User manual

Introduction

The Fast Fourier Transform (FFT) is an efficient algorithm for computing the Discrete Fourier Transform (DFT). This Intellectual Property (IP) core was designed to offer very fast transform times while keeping a floating point accuracy at all computational stages. 4DSP's core is the fastest and the most efficient available in the FPGA world. Based on a radix-32 architecture, it also saves memory resources compared to other floating point cores available on the market.

Features

- This IP core targets the following devices:
  - **Xilinx**: Virtex-II™, Virtex-II Pro™, Spartan-3™ and Virtex-4™
  - **Altera**: Stratix™, Cyclone-II™ and StratixII™
- Forward and inverse complex FFT
- Transform sizes: 2^m with m = 8 to 20
  (256, 512, 1024, …, 1M points)
- Arithmetic type : floating point
- Data formats
  - IEEE-754
  - 24-bit mantissa, 8-bit exponent, 2's complement
  - 14-bit mantissa, 8-bit exponent, 2's complement
  - Any mantissa and exponent precision upon request
- Configurable on the fly forward or inverse operation
- Configurable on the fly transform length
- Fully functional VHDL testbench and the related Matlab functions delivered along the FFT/IFFT core for simulation purposes and specific performance characterization.

Functional description

The Discrete Fourier Transform (DFT), of length N ($N=2^m$), calculates the sampled Fourier transform of a discrete-time sequence with N points evenly distributed.

The forward DFT with N points of a sequence x(n) can be written as follows:

$$X(k) = \sum_{n=0}^{N-1} x(n)\, e^{\frac{-j2\pi nk}{N}} \quad \text{with } k = 0, 1, \ldots, N-1$$

Equation 1: DFT

The inverse DFT is given by the following equation:

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k)\, e^{\frac{j2\pi nk}{N}} \quad \text{with } n = 0, 1, \ldots, N-1$$

Equation 2 : Inverse DFT

Algorithm

The FFT core uses a decomposition of radix-2 butterflies for computing the DFT. With 5 different stages, the processing of the transform requires log32(N) stages. To maintain an optimal signal-to-noise ratio throughout the transform calculation, the FFT core uses a floating point architecture with 8-bit exponent for the real and imaginary part of each complex sample. This FFT core employs the decimation in frequency (DIF) method.

This FFT core is designed for FFT computation larger or equal to 1k points and up to 1M points. Since FPGAs memory resources are limited and relatively small, the memory banks used for the processing of the transform are not integrated in the core. External memory, such as QDR SRAM, ZBT RAM, DDR SDRAM or SDRAM is most suited for transforms larger than 16384 points. For shorter transforms, memory banks can likely be implemented inside the FPGA depending on which device is used.

Data format

This core is compliant to the IEEE-754 standard for Binary floating-point arithmetic. There is however one restriction regarding the optional denormalized numbers of this standard that are not supported by the Floating Point FFT core. Therefore, when intermediate calculations and results data are very small there might be some differences between the values generated by the core and 4DSP's GUI. Please note however that the denormalized numbers option can be turned off on many processors.

Other data formats available for this core are coded in 2's complement for both the mantissa and exponent.
The 8-bit exponent ranges from -128 to 127
The p-bit mantissa ranges from -2^{p-1} to $2^{p-1}-1$
The exponent bit width is noted Ebw. The mantissa bit width is noted Mbw.

Parameters and Ports definitions

Parameter name	type	Value	Description
addr_width	integer	≥ 8 and ≤ 20	Address width. This parameter (also noted Abw) indicates the width of the address bus for twiddle factors and data. If N is the maximum transform length used for computing the FFT, then Abw=log2(N). Please note that the transform length can be changed on the fly by assigning a new FFT length when restarting the core. However this new transform length cannot be larger than 2^{Abw}. Assigning the smallest address width as possible is recommended for achieving higher clock frequencies during synthesis.

Table 1 : Parameters definition

Port name	Port width	Direction	Description
clk	1	Input	Clock
reset	1	Input	Asynchronous reset (active high)
cke	1	Input	Clock enable (active high). Refer to the clock enable section of this document for more information about this signal.
start	1	Input	FFT start signal (active high). The signal *start* is asserted for one clock cycle to start the core and the address generators. It is only asserted once for continuous data processing (the core will restart automatically every time a transform is complete). A new *start* pulse will act as a synchronous reset, will restart the core and discard the transform that was currently computed.
stop	1	Input	FFT stop signal (active high). The signal *stop* is asserted for one clock cycle to stop the FFT core. The results of the current FFT will be discarded once stop has been asserted.
done	1	Output	FFT done signal (active high). A *done* pulse indicates that the results of the current transform are ready. The done pulse is active one clock cycle after the last active cycle of the *result_valid* signal.
FFTlength	5	input	FFT transform length. Please refer to the Transform length section of this document for more details.
FFT_nIFFT	1	Input	FFT direction. High ⇔ FFT, Low ⇔ IFFT. This signal is registered inside the core on a start pulse.
empty_pipeline	1	Input	Empty the core pipeline before processing the next FFT/IFFT pass. If High, this signal will force the core to wait for all the data of an FFT/IFFT pass to be output before the next pass can be started. This is useful in a configuration where the single port processing memory banks are swapped every new pass. If Low the FFT core will start reading the data from the memory before the core has completed the calculations from the previous pass. This signal is registered inside the core on a *start* pulse.
tw_din_addr_valid	1	Output	Address valid strobe. This signal indicates that the current addresses on *tw_addr* and *din_addr* are valid.
tw_addr	Abw	Output	Twiddle factors address bus. This bus gives the address in the memory where the twiddle factors must be read from.
din_addr	Abw	Output	Data input address bus. This bus gives the address in the memory where the input data must be read from.
din_bank	1	Output	Data input memory bank. This signal indicates which data memory bank is used as the input bank.
tw	2.Mbw+2.Ebw or 64 for IEEE-754	Input	Twiddle factors input. This bus should be connected to the memory containing the twiddle factors. The data decomposition is as follows for **2's complement formats:** Real mantissa: bits Mbw-1 down to 0 Imag mantissa: bits 2.Mbw-1 down to Mbw Real exponent: bits 2.Mbw+ Ebw-1 down to 2.Mbw Imag exponent: bits 2.Mbw+ 2.Ebw-1 down to 2.Mbw+ Ebw **IEEE-754:** Real : bits 31 down to 0 Imag : bits 63 down to 32

