Stream Channel**

AXI Channel Rules

Note that all of the AXI channels follow the same rules:

- All TDATA and TUSER fields are packed in little endian format. That is, bit 0 of a sub-field is aligned to the same side as bit 0 of TDATA or TUSER
- Fields are not included in TDATA or TUSER unless the core is configured in such a way that it needs the fields to be present. For example, if the FFT is configured to have a fixed point size, no bits are allocated to the NFFT field that specifies the point size
- All TDATA and TUSER vectors are multiples of 8 bits. Once all fields in a TDATA or TUSER vector have been concatenated, the overall vector is padded to bring it up to an 8 bit boundary.

Configuration Channel

Pinout

Table 7: **Configuration Channel Pinout**

Port Name	Port Width	Direction	Description
s_axis_config_tdata	Variable. Please refer to the CORE Generator GUI when configuring the FFT.	In	Carries the configuration information: `CP_LEN, FWD/INV, NFFT and SCALE_SCH`. See section Run-Time Transfer Configuration for more information
s_axis_config_tvalid	1	In	Asserted by the external master to signal that it is able to provide data.
s_axis_config_tready	1	Out	Asserted by the FFT to signal that it is able to accept data.

TDATA Fields

The Configuration channel (`s_axis_config`) is an AXI channel that carries the following fields in its TDATA vector:

Table 8: **Configuration Channel TDATA Fields**

Field Name	Width	Padded	Description
NFFT	5	Yes	Point size of the transform: NFFT can be the size of the maximum transform or any smaller point size. For example, a 1024-point FFT can compute point sizes 1024, 512, 256, and so on. The value of NFFT is $\log_2$ (point size). This field is only present with run-time configurable transform point size. For more information, see Transform Size
CP_LEN	$\log_2$ (maximum point size)	Yes	Cyclic prefix length: The number of samples from the end of the transform that are initially output as a cyclic prefix, before the whole transform is output. CP_LEN can be any number from zero to one less than the point size. This field is only present with cyclic prefix insertion. For more information, see Cyclic Prefix Insertion
FWD_INV	1 bit per FFT data channel	No	Indicates if a forward FFT transform or an inverse FFT transform is performed. When FWD_INV = 1, a forward transform is computed. If FWD_INV = 0, an inverse transform is computed. The field contains 1 bit per FFT data channel, bit 0 (LSB) representing channel 0, bit 1 representing channel 1, etc. For more information, see Forward/Inverse and Scaling Schedule
SCALE_SCH	$2 \times ceil\left(\frac{NFFT}{2}\right)$ for Pipelined Streaming I/O and Radix-4 Burst I/O architectures or *2* x *NFFT* for Radix-2, Burst I/O and Radix-2 Lite Burst I/O architectures where NFFT is $\log_2$ (maximum point size) or the number of stages	No	Scaling schedule: For Burst I/O architectures, the scaling schedule is specified with two bits for each stage, with the scaling for the first stage given by the two LSBs. The scaling can be specified as 3, 2, 1, or 0, which represents the number of bits to be shifted. An example scaling schedule for N =1024, Radix-4 Burst I/O is [1 0 2 3 2] (ordered from last to first stage). For N =128, Radix-2 Burst I/O or Radix-2 Lite Burst I/O, one possible scaling schedule is [1 1 1 1 0 1 2] (ordered from last to first stage). For Pipelined Streaming I/O architecture, the scaling schedule is specified with two bits for every pair of Radix-2 stages, starting at the two LSBs. For example, a scaling schedule for N = 256 could be [2 2 2 3]. When N is not a power of 4, the maximum bit growth for the last stage is one bit. For instance, [0 2 2 2 2] or [1 2 2 2 2] are valid scaling schedules for N = 512, but [2 2 2 2 2] is invalid. For this transform length the two MSBs of SCALE_SCH can only be 00 or 01. This field is only available with scaled arithmetic (not unscaled, block floating-point or single precision floating-point). For more information, see Forward/Inverse and Scaling Schedule

All fields with padding should be extended to the next 8 bit boundary if they don't already finish on an 8 bit boundary. The FFT core ignores the value of the padding bits, so they can be driven to any value. Connecting them to constant values may help reduce device resource usage.

TDATA Format

The configuration fields are packed into the `s_axis_config_tdata` vector in the following order (starting from the LSB):

1. (optional) NFFT plus padding
2. (optional) CP_LEN plus padding
3. FWD/INV
4. (optional) SCALE_SCH

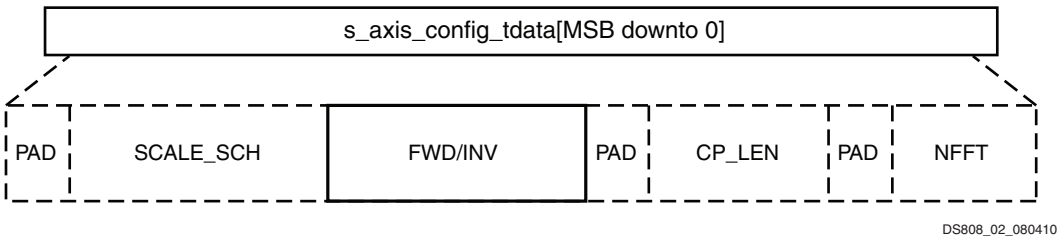

Figure 36: **Configuration Channel TDATA (s_axis_config_tdata) Format**

Optional fields are shown as dotted.

TDATA Example

A core has a configurable transform size with a maximum size of 128 points, cyclic prefix insertion and 3 FFT channels. The core needs to be configured to do an 8 point transform, with an inverse transform performed on channels 0 and 1, and a forward transform performed on channel 2. A 4 point cyclic prefix is required. The fields take on the following values:

Table 9: **Configuration Channel TDATA Example**

Field Name	Padding	Value	Notes
NFFT	000	00011	3 gives an 8-point FFT
CP_LEN	0	1000000	The FFT selects the top NFFT bits of the CP_LEN field (not including the padding) to determine the cyclic prefix length. As we want a Cyclic Prefix length of 4, and NFFT is 3, the field has to be set to 64
FWD_INV	N/A	100	Channel 2: Forward Channel 1: Inverse Channel 0: Inverse

This gives a vector length of 19 bits. As all AXI channels must be aligned to byte boundaries, 5 padding bits are required, giving an `s_axis_config_tdata` length of 24 bits.

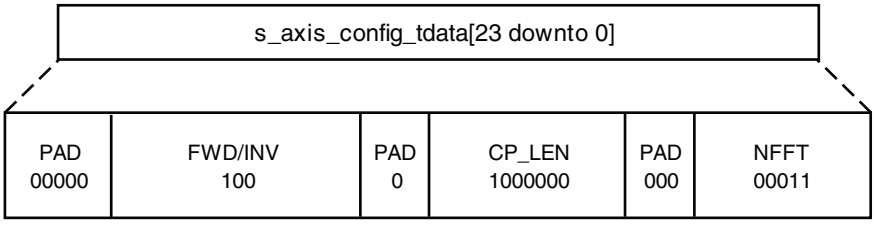

Figure 37: **Configuration Channel TDATA Example**

Data Input Channel

The Data Input channel contains the real and imaginary sample data to be transformed.

Pinout

Table 10: **Data Input Channel Pinout**

Port Name	Port Width	Direction	Description
s_axis_data_tdata	Variable. Please refer to the CORE Generator GUI when configuring the FFT.	In	Carries the sample data: `XN_RE` and `XN_IM`
s_axis_data_tvalid	1	In	Asserted by the upstream master to signal that it is able to provide data
s_axis_data_tlast	1	In	Asserted by the upstream master on the last sample of the frame. This is not used by the FFT except to generate the events: • `event_tlast_unexpected` • `event_tlast_missing` events
s_axis_data_tready	1	Out	Used by the FFT to signal that it is ready to accept data

TDATA Fields

The Data Input channel (`s_axis_data`) is an AXI channel that carries the following fields in its TDATA vector:

Table 11: **Data Input Channel TDATA Fields**

Field Name	Width	Padded	Description
XN_RE	b_{xn}	Yes	Real component (b_{xn} = 8 - 34) in two's complement or single precision floating-point format.
XN_IM	b_{xn}	Yes	Imaginary component (b_{xn} = 8 - 34) in two's complement or single precision floating-point format.

All fields with padding should be extended to the next 8-bit boundary if they do not already finish on an 8-bit boundary. The FFT core ignores the value of the padding bits, so they can be driven to any value. Connecting them to constant values may help reduce device resource usage.

These fields are then repeated for each FFT channel that the design is configured to have.

TDATA Format

The data fields are packed into the `s_axis_data_tdata` vector in the following order (starting from the LSB):

1. XN_RE plus padding for channel 0
2. XN_IM plus padding for channel 0
3. (optional) XN_RE plus padding for channel 1
4. (optional) XN_IM plus padding for channel 1
5. (optional) XN_RE plus padding for channel 2
6. (optional) XN_IM plus padding for channel 2
7. etc, up to channel 11

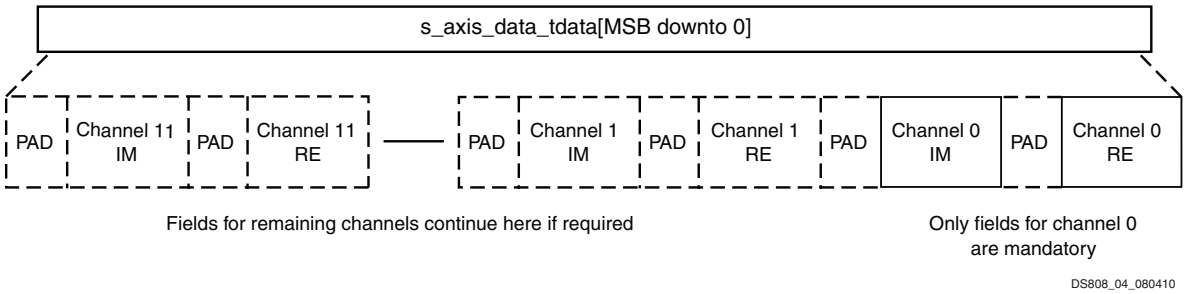

Figure 38: **Data Input Channel TDATA (s_axis_data_tdata) Format**

Optional fields are shown as dotted.

TDATA Example

The core has been configured to have two FFT data channels with 12 bit data. Channel 0 has the following sample value:

- Re = 0010 1101 1001
- IM = 0011 1110 0110

Channel 1 has the following sample value:

- Re = 0111 0000 0000
- IM = 0000 0000 0000

The fields take on the following values:

Table 12: **Data Input Channel TDATA Example**

Field Name	Padding	Value
XN_RE (channel 0)	0000	0010 1101 1001
XN_IM (channel 0)	0000	0011 1110 0110
XN_RE (channel 1)	0000	0111 0000 0000
XN_IM (channel 1)	0000	0000 0000 0000

This gives a vector length of 64 bits.

