FFT Core User Guide

UG-CORE-FFT-v1.2
June 2023
www.elitestek.com

Contents

Introduction

The fast Fourier transform (FFT) is a common and efficient method to calculate the discrete Fourier transform (DFT). The FFT core computes the FFT using the 2-parallel radix-2^2 feedforward algorithm. The FFT core takes in a complex data vector as input and outputs the complex vector in the natural order in the frequency domain.

Note: The FFT core is provided as an early access download available in the Support Portal.

Features

- Forward and inverse FFT (Run-time configurable)
- Fixed-point, and Bfloat16 or FP32 floating-point input format
- 16 to 4096 FFT points
- Scaling modes for fixed-point format
- Verilog HDL RTL and simulation testbench

Device Support

Table 1: FFT Core Device Support

FPGA Family	Supported Device
Trion	All
钛金系列	All

Resource Utilization and Performance

Note: The resources and performance values provided are based on some of the supported FPGAs. These values are just guidance and change depending on the device resource utilization, design congestion, and user design.

Table 2: 钛金系列 Resource Utilization and Performance

Generated with Verilog HDL in Efinity v2022.2

FPGA	Data Width	FFT Points	Logic Elements (Logic, Adders, Flipflops, etc.)	Memory Block	DSP Block	f_{MAX} (MHz)	
						clk	clk_fast
Ti180 M484 C4	Fixed-Point without Scaling						
	16-bit	64	3227/172800 (1.9%)	46/1280 (3.6%)	48/640 (7.5%)	133	-
		256	5251/172800 (3%)	62/1280 (4.8%)	72/640 (11.3%)	132	-
	32-bit	64	9327/172800 (5.4%)	68/1280 (5.3%)	48/640 (7.5%)	103	-
		256	9340/172800 (5.4%)	92/1280 (7.2%)	144/640 (22.5%)	100	-
	Fixed-Point with Scaling Mode 1 or 2						
	16-bit	64	2907/172800 (1.7%)	34/1280 (2.7%)	12/640 (1.9%)	232	-
		256	4478/172800 (2.6%)	46/1280 (3.6%)	18/640 (2.8%)	230	-
		1024	6792/172800 (3.9%)	96/1280 (7.5%)	24/640 (3.8%)	228	-
	32-bit	64	5283/172800 (3.1%)	68/1280 (5.3%)	48/640 (7.5%)	141	-
		256	7805/172800 (4.5%)	92/1280 (7.2%)	72/640 (11.3%)	131	-
		1024	11059/172800 (6.4%)	173/1280 (13.5%)	96/640 (15%)	128	-
	Floating-Point						
	16-bit	64	3905/172800 (2.3%)	38/1280 (3%)	28/640 (4.4%)	250	500
		256	5911/172800 (3.4%)	52/1280 (4%)	40/640 (6.3%)	249	498
		1024	8659/172800 (5%)	104/1280 (8.1%)	52/640 (8.1%)	248	496
	32-bit	64	21616/172800 (12.5%)	76/1280 (5.9%)	32/640 (5%)	118	236
		256	31049/172800 (18%)	104/1280 (8.1%)	48/640 (7.5%)	118	236
		1024	41512/172800 (24%)	189/1280 (14.8%)	64/640 (10%)	118	236

Table 3: Trion Resource Utilization and Performance

Generated with Verilog HDL in Efinity v2022.2

FPGA	Data Width	FFT Point	Logic Elements (Logic, Adders, Flipflops, etc.)	Memory Block	Multiplier Block	f_{MAX} (MHz)	
						clk	clk_fast
T120 F576 C4	Fixed-Point without Scaling						
	16-bit	64	3620 / 112128 (3.2%)	46 / 1056 (4.4%)	48 / 320 (15%)	65	-
		256	5617 / 112128 (5%)	62 / 1056 (5.9%)	72 / 320 (22.5%)	63	-
	Fixed-Point with Scaling Mode 1 or 2						
	16-bit	64	2656 / 112128 (2.4%)	34 / 1056 (3.2%)	12 / 320 (3.8%)	97	-
		256	4133 / 112128 (3.7%)	46 / 1056 (4.4%)	18 / 320 (5.6%)	97	-
	32-bit	64	5908 / 112128 (5.3%)	68 / 1056 (6.4%)	48 / 320 (15%)	61	-
		256	8771 / 112128 (7.8%)	92 / 1056 (8.7%)	72 / 320 (22.5%)	60	-
	Floating-Point						
	32-bit	64	22868 / 112128 (20.4%)	76 / 1056 (7.2%)	32 / 320 (10%)	35	70
		256	32021 / 112128 (28.6%)	104 / 1056 (9.8%)	48 / 320 (15%)	35	70

Installing the Core

The FFT core is an Early Access core and it is not included in the Efinity IP Manager. To obtain the core, download the **efx_fft-v**<version>**.zip** file from the Support Center.