din	2.Mbw+2.Ebw or 64 for IEEE-754	Input	Data input. This bus should be connected to the input data bank currently used for processing. The data decomposition is as follows for **2's complement formats:** Real mantissa: bits Mbw-1 down to 0 Imag mantissa: bits 2.Mbw-1 down to Mbw Real exponent: bits 2.Mbw+ Ebw-1 down to 2.Mbw Imag exponent: bits 2.Mbw+ 2.Ebw-1 down to 2.Mbw+ Ebw **IEEE-754:** Real : bits 31 down to 0 Imag : bits 63 down to 32
tw_din_valid	1	Input	Twiddle factors, data input valid. This signal should be asserted high when the data input (*din*) and twiddle factors (*tw*) are valid on the bus.
dout_addr	Abw	Output	Data output and results address. This bus gives the address in the memory where the output data (*dout*) must be written to.
dout_bank	1	Output	Data output memory bank. This signal indicates which data memory bank is used as the output bank.
dout	2.Mbw+2.Ebw or 64 for IEEE-754	Output	Data output. This bus should be connected to the output data bank currently used for processing. The data decomposition is as follows for **2's complement formats:** Real mantissa: bits Mbw-1 down to 0 Imag mantissa: bits 2.Mbw-1 down to Mbw Real exponent: bits 2.Mbw+ Ebw-1 down to 2.Mbw Imag exponent: bits 2.Mbw+ 2.Ebw-1 down to 2.Mbw+ Ebw **IEEE-754:** Real : bits 31 down to 0 Imag : bits 63 down to 32
dout_valid	1	Output	Data out valid strobe. This signal indicates that the data on the *dout* bus are valid and can be written to a memory bank for further processing.
result_valid	1	Output	Result valid strobe. This signal indicates that the data on the *dout* bus are the final results of the transform and must be written to the results memory bank.

Table 2 : Ports definition

Clock enable

The clock enable signal should be handled with great care. Due to the internal complexity of the FFT core, the use of the clock enable signal is subject to the following rules in order to ensure proper operation of the core:

- When a new FFT is started (start pulse), the cke signal must remain high at least till the first sample is written to the core.
- The cke signal can be driven low only during the data loading phase (first pass through the core) and during the results offloading phase (last pass though the core). During intermediate passes, the cke signal must remain high.
- When forced low, the cke signal must remain low for at least 4 consecutive clock cycles.

Transform length

The FFT transform length is a parameter fed to the core. This parameter can be either constant or can be changed on the fly in order to perform an FFT or Inverse FFT with a different transform length.

The FFT length parameter as well as the FFT direction (*FFT_nIFFT*) is registered when a start pulse is sent to the core. In the case the FFT transform length is a constant parameter passed to the core, it is recommended to match the address bit width (addr_width) with the length N of the transform: addr_width=log2(N). This will yield the best synthesis results and guarantee an optimal clock frequency for this implementation. In any other case 2^{addr_width} must be bigger or equal to the longest transform length N.
The following table shows the FFTlength code for a given transform length:

Transform length	FFTlength code	Number of passes through the core
256	00010	2
512	00011	2
1024	00100	2
2048	00101	3
4096	00110	3
8192	00111	3
16384	01000	3
32768	01001	3
65536	01010	4
131072	01011	4
262144	01100	4
524288	01101	4
1048576	01110	4

Table 3 : FFTlength codes

Twiddle factors

The twiddle factors used during the transform computation must be stored in a memory accessible by the FFT core. The twiddle factors for a forward FFT of length N are given by the following equation:

$$Tw(k) = e^{\frac{-j2\pi k}{N}} \quad \text{with } k = 0, 1, \ldots, N-1$$

Equation 3: Twiddle factors DFT

The inverse FFT twiddle factors can be calculated as follows.

$$Tw(k) = e^{\frac{j2\pi k}{N}} \quad \text{with } k = 0, 1, \ldots, N-1$$

Equation 4: Twiddle factors IDFT

The FFT core package comprises a Matlab program (FFT_test.m) and subroutines that generate the twiddles factors and write them to a file (fftcore_twiddle) in the floating point format required.

Memory

The memory banks are external to the FFT core. Two banks are dedicated to data processing. The signals din_bank and dout_bank indicate which bank is used for input and which bank is used for output. Every new pass, the banks are swapped as the FFT core needs to access the data calculated from the previous pass.

Minimal memory usage architecture

The block diagram below shows a configuration that uses as few memory banks as possible. Please note that a system using dual port memory or QDR SRAM will only require one data bank.

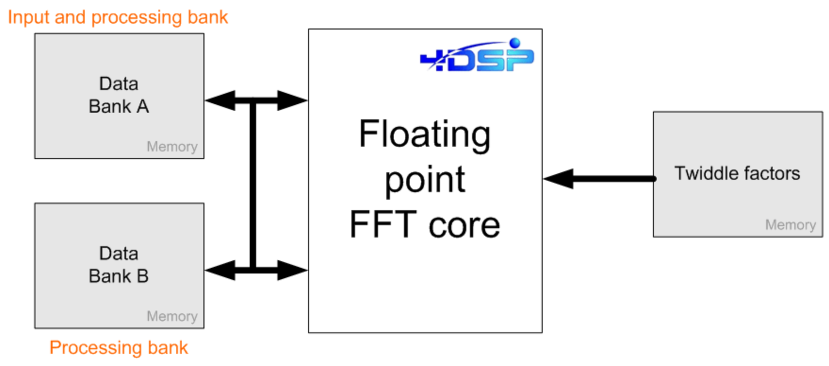


Figure 1 : Minimum memory usage architecture

The output data bank is either A or B. The number of passes through the core will help to determine which one is the output data bank. Table 3 shows the number of passes in function of the transform length. If the number is odd for a given transform length, the FFT results will be in data bank B. If even, the results will be stored in data bank A.