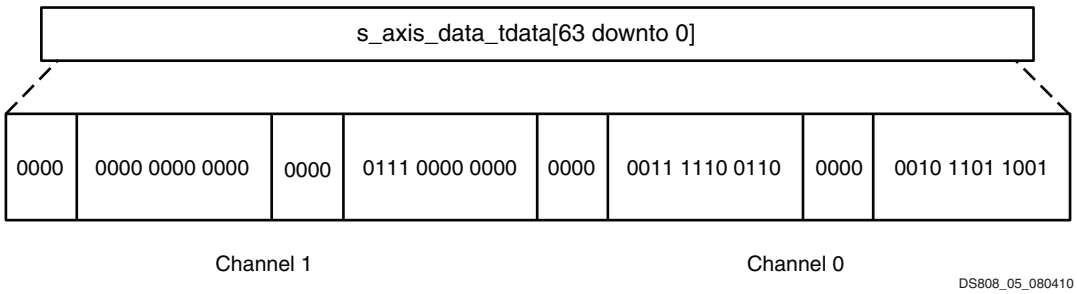

Figure 39: **Data Input Channel TDATA Example**

Data Output Channel

The Data Output channel contains the real and imaginary results of the transform, which are carried on TDATA. In addition, TUSER carries per-sample status information relating to the sample data on TDATA. This status information is intended for use by downstream slaves that directly process data samples. It cannot get out of synchronisation with the data as it is transferred in the same channel. The following information is classed as per-sample status:

1. XK_INDEX
2. Block Exponent (BLK_EXP) for each FFT channel
3. Overflow (OVFLO) for each FFT channel

Pinout.

Table 13: **Data Output Channel Pinout**

Port Name	Port Width	Direction	Description
m_axis_data_tdata	Variable. Please refer to the CORE Generator GUI when configuring the FFT.	Out	Carries the sample data: XK_RE and XK_IM.
m_axis_data_tuser	Variable. Please refer to the CORE Generator GUI when configuring the FFT.	Out	Carries additional per-sample: XK_RE and XK_IM.
m_axis_data_tvalid	1	Out	Asserted by the FFT to signal that it is able to provide sample data
m_axis_data_tlast	1	Out	Asserted by the FFT on the last sample of the frame
m_axis_data_tready	1	In	Asserted by the external slave to signal that it is ready to accept data

TDATA Fields

The Data Output channel (m_axis_data) is an AXI channel that carries the following fields in its TDATA vector:

Table 14: **Data Output Channel TDATA Fields**

Field Name	Width	Padded	Description
XK_RE	b_{xk}	Yes - sign extended	Output data: Real component in two's complement or floating-point format. (For scaled arithmetic and block floating-point arithmetic, $b_{xk} = b_{xn}$. For unscaled arithmetic, $b_{xk} = b_{xn} + \log_2$ (maximum point size) +1. For single precision floating-point $b_{xk} = 32$).
XK_IM	b_{xk}	Yes - sign extended	Output data: Imaginary component in two's complement or single precision floating-point format. (For scaled arithmetic and block floating-point arithmetic, $b_{xk} = b_{xn}$. For unscaled arithmetic, $b_{xk} = b_{xn} + \log_2$ (maximum point size) +1. For single precision floating-point $b_{xk} = 32$)

All fields are sign extended to the next 8 bit boundary if they don't already finish on an 8 bit boundary.

These fields are then repeated for each FFT channel that the design is configured to have.

TDATA Format

The data fields are packed into the s_axis_data_tdata vector in the following order (starting from the LSB):

1. XK_RE plus padding for channel 0
2. XK_IM plus padding for channel 0

3. (optional) XK_RE plus padding for channel 1
4. (optional) XK_IM plus padding for channel 1
5. (optional) XK_RE plus padding for channel 2
6. (optional) XK_IM plus padding for channel 2
7. etc, up to channel 11

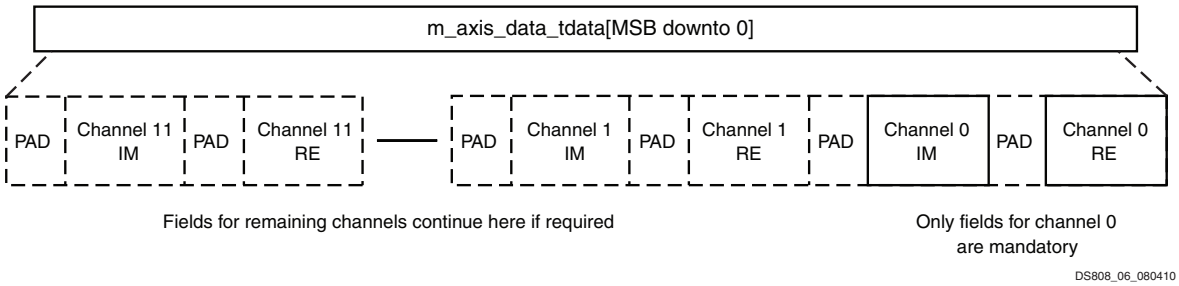

Figure 40: **Data Output Channel TDATA (m_axis_data_tdata) Format**

Optional fields are shown as dotted.

TDATA Example

The core has been configured to have two FFT data channels with 12 bit output data. The FFT produces the following sample result for channel 0:

- Re = 0010 1101 1001
- IM = 1011 1110 0110

The FFT produces the following sample result for channel 1:

- Re = 0111 0000 0000
- IM = 1000 0000 0000

The fields take on the following values:

Table 15: **Data Output Channel TDATA Example**

Field Name	Padding	Value
XK_RE (channel 0)	0000	0010 1101 1001
XK_IM (channel 0)	1111	1011 1110 0110
XK_RE (channel 1)	0000	0111 0000 0000
XK_IM (channel 1)	1111	1000 0000 0000

This gives a vector length of 64 bits.

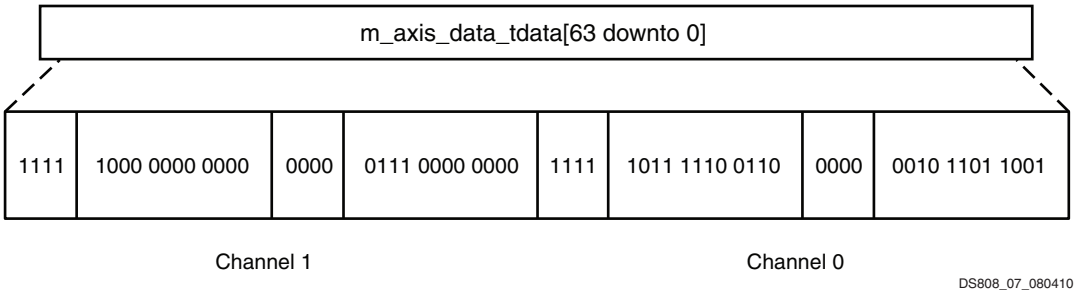

Figure 41: **Data Output Channel TDATA Example**

TUSER Fields

The Data Output channel carries the following fields in its TUSER vector:

Table 16: **Data Output Channel TUSER Fields**

Field Name	Width	Padded	Description
XK_INDEX	$\log_2$ (maximum point size)	Yes - zero extended	Index of output data. This field is optional, and only included when XK_INDEX is enabled in the GUI.
BLK_EXP	b_{xk}	Yes - zero extended	Block exponent: The amount of scaling applied. A separate BLK_EXP field is included for each FFT channel that the core has. Available only when block floating-point is used. For more information on BLK_EXP, see Block Exponent
OVFLO	1	No	Arithmetic overflow indicator (active high): OVFLO is high during result unloading if any value in the data frame overflowed. The OVFLO signal is reset at the beginning of a new frame of data. A separate OVFLO field is included for each FFT channel that the core has. This port is optional and only available with scaled arithmetic or single precision floating-point I/O. For more information on OVFLO, see Overflow

All fields with padding should be 0 extended to the next 8 bit boundary if they don't already finish on an 8 bit boundary.

TUSER Format

The data fields are packed into the m_axis_data_tuser vector in the following order (starting from the LSB):

1. (optional) XK_INDEX plus padding
2. (optional) BLK_EXP plus padding for channel 0
3. (optional) BLK_EXP plus padding for channel 1
4. etc
5. (optional) OVFLO for channel 0
6. (optional) OVFLO for channel 1
7. etc
8. Padding to make TUSER 8 bit aligned. Only needed when OVFLO is present

Note that the FFT cannot be configured to have both BLK_EXP and OVFLO.

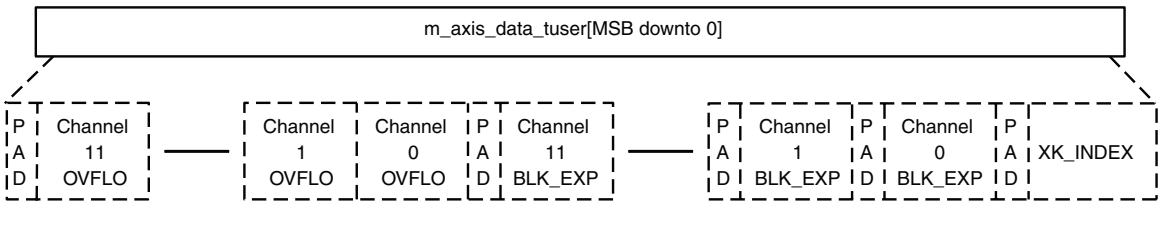

Figure 42: **Data Output Channel TUSER (m_axis_data_tuser) Format**

Optional fields are shown as dotted. As all fields are optional, it's possible to configure the core such that TUSER would have no fields. In this case it is automatically removed from the core's interface.

TUSER Examples

Example 1

The core has been configured to have two FFT data channels, a 128 point transform size, overflow, and XK_INDEX. The third sample (XK_INDEX = 3) has an overflow on channel 0 but not on channel 1. XK_INDEX is 7 bits long.

The fields take on the following values:

Table 17: **Data Output Channel TUSER Example 1**

Field Name	Padding	Value
XK_INDEX	0	000 0011
OVFLO (channel 0)	None	1
OVFLO (channel 1)	None	0

This gives a vector length of 10 bits. As all AXI channels must be aligned to byte boundaries, 6 padding bits are required, giving an m_axis_data_tuser length of 16 bits.

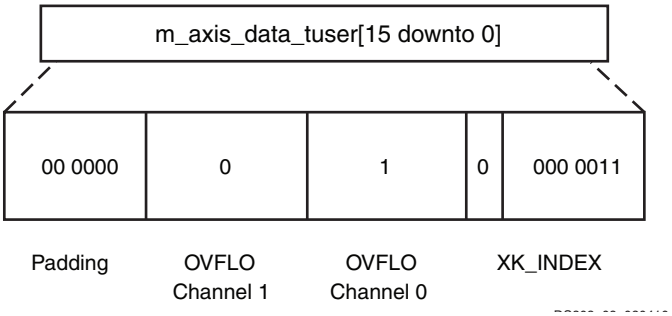

Figure 43: **Data Output Channel TUSER Example 1**

Example 2

The core has been configured to have two FFT data channels, block exponent, but no XK_INDEX. The output sample for channel 0 has a block exponent of 4, and the output sample for channel 1 has a block exponent of 31.

The fields take on the following values:

Table 18: **Data Output Channel TUSER Example 2**

Field Name	Padding	Value
BLK_EXP (channel 0)	000	0 0100
BLK_EXP (channel 1)	000	1 1111

This gives a vector length of 16 bits, so no more padding is required.

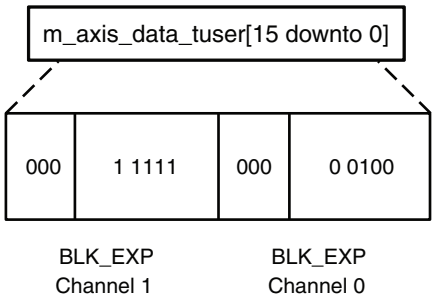

Figure 44: **Data Output Channel TUSER Example 2**

Status Channel

The Status channel contains per-frame status information. That is, information that relates to an entire frame's worth of data. This is intended for downstream slaves that don't operate on the data directly but might need to know the information to control another part of the system. The exact position in the frame where the status is sent depends on the nature of the status information. The following information is classed as per-frame status:

1. BLK_EXP for each channel
2. OVFLO for each channel

Note that the FFT cannot be configured to have both BLK_EXP and OVFLO.

