Pipelined FFT/IFFT 128 points (Fast Fourier Transform)
IP Core User Manual

Unicore Systems Ltd
60-A Saksaganskogo St
Office 1
Kiev 01033
Ukraine
Phone: +38-044-289-87-44
Fax: : +38-044-289-87-44
E-mail: o.uzenkov@unicore.co.ua
URL: www.unicore.co.ua

Overview

The FFT128 User Manual contains description of the FFT128 core architecture to explain its proper use. FFT128 soft core is the unit to perform the Fast Fourier Transform (FFT). It performs one dimensional 128 – complex point FFT. The data and coefficient widths are adjustable in the range 8 to 16.

Features

- 128 -point radix-8 FFT.
- Forward and inverse FFT.
- Pipelined mode operation, each result is outputted in one clock cycle, the latent delay from input to output is equal to 310 clock cycles (440 clock cycles when the direct output data order), simultaneous loading/downloading supported.
- Input data, output data, and coefficient widths are parametrizable in range 8 to 16 and more.
- Two and three data buffers are selected.
- FFT for 10 bit data and coefficient width is calculated on Xilinx XC4SX35-12 FPGA at 215 MHz clock cycle, and on Xilinx XC5VLX30-3 FPGA at 260 MHz clock cycle, respectively.
- FFT unit for 10 bit data and coefficients, and 3 data buffers occupies 4147 CLB slices, 4 DSP48 blocks, and 5 kbit of RAM in Xilinx XC4SX35 FPGA, and 1254 CLB slices 4 DSP48E blocks, and 5 kbit of RAM in Xilinx XC5VLX30 FPGA.
- Overflow detectors of intermediate and resulting data are present.
- Two normalizing shifter stages provide the optimum data magnitude bandwidth.
- Structure can be configured in Xilinx, Altera, Actel, Lattice FPGA devices, and ASIC.
- Can be used in OFDM modems, software defined radio, multichannel coding, wideband spectrum analysis.

Design Features

FFT is an algorithm for the effective Discrete Fourier Transform calculation

Discrete Fourier Transform (DFT) is a fundamental digital signal processing algorithm used in many applications, including frequency analysis and frequency domain processing. DFT is the decomposition of a sampled signal in terms of sinusoidal (complex exponential) components. The symmetry and periodicity properties of the DFT are exploited to significantly lower its computational requirements. The resulting algorithms are known as Fast Fourier Transforms (FFTs). An 128-point DFT computes a sequence x(n) of 128 complex-valued numbers given another sequence of data X(k) of length 128 according to the formula

$$X(k) = \sum_{n=0}^{127} x(n) e^{-j2\pi nk/128} \quad ; \quad k = 0 \text{ to } 127.$$

To simplify the notation, the complex-valued phase factor e –j2□nk/128 is usually defined as W128n where: W128 = cos(2□/128) – j sin(2□/128). The FFT algorithms take advantage of the symmetry and periodicity properties of W128n to greatly reduce the number of calculations that the DFT requires. In an FFT implementation the real and imaginary components of WnN are called twiddle factors.

The basis of the FFT is that a DFT can be divided into smaller DFTs. In the processor FFT128 a mixed radix 8 and 16 FFT algorithm is used. It divides DFT into two smaller DFTs of the length 8 and 16, as it is shown in the formula:

