

Features

- Uses the Altera® fft MegaCore™ function
- Optimized for the Altera FLEX® 10K device architecture
- Uses FLEX 10K embedded array blocks (EABs) to store both data and twiddle factors
- Supported by the Altera MAX+PLUS® II software
- Parameterized number of points and data width

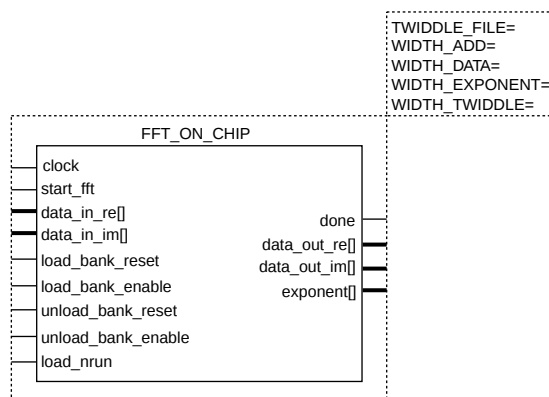
General Description

The `fft_on_chip` reference design implements a fast Fourier transform (FFT) function using the Altera fft MegaCore function. The `fft` function provides maximum flexibility by allowing the designer to choose the memory and I/O interface.

The `fft_on_chip` design has a memory architecture that consists of left, right, and twiddle memory banks. Each memory bank has both real and imaginary elements that are read in parallel. The `fft_on_chip` uses FLEX 10K EABs for all the memory banks, allowing an EPF10K100 device to implement an `fft` function with up to 256 data points and a 16-bit data width.

Figure 1 shows the symbol for the `fft_on_chip` reference design.

Figure 1. `fft_on_chip` Symbol



Function Prototype

The Altera Hardware Description Language (AHDL™) Function Prototype for the fft_on_chip reference design is shown below:

```
FUNCTION fft_on_chip (clock, start_fft,
    data_in_re[WIDTH_DATA-1..0],
    data_in_im[WIDTH_DATA-1..0], load_bank_reset,
    load_bank_enable, unload_bank_reset,
    unload_bank_enable, load_nrun)
WITH (WIDTH_DATA, TWIDDLE_FILE, WIDTH_TWIDDLE,
    WIDTH_EXPONENT, WIDTH_ADD)
RETURNS (done, data_out_re[WIDTH_DATA-1..0],
    data_out_im[WIDTH_DATA-1..0],
    exponent[WIDTH_EXPONENT-1..0]);
```

VHDL Component Declaration

The VHDL Component Declaration for the fft_on_chip reference design is shown below:

```
COMPONENT fft_on_chip
GENERIC(
    TWIDDLE_FILE      : STRING;
    WIDTH_ADD         : POSITIVE;
    WIDTH_DATA        : POSITIVE;
    WIDTH_EXPONENT    : POSITIVE;
    WIDTH_TWIDDLE     : POSITIVE;
);
PORT(
    clock : IN STD_LOGIC := '0';
    start_fft, load_bank_reset, load_bank_enable,
    unload_bank_reset, unload_bank_enable,
    load_nrun : IN STD_LOGIC;
    data_in_re,
    data_left_in_im :
    IN STD_LOGIC_VECTOR(WIDTH_DATA-1 DOWNTO 0);
    done : OUT STD_LOGIC;
    data_out_re, data_out_im : OUT
    STD_LOGIC_VECTOR(WIDTH_ADD - 1 DOWNTO 0);
    exponent : OUT
    STD_LOGIC_VECTOR(WIDTH_EXPONENT-1 DOWNTO 0));
END COMPONENT;
```

Ports

Table 1 shows the ports for the fft_on_chip reference design.

Table 1. fft_on_chip Ports			
Name	Type	Required	Description
clock	Input	Yes	Clock signal.
start_fft	Input	Yes	Starts the fft processor after data is loaded.
data_in_re[]	Input	Yes	Real data input.
data_in_im[]	Input	Yes	Imaginary data input.
load_bank_reset	Input	Yes	Resets the load counter.
load_bank_enable	Input	Yes	Enables the load counter.
unload_bank_reset	Input	Yes	Resets the unload counter.
unload_bank_enable	Input	Yes	Enables the unload counter.
load_nrun	Input	Yes	Places fft_on_chip in either load or run mode. When load_nrun is high, the left and right memory banks are controlled by the load and unload counters and other input signals. When load_nrun is low, the memory banks are controlled by the fft MegaCore function.
done	Output	Yes	Goes high when the fft function has completed the calculation and data is available for unloading.
data_out_re[]	Output	Yes	Real data output.
data_out_im[]	Output	Yes	Imaginary data output.
exponent[]	Output	Yes	Exponent of the resultant data.