BLK_EXP status information is sent at the start of the frame and OVFLO status information is sent at the end of the frame.

Pinout

Table 19: **Status Channel Pinout**

Port Name	Port Width	Direction	Description
m_axis_status_tdata	Variable. Please refer to the CORE Generator GUI when configuring the FFT.	Out	Carries the status data: BLK_EXP or OVFLO
m_axis_status_tvalid	1	Out	Asserted by the FFT to signal that it is able to provide status data
m_axis_status_tready	1	In	Asserted by the external slave to signal that it is ready to accept data

TDATA Fields

The Status Channel carries the following fields in its TDATA vector:

Table 20: **Status Channel TDATA Fields**

Field Name	Width	Padded	Description
BLK_EXP	5	Yes - zero extended	Block exponent: The amount of scaling applied. A separate BLK_EXP field is included for each FFT channel that the core has. Available only when block floating-point is used. For more information on BLK_EXP, see Block Exponent.
OVFLO	1	No	Arithmetic overflow indicator (active high): OVFLO is high during result unloading if any value in the data frame overflowed. The OVFLO signal is reset at the beginning of a new frame of data. A separate OVFLO field is included for each FFT channel that the core has. This port is optional and only available with scaled arithmetic or single precision floating-point I/O. For more information on OVFLO, see Overflow.

All fields with padding should be 0 extended to the next 8 bit boundary if they don't already finish on an 8 bit boundary.

TDATA Format

The data fields are packed into the `m_axis_status_tdata` vector in the following order (starting from the LSB):

1. (optional) `BLK_EXP` plus padding for channel 0
2. (optional) `BLK_EXP` plus padding for channel 1
3. etc
4. (optional) `OVFLO` for channel 0
5. (optional) `OVFLO` for channel 1
6. etc
7. Padding to make `TDATA` 8 bit aligned. Only needed when `OVFLO` is present

Note that the FFT cannot be configured to have both `BLK_EXP` and `OVFLO`.

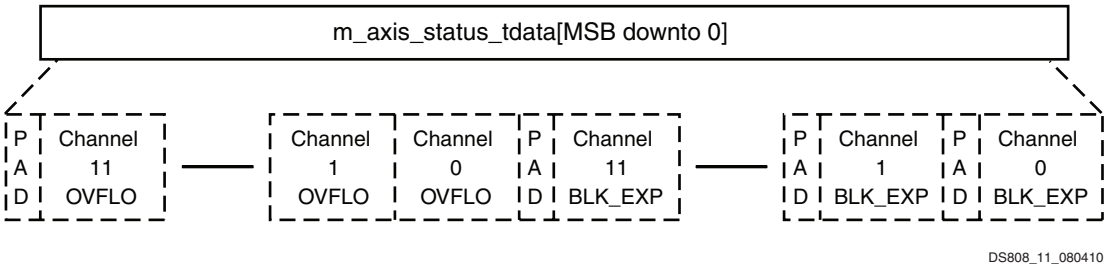

Figure 45: **Status channel TDATA (m_axis_status_tdata) Format**

Optional fields are shown as dotted. As all fields are optional, it's possible to configure the core such that TDATA would have no fields. In this case the entire Status channel is automatically removed from the core's interface.

TDATA Example

Example 1: The core has been configured to have four FFT data channels and overflow. The current frame contains an overflow in channels 2 and 3.

Table 21: **Status Channel TDATA Example 1**

Field Name	Padding	Value
OVFLO (channel 0)	None	0
OVFLO (channel 1)	None	0
OVFLO (channel 2)	None	1
OVFLO (channel 3)	None	1

This gives a vector length of 4 bits. As all AXI channels must be aligned to byte boundaries, 4 padding bits are required, giving an `m_axis_status_tdata` length of 8 bits.

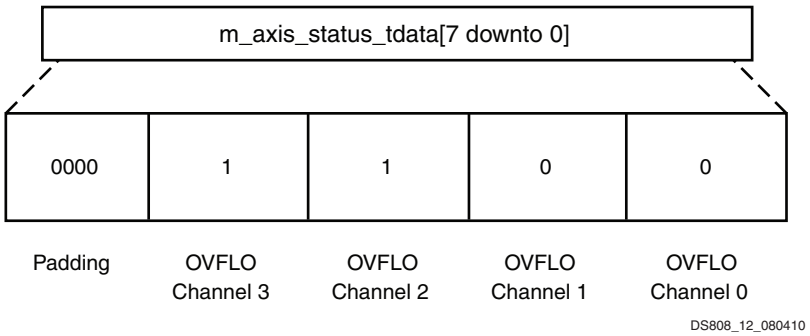

Figure 46: **Status Channel TDATA Example 1**

Example 2: The core has been configured to have one FFT data channel and overflow. The current frame contains no overflow.

Table 22: **Status Channel TDATA Example 2**

Field Name	Padding	Value
OVFLO (channel 0)	None	0

This gives a vector length of 1 bit. As all AXI channels must be aligned to byte boundaries, 7 padding bits are required, giving an `m_axis_status_tdata` length of 8 bits.

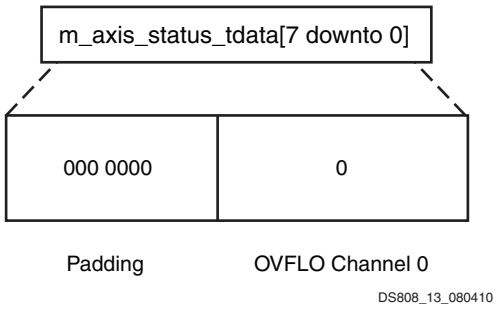

Figure 47: **Status Channel TDATA Example 2**

Migrating to FFT v8.0 from earlier versions

XCO Parameters Changes

The CORE Generator core update functionality may be used to update an existing XCO file from v7.1 to v8.0, but it should be noted that the update mechanism alone does not create a core compatible with v7.1. See Instructions for minimum change migration. FFT v8.0 has additional parameters for AXI4-Stream support. The following table shows the changes to XCO parameters from version 7.1 to version 8.0.

Table 23: **XCO Parameter Changes from v7.1 to v8.0**

Version 7.1	Version 8.0	Notes
component_name	component_name	Unchanged
channels	channels	Unchanged
transform_length	transform_length	Unchanged
implementation_options	implementation_options	Unchanged
target_clock_frequency	target_clock_frequency	Unchanged
target_data_throughput	target_data_throughput	Unchanged
run_time_configurable_transform_length	run_time_configurable_transform_length	Unchanged
data_format	data_format	Unchanged
input_width	input_width	Unchanged
phase_factor_width	phase_factor_width	Unchanged
scaling_options_scaled	scaling_options_scaled	Unchanged
rounding_modes	rounding_modes	Unchanged
ce	aclken	Renamed
sclr	aresetn	Renamed
ovflo	ovflo	Unchanged
	xk_index	New to version 8.0
	throttle_scheme	New to version 8.0
output_ordering	output_ordering	Unchanged
cyclic_prefix_ordering	cyclic_prefix_ordering	Unchanged
memory_options_data	memory_options_data	Unchanged
memory_options_phase_factors	memory_options_phase_factors	Unchanged
memory_options_reorder	memory_options_reorder	Unchanged
number_of_stages_using_block_ram_for_data_and_phase_factors	number_of_stages_using_block_ram_for_data_and_phase_factors	Unchanged
memory_options_hybrid	memory_options_hybrid	Unchanged
input_data_offset		Obsolete
complex_mult_type	complex_mult_type	Unchanged
butterfly_type	butterfly_type	Unchanged

Port Changes

The following table details the changes to port naming, additional or deprecated ports and polarity changes from v7.1 to v8.0.

Table 24: **Port Changes from v7.1 to v8.0**

Version 7.1	Version 8.0	Notes
CLK	aclk	Rename only
CE	alcken	Rename only
SCLR	aresetn	Renamed Polarity change (now active low) Minimum length now two clock cycles
XN_RE XN_RE0 to XN_RE11	s_axis_data_tdata	See section Data Input Channel for more information on how the sample data is packed into `s_axis_data_tdata`
XN_IM XN_IM0 to XN_IM11		
RFD	s_axis_data_tready	Renamed May now go low during the loading of data into the FFT. Data is still only transferred when high. Can assert a significant time before sample processing actually begins (`s_axis_data_tready` signals that the *buffer* in the Data Input Channel is ready for data. That does not mean the FFT is immediately ready to load and process that data).
NFFT	s_axis_config_tdata	See section Configuration Channel for more information
FWD_INV FWD_INV0 to FWD_INV11		
SCALE_SCH SCALE_SCH0 to SCALE_SCH11		
CP_LEN		
DV	m_axis_data_tvalid	Renamed May now go low during the unloading of data from the FFT. Data is still only transferred when high
XK_RE XK_RE0 to XK_RE11	m_axis_data_tdata	See section Data Output Channel for more information on how the processed sample data and `XK_INDEX` are packed into `m_axis_data_tdata` and `m_axis_data_tuser`
XK_IM XK_IM0 to XK_IM11		
XK_INDEX	m_axis_data_tuser	
BLK_EXP BLK_EXP0 to BLK_EXP11	m_axis_data_tuser m_axis_status_tdata	See section Data Output Channel for more information on how `BLK_EXP` and `OVFLO` are packed into `m_axis_data_tuser`. See section Status Channel for more information on how `BLK_EXP` and `OVFLO` are packed into `m_axis_status_tdata`.
OVFLO OVFLO0 to OVFLO11		
START		Obsolete. The FFT automatically starts (although it doesn't immediately start) when sample data is supplied on the Data Input channel.
RFS		

Table 24: **Port Changes from v7.1 to v8.0 *(Cont'd)***

Version 7.1	Version 8.0	Notes
UNLOAD		Obsolete. The FFT automatically starts to unload processed sample data when it is available. Data is not actually unloaded until `m_axis_data_tready` is asserted
NFFT_WE		Obsolete. The application of new configuration information is now handled automatically by the core. See section Configuration Channel for more information
FWD_INV_WE FWD_INV_WE0 to FWD_INV_WE11		
SCALE_SCH_WE SCALE_SCH_WE0 to SCALE_SCH_WE11		
CP_LEN_WE		
XN_INDEX		Obsolete
BUSY		Obsolete
EDONE		Obsolete
DONE		Obsolete
CPV		Obsolete
	s_axis_data_tlast	New in v8.0
	m_axis_data_tlast	New in v8.0
	s_axis_config_tvalid	New in v8.0
	s_axis_config_tready	New in v8.0
	s_axis_data_tvalid	New in v8.0
	m_axis_data_tready	New in v8.0
	event_frame_started	New in v8.0
	event_tlast_missing	New in v8.0
	event_tlast_unexpected	New in v8.0
	event_fft_overflow	New in v8.0
	event_status_channel_halt	New in v8.0
	event_data_in_channel_halt	New in v8.0
	event_data_out_channel_halt	New in v8.0

Latency Changes

The latency of FFT v8.0 is greater than that of v7.1. The update process cannot account for this and guarantee equivalent performance. Importantly, the latency of the core is variable, so that only the minimum possible latency can be determined.

When in Non-Realtime mode the latency is 7 cycles longer than for the equivalent configuration of v7.1.

When in Realtime mode, the latency of the core for equivalent performance is 3 cycles longer than for the equivalent configuration of v7.1.

See section Information Tabs for the definition of latency.