Streaming IO architecture

A streaming IO architecture is presented below for continuous data processing. Please note that a system using Dual Port Memory or QDR SRAM will only require two data banks.

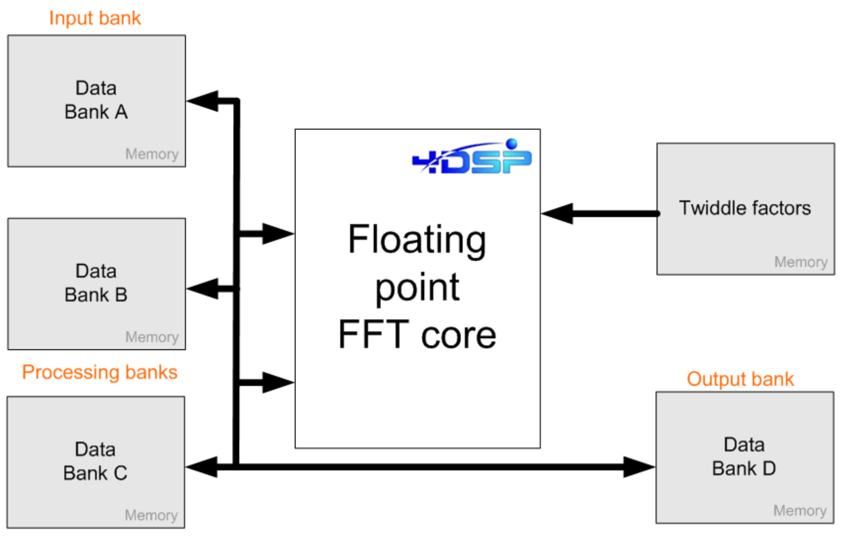


Figure 2 : Streaming IO memory architecture

Streaming IO processing with concurrent data input and data output requires 5 memory banks to be connected to the FFT core. In this type of architectures, the maximum continuous throughput depends on the number of passes through the FFT engine and the clock rate is it running at. The table below shows how the memory banks are used when performing several transforms in a row.

Bank	**Pass 1 FFT 1**	**Pass 2 FFT 1**	**Pass 3 FFT 1**	**Pass 1 FFT 2**	**Pass 2 FFT 2**	**Pass 3 FFT 2**	**Pass 1 FFT 3**	**Pass 2 FFT 3**	**Pass 3 FFT 3**
Data A	Write input data for FFT 2			FFT read	FFT write	FFT read	FFT write	FFT read	FFT write
Data B	FFT read	FFT write	FFT read	FFT write	FFT read	FFT write	Read output results of FFT 2		
Data C	FFT write	FFT read	FFT write	Read output results of FFT 1			Write input data for FFT 4		
Data D	Read output results of FFT 0			Write input data for FFT 3			FFT read	FFT write	FFT read
Twiddles	read	read	read	read	read	read	read	read	read

Table 4 : Memory banks for a streaming IO architecture

Memory latency

The FFT core generates the addresses for twiddles factors, data input and data output. The memory latency is calculated as the number of clock cycles it takes between the address is valid on the core address bus and the twiddle factors or data are available at the input of the FFT core. This latency can be up to 15 clock cycles. The FFT core expects the latency to be the same for the twiddle factors and the data input and to remain the same during the transform computation. This latency is automatically calculated inside the FFT core by monitoring the tw_din_valid signal (driven high by the user few clock cycles after tw_din_addr_valid goes high).

Radix-32 vs Radix 2

4DSP's radix-32 butterfly architecture allows the core to be connected to much less memory for the same processing performances than designs with radix-2 butterflies implemented in parallel. The following table shows how much memory is required to perform an FFT in both configurations.

FFT length	radix-32 memory required (in Mbytes)	radix-2 memory required (in Mbytes)
256	0.02	0.08
512	0.04	0.18
1024	0.08	0.39
2048	0.23	0.86
4096	0.47	1.88
8192	0.94	4.06
16384	1.88	8.75
32768	3.75	18.75
65536	10.00	40.00
131072	20.00	85.00
262144	40.00	180.00
524288	80.00	380.00
1048576	160.00	800.00

Table 5: Radix-32 vs Radix-2 memory usage

Data throughput=maximum data throughput as shown in Table 7

Using a radix-32 architecture substantially reduces the number of memory resources required. The main benefit is seen at the system level. A single-width PMC module used to perform long transforms with 4DSP's FFT core, achieves the same level of processing performances as a radix-2 implementation spread over two 6U CompactPCI boards bundled with multiple FPGAs and memory devices.

Resources usage and performances

The following table summarizes the resources usage and performances of a 24-bit mantissa, 8-bit exponent floating point FFT/IFFT core.

Device	Slices	Multipliers 18x18	Block RAMs 18Kb	Fmax
Virtex-4 *XC4VLX40 -12*	12394	40	36	200.2 MHz
Virtex-II Pro *XC2VP40 -7*	12293	40	36	175 MHz
Spartan-3 *XC3S4000-5*	12835	40	36	105.3 MHz

Table 6 : Core resources usage

The FFT/IFFT processing time with an FPGA internal clock running at 200MHz is shown in the table below.