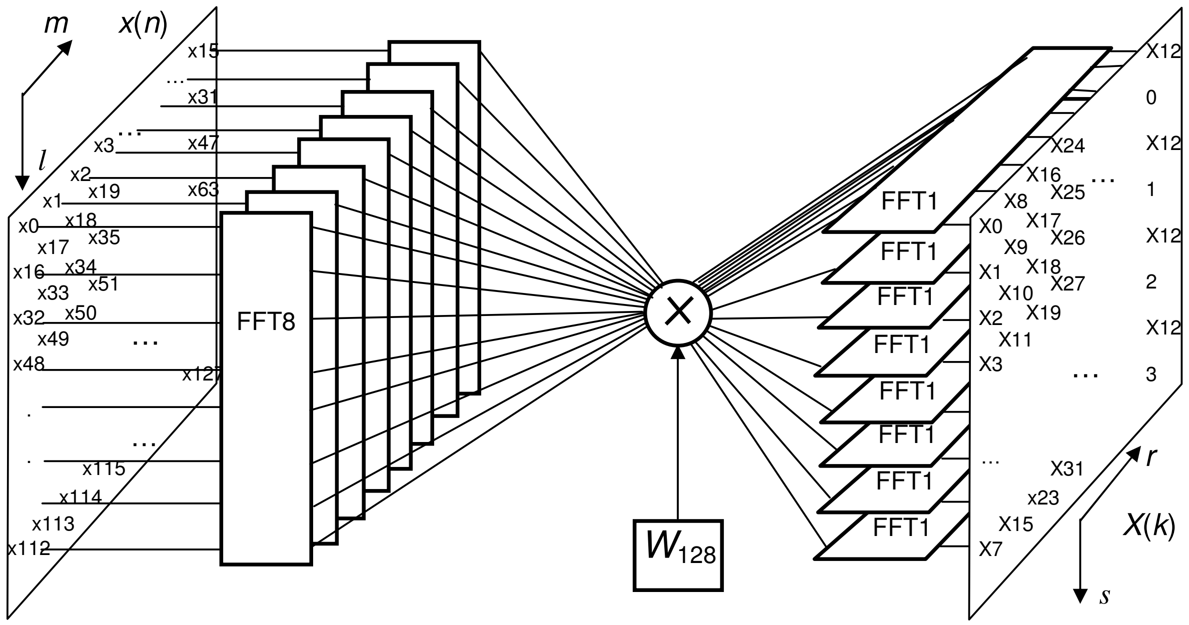
$$X(k) = X(16r+s) = \sum_{m=0}^{15} W_{16}^{mr} W_{128}^{ms} \sum_{l=0}^{7} x(16l+m)\, W_8^{sl},\ r = 0 \text{ to } 15,\ s = 0 \text{ to } 7,$$

which shows that 128-point DFT is divided into two smaller 8- and16-point DFTs. This algorithm is illustrated by the graph which is shown in the Fig.1. The input complex data x(n) are represented by the 2-dimensional array of data x(16l+m). The columns of this array are computed by 8-point DFTs. The results of them are multiplied by the twiddle factors W128ms . And the resulting array of data X(16r+s) is derived by 16-point DFTs of rows of the intermediate result array.

The 8- and 16-point DFTs are implemented by the Winograd small point FFT algorithms, which provide the minimum additions and multiplications. As a result, the radix-16 FFT algorithm needs only 128 complex multiplications to the twiddle factors W128ms and a set of multiplications to the twiddle factors W16sl except of 32768 complex multiplications in the origin DFT. Note that the well known radix-2 128-point FFT algorithm needs 896 complex multiplications.

Highly pipelined calculations

Each base FFT operation is computed by the datapaths called FFT8 and FFT16. FFT8 and FFT16 calculates the 8- and 16-point DFTs in the high pipelined mode. Therefore in each clock cycle one complex number is read from the input data buffer RAM and the complex result is written in the output buffer RAM. The 8- and 16-point DFT algorithm is divided into several stages which are implemented in the stages of the FFT8 and FFT16 pipelines. This supports the increasing the clock frequency up to 200 MHz and higher. The latent delay of the FFT8 unit from input of the first data to output of the first result is equal to 30 clock cycles. The latent delay of the FFT16 unit from input of the first data to output of the first result is equal to 30 clock cycles.



High precision computations

In the core the inner data bit width is higher to 4 digits than the input data bit width. The main error source is the result truncation after multiplication to the factors W_{64}^{ms}. Because the most of base FFT operation calculations are additions, they are calculated without errors. The FFT results have the data bit width which is higher in 3 digits than the input data bit width, which provides the high data range of results when the input data is the sinusoidal signal. The maximum result error is less than the 1 least significant bit of the input data.