Instructions for minimum change migration

To configure the FFT v8.0 to most closely mimic the behaviour of v7.1 the translation is as follows:

XCO Parameters - Set throttle_scheme to realtime. If you previously had sclr set to true then remember that the reset pulse is now active low and must be a minimum of two clock cycles long.

Ports - Rename and map signals as detailed in Port Changes. Tie `s_axis_data_tvalid` to 1. This tells the core that you are always able to supply data when requested. Note, however, that the FFT cannot always consume data on consecutive clock cycles, so `s_axis_data_tready` has to be used to control the flow of data into the FFT.

Performance and Resource Usage

The following tables list the resource usage and transform time for a selected set of parameters. This core does not use placement constraints, allowing Place and Route full flexibility. The slice count, block RAM count, and XtremeDSP slice count are listed. The maximum clock frequency is listed with the transform latency. The latency is from the Upstream Master supplying the first sample of a frame to the last sample of output data coming out of the core, assuming that the FFT core was idle and neither the Upstream Master or the Downstream Slave inserted wait states. The following device architectures are represented:

- Virtex-6 Family
- Spartan-6 Family

The maximum clock frequency for each test was determined iteratively. For the determination of maximum frequency, the core was generated with double registers on each input and output. The registers directly connected to the core run on the core clock, whereas the outer registers run off a separate clock. This ensures that all paths in the core are included in the timing constraint without artificially distorting the design to fit the chip. The slowest speed grade is used for each family. The parameters used for map and par are as follows:

```
map -pr b -ol high
par -ol high
```

The maximum achievable clock frequency and the resource counts may also be affected by other tools options, additional logic in the FPGA device, using a different version of Xilinx tools, and other factors.

Improved performance or resource usage may be achieved by applying an area group, or using map arguments such as “-lc area.” Consult the ISE Design Suite 13.1 documentation for more details on available options.

Virtex-6 Family

Table 25 and Table 26 show performance and resource usage numbers for Virtex-6 FPGAs for both realtime and non-realtime modes. A range of FFT cores is shown for several typical applications: Baseband 3GPP LTE, Baseband OFDM, CT scanners, Ultrasound, Test and measurement, and Radar. The parameters for each core are shown in the tables. None of the optional pins (ACLKEN, ARESETN, OVFLO) are used and hybrid RAM is not used. The performance and resource usage numbers were produced using ISE 13.1 software, with speed file versions:

- XC6VLX75T : "PRELIMINARY 1.08 2010-07-20"
- XC6VLX130T : "PRODUCTION 1.08 2010-07-20"
- XC6VLX550T : "PRELIMINARY 1.08 2010-07-20"

Table 25: **Virtex-6 FPGA Family Performance and Resource Usage in Realtime Mode**

Application	Channels	Point Size	Implementation(1)	Variable Point Size	Input Data Width	Phase Factor Width	Scaling Type(2)	Rounding Mode(3)	Output Ordering(4)	Cyclic Prefix Insertion	Memory Type(5)	Stages Using Block RAM	Optimize for Speed(6)	Xilinx Part(7)	LUT/FF Pairs	LUTs	FFs	18k Block RAMs(8)	XtremeDSP Slices	Max Clock Frequency(9)	Latency (cycles)(10)	Latency (µs)(11)
Baseband 3GPP LTE	1	1k	R2L	Y	16	16	S	C	N	Y	B	-	N	XC6VLX75T	1033	853	1188	3	2	395	12456	31.53
	1	1k	R2L	Y	16	16	S	C	N	Y	B	-	Y	XC6VLX75T	1025	839	1190	3	3	395	12476	31.58
	1	2k	R2L	Y	16	16	S	C	N	Y	B	-	N	XC6VLX75T	1175	884	1249	5	2	395	26807	67.87
	1	2k	R2L	Y	16	16	S	C	N	Y	B	-	Y	XC6VLX75T	1137	882	1251	5	3	402	26829	66.74
	4	1k	R2L	Y	16	16	S	C	N	Y	B	-	N	XC6VLX130T	2214	1837	3015	9	8	380	12456	32.78
	8	1k	R2L	Y	16	16	S	C	N	Y	B	-	N	XC6VLX550T	4065	3031	5451	17	16	379	12456	32.87
	4	2k	R2L	Y	16	16	S	C	N	Y	B	-	N	XC6VLX130T	2513	1817	3100	17	8	401	26807	66.85
	8	2k	R2L	Y	16	16	S	C	N	Y	B	-	N	XC6VLX550T	4253	3069	5568	33	16	351	26807	76.37
	1	1k	R2	Y	16	16	S	C	N	Y	B	-	N	XC6VLX75T	1199	1040	1362	3	3	395	7367	18.65
	1	1k	R2	Y	16	16	S	C	N	Y	B	-	Y	XC6VLX75T	1106	882	1248	3	6	395	7357	18.63
	1	2k	R2	Y	16	16	S	C	N	Y	B	-	N	XC6VLX75T	1233	1100	1431	5	3	395	15578	39.44
	1	2k	R2	Y	16	16	S	C	N	Y	B	-	Y	XC6VLX75T	1082	952	1319	5	6	395	15567	39.41
	2	1k	R2	Y	16	16	S	C	N	Y	B	-	N	XC6VLX130T	1868	1605	2221	5	6	402	7367	18.33
	4	1k	R2	Y	16	16	S	C	N	Y	B	-	N	XC6VLX130T	3237	2714	3939	9	12	384	7367	19.18
	8	1k	R2	Y	16	16	S	C	N	Y	B	-	N	XC6VLX550T	5644	4977	7375	17	24	368	7367	20.02
	2	2k	R2	Y	16	16	S	C	N	Y	B	-	N	XC6VLX130T	1918	1687	2330	9	6	402	15578	38.75
	4	2k	R2	Y	16	16	S	C	N	Y	B	-	N	XC6VLX130T	3415	2868	4128	17	12	374	15578	41.65
	8	2k	R2	Y	16	16	S	C	N	Y	B	-	N	XC6VLX550T	6111	5275	7724	33	24	334	15578	46.64
OFDM	1	256	R2	Y	12	12	S	T	N	Y	D	-	N	XC6VLX75T	1048	850	947	0	3	444	1655	3.73
	1	256	R2	Y	12	12	S	T	N	Y	B	-	N	XC6VLX75T	842	650	933	3	3	400	1673	4.18
	1	256	R2	Y	12	12	B	T	N	Y	B	-	N	XC6VLX75T	863	719	937	3	3	400	1673	4.18

Table 25: **Virtex-6 FPGA Family Performance and Resource Usage in Realtime Mode *(Cont'd)***

Application	Channels	Point Size	Implementation[1]	Variable Point Size	Input Data Width	Phase Factor Width	Scaling Type[2]	Rounding Mode[3]	Output Ordering[4]	Cyclic Prefix Insertion	Memory Type[5]	Stages Using Block RAM	Optimize for Speed[6]	Xilinx Part[7]	LUT/FF Pairs	LUTs	FFs	18k Block RAMs[8]	XtremeDSP Slices	Max Clock Frequency[9]	Latency (cycles)[10]	Latency (µs)[11]
CT Scanners	1	1k	Str	N	24	24	S	C	R	N	-	3	Y	XC6VLX75T	3437	2994	4989	8	62	399	2182	5.47
	1	1k	Str	N	24	24	S	C	R	N	-	3	N	XC6VLX75T	4141	3726	5472	8	32	395	2170	5.49
	1	1k	Str	N	24	24	S	C	R	N	-	1	Y	XC6VLX75T	4076	3713	5026	2	62	395	2182	5.52
	1	1k	Str	N	24	24	S	C	R	N	-	5	Y	XC6VLX75T	3455	2834	4970	14	62	395	2182	5.52
	1	1k	Str	N	24	24	S	C	N	N	-	3	Y	XC6VLX75T	3687	3030	5130	11	62	386	3210	8.32
	1	1k	Str	N	24	24	U	C	N	N	-	3	Y	XC6VLX75T	3542	3081	5582	12	62	401	3206	8.00
	1	1k	Str	Y	24	24	S	C	N	N	-	3	Y	XC6VLX75T	4343	3792	6114	11	62	369	3219	8.72
	1	1k	Str	Y	16	24	U	C	R	N	-	3	Y	XC6VLX75T	3420	2923	4990	6	52	396	2184	5.52
	1	1k	Str	N	16	16	S	C	R	N	-	3	Y	XC6VLX75T	2489	2127	3416	4	40	395	2174	5.50
	1	1k	Str	N	16	16	S	C	R	N	-	3	N	XC6VLX75T	3404	3049	4286	4	12	395	2174	5.50
	1	1k	Str	N	16	16	S	C	N	N	-	3	Y	XC6VLX75T	2445	2190	3541	6	40	395	3202	8.11
	1	1k	Str	N	16	16	U	C	N	N	-	3	Y	XC6VLX75T	2475	2255	4008	8	40	395	3198	8.10
	1	1k	Str	N	34	34	S	C	N	N	-	3	Y	XC6VLX75T	4902	4423	7514	12	94	374	3226	8.63
	1	1k	Str	N	34	34	U	C	N	N	-	3	Y	XC6VLX75T	5038	4389	8110	14	102	358	3226	9.01
	1	1k	Str	N	34	34	B	C	N	N	-	3	Y	XC6VLX75T	5371	4820	8303	14	102	347	3228	9.30
	1	4k	Str	N	24	24	S	C	N	N	-	5	Y	XC6VLX75T	4528	3714	6350	30	78	395	12448	31.51
	1	4k	Str	N	24	24	U	C	N	N	-	5	Y	XC6VLX75T	4453	3979	7110	36	78	384	12444	32.41
	1	8k	Str	N	24	24	S	C	N	N	-	6	Y	XC6VLX75T	4720	4355	7052	56	90	366	24751	67.63
	1	8k	Str	N	24	24	U	C	N	N	-	6	Y	XC6VLX75T	5088	4666	8095	69	94	292	24749	84.76
U[12]	1	512	R2L	Y	24	24	S	C	N	N	B	-	N	XC6VLX75T	1173	790	1398	4	4	400	5803	14.51
Test	1	8k	Str	Y	24	24	U	C	N	N	-	6	Y	XC6VLX75T	6148	5616	9582	69	94	307	24761	80.65
	1	8k	Str	Y	24	24	U	C	N	N	-	6	N	XC6VLX75T	7499	7186	10889	69	51	347	24748	71.32
	1	16k	Str	Y	24	24	U	C	N	N	-	7	Y	XC6VLX75T	7041	6086	10355	132	98	289	49344	170.74
	1	16k	Str	Y	24	24	U	C	N	N	-	7	N	XC6VLX75T	9034	7909	11723	132	51	310	49330	159.13
	1	256	R4	N	32	24	F	-	R	N	B	-	Y	XC6VLX75T	3419	3015	4505	16	40	399	1414	3.54
	1	1k	R4	N	32	24	F	-	R	N	B	-	Y	XC6VLX75T	3469	3148	4661	18	40	398	5532	13.90
	1	4k	R4	N	32	24	F	-	R	N	B	-	Y	XC6VLX75T	3477	3259	4768	37	40	400	22706	56.77
	1	256	R2	N	32	24	F	-	R	N	B	-	Y	XC6VLX75T	2127	1927	2842	8	12	395	2228	5.64
	1	1k	R2	N	32	24	F	-	R	N	B	-	Y	XC6VLX75T	2264	1987	2917	10	12	395	9430	23.87
	1	4k	R2	N	32	24	F	-	R	N	B	-	Y	XC6VLX75T	2181	2032	2979	31	12	395	41208	104.32
	1	256	R2L	N	32	24	F	-	R	N	B	-	Y	XC6VLX75T	2105	1697	2728	6	6	395	3172	8.03
	1	1k	R2L	N	32	24	F	-	R	N	B	-	Y	XC6VLX75T	2145	1797	2797	10	6	395	14444	36.57
	1	4k	R2L	N	32	24	F	-	R	N	B	-	Y	XC6VLX75T	2093	1870	2850	31	6	400	65652	164.13
	1	256	Str	N	32	24	F	-	R	N	B	1	Y	XC6VLX75T	3570	3217	5315	4	46	378	888	2.35
	1	1k	Str	N	32	24	F	-	R	N	B	3	Y	XC6VLX75T	4271	3949	6678	12	66	386	3212	8.32
	1	4k	Str	N	32	24	F	-	R	N	B	5	Y	XC6VLX75T	5177	4739	8124	36	86	346	12448	35.98