FFT/IFFT transform size	Processing time	Sustained throughput in MSPS
256	3.68µs	69.6
512	6.24µs	82.1
1024	11.4µs	90.1
2048	31.8µs	64.3
4096	61.4µs	66.7
8192	123µs	66.7
16384	246µs	66.7
32768	492µs	66.7
65536	1.31ms	50.0
131072	2.62ms	50.0
262144	5.24ms	50.0
524288	10.5ms	50.0
1048576	21ms	50.0

Table 7: Core performances

The following graph displays the Signal to Noise Ratio of a Fast Fourier Transform performed over a 1024 points random vector with IEEE-754 accuracy. The software Discrete Fourier Transform was calculated using the FFT Matlab function in double precision mode.

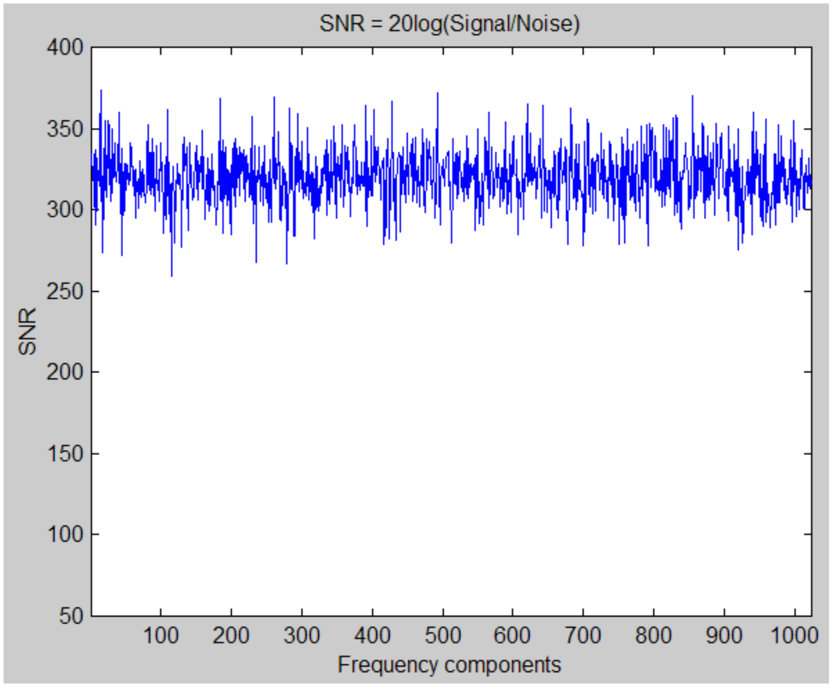


Figure 3: FFT SNR

Graphical User Interface program

The Graphical User Interface (GUI) program is a tool made for assessing the performances of the FFT/IFFT when IEEE-754 accuracy is used.
Users can perform the following from the GUI:

- Choose the transform length
- Choose the transform direction (Forward or Inverse FFT)
- Choose the number of transforms to be calculated.
- Generate data (random function, Dirac, exp(i.x))
- Check the transform results against C model

After generating data with the GUI, users should run a simulation using the VHDL testbench and check the transform results in the GUI

The GUI generates data files that can be used for further analysis using a Matlab program.

-

Testbench and Matlab programs

The FFT core package comprises a VHDL testbench and two Matlab programs to generate data and check results.

fftcore_TB.vhd: This testbench is designed to work with the FFT core. It reads the twiddle factors from a file ('fftcore_twiddle.txt') and stores them in the twiddle factors memory bank connected to the core. The input data are also read from a file ('fftcore_data_in.txt') and stored in a memory bank that will be accessed by the core once started. Upon the transform completion, the results, available in one of the processing memory banks, are written to a file ('fftcore_results.txt').
Please make sure that the file paths in the testbench VHDL file are pointing to the folder where the files are being stored.

FFT_test.m : This Matlab program generates data and twiddle factors in the floating point format expected by the core (see Data format). The FFT core data and the twiddle factors are saved in a text format respectively in the 'fftcore_data_in.txt' and 'fftcore_twiddle.txt' files. A third file, 'fftcore_parmaters.txt', contains the parameters for the FFT core simulation.
Please make sure that the file paths in the FFT_test.m Matlab program are pointing to the same files as the VHDL testbench.

Analyse_FFT_results.m : This Matlab program reads the input data file ('fftcore_data_in.txt'), the output result file ('fftcore_results.txt') from the testbench, calculates the expected results with the fft Matlab function and saves the Signal-to-Noise Ratio for each FFT/IFFT in a file (fftcore_SNR.txt).
Please make sure that the file paths in the Analyse_FFT_results.m Matlab program are pointing to the same files as the VHDL testbench.