Besides, the normalizing shifters are attached to the outputs of FFT8 pipelines, which provide the proper bandwidth of the resulting data. The overflow detector outputs provide the opportunity to input the proper shift left bit number for these shifters.

Low hardware volume

The FFT128 processor has the minimum multiplier number which is equal to 4. This fact makes this core attractive to implement in ASIC. When configuring in Xilinx FPGA, these multipliers are implemented in 4 DSP48 units respectively. The customer can select the input data, output data, and coefficient widths which provide application dynamic range needs. This can minimize both logic hardware and memory volume.

Interface:

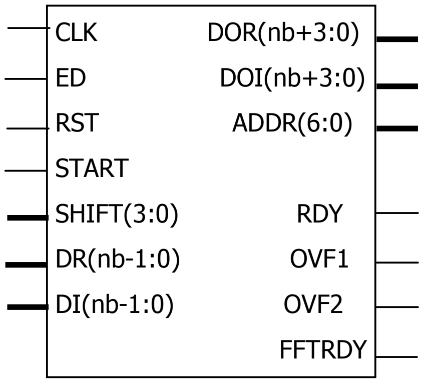


Figure 2. FFT 128 Symbol

Signal Description:

The descriptions of the core signals are represented in the table 1.

SIGNAL	TYPE	DESCRIPTION
CLK	input	Global clock
RST	input	Global reset
START	input	FFT start
ED	input	Input data and operation enable strobe
DR [nb-1:0]	input	Input data real sample
DI [nb-1:0]	input	Input data imaginary sample
SHIFT	input	Shift left code
RDY	output	Result ready strobe
WERES	output	Result write enable strobe
FFTRDY	output	Input data accepting ready strobe
ADDR [6:0]	output	Result number or address
DOR [nb+3:0]	output	Output data real sample
DOI [nb+3:0]	output	Output data imaginary sample
OVF1	output	Overflow flag
OVF2	output	Overflow flag

Table 1. IP core signals

Data representation

Input and output data are represented by *nb* and *nb*+4 bit twos complement complex integers, respectively. The twiddle coefficients are *nw* –bit wide numbers. $nb \le 16$, $nw \le 16$

Typical Core Interconnection

The core interconnection depends on the application nature where it is used. The simple core interconnection considers the calculation of the unlimited data stream which are inputted in each clock cycle. This interconnection is shown on the figure 3.

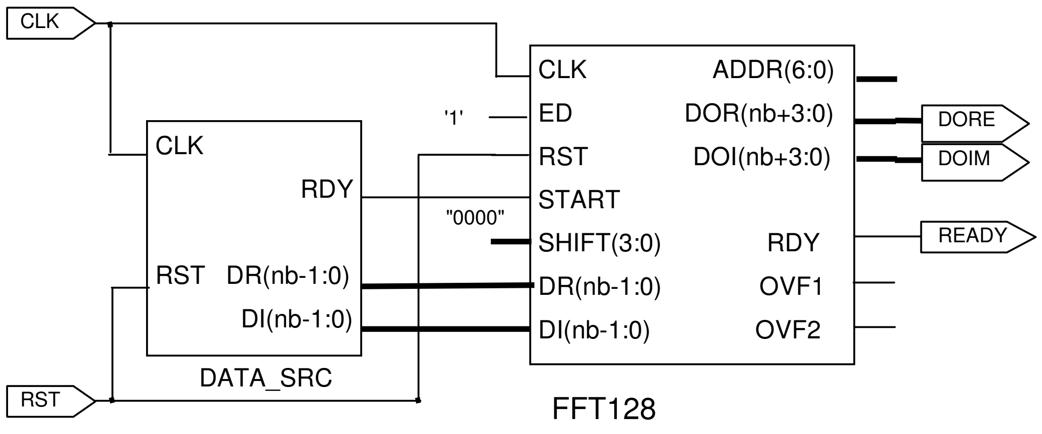


Fig. 3. Simple core interconnection

Here DATA_SRC is the data source, for example, the analog-to-digital converter, FFT128 is the core, which is customized as one with 3 inner data buffers. The FFT algorithm starts with the impulse START. The respective results are outputted after the READY impulse and followed by the address code ADDR. The signal START is needed for the global synchronization, and can be generated once before the system operation.