Parameters

Table 2 shows the parameters for the fft_on_chip reference design.

Table 2. fft_on_chip Parameters		
Name	Value	Description
TWIDDLE_FILE	String	Contains the ROM for the twiddle memory bank.
WIDTH_ADD	Integer	Data address bus width within fft_on_chip. The number of points is $2^{\text{WIDTH_ADD}}$.
WIDTH_DATA	Integer	Data width.
WIDTH_EXPONENT	Integer	Number of bits in the exponent, which is calculated by $\text{ceil}(\log_2(2 \times \text{WIDTH_ADD})) + 1$.
WIDTH_TWIDDLE	Integer	Twiddle width. There are $2^{\text{WIDTH_ADD} - 1}$ twiddles.

Functional Description

The `fft_on_chip` reference design is made up of several essential parts, which are described below:

- *fft MegaCore function*—Controls all aspects of the actual FFT computation. The `fft` MegaCore function is active when the `load_nrun` input is driven low.
- *Left and right memory banks*—A dual memory architecture in which data is simultaneously read from one memory bank and written to the other. The memory banks are made up of RAM blocks that are $2 \times \text{WIDTH_DATA}$ bits wide and $2^{\text{WIDTH_ADD}}$ words deep, and the memory banks hold both real and imaginary data elements. The width of the address bus port is `WIDTH_ADD`.
- *Twiddle memory bank*—A ROM block that is $2 \times \text{WIDTH_TWIDDLE}$ bits wide and $2^{\text{WIDTH_ADD} - 1}$ words deep. It holds the real and imaginary elements of the twiddle factors. The twiddle memory does not require a write enable signal or an I/O interface; it connects directly to the `fft` MegaCore function.
- *Load and unload counters*—Both counters are `WIDTH_ADD` bits wide. The load counter counts through the addresses when loading the right memory bank in normal bit order. The unload counter counts through the addresses for the left memory bank in bit-reversed order.

To use the `fft_on_chip` reference design, follow these steps:

1. Reset the load and unload counters by asserting the `load_bank_enable` and `unload_bank_reset` inputs for one or more clock cycles.
2. Drive the `load_nrun` input high to place the `fft` MegaCore into load mode.
3. Drive the `load_bank_enable` input high for $2^{\text{WIDTH_ADD} - 1}$ clock cycles while applying data to the `data_in_re[]` and `data_in_im[]` inputs. The `load_bank_enable` input can be driven high and low to accommodate the incoming data.

If the `fft` function has already completed a calculation, the `unload_bank_enable` input may be asserted to retrieve the data from the `data_out_re[]` and `data_out_im[]` outputs. The loading and unloading of data are independent and carried out in any desirable order. After the `unload_bank_enable` input is driven high, there are four (`WIDTH_ADD` is odd) or five (`WIDTH_ADD` is even) pipeline stages, and valid data is received on the `data_out_re[]` and `data_out_im[]` outputs.

4. Drive the `load_nrun` input low to place the entire design into run mode.
5. Drive the `start_fft` input high for one clock cycle to instruct the `fft` MegaCore function to begin processing the loaded data.



Remember to unload all data currently in the memory, because the `fft` MegaCore function will overwrite any old data.

6. Repeat steps 2 to 6 after `done` output goes high.



For instructions on how to instantiate and simulate the `fft_on_chip` reference design, see *Application Note 84 (Implementing `fft` with On-Chip RAM in FLEX 10K Devices)*.

Example Implementations

Figures 2 and 3 show two possible implementations of the `fft_on_chip` reference design. Both implementations are embedded in the MAX+PLUSII `fft_on_chip` Text Design File (`fft_on_chip.tdf`). The parity of `WIDTH_ADD` (i.e., odd or even) determines which configuration will be used.

Figure 2 shows an implementation of the `fft_on_chip` reference design when `WIDTH_ADD` is odd.