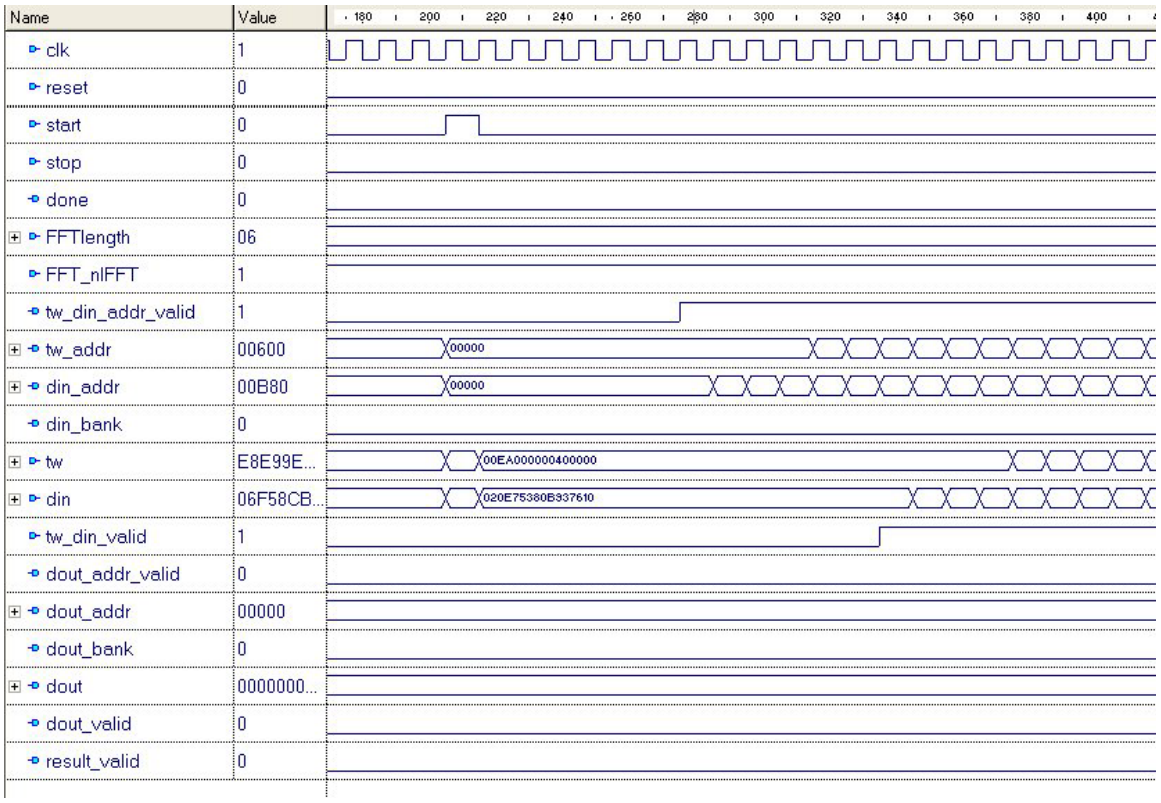
The data input file is coded in integer format for the mantissa/exponent and is organized as follow:

```
Line1: Mantissa Real0
Line2: Mantissa Imag0
Line3: Exponent Real0
Line4: Exponent Imag0
Line5: Mantissa Real1
…
```

The data output file has the same format as the data input file.

Waveforms

Start

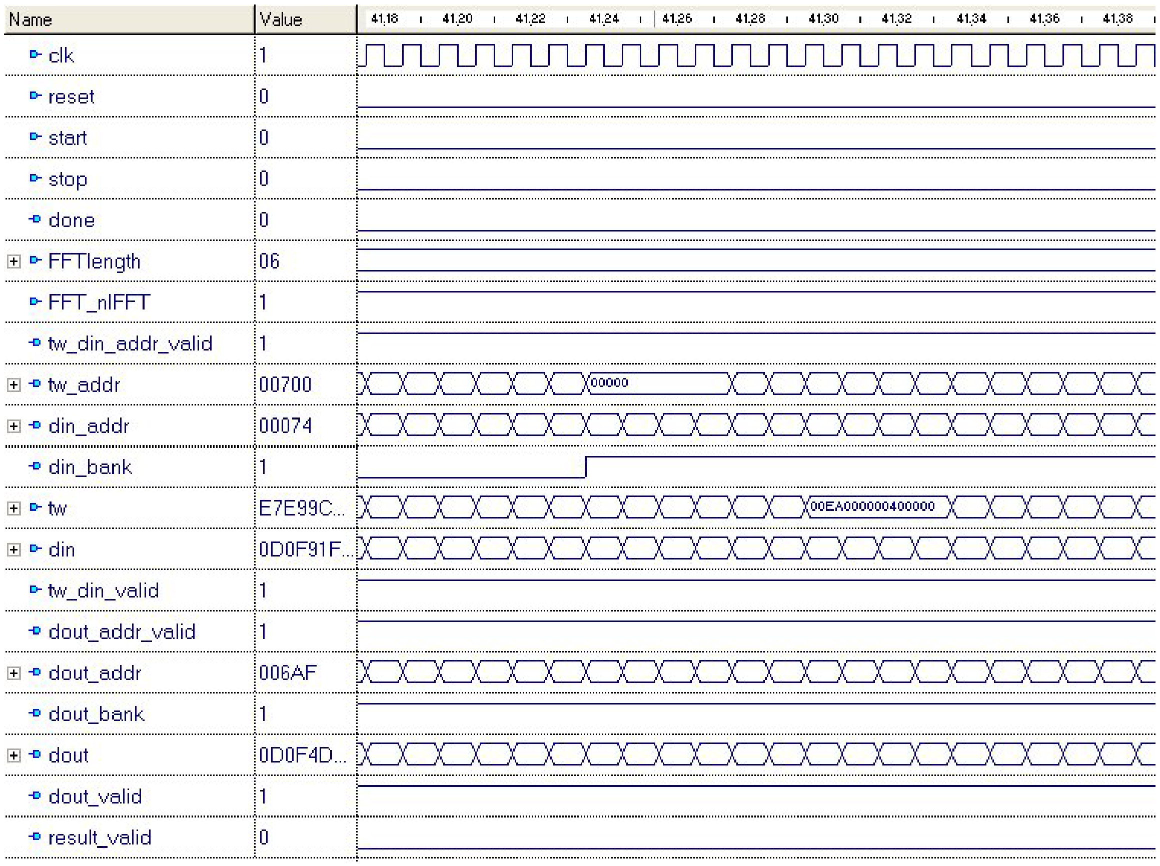


Name	Value
clk	1
reset	0
start	0
stop	0
done	0
FFTlength	06
FFT_nIFFT	1
tw_din_addr_valid	1
tw_addr	00600
din_addr	00B80
din_bank	0
tw	E8E99E...
din	06F58CB...
tw_din_valid	1
dout_addr_valid	0
dout_addr	00000
dout_bank	0
dout	0000000...
dout_valid	0
result_valid	0

Figure 4: Start

Figure 3 shows how the FFT core must be started. The start signal is driven high for one clock cycle. The first address for the data and twiddles is generated after 7 clock cycles. The user then fetches the twiddles and data in the memory and drives the signal tw_din_valid high. A new data and twiddle are then expected every new clock cycle.