The input data have the natural order, and can be numbered from 0 to 63. When 3 inner data buffers are configured then the output data have the natural order. When 2 inner data buffers are configured then the output data have the 8-th inverse order, i.e. the order is 0,8,16,...56,1,9,17,... .

Input and Output Waveforms

The input data array starts to be inputted after the falling edge of the START signal. When the enable signal ED is active high then the data samples are latched by the each rising edge of the clock signal CLK. When all the 128 data are inputted and the START signal is low then the data samples of the next input array start to be inputted (see the Fig.4).

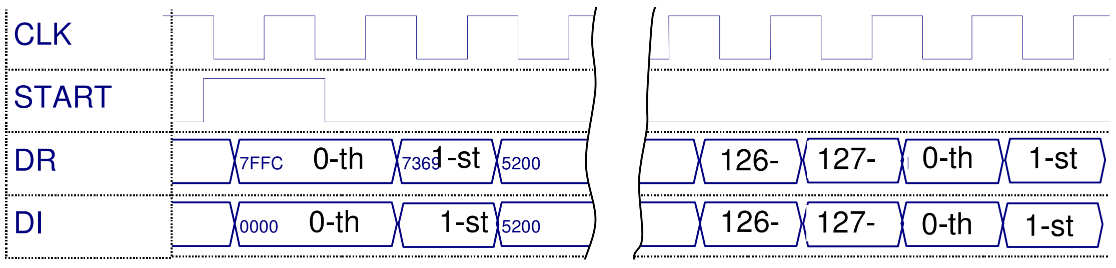


Figure *4*. Waveforms of data input

When ED signal is controlled then the throughput of the processor is slowed down. In the Fig.5 the input waveforms are shown when the ED signal is the alternating signal with the frequency which is in 2 times less than one of the clock signal. The input data are sampled when ED is high and the rising clock signal.

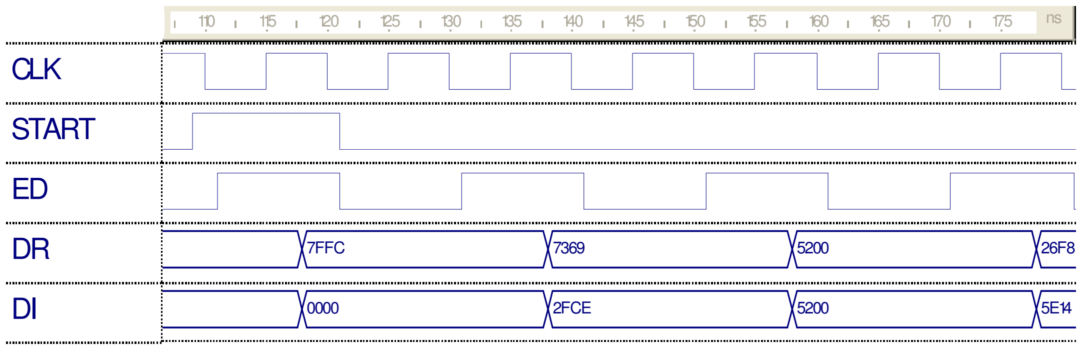


Fig. 5. Waveforms of data input when the throughput is slowed down in 2 times

The result samples start to be outputted after the RDY signal. They are followed by the result number which is given by the signal ADDR (see Fig.6). When the START signal is not active for the long time period then just after output of the 127-th couple of results the 0-th couple of results for the next data array is outputted.