Figure 2. fft_on_chip Block Diagram When WIDTH_ADD Is Odd

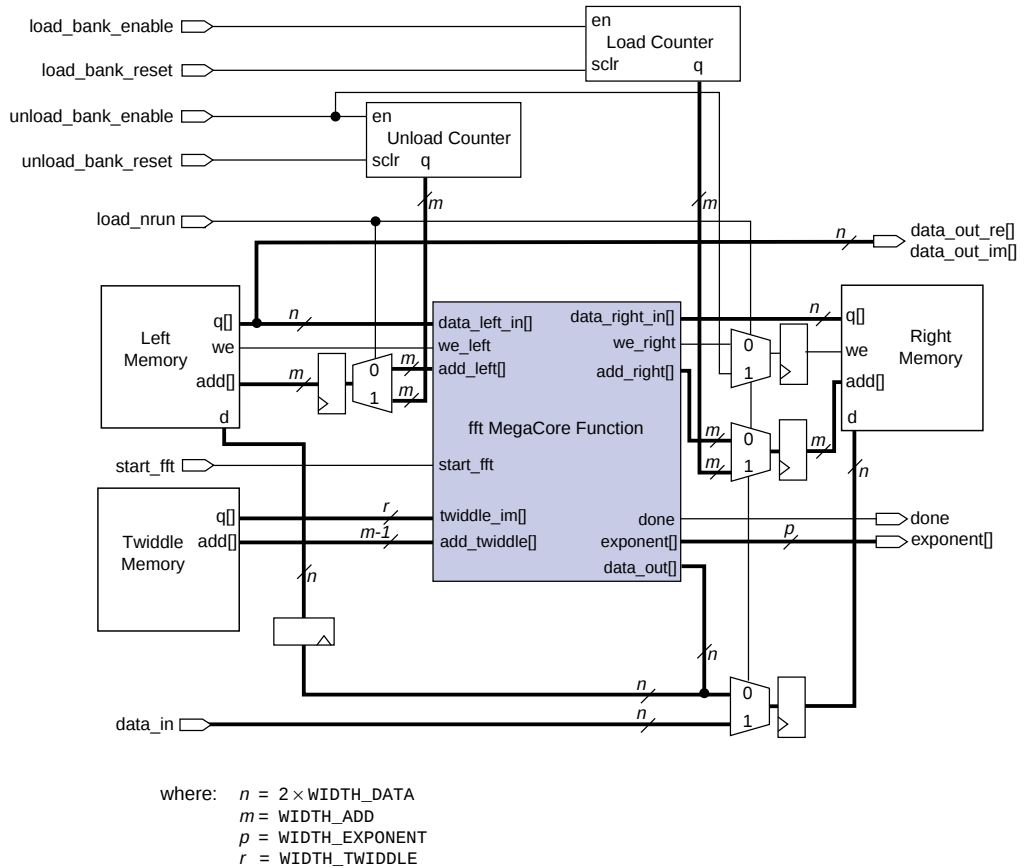


Figure 3 shows an implementation of the fft_on_chip reference design when WIDTH_ADD is even.