The file contains:

File or Folder	Description
/source/efinity/efx_fft.sv	FFT core RTL source file.
/script	Contains the twiddle factor scripts.
/testbench	Contains simulation testbench files.

To install the FFT core:

1. Unzip and copy the files into your project directory.
2. Instantiate the FFT core in your top-level wrapper file.

   You can refer to the **top.v** wrapper file included in the downloaded IP files. Ensure to check the UUID used is the latest one.

   Example:

```
module top (
     <ports>
 );

efx_fft_<uuid> #  (
    <parameters>
)  fft_inst  (
    <ports>
);
endmodule
```

   The `<uuid>` is included in the /source/efinity/efx_fft.sv file. For example, `` `define IP_UUID _06b4d75bf4ee4adc977f2f23caa2abda `` then the `<uuid>` is `06b4d75bf4ee4adc977f2f23caa2abda`.

   Note: You can also customize the core using parameters in your design file. The core uses the default settings if no parameter is set.

3. Generate the twiddle factor by running the **Twiddle Factor Script** on page 7 and include the generated **.hex** files in the same folder as your Efinity project file.
4. Open your project in the Efinity software, click **File** > **Edit Project**, and in the **Design** tab, add design file and select **/source/efinity/efx_fft.sv**.

   Note: If required, update the Synopsys Design Constraints (**.sdc**) file before compiling your design.

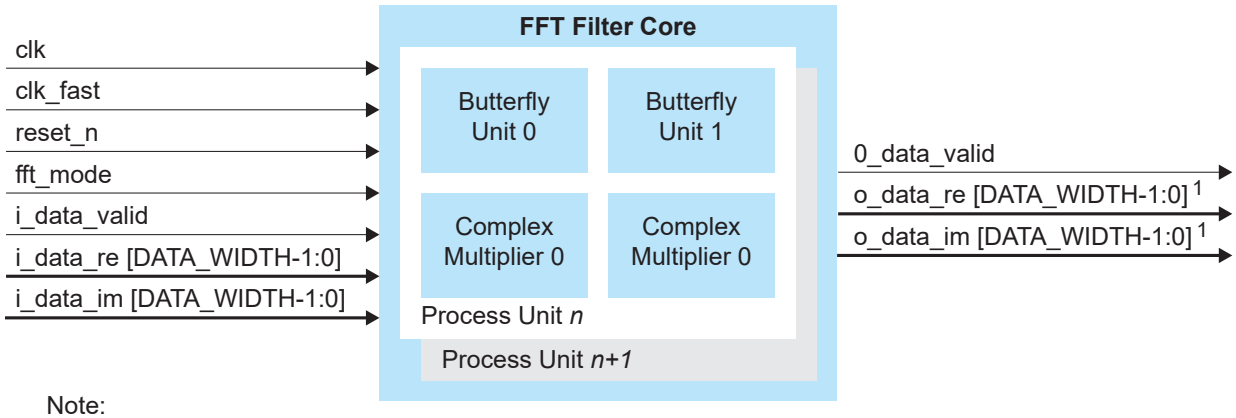
5. Compile your project.

Functional Description

The FFT core consists of the following main blocks:

- Process Unit—Instantiated based on the number of FFT stages
- Butterfly Unit—Performs addition and subtraction operations
- Complex Multiplier Unit—Performs arithmetic operations for complex numbers

Figure 1: FFT System Block Diagram



Note:

1 For fixed-point with no scaling, the interface width is [(DATA_WIDTH + $\log_2$(FFT_POINT))-1:0].

Ports

Table 4: FFT Ports

Port	Direction	Description
clk	Input	System clock that drives input and samples the output.
clk_fast	Input	Input clock that is double the clk and drives the DSP block when you enable the floating-point mode. The core ignores this port if you use the fixed-point mode.
reset_n	Input	Asynchronous reset signal.
fft_mode	Input	Set the FFT mode: 0: Forward FFT 1: Inverse FFT
i_data_valid	Input	Input data valid signal. Drive this signal high to indicate the input data is valid.
i_data_re [DATA_WIDTH-1:0]	Input	Real input data that represent a signed number of data precision bits.
i_data_im [DATA_WIDTH-1:0]	Input	Imaginary input data that represent a signed number of data precision bits.
o_data_valid	Output	Output data valid signal. The core drives this signal high when the output data is ready.
o_data_re [DATA_WIDTH-1:0]	Output	Real output data. For fixed-point operation without scaling, n = DATA_WIDTH + $\log_2$(FFT_POINT)
o_data_im [DATA_WIDTH-1:0]	Output	Imaginary output data. For fixed-point operation without scaling, n = DATA_WIDTH + $\log_2$(FFT_POINT)