Table 25: **Virtex-6 FPGA Family Performance and Resource Usage in Realtime Mode *(Cont'd)***

Application	Channels	Point Size	Implementation[1]	Variable Point Size	Input Data Width	Phase Factor Width	Scaling Type[2]	Rounding Mode[3]	Output Ordering[4]	Cyclic Prefix Insertion	Memory Type[5]	Stages Using Block RAM	Optimize for Speed[6]	Xilinx Part[7]	LUT/FF Pairs	LUTs	FFs	18k Block RAMs[8]	XtremeDSP Slices	Max Clock Frequency[9]	Latency (cycles)[10]	Latency (µs)[11]
Radar	1	1k	R4	Y	16	16	S	C	N	N	B	-	N	XC6VLX75T	2237	1889	2586	7	9	395	3449	8.73
	1	1k	R4	Y	16	16	S	C	N	N	B	-	Y	XC6VLX75T	1672	1529	2280	7	20	403	3454	8.57
	1	1k	R4	Y	16	16	B	C	N	N	B	-	N	XC6VLX75T	2091	2001	2615	7	9	395	3449	8.73
	1	1k	R4	Y	20	16	S	C	N	N	B	-	N	XC6VLX75T	2530	2243	3028	11	9	395	3449	8.73
	1	1k	R4	Y	20	16	B	C	N	N	B	-	N	XC6VLX75T	2624	2321	3057	11	9	395	3449	8.73
	1	32k	R4	Y	16	16	S	C	N	N	B	-	N	XC6VLX75T	2548	2388	2994	84	9	327	131259	401.40
	1	32k	R4	Y	16	16	B	C	N	N	B	-	N	XC6VLX75T	2750	2463	3005	84	9	318	131259	412.76
	1	32k	Str	Y	16	16	U	C	N	N	-	10	N	XC6VLX75T	8364	7417	10872	201	31	248	98500	397.18

Notes:

1. Implementations: Str = Pipelined Streaming I/O; R4 = Radix-4, Burst I/O; R2 = Radix-2, Burst I/O; R2L = Radix-2 Lite, Burst I/O.
2. Scaling types: S = scaled; U = unscaled; B = block floating-point; F = single precision floating-point.
3. Rounding modes: C = convergent rounding; T = truncation.
4. Output ordering: N = Natural Order; R = Bit/Digit Reversed Order.
5. Memory types: B = block RAM, D = distributed RAM. Applies to data and phase factor storage in Burst I/O architectures, and to the output reorder buffer in the Pipelined Streaming I/O architecture.
6. Optimize for Speed using XtremeDSP slices in both Complex Multipliers (4-multiplier structure) and Butterfly Arithmetic.
7. The -1 speedgrade was used in all cases.
8. Virtex-6 FPGAs have 18K block RAMs that may be packed in pairs to form 36K block RAMs. map reports the number of 36K block RAMs + 18K block RAMs, which may not match the number of 18K block RAMs given here.
9. Area and maximum clock frequencies are provided as a guide. They may vary with the amount of other logic in the FPGA device, tools options, and other releases of Xilinx implementation tools. Clock frequency does not take jitter into account and should be de-rated by an amount appropriate to the clock source jitter specification.
10. Latency in clock cycles for the largest transform size.
11. Latency in microseconds for the largest transform size, when running at the maximum achievable clock frequency.
12. Ultrasound.

Table 26: **Virtex-6 FPGA Family Performance and Resource Usage in Non-Realtime Mode**

Application	Channels	Point Size	Implementation(1)	Variable Point Size	Input Data Width	Phase Factor Width	Scaling Type(2)	Rounding Mode(3)	Output Ordering(4)	Cyclic Prefix Insertion	Memory Type(5)	Stages Using Block RAM	Optimize for Speed(6)	Xilinx Part(7)	LUT/FF Pairs	LUTs	FFs	18k Block RAMs(8)	XtremeDSP Slices	Max Clock Frequency(8)	Latency (cycles)(10)	Latency (µs)(11)
Baseband 3GPP LTE	1	1k	R2L	Y	16	16	S	C	N	Y	B	-	N	XC6VLX75T	1192	1017	1350	3	2	397	12460	31.39
	1	1k	R2L	Y	16	16	S	C	N	Y	B	-	Y	XC6VLX75T	1192	1005	1352	3	3	404	12480	30.89
	1	2k	R2L	Y	16	16	S	C	N	Y	B	-	N	XC6VLX75T	1313	1085	1410	5	2	396	26811	67.70
	1	2k	R2L	Y	16	16	S	C	N	Y	B	-	Y	XC6VLX75T	1324	1060	1412	5	3	402	26833	66.75
	4	1k	R2L	Y	16	16	S	C	N	Y	B	-	N	XC6VLX130T	2981	2146	3528	9	8	367	12460	33.95
	8	1k	R2L	Y	16	16	S	C	N	Y	B	-	N	XC6VLX550T	5043	3864	6432	17	16	324	12460	38.46
	4	2k	R2L	Y	16	16	S	C	N	Y	B	-	N	XC6VLX130T	3041	2272	3618	17	8	350	26811	76.60
	8	2k	R2L	Y	16	16	S	C	N	Y	B	-	N	XC6VLX550T	5302	3886	6562	33	16	323	26811	83.01
	1	1k	R2	Y	16	16	S	C	N	Y	B	-	N	XC6VLX75T	1376	1206	1507	3	3	396	7371	18.61
	1	1k	R2	Y	16	16	S	C	N	Y	B	-	Y	XC6VLX75T	1272	1094	1410	3	6	400	7361	18.40
	1	2k	R2	Y	16	16	S	C	N	Y	B	-	N	XC6VLX75T	1428	1292	1594	5	3	396	15582	39.35
	1	2k	R2	Y	16	16	S	C	N	Y	B	-	Y	XC6VLX75T	1309	1154	1481	5	6	396	15571	39.32
	2	1k	R2	Y	16	16	S	C	N	Y	B	-	N	XC6VLX130T	2174	1867	2483	5	6	378	7371	19.50
	4	1k	R2	Y	16	16	S	C	N	Y	B	-	N	XC6VLX130T	3563	3171	4435	9	12	368	7371	20.03
	8	1k	R2	Y	16	16	S	C	N	Y	B	-	N	XC6VLX550T	6649	5751	8339	17	24	317	7371	23.25
	2	2k	R2	Y	16	16	S	C	N	Y	B	-	N	XC6VLX130T	2337	1991	2612	9	6	373	15582	41.77
	4	2k	R2	Y	16	16	S	C	N	Y	B	-	N	XC6VLX130T	3674	3270	4648	17	12	362	15582	43.04
	8	2k	R2	Y	16	16	S	C	N	Y	B	-	N	XC6VLX550T	7192	6179	8720	33	24	278	15582	56.05
OFDM	1	256	R2	Y	12	12	S	T	N	Y	D	-	N	XC6VLX75T	1177	1005	1078	0	3	422	1659	3.93
	1	256	R2	Y	12	12	S	T	N	Y	B	-	N	XC6VLX75T	948	784	1051	3	3	400	1677	4.19
	1	256	R2	Y	12	12	B	T	N	Y	B	-	N	XC6VLX75T	983	820	1044	3	3	360	1677	4.66

Table 26: **Virtex-6 FPGA Family Performance and Resource Usage in Non-Realtime Mode *(Cont'd)***

Application	Channels	Point Size	Implementation[1]	Variable Point Size	Input Data Width	Phase Factor Width	Scaling Type[2]	Rounding Mode[3]	Output Ordering[4]	Cyclic Prefix Insertion	Memory Type[5]	Stages Using Block RAM	Optimize for Speed[6]	Xilinx Part[7]	LUT/FF Pairs	LUTs	FFs	18k Block RAMs[8]	XtremeDSP Slices	Max Clock Frequency[8]	Latency (cycles)[10]	Latency (µs)[11]
CT Scanners	1	1k	Str	N	24	24	S	C	R	N	-	3	Y	XC6VLX75T	3421	3217	5136	8	62	366	2186	5.97
	1	1k	Str	N	24	24	S	C	R	N	-	3	N	XC6VLX75T	4122	3893	5628	8	32	379	2174	5.74
	1	1k	Str	N	24	24	S	C	R	N	-	1	Y	XC6VLX75T	4402	4029	5176	2	62	284	2186	7.7
	1	1k	Str	N	24	24	S	C	R	N	-	5	Y	XC6VLX75T	3495	3092	5117	14	62	280	2186	7.81
	1	1k	Str	N	24	24	S	C	N	N	-	3	Y	XC6VLX75T	3784	3310	5279	11	62	297	3214	10.82
	1	1k	Str	N	24	24	U	C	N	N	-	3	Y	XC6VLX75T	3669	3308	5680	12	62	298	3210	10.77
	1	1k	Str	Y	24	24	S	C	N	N	-	3	Y	XC6VLX75T	4449	4009	6266	11	62	324	3223	9.95
	1	1k	Str	Y	16	24	U	C	R	N	-	3	Y	XC6VLX75T	3612	3109	5111	6	52	354	2188	6.18
	1	1k	Str	N	16	16	S	C	R	N	-	3	Y	XC6VLX75T	2445	2272	3551	4	40	393	2178	5.54
	1	1k	Str	N	16	16	S	C	R	N	-	3	N	XC6VLX75T	3390	3209	4402	4	12	387	2178	5.63
	1	1k	Str	N	16	16	S	C	N	N	-	3	Y	XC6VLX75T	2531	2316	3678	6	40	379	3206	8.46
	1	1k	Str	N	16	16	U	C	N	N	-	3	Y	XC6VLX75T	2588	2415	4111	8	40	363	3202	8.82
	1	1k	Str	N	34	34	S	C	N	N	-	3	Y	XC6VLX75T	4922	4676	7651	12	94	273	3230	11.83
	1	1k	Str	N	34	34	U	C	N	N	-	3	Y	XC6VLX75T	5279	4531	8096	14	102	307	3230	10.52
	1	1k	Str	N	34	34	B	C	N	N	-	3	Y	XC6VLX75T	5689	4998	8300	14	102	271	3232	11.93
	1	4k	Str	N	24	24	S	C	N	N	-	5	Y	XC6VLX75T	4475	4062	6485	30	78	326	12452	38.20
	1	4k	Str	N	24	24	U	C	N	N	-	5	Y	XC6VLX75T	4701	4234	7180	36	78	287	12448	43.37
	1	8k	Str	N	24	24	S	C	N	N	-	6	Y	XC6VLX75T	4864	4510	7203	56	90	290	24755	85.36
	1	8k	Str	N	24	24	U	C	N	N	-	6	Y	XC6VLX75T	5624	4763	8131	69	94	265	24753	93.41
U[12]	1	512	R2L	Y	24	24	S	C	N	N	B	-	N	XC6VLX75T	1324	1034	1597	4	4	385	5807	15.08
Test	1	8k	Str	Y	24	24	U	C	N	N	-	6	Y	XC6VLX75T	6369	5870	9621	69	94	287	24765	86.29
	1	8k	Str	Y	24	24	U	C	N	N	-	6	N	XC6VLX75T	7812	7500	11015	69	51	309	24752	80.10
	1	16k	Str	Y	24	24	U	C	N	N	-	7	Y	XC6VLX75T	6679	6384	10361	132	98	253	49348	195.05
	1	16k	Str	Y	24	24	U	C	N	N	-	7	N	XC6VLX75T	9363	8214	11778	132	51	283	49334	174.33
	1	256	R4	N	32	24	F	-	R	N	B	-	Y	XC6VLX75T	3526	3331	4730	16	40	350	1418	4.05
	1	1k	R4	N	32	24	F	-	R	N	B	-	Y	XC6VLX75T	3737	3429	4885	18	40	328	5536	16.88
	1	4k	R4	N	32	24	F	-	R	N	B	-	Y	XC6VLX75T	3831	3510	4993	37	40	343	22710	66.21
	1	256	R2	N	32	24	F	-	R	N	B	-	Y	XC6VLX75T	2434	2162	3066	8	12	384	2232	5.81
	1	1k	R2	N	32	24	F	-	R	N	B	-	Y	XC6VLX75T	2508	2261	3140	10	12	377	9434	25.02
	1	4k	R2	N	32	24	F	-	R	N	B	-	Y	XC6VLX75T	2608	2318	3203	31	12	378	41212	109.03
	1	256	R2L	N	32	24	F	-	R	N	B	-	Y	XC6VLX75T	2330	1998	2952	6	6	395	3176	8.04
	1	1k	R2L	N	32	24	F	-	R	N	B	-	Y	XC6VLX75T	2409	2055	3020	10	6	370	14448	39.05
	1	4k	R2L	N	32	24	F	-	R	N	B	-	Y	XC6VLX75T	2573	2075	3074	31	6	385	65656	170.54
	1	256	Str	N	32	24	F	-	R	N	B	1	Y	XC6VLX75T	3789	3581	5513	4	46	348	892	2.56
	1	1k	Str	N	32	24	F	-	R	N	B	3	Y	XC6VLX75T	4723	4237	6843	12	66	299	3216	10.76
	1	4k	Str	N	32	24	F	-	R	N	B	5	Y	XC6VLX75T	5400	4951	8258	36	86	305	12452	40.83