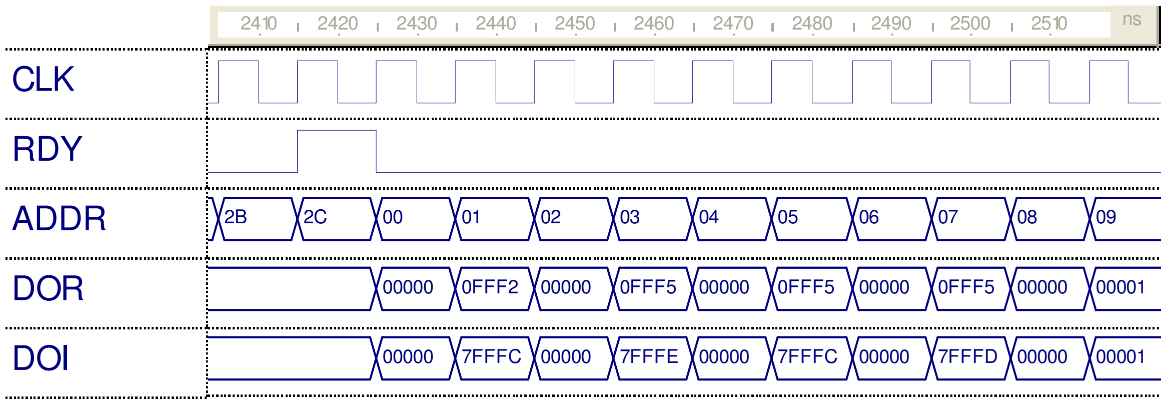


Figure 6. Waveforms of data output

The latent delay of the FFT128 processor core is estimated by ED = 1 as the delay between impulses START and RDY, and it is equal to 839 clock cycles when 3 buffer units are used and to 580 clock cycles when 2 buffer units are instantiated.

When the throughput is slowed down by the signal ED controlling then the result output is slowed down respectively, as it is shown in Fig.7.

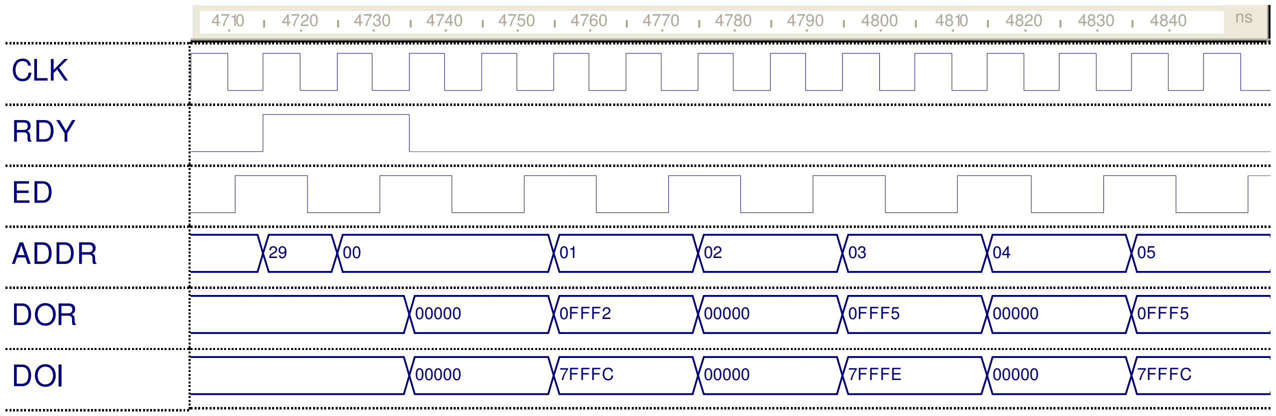


Figure 7. Waveforms of data output when the throughput is slowed down in 2 times

Block Diagram

The basic block diagram of the FFT128 core with two data buffers is shown in the fig.8.