Data input memory bank swap

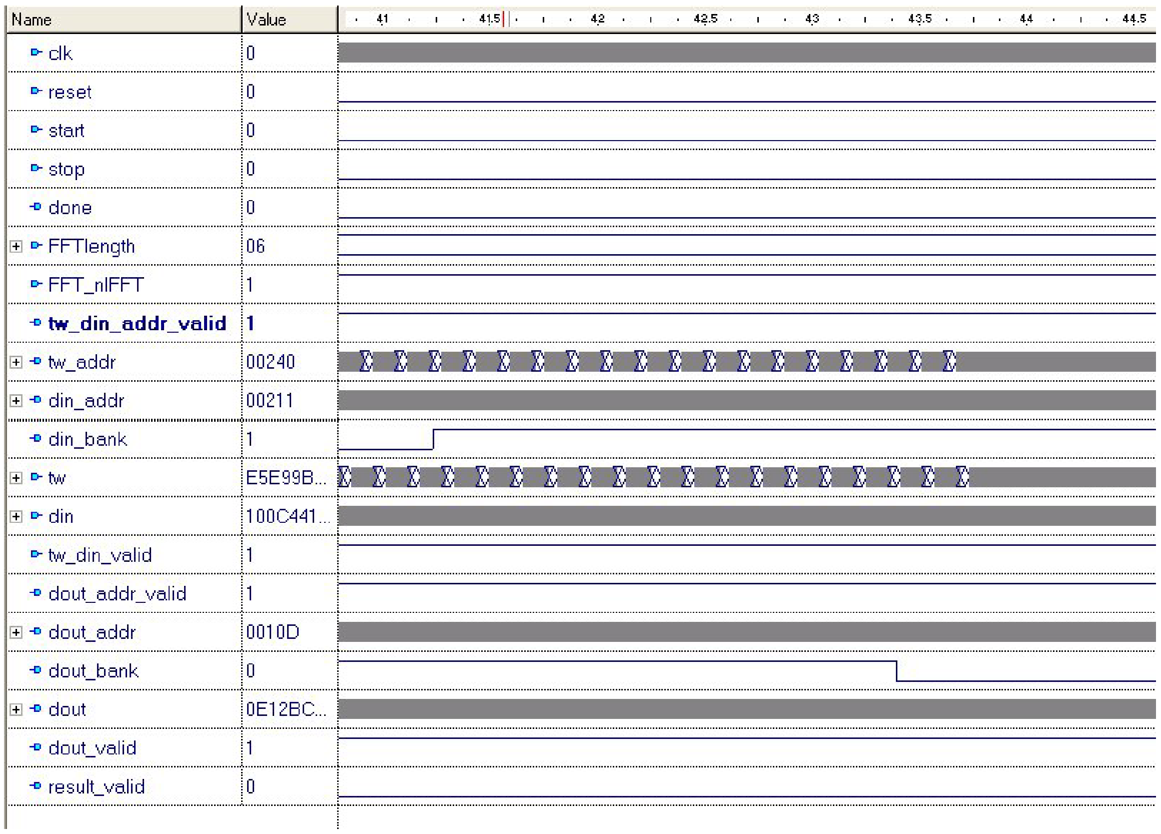


Name	Value
clk	1
reset	0
start	0
stop	0
done	0
FFTlength	06
FFT_nIFFT	1
tw_din_addr_valid	1
tw_addr	00700
din_addr	00074
din_bank	1
tw	E7E99C...
din	0D0F91F...
tw_din_valid	1
dout_addr_valid	1
dout_addr	006AF
dout_bank	1
dout	0D0F4D...
dout_valid	1
result_valid	0

Figure 5 : Memory bank swap

When the core requires a new pass to be computed, it needs to get the results data from the previous pass as input data. A pass transition is indicated by an inversion of the din_bank signal. This signal can be used to multiplex the memory banks connected to the core during processing.

Continuous processing between two consecutive passes



Name	Value
clk	0
reset	0
start	0
stop	0
done	0
FFTlength	06
FFT_nIFFT	1
tw_din_addr_valid	**1**
tw_addr	00240
din_addr	00211
din_bank	1
tw	E5E99B...
din	100C441...
tw_din_valid	1
dout_addr_valid	1
dout_addr	0010D
dout_bank	0
dout	0E12BC...
dout_valid	1
result_valid	0

Figure 6 : empty_pipeline low

When a pass transition occurs, the din_bank and dout_bank signals are inverted. However, due to the core latency, the dout_bank signal is inverted after the din_bank signal, when all the data for the previous pass have been processed through the core. Forcing the empty_pipeline signal low when starting the core will enable to continuously process data through the core without pausing between two consecutive passes. As a result the core will need to access the same memory bank for read and write operations simultaneously. Therefore, if this mode is used, the processing memory banks connected to the core must be dual port.