Table 26: **Virtex-6 FPGA Family Performance and Resource Usage in Non-Realtime Mode *(Cont'd)***

Application	Channels	Point Size	Implementation(1)	Variable Point Size	Input Data Width	Phase Factor Width	Scaling Type(2)	Rounding Mode(3)	Output Ordering(4)	Cyclic Prefix Insertion	Memory Type(5)	Stages Using Block RAM	Optimize for Speed(6)	Xilinx Part(7)	LUT/FF Pairs	LUTs	FFs	18k Block RAMs(8)	XtremeDSP Slices	Max Clock Frequency(8)	Latency (cycles)(10)	Latency (µs)(11)
Radar	1	1k	R4	Y	16	16	S	C	N	N	B	-	N	XC6VLX75T	2133	2023	2727	7	9	400	3453	8.63
	1	1k	R4	Y	16	16	S	C	N	N	B	-	Y	XC6VLX75T	1973	1665	2422	7	20	387	3458	8.94
	1	1k	R4	Y	16	16	B	C	N	N	B	-	N	XC6VLX75T	2346	2150	2753	7	9	386	3453	8.95
	1	1k	R4	Y	20	16	S	C	N	N	B	-	N	XC6VLX75T	2738	2492	3198	11	9	378	3453	9.135
	1	1k	R4	Y	20	16	B	C	N	N	B	-	N	XC6VLX75T	2750	2505	3220	11	9	387	3453	8.925
	1	32k	R4	Y	16	16	S	C	N	N	B	-	N	XC6VLX75T	2882	2572	3140	84	9	337	131263	389.50
	1	32k	R4	Y	16	16	B	C	N	N	B	-	N	XC6VLX75T	2934	2658	3142	84	9	296	131263	443.46
	1	32k	Str	Y	16	16	U	C	N	N	-	10	N	XC6VLX75T	9279	7538	10878	201	31	249	98504	395.60

Notes:

1. Implementations: Str = Pipelined Streaming I/O; R4 = Radix-4, Burst I/O; R2 = Radix-2, Burst I/O; R2L = Radix-2 Lite, Burst I/O.
2. Scaling types: S = scaled; U = unscaled; B = block floating-point; F = single precision floating-point.
3. Rounding modes: C = convergent rounding; T = truncation.
4. Output ordering: N = Natural Order; R = Bit/Digit Reversed Order.
5. Memory types: B = block RAM, D = distributed RAM. Applies to data and phase factor storage in Burst I/O architectures, and to the output reorder buffer in the Pipelined Streaming I/O architecture.
6. Optimize for Speed using XtremeDSP slices in both Complex Multipliers (4-multiplier structure) and Butterfly Arithmetic.
7. The -1 speedgrade was used in all cases.
8. Virtex-6 FPGAs have 18K block RAMs that may be packed in pairs to form 36K block RAMs. map reports the number of 36K block RAMs + 18K block RAMs, which may not match the number of 18K block RAMs given here.
9. Area and maximum clock frequencies are provided as a guide. They may vary with the amount of other logic in the FPGA device, tools options, and other releases of Xilinx implementation tools. Clock frequency does not take jitter into account and should be de-rated by an amount appropriate to the clock source jitter specification.
10. Latency in clock cycles for the largest transform size.
11. Latency in microseconds for the largest transform size, when running at the maximum achievable clock frequency.
12. Ultrasound.

Spartan-6 Family

Table 27 and Table 28 show performance and resource usage numbers for Spartan-6 FPGAs for both realtime and non-realtime modes. A range of FFT cores is shown for several typical applications: Baseband 3GPP LTE, Baseband OFDM, CT scanners, Ultrasound, Test and measurement, and Radar. The parameters for each core are shown in both tables. Some rows of the table are grayed-out to indicate that these cores would not fit on the device due to FPGA resource requirements (typically insufficient I/O pins to route all core signals outside the device). None of the optional pins (ACLKN, ARESETN, OVFLO) are used and hybrid RAM is not used. The performance and resource usage numbers were produced using ISE 13.1 software, with speed file version "PRELIMINARY 1.11 2010-07-20".

Table 27: **Spartan-6 Family Performance and Resource Usage in Realtime Mode**

Application	Channels	Point Size	Implementation(1)	Configurable Point Size	Input Data Width	Phase Factor Width	Scaling Type(2)	Rounding Mode(3)	Output Ordering(4)	Cyclic Prefix Insertion	Memory Type(5)	Stages Using Block RAM	Optimize for Speed(6)	Xilinx Part(7)	LUT/FF Paris	LUTs	FFs	9k Block RAMs(8)	XtremeDSP slices	Max clock frequency(9)	Latency (clock cycles)(10)	Latency(µs)(11)
Baseband 3GPP LTE	1	1k	R2L	Y	16	16	S	C	N	Y	B	-	N	XC6SLX150T	1154	753	1187	6	2	246	12456	50.63
	1	1k	R2L	Y	16	16	S	C	N	Y	B	-	Y	XC6SLX150T	1149	728	1189	6	3	225	12476	55.45
	1	2k	R2L	Y	16	16	S	C	N	Y	B	-	N	XC6SLX150T	1168	894	1245	9	2	232	26807	115.55
	1	2k	R2L	Y	16	16	S	C	N	Y	B	-	Y	XC6SLX150T	1141	891	1247	9	3	232	26829	115.64
	4	1k	R2L	Y	16	16	S	C	N	Y	B	-	N	XC6SLX150T	2554	1586	3020	18	8	213	12456	58.48
	8	1k	R2L	Y	16	16	S	C	N	Y	B	-	N									
	4	2k	R2L	Y	16	16	S	C	N	Y	B	-	N	XC6SLX150T	2605	1742	3096	33	8	215	26807	124.68
	8	2k	R2L	Y	16	16	S	C	N	Y	B	-	N									
	1	1k	R2	Y	16	16	S	C	N	Y	B	-	N	XC6SLX150T	1292	989	1361	6	3	237	7367	31.08
	1	1k	R2	Y	16	16	S	C	N	Y	B	-	Y	XC6SLX150T	1197	842	1247	6	6	237	7357	31.04
	1	2k	R2	Y	16	16	S	C	N	Y	B	-	N	XC6SLX150T	1374	1072	1428	9	3	228	15578	68.32
	1	2k	R2	Y	16	16	S	C	N	Y	B	-	Y	XC6SLX150T	1221	935	1316	9	6	226	15567	68.88
	2	1k	R2	Y	16	16	S	C	N	Y	B	-	N	XC6SLX150T	2006	1529	2220	10	6	226	7367	32.60
	4	1k	R2	Y	16	16	S	C	N	Y	B	-	N	XC6SLX150T	3205	2654	3938	18	12	212	7367	34.75
	8	1k	R2	Y	16	16	S	C	N	Y	B	-	N									
	2	2k	R2	Y	16	16	S	C	N	Y	B	-	N	XC6SLX150T	2149	1622	2327	17	6	219	15578	71.13
	4	2k	R2	Y	16	16	S	C	N	Y	B	-	N	XC6SLX150T	3588	2785	4125	33	12	221	15578	70.49
	8	2k	R2	Y	16	16	S	C	N	Y	B	-	N									
OFDM	1	256	R2	Y	12	12	S	T	N	Y	D	-	N	XC6SLX150T	1054	868	946	0	3	236	1655	7.01
	1	256	R2	Y	12	12	S	T	N	Y	B	-	N	XC6SLX150T	872	665	932	3	3	228	1673	7.34
	1	256	R2	Y	12	12	B	T	N	Y	B	-	N	XC6SLX150T	924	700	934	3	3	218	1673	7.67

Table 27: **Spartan-6 Family Performance and Resource Usage in Realtime Mode *(Cont'd)***