Figure 3. fft_on_chip Block Diagram Reference Design When WIDTH_ADD Is Even

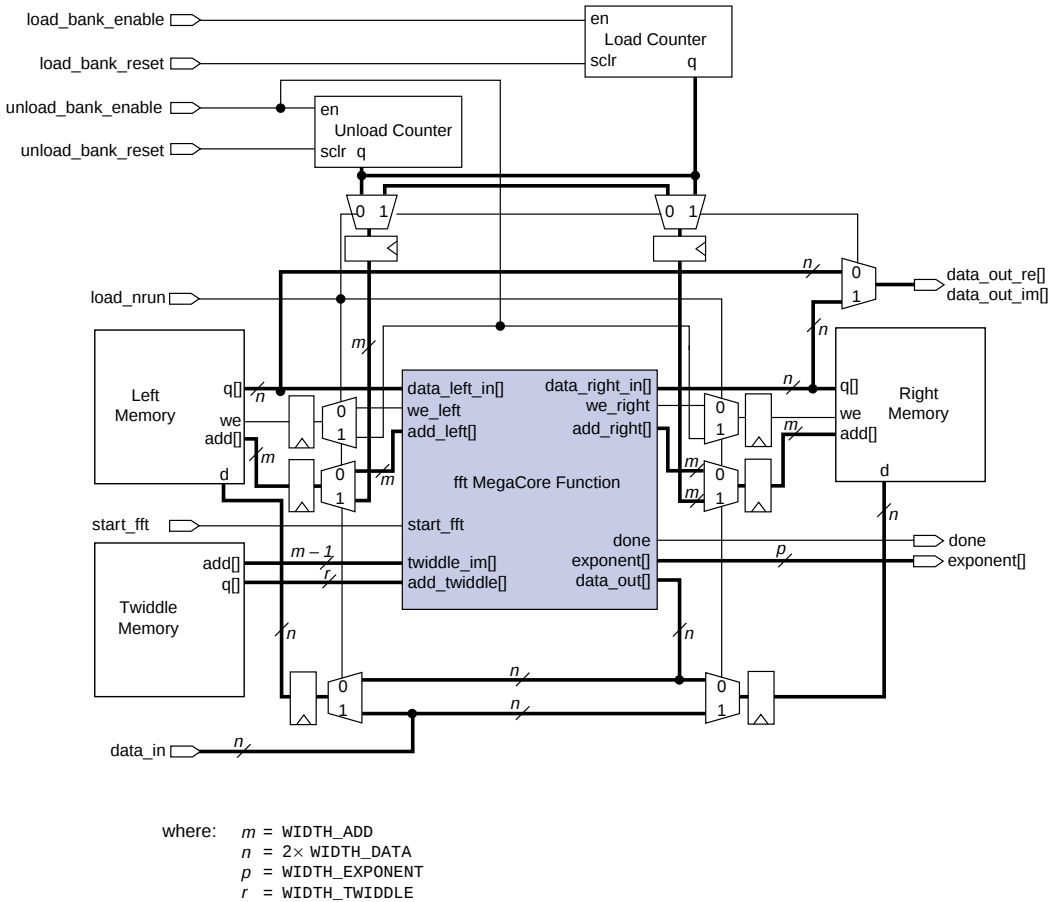


Figure 4 shows a load and unload cycle for the `fft_on_chip` function when `WIDTH_ADD` is 3. The data is loaded, and `start_fft` is asserted. When `done` goes high, the process is repeated. If desired, data can be individually loaded and unloaded by asserting the `load_bank_enable` and `unload_bank_enable` inputs at different times (i.e., once the `done` output goes high).

Figure 4. `fft_on_chip` Whole Load & Unload Cycle

X indicates “don’t care.” DV indicates “data valid.”

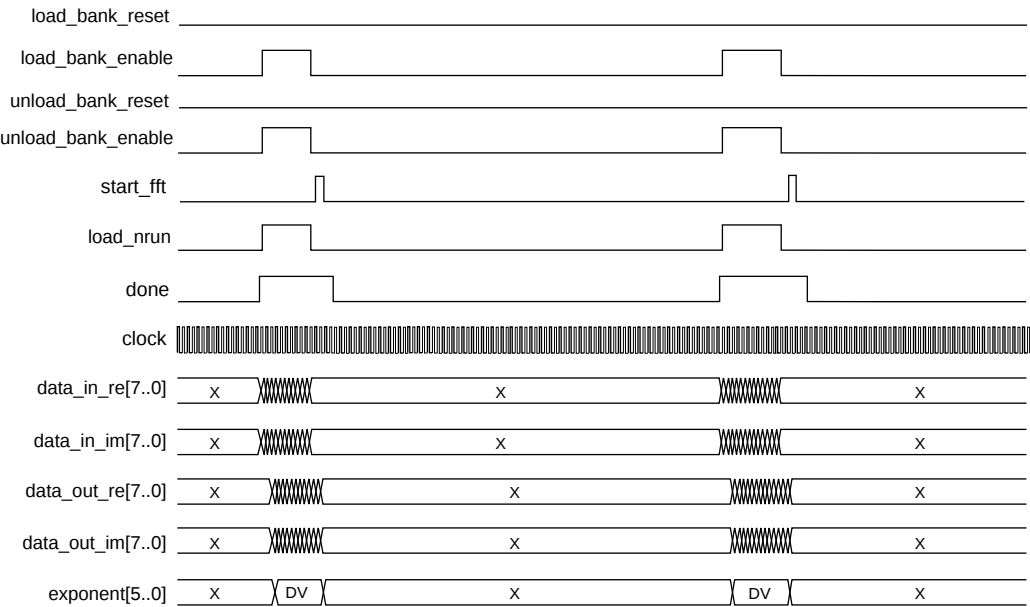


Figure 5 shows the simultaneous loading and unloading of data when WIDTH_ADD is 3. A pipeline delay of three clock cycles exists between the time when the unload_bank_enable input is asserted and valid data is received on the data_out_re[] and data_out_im[] outputs, whenever WIDTH_ADD is odd.

Figure 5. fft_on_chip Simultaneous Load & Unload Cycle When WIDTH_ADD is 3

X indicates "don't care."