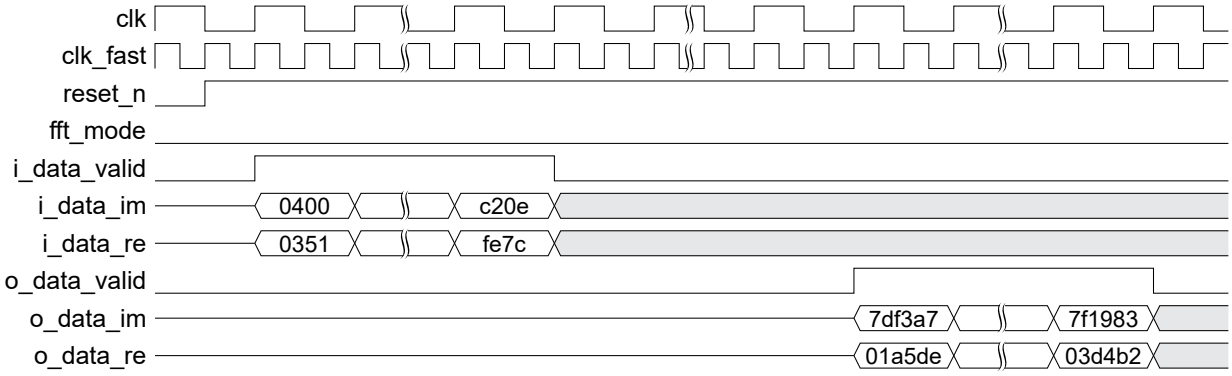
Operations

The following waveform example illustrates an FFT core operation with the following settings:

- FFT points: 128
- Data width: 16
- FFT mode: Forward FFT
- Scaling: No
- Input format: Fixed-point

The FFT core starts receiving inputs on `i_data_re` and `i_data_im` buses when you assert the `i_data_valid` signal. Deassert the `i_data_valid` signal to indicate the end of input data. The FFT then starts processing the input and asserts the `o_data_valid` signal when the operation is done. The `o_data_re` and `o_data_im` then output the FFT core results.

Figure 2: FFT Operation Example Waveform



Twiddle Factor Script

易灵思 provides the following Python scripts to generate the twiddle factor for the FFT core:

- **twiddle.py**—For fixed-point mode
- **twiddle_float.py**—For floating-point mode

You need to run the script before compiling your design in Efinity. The script outputs **.hex** files that contain the twiddle factor based on the parameters you set.

Include the number of FFT points (NUM_FFT_POINT) and data width (DATA_WIDTH) parameters when running the script:

```
python twiddle.py <NUM_FFT_POINT> <DATA_WIDTH>
```

or

```
python twiddle_float.py <NUM_FFT_POINT> <DATA_WIDTH>
```

Customizing the FFT

The FFT core has parameters so you can customize its function.

Table 5: FFT Core Parameters

Parameters	Options	Description
NUM_POINT	16, 32, 64, 128, 256, 512, 1024, 2048, 4096	Number of FFT points.
ENABLE_FLOATING_POINT	0, 1	0: Fixed-point mode 1: Floating-point mode
DATA_WIDTH	8, 16, 32	Set the data width. Fixed-point mode supports all DATA_WIDTH options. Floating-point mode supports: • 16—The core uses Bfloat16 floating-point format(1) • 32—The core uses FP32 floating-point format
SCALING_MODE	0, 1, 2	Set the scaling mode that is only supported in fixed-point mode. 0: No scaling 1: Mode 1 scaling. The output of each butterfly stage is shifted right by 1. 2: Mode 2 scaling. The output of the first butterfly stage is shifted right by 2, while other stages are shifted right by 1. The FFT core produces more accurate results when no scaling is used but utilizes more FPGA resources.

(1) Available in 钛金系列 FPGAs only.

FFT Core Testbench

The FFT includes a simulation testbench that simulates an FFT operation.

易灵思 provides a simulation script for you to run the testbench quickly using either Modelsim or NCSim. To run the testbench script, run `vsim -do modelsim.do` or `sh run_ncsim.sh` in a terminal application. You must have Modelsim or NCSim installed on your computer to use these scripts.

The testbench outputs the FFT values in **output_im.txt** and **output_re.txt**. You can then compare the values with the expected values included in **expected_output_im** and **expected_output_re**.

The testbench is set to the following default settings:

- 128 FFT points
- 16-bit data width
- Fixed-point mode
- Forward FFT mode
- No scaling

You can modify the testbench settings. However, 易灵思 only provides the expected value file for the testbench with default values.

To run the testbench with different values, perform the following task before running the testbench script:

1. Change the parameters in the **top_tb.sv** file to modify the testbench settings.
2. Generate the twiddle factor by running the **Twiddle Factor Script** on page 7.
3. Prepare the input data in **input_im** and **input_re** files.