Application	Channels	Point Size	Implementation[1]	Configurable Point Size	Input Data Width	Phase Factor Width	Scaling Type[2]	Rounding Mode[3]	Output Ordering[4]	Cyclic Prefix Insertion	Memory Type[5]	Stages Using Block RAM	Optimize for Speed[6]	Xilinx Part[7]	LUT/FF Paris	LUTs	FFs	9k Block RAMs[8]	XtremeDSP slices	Max clock frequency[9]	Latency (clock cycles)[10]	Latency(µs)[11]
CT scanners	1	1k	Str	N	24	24	S	C	R	N	-	3	Y	XC6SLX150T	3850	3301	5563	9	94	162	2182	13.47
	1	1k	Str	N	24	24	S	C	R	N	-	3	N	XC6SLX150T	5614	5174	7293	9	48	198	2170	10.96
	1	1k	Str	N	24	24	S	C	R	N	-	1	Y	XC6SLX150T	4477	4069	5600	3	94	166	2182	13.14
	1	1k	Str	N	24	24	S	C	R	N	-	5	Y	XC6SLX150T	3775	3147	5545	15	94	171	2182	12.76
	1	1k	Str	N	24	24	S	C	N	N	-	3	Y	XC6SLX150T	4013	3354	5704	15	94	167	3210	19.22
	1	1k	Str	N	24	24	U	C	N	N	-	3	Y	XC6SLX150T	4138	3409	6254	17	94	169	3206	18.97
	1	1k	Str	Y	24	24	S	C	N	N	-	3	Y	XC6SLX150T	4990	4001	6691	15	94	184	3219	17.49
	1	1k	Str	Y	16	24	U	C	R	N	-	3	Y	XC6SLX150T	3969	3153	5449	7	80	177	2184	12.34
	1	1k	Str	N	16	16	S	C	R	N	-	3	Y	XC6SLX150T	2545	2164	3402	6	40	176	2174	12.35
	1	1k	Str	N	16	16	S	C	R	N	-	3	N	XC6SLX150T	3063	2621	3739	6	16	219	2174	9.93
	1	1k	Str	N	16	16	S	C	N	N	-	3	Y	XC6SLX150T	2633	2243	3527	10	40	202	3202	15.85
	1	1k	Str	N	16	16	U	C	N	N	-	3	Y	XC6SLX150T	2874	2340	4169	13	52	168	3198	19.04
	1	1k	Str	N	34	34	S	C	N	N	-	3	Y	XC6SLX150T	5361	4745	8080	18	126	146	3226	22.10
	1	1k	Str	N	34	34	U	C	N	N	-	3	Y	XC6SLX150T	5450	4629	8570	21	126	131	3226	24.63
	1	1k	Str	N	34	34	B	C	N	N	-	3	Y	XC6SLX150T	5750	5028	8765	21	126	134	3228	24.09
	1	4k	Str	N	24	24	S	C	N	N	-	5	Y	XC6SLX150T	4884	4112	7069	52	118	124	12448	100.39
	1	4k	Str	N	24	24	U	C	N	N	-	5	Y	XC6SLX150T	5432	4191	7983	64	118	154	12444	80.81
	1	8k	Str	N	24	24	S	C	N	N	-	6	Y	XC6SLX150T	5342	4675	7921	101	138	154	24751	160.72
	1	8k	Str	N	24	24	U	C	N	N	-	6	Y	XC6SLX150T	5814	5077	9186	128	146	144	24749	171.87
U[12]	1	512	R2L	Y	24	24	S	C	N	N	B	-	N	XC6SLX150T	1130	906	1469	6	8	224	5803	25.91
Test	1	8k	Str	Y	24	24	U	C	N	N	-	6	Y	XC6SLX150T	7435	6187	10676	128	146	152	24761	162.90
	1	8k	Str	Y	24	24	U	C	N	N	-	6	N	XC6SLX150T	10561	9299	13696	128	80	159	24748	155.65
	1	16k	Str	Y	24	24	U	C	N	N	-	7	Y	XC6SLX150T	8325	6622	11442	254	150	134	49344	368.24
	1	16k	Str	Y	24	24	U	C	N	N	-	7	N	XC6SLX150T	11273	10105	14535	254	80	129	49330	382.40
	1	256	R4	N	32	24	F	-	R	N	B	-	Y	XC6SLX150T	3824	3215	4932	16	64	155	1414	9.12
	1	1k	R4	N	32	24	F	-	R	N	B	-	Y	XC6SLX150T	4133	3207	5096	22	64	148	5532	37.38
	1	4k	R4	N	32	24	F	-	R	N	B	-	Y	XC6SLX150T	4012	3452	5203	66	64	134	22706	169.45
	1	256	R2	N	32	24	F	-	R	N	B	-	Y	XC6SLX150T	2428	1964	2993	8	20	180	2228	12.38
	1	1k	R2	N	32	24	F	-	R	N	B	-	Y	XC6SLX150T	2488	2086	3066	18	20	166	9430	56.81
	1	4k	R2	N	32	24	F	-	R	N	B	-	Y	XC6SLX150T	2558	2121	3128	58	20	185	41208	222.75
	1	256	R2L	N	32	24	F	-	R	N	B	-	Y	XC6SLX150T	2374	1685	2824	6	10	215	3172	14.75
	1	1k	R2L	N	32	24	F	-	R	N	B	-	Y	XC6SLX150T	2437	1781	2891	18	10	220	14444	65.65
	1	4k	R2L	N	32	24	F	-	R	N	B	-	Y	XC6SLX150T	2408	1891	2944	28	10	208	65652	315.63
	1	256	Str	N	32	24	F	-	R	N	B	1	Y	XC6SLX150T	4121	3452	5822	4	70	164	888	5.41
	1	1k	Str	N	32	24	F	-	R	N	B	3	Y	XC6SLX150T	5043	4348	7386	18	102	153	3212	20.99
	1	4k	Str	N	32	24	F	-	R	N	B	5	Y	XC6SLX150T	6094	5107	9051	62	134	122	12448	102.03

Table 27: **Spartan-6 Family Performance and Resource Usage in Realtime Mode *(Cont'd)***

Application	Channels	Point Size	Implementation(1)	Configurable Point Size	Input Data Width	Phase Factor Width	Scaling Type(2)	Rounding Mode(3)	Output Ordering(4)	Cyclic Prefix Insertion	Memory Type(5)	Stages Using Block RAM	Optimize for Speed(6)	Xilinx Part(7)	LUT/FF Paris	LUTs	FFs	9k Block RAMs(8)	XtremeDSP slices	Max clock frequency(9)	Latency (clock cycles)(10)	Latency(µs)(11)
Radar	1	1k	R4	Y	16	16	S	C	N	N	B	-	N	XC6SLX150T	2158	1923	2585	10	9	206	3449	16.74
	1	1k	R4	Y	16	16	S	C	N	N	B	-	Y	XC6SLX150T	1912	1484	2277	10	20	198	3454	17.44
	1	1k	R4	Y	16	16	B	C	N	N	B	-	N	XC6SLX150T	2349	1942	2614	10	9	209	3449	16.50
	1	1k	R4	Y	20	16	S	C	N	N	B	-	N	XC6SLX150T	2757	2369	3153	14	18	197	3449	17.51
	1	1k	R4	Y	20	16	B	C	N	N	B	-	N	XC6SLX150T	2874	2455	3182	14	18	201	3449	17.16
	1	32k	R4	Y	16	16	S	C	N	N	B	-	N	XC6SLX150T	2779	2368	2993	164	9	165	131259	795.51
	1	32k	R4	Y	16	16	B	C	N	N	B	-	N	XC6SLX150T	2800	2419	3005	164	9	148	131259	886.89
	1	32k	Str	Y	16	16	U	C	N	N	-	10	N	XC6SLX150T	8761	7852	10881	389	40	104	98500	947.12

Notes:

1. Implementations: Str = Pipelined Streaming I/O; R4 = Radix-4, Burst I/O; R2 = Radix-2, Burst I/O; R2L = Radix-2 Lite, Burst I/O.
2. Scaling types: S = scaled; U = unscaled; B = block floating-point; F = single precision floating-point.
3. Rounding modes: C = convergent rounding; T = truncation.
4. Output ordering: N = Natural Order; R = Bit/Digit Reversed Order.
5. Memory types: B = block RAM, D = distributed RAM. Applies to data and phase factor storage in Burst I/O architectures, and to the output reorder buffer in the Pipelined Streaming I/O architecture.
6. Optimize for Speed using XtremeDSP slices in both Complex Multipliers (4-multiplier structure) and Butterfly Arithmetic.
7. The -2 speedgrade was used in all cases.
8. Spartan-6 FPGAs have 9K block RAMs that may be packed in pairs to form 18K block RAMs. map reports the number of 18K block RAMs + 9K block RAMs, which may not match the number of 9K block RAMs given here.
9. Area and maximum clock frequencies are provided as a guide. They may vary with the amount of other logic in the FPGA device, tools options, and other releases of Xilinx implementation tools. Clock frequency does not take jitter into account and should be de-rated by an amount appropriate to the clock source jitter specification.
10. Latency in clock cycles for the largest transform size.
11. Latency in microseconds for the largest transform size, when running at the maximum achievable clock frequency.
12. Ultrasound.

Table 28: **Spartan-6 Family Performance and Resource Usage in Non-Realtime Mode**

Application	Channels	Point Size	Implementation(1)	Configurable Point Size	Input Data Width	Phase Factor Width	Scaling Type(2)	Rounding Mode(3)	Output Ordering(4)	Cyclic Prefix Insertion	Memory Type(5)	Stages Using Block RAM	Optimize for Speed(6)	Xilinx Part(7)	LUT/FF Paris	LUTs	FFs	9k Block RAMs(8)	XtremeDSP slices	Max clock frequency(9)	Latency (clock cycles)(10)	Latency(µs)(11)
Baseband 3GPP LTE	1	1k	R2L	Y	16	16	S	C	N	Y	B	-	N	XC6SLX150T	1280	933	1354	6	2	196	12460	63.57
	1	1k	R2L	Y	16	16	S	C	N	Y	B	-	Y	XC6SLX150T	1274	914	1356	6	3	181	12480	68.95
	1	2k	R2L	Y	16	16	S	C	N	Y	B	-	N	XC6SLX150T	1298	1048	1412	9	2	204	26811	131.43
	1	2k	R2L	Y	16	16	S	C	N	Y	B	-	Y	XC6SLX150T	1302	1054	1416	9	3	202	26833	132.84
	4	1k	R2L	Y	16	16	S	C	N	Y	B	-	N	XC6SLX150T	2825	2411	3542	18	8	196	12460	63.57
	8	1k	R2L	Y	16	16	S	C	N	Y	B	-	N									
	4	2k	R2L	Y	16	16	S	C	N	Y	B	-	N	XC6SLX150T	3061	2224	3623	33	8	172	26811	155.88
	8	2k	R2L	Y	16	16	S	C	N	Y	B	-	N									
	1	1k	R2	Y	16	16	S	C	N	Y	B	-	N	XC6SLX150T	1453	1127	1507	6	3	208	7371	35.44
	1	1k	R2	Y	16	16	S	C	N	Y	B	-	Y	XC6SLX150T	1344	1007	1410	6	6	206	7361	35.73
	1	2k	R2	Y	16	16	S	C	N	Y	B	-	N	XC6SLX150T	1443	1260	1594	9	3	213	15582	73.15
	1	2k	R2	Y	16	16	S	C	N	Y	B	-	Y	XC6SLX150T	1305	1117	1482	9	6	212	15571	73.45
	2	1k	R2	Y	16	16	S	C	N	Y	B	-	N	XC6SLX150T	2175	1852	2484	10	6	194	7371	37.99
	4	1k	R2	Y	16	16	S	C	N	Y	B	-	N	XC6SLX150T	3702	3177	4440	18	12	170	7371	43.36
	8	1k	R2	Y	16	16	S	C	N	Y	B	-	N									
	2	2k	R2	Y	16	16	S	C	N	Y	B	-	N	XC6SLX150T	2205	1977	2613	17	6	179	15582	87.05
	4	2k	R2	Y	16	16	S	C	N	Y	B	-	N	XC6SLX150T	3941	3307	4652	33	12	179	15582	87.05
	8	2k	R2	Y	16	16	S	C	N	Y	B	-	N									
OFDM	1	256	R2	Y	12	12	S	T	N	Y	D	-	N	XC6SLX150T	1163	977	1079	0	3	226	1659	7.34
	1	256	R2	Y	12	12	S	T	N	Y	B	-	N	XC6SLX150T	980	746	1051	3	3	217	1677	7.73
	1	256	R2	Y	12	12	B	T	N	Y	B	-	N	XC6SLX150T	990	777	1044	3	3	218	1677	7.69