Halted processing between two consecutive passes

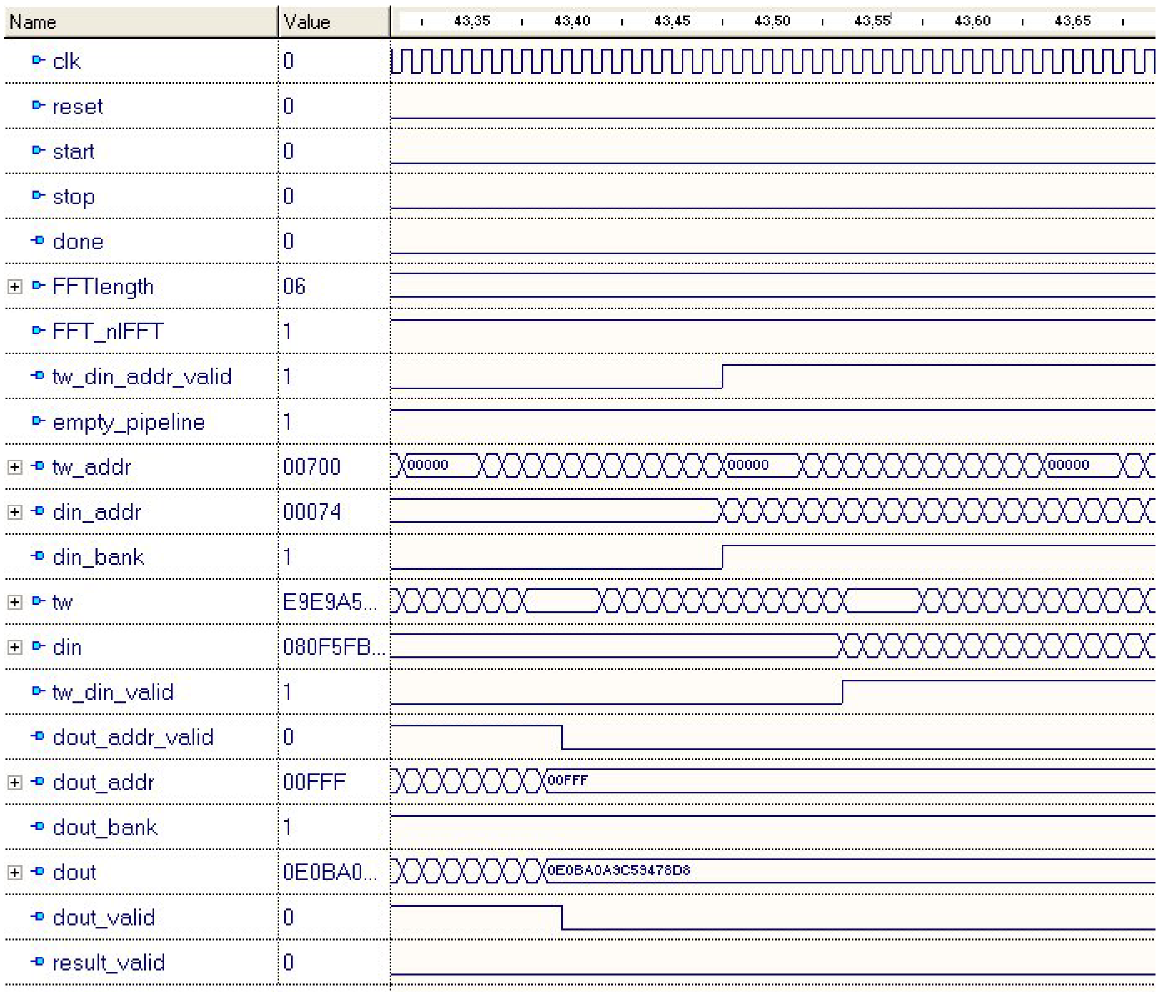
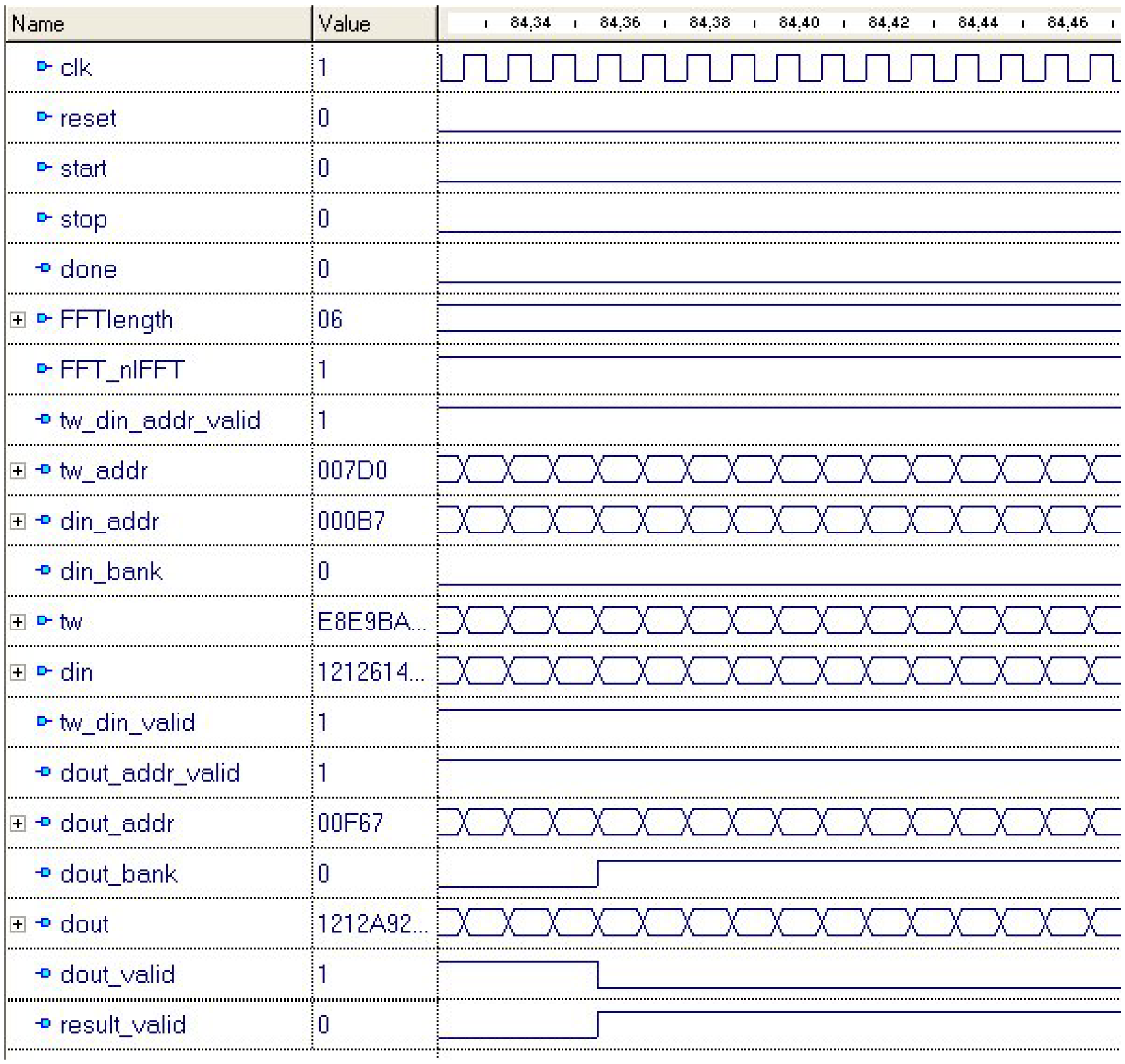