FFT Core Resource Utilization and Output Accuracy

The FFT core supports various data widths, FFT points, and input data ranges. However, these settings will affect the FFT accuracy and the FPGA resource utilization. For example, using an FFT point that is too large, or feeding the FFT core with a high input value makes the FFT core output less accurate and also utilizes more FPGA resources.

The following tables list the supported data width, FFT points, and input data range based on the following requirements:

- Resource utilization—Less than 60% utilization of the FPGA logic, DSP/multiplier blocks, and memory blocks
- Accuracy tolerance of:
  - FP32 floating-point format—5%
  - Other formats—10% (input value > 50) or 5 (input value ≤ 50)

Table 6: Supported FFT Settings in 钛金系列 FPGAs

Data Width	FFT Point	FPGA					Valid Input Range (Min, Max)
		Ti35	Ti60	Ti90	Ti120	Ti180	
Fixed-Point without Scaling							
8-bit	8 - 64	✓	✓	✓	✓	✓	$(-2^6, 2^6 - 1)$
	128 - 256	✓	✓	✓	✓	✓	$(-2^3, 2^3 - 1)$
16-bit	8 - 64	✓	✓	✓	✓	✓	$(-2^{14}, 2^{14} - 1)$
	128		✓	✓	✓	✓	
	256 - 512		✓	✓	✓	✓	$(-2^{11}, 2^{11} - 1)$
32-bit	8 - 64	✓	✓	✓	✓	✓	$(-2^{30}, 2^{30} - 1)$
	128 - 512			✓	✓	✓	
	1024				✓	✓	
	2048					✓	
Fixed-Point with Scaling Mode 1 or 2							
8-bit	8 - 64	✓	✓	✓	✓	✓	$(-2^6, 2^6 - 1)$
16-bit	8 - 512	✓	✓	✓	✓	✓	$(-2^{14}, 2^{14} - 1)$
	1024		✓	✓	✓	✓	
	2048			✓	✓	✓	
	4096				✓	✓	
32-bit	8 - 64	✓	✓	✓	✓	✓	$(-2^{30}, 2^{30} - 1)$
	128 - 512		✓	✓	✓	✓	
	1024 - 2048			✓	✓	✓	
Floating-Point							
16-bit	8 - 16	✓	✓	✓	✓	✓	EXP: (10, 130) MAN: $(0, 2^7 - 1)$
	64 - 512	✓	✓	✓	✓	✓	EXP: (10, 120) MAN: $(0, 2^7 - 1)$
	1024		✓	✓	✓	✓	
	2048			✓	✓	✓	
	4096				✓	✓	
32-bit	8 - 32		✓	✓	✓	✓	EXP: (60, 240) MAN: $(0, 2^{23} - 1)$
	64 - 256			✓	✓	✓	
	512 - 1024			✓	✓	✓	
	2048			✓	✓	✓	EXP: (60, 130) MAN: $(0, 2^{23} - 1)$

Table 7: Supported FFT Settings in Trion FPGAs

Data Width	FFT Point	FPGA					Valid Input Range (Min, Max)
		T20	T35	T55	T85	T120	
Fixed-Point without Scaling							
8-bit	8 - 64	✓	✓	✓	✓	✓	$(-2^6, 2^6-1)$
	128 - 256	✓	✓	✓	✓	✓	$(-2^3, 2^3-1)$
16-bit	8	✓	✓	✓	✓	✓	$(-2^{14}, 2^{14}-1)$
	16 - 128		✓	✓	✓	✓	
	256		✓	✓	✓	✓	$(-2^{11}, 2^{11}-1)$
	512				✓	✓	
32-bit	8 - 16		✓	✓	✓	✓	$(-2^{30}, 2^{30}-1)$
	32 - 64				✓	✓	
	128 - 256					✓	
Fixed-Point with Scaling Mode 1 or 2							
8-bit	8 - 64	✓	✓	✓	✓	✓	$(-2^6,2^6-1)$
16-bit	8 - 256	✓	✓	✓	✓	✓	$(-2^{14},2^{14}-1)$
	512		✓	✓	✓	✓	
	1024			✓	✓	✓	
	2048				✓	✓	
32-bit	8 - 256		✓	✓	✓	✓	$(-2^{30},2^{30}-1)$
	512 - 1024				✓	✓	
Floating-Point							
32-bit	8	✓	✓	✓	✓	✓	EXP: (60, 240) MAN: $(0,2^{23}-1)$
	16		✓	✓	✓	✓	
	32 - 256			✓	✓	✓	
	512 - 1024				✓	✓	

Revision History

Table 8: Revision History

Date	Version	Description
June 2023	1.2	Added Device Support section. (DOC-1234) Improved Installing the Core description. (DOC-1287)
May 2023	1.1	Improved steps to install the core. (DOC-1244)
March 2023	1.0	Initial release.