Table 28: **Spartan-6 Family Performance and Resource Usage in Non-Realtime Mode *(Cont'd)***

Application	Channels	Point Size	Implementation[1]	Configurable Point Size	Input Data Width	Phase Factor Width	Scaling Type[2]	Rounding Mode[3]	Output Ordering[4]	Cyclic Prefix Insertion	Memory Type[5]	Stages Using Block RAM	Optimize for Speed[6]	Xilinx Part[7]	LUT/FF Paris	LUTs	FFs	9k Block RAMs[8]	XtremeDSP slices	Max clock frequency[9]	Latency (clock cycles)[10]	Latency(µs)[11]
CT scanners	1	1k	Str	N	24	24	S	C	R	N	-	3	Y	XC6SLX150T	4127	3764	5770	9	94	131	2186	16.69
	1	1k	Str	N	24	24	S	C	R	N	-	3	N	XC6SLX150T	8119	7800	9290	9	48	160	2174	13.59
	1	1k	Str	N	24	24	S	C	R	N	-	1	Y	XC6SLX150T	4797	4373	5810	3	94	129	2186	16.95
	1	1k	Str	N	24	24	S	C	R	N	-	5	Y	XC6SLX150T	3907	3385	5676	15	94	136	2186	16.07
	1	1k	Str	N	24	24	S	C	N	N	-	3	Y	XC6SLX150T	4461	4010	5908	15	94	140	3214	22.96
	1	1k	Str	N	24	24	U	C	N	N	-	3	Y	XC6SLX150T	4406	4025	6269	17	94	136	3210	23.60
	1	1k	Str	Y	24	24	S	C	N	N	-	3	Y	XC6SLX150T	5083	4457	6839	15	94	144	3223	22.38
	1	1k	Str	Y	16	24	U	C	R	N	-	3	Y	XC6SLX150T	4044	3337	5462	7	80	151	2188	14.49
	1	1k	Str	N	16	16	S	C	R	N	-	3	Y	XC6SLX150T	2661	2274	3540	6	40	187	2178	11.65
	1	1k	Str	N	16	16	S	C	R	N	-	3	N	XC6SLX150T	3138	2752	3858	6	16	206	2178	10.57
	1	1k	Str	N	16	16	S	C	N	N	-	3	Y	XC6SLX150T	2751	2378	3667	10	40	175	3206	18.32
	1	1k	Str	N	16	16	U	C	N	N	-	3	Y	XC6SLX150T	3049	2594	4222	13	52	140	3202	22.87
	1	1k	Str	N	34	34	S	C	N	N	-	3	Y	XC6SLX150T	5696	5288	8311	18	126	119	3230	27.14
	1	1k	Str	N	34	34	U	C	N	N	-	3	Y	XC6SLX150T	5622	5054	8548	21	126	114	3230	28.33
	1	1k	Str	N	34	34	B	C	N	N	-	3	Y	XC6SLX150T	6008	5423	8751	21	126	101	3232	32.00
	1	4k	Str	N	24	24	S	C	N	N	-	5	Y	XC6SLX150T	5249	4727	7266	52	118	111	12452	112.18
	1	4k	Str	N	24	24	U	C	N	N	-	5	Y	XC6SLX150T	5689	5166	7943	64	118	125	12448	99.58
	1	8k	Str	N	24	24	S	C	N	N	-	6	Y	XC6SLX150T	6007	5315	8135	101	138	125	24755	198.04
	1	8k	Str	N	24	24	U	C	N	N	-	6	Y	XC6SLX150T	6068	5533	9087	128	146	106	24753	233.52
U[12]	1	512	R2L	Y	24	24	S	C	N	N	B	-	N	XC6SLX150T	1324	1187	1673	6	8	199	5807	29.18
Test	1	8k	Str	Y	24	24	U	C	N	N	-	6	Y	XC6SLX150T	7605	6392	10486	128	146	107	24765	231.45
	1	8k	Str	Y	24	24	U	C	N	N	-	6	N	XC6SLX150T	16260	15006	17713	128	80	130	24752	190.40
	1	16k	Str	Y	24	24	U	C	N	N	-	7	Y	XC6SLX150T	8086	6950	11209	254	150	117	49348	421.78
	1	16k	Str	Y	24	24	U	C	N	N	-	7	N	XC6SLX150T	17150	15724	18501	254	80	114	49334	432.75
	1	256	R4	N	32	24	F	-	R	N	B	-	Y	XC6SLX150T	4477	3618	5086	16	64	144	1418	9.85
	1	1k	R4	N	32	24	F	-	R	N	B	-	Y	XC6SLX150T	4436	3926	5246	22	64	146	5536	37.92
	1	4k	R4	N	32	24	F	-	R	N	B	-	Y	XC6SLX150T	4747	3856	5356	66	64	140	22710	162.21
	1	256	R2	N	32	24	F	-	R	N	B	-	Y	XC6SLX150T	2697	2263	3186	8	20	180	2232	12.40
	1	1k	R2	N	32	24	F	-	R	N	B	-	Y	XC6SLX150T	2833	2326	3259	18	20	176	9434	53.60
	1	4k	R2	N	32	24	F	-	R	N	B	-	Y	XC6SLX150T	2896	2371	3321	58	20	177	41212	232.84
	1	256	R2L	N	32	24	F	-	R	N	B	-	Y	XC6SLX150T	2522	2133	3060	6	10	197	3176	16.12
	1	1k	R2L	N	32	24	F	-	R	N	B	-	Y	XC6SLX150T	2588	2240	3125	18	10	188	14448	76.85
	1	4k	R2L	N	32	24	F	-	R	N	B	-	Y	XC6SLX150T	2676	2253	3176	58	10	176	65656	373.05
	1	256	Str	N	32	24	F	-	R	N	B	1	Y	XC6SLX150T	4607	4239	6091	4	70	143	892	6.24
	1	1k	Str	N	32	24	F	-	R	N	B	3	Y	XC6SLX150T	5527	4851	7637	18	102	125	3216	25.73
	1	4k	Str	N	32	24	F	-	R	N	B	5	Y	XC6SLX150T	6485	5281	9168	62	134	108	12452	115.30

Table 28: **Spartan-6 Family Performance and Resource Usage in Non-Realtime Mode *(Cont'd)***

Application	Channels	Point Size	Implementation(1)	Configurable Point Size	Input Data Width	Phase Factor Width	Scaling Type(2)	Rounding Mode(3)	Output Ordering(4)	Cyclic Prefix Insertion	Memory Type(5)	Stages Using Block RAM	Optimize for Speed(6)	Xilinx Part(7)	LUT/FF Paris	LUTs	FFs	9k Block RAMs(8)	XtremeDSP slices	Max clock frequency(9)	Latency (clock cycles)(10)	Latency(µs)(11)
Radar	1	1k	R4	Y	16	16	S	C	N	N	B	-	N	XC6SLX150T	2420	2115	2731	10	9	189	3453	18.27
	1	1k	R4	Y	16	16	S	C	N	N	B	-	Y	XC6SLX150T	2064	1679	2423	10	20	205	3458	16.87
	1	1k	R4	Y	16	16	B	C	N	N	B	-	N	XC6SLX150T	2285	2125	2753	10	9	190	3453	18.17
	1	1k	R4	Y	20	16	S	C	N	N	B	-	N	XC6SLX150T	2921	2732	3323	14	18	181	3453	19.08
	1	1k	R4	Y	20	16	B	C	N	N	B	-	N	XC6SLX150T	3026	2505	3345	14	18	203	3453	17.01
	1	32k	R4	Y	16	16	S	C	N	N	B	-	N	XC6SLX150T	2976	2662	3144	164	9	148	131263	886.91
	1	32k	R4	Y	16	16	B	C	N	N	B	-	N	XC6SLX150T	2956	2625	3142	164	9	144	131263	911.55
	1	32k	Str	Y	16	16	U	C	N	N	-	10	N	XC6SLX150T	9019	7987	10896	389	40	103	98504	956.35

Notes:

1. Implementations: Str = Pipelined Streaming I/O; R4 = Radix-4, Burst I/O; R2 = Radix-2, Burst I/O; R2L = Radix-2 Lite, Burst I/O.
2. Scaling types: S = scaled; U = unscaled; B = block floating-point; F = single precision floating-point.
3. Rounding modes: C = convergent rounding; T = truncation.
4. Output ordering: N = Natural Order; R = Bit/Digit Reversed Order.
5. Memory types: B = block RAM, D = distributed RAM. Applies to data and phase factor storage in Burst I/O architectures, and to the output reorder buffer in the Pipelined Streaming I/O architecture.
6. Optimize for Speed using XtremeDSP slices in both Complex Multipliers (4-multiplier structure) and Butterfly Arithmetic.
7. The -2 speedgrade was used in all cases.
8. Spartan-6 FPGAs have 9K block RAMs that may be packed in pairs to form 18K block RAMs. map reports the number of 18K block RAMs + 9K block RAMs, which may not match the number of 9K block RAMs given here.
9. Area and maximum clock frequencies are provided as a guide. They may vary with the amount of other logic in the FPGA device, tools options, and other releases of Xilinx implementation tools. Clock frequency does not take jitter into account and should be de-rated by an amount appropriate to the clock source jitter specification.
10. Latency in clock cycles for the largest transform size.
11. Latency in microseconds for the largest transform size, when running at the maximum achievable clock frequency.
12. Ultrasound.

Support

Xilinx provides technical support for this LogiCORE IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support of product if implemented in devices that are not defined in the documentation, if customized beyond that allowed in the product documentation, or if changes are made to any section of the design labeled *DO NOT MODIFY*.

Refer to the IP Release Notes Guide (XTP025) for further information on this core. There is a link to all the DSP IP and then to each core. For each core, there is a master Answer Record that contains the Release Notes and Known Issues list for each core. The following information is listed for each version of the core:

- New Features
- Bug Fixes
- Known Issues

Ordering Information

This LogiCORE IP module is included at no additional cost with the Xilinx ISE Design Suite software and is provided under the terms of the Xilinx End User License Agreement. Use the CORE Generator software included with the ISE Design Suite to generate the core. For more information, please visit the core page.

Please contact your local Xilinx sales representative for pricing and availability of additional Xilinx LogiCORE modules and software. Information about additional Xilinx LogiCORE modules is available on the Xilinx IP Center.

References

1. W. R. Knight and R. Kaiser, *A Simple Fixed-Point Error Bound for the Fast Fourier Transform, IEEE Trans. Acoustics, Speech and Signal Proc.*, Vol. 27, No. 6, pp. 615-620, December 1979.
2. L. R. Rabiner and B. Gold, *Theory and Application of Digital Signal Processing*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1975.
3. Quang Hung Nguyen and Istvan Kollar, *Limited Dynamic Range of Spectrum Analysis Due To Round off Errors Of The FFT*, available at: home.mit.bme.hu/~kollar/papers/IMTC-FFT.pdf
4. I. Szolik, K. Kovac, V. Smiesko, *Influence of Digital Signal Processing on Precision of Power Quality Parameters Measurement*, available at: www.measurement.sk/2003/S1/Szolik.pdf.
5. *Xilinx AXI Reference Guide* (UG761) available at www.xilinx.com/support/ip_documentation/ug761_axi_reference_guide.pdf.

Revision History

Date	Version	Description of Revisions
09/21/10	1.0	First release of the core with AXI interface support. The previous release of this document was ds260.
03/01/11	1.1	Support added for Virtex-7 and Kintex-7. ISE Design Suite 13.1.
7/25/12	1.2	Updated m_axis_data_tvalid data direction on page 32.

Notice of Disclaimer