Figure 7 : empty_pipeline high

When the empty_pipeline signal is driven high, the core will pause the processing between two consecutive passes in order to empty the data pipeline. As shown on the waveform above, a new pass is started only when all the data from the previous pass have been processed through the core and written to memory. This mode should be used when the data processing memory banks are single port.

Results



Name	Value
clk	1
reset	0
start	0
stop	0
done	0
FFTlength	06
FFT_nIFFT	1
tw_din_addr_valid	1
tw_addr	007D0
din_addr	000B7
din_bank	0
tw	E8E9BA...
din	1212614...
tw_din_valid	1
dout_addr_valid	1
dout_addr	00F67
dout_bank	0
dout	1212A92...
dout_valid	1
result_valid	0

Figure 8 : Results

When the last pass of the algorithm is processed, the data coming out of the core are the results of the transform. These results are in a non-sequential order and must be written in memory at the addresses given on the dout_addr bus. The transform results are stored in memory in a bit-reversed order.

Done

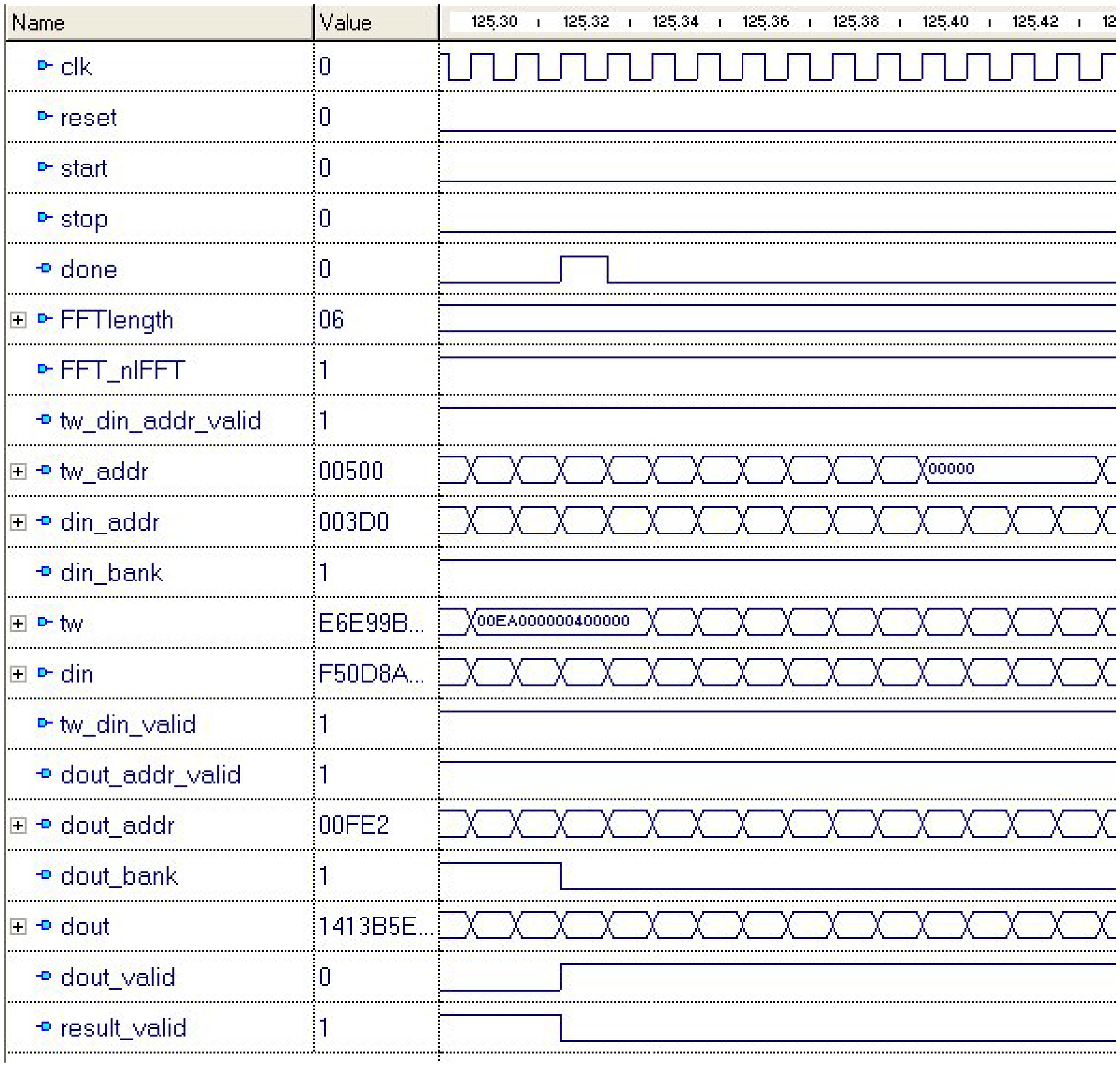


Figure 9 : Done

After the last result data has been output from the core, the done signal is high for one clock cycle, indicating the completion of the transform. A new transform is then processed through the core.