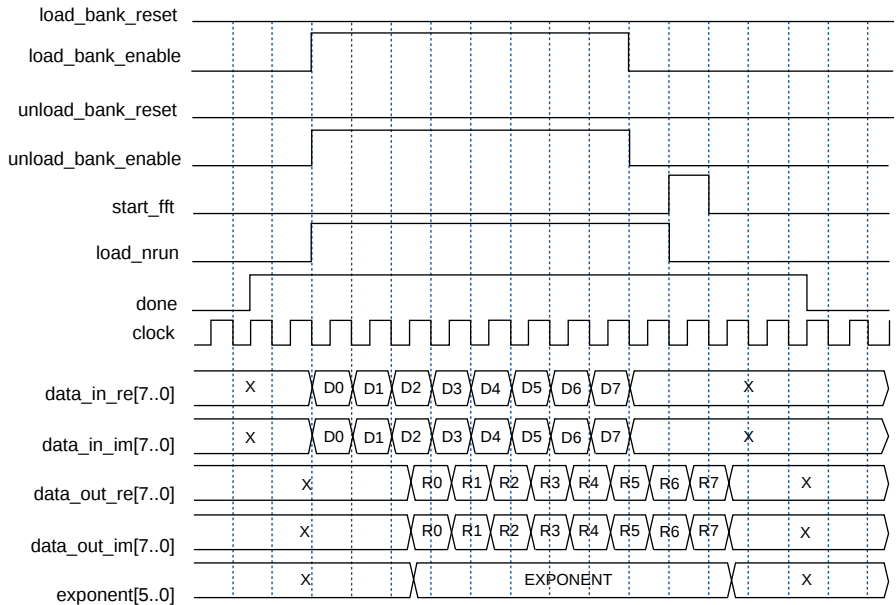
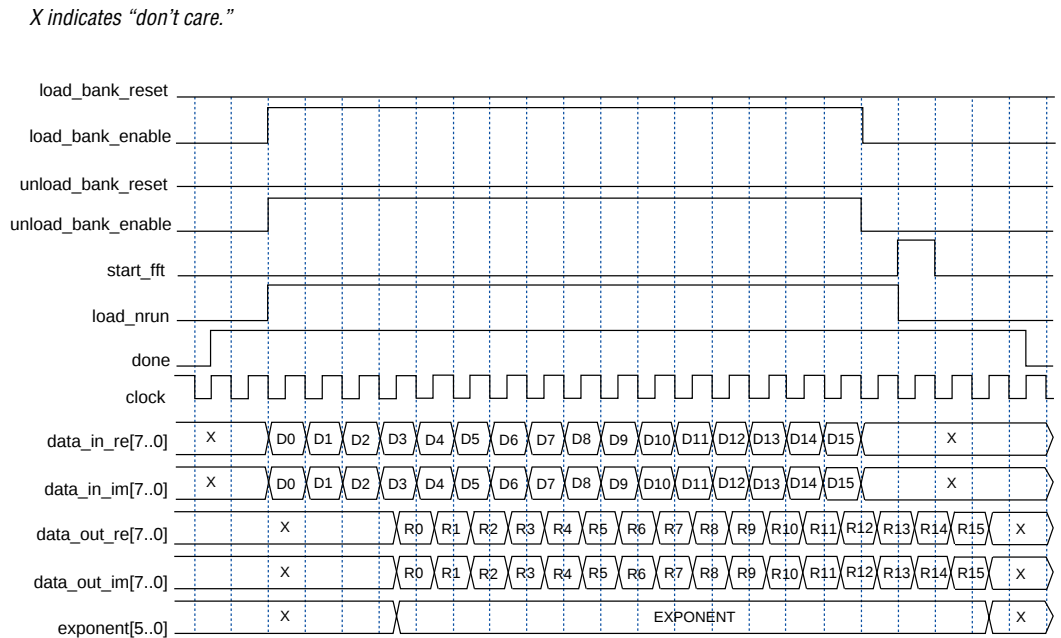


Figure 6 shows the simultaneous loading and unloading of data when `WIDTH_ADD` is 4. A pipeline delay of three clock cycles exists between the time `unload_bank_enable` input is asserted and valid data is received on the `data_out_im[]` and `data_out_re[]` outputs.

Figure 6. `fft_on_chip` Simultaneous Load & Unload Cycle When `WIDTH_ADD` is 4





2610 Orchard Parkway
San Jose, CA 95134-2020
(408) 544-7000
<http://www.altera.com>
Applications Hotline:
(800) 800-EPLD
Customer Marketing:
(408) 894-7104
Literature Services:
(888) 3-ALTERA
lit_req@altera.com

Altera, AHDL, MAX, MAX+PLUS, MAX+PLUS II, MegaCore, FLEX, FLEX 10K, and EPF10K100 are trademarks and/or service marks of Altera Corporation in the United States and other countries. Altera acknowledges the trademarks of other organizations for their respective products or services mentioned in this document. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.



Copyright © 1997 Altera Corporation. All rights reserved.

I.S. EN ISO 9001