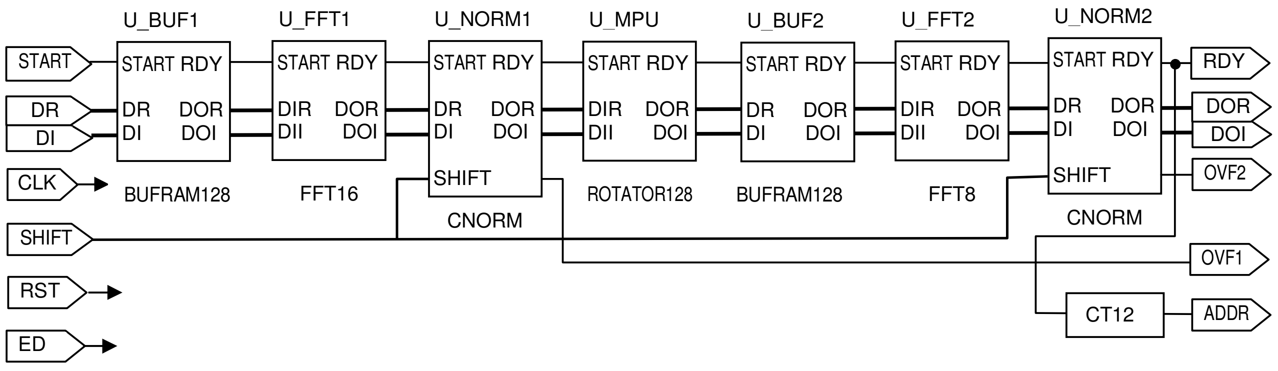


Fig.8. Block diagram of the FFT128 core with two data buffers

Components:

BUFRAM128 – data buffer with row writing and column reading, described in BUFRAM128C.v, RAM2x128C.v, RAM128.v;
FFT8 – datapath, which calculates the 8-point DFT, described in FFT8.v, MPUC707.v;
FFT16 – datapath, which calculates the 16-point DFT, described in FFT16.v, MPUC707.v, MPUC383.v, MPUC1307.v, MPUC541.v;
CNORM – shifter to 0,1,2,3 bit left shift, described in CNORM.v;
ROTATOR128 – complex multiplier with twiddle factor ROM, described in ROTATOR128.v, WROM128.v;
CT128 – counter modulo 128.
Below all the components are described more precisely.

BUFRAM128

BUFRAM128 is the data buffer, which consists of the two port synchronous RAM of the volume 512 complex data, and the write-read address counter. The real and imaginary parts of the data are stored in the natural ascending order as in the diagram in the Fig. 9. By the START impulse the address counter is reset and then starts to count (signal addrw). The input data DR and DI are stored to the respective address place by the rising edge of the clock signal.

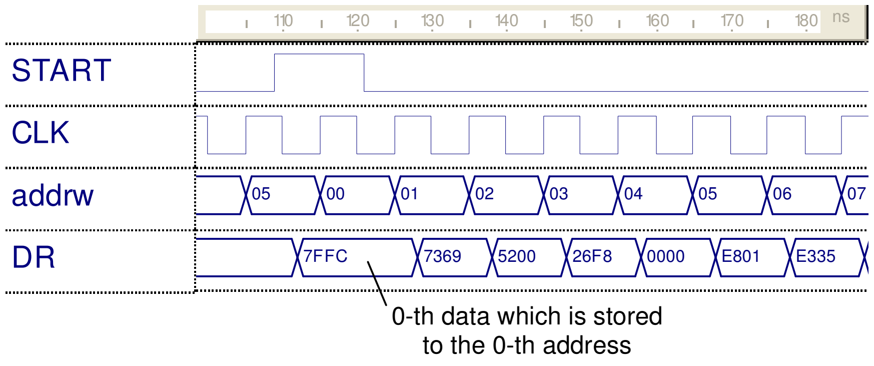


Fig.9. Waveforms of data writing to BUFRAM128

After writing 128 data beginning at the START signal, the unit outputs the ready signal RDY and starts to write the next 128 data to the second half of the memory. At this period of time it outputs the data stored in the first half of the memory. When this data reading is finished then the reading of the next array is starting. This process is continued until the next START signal or RST signal are entered. The reading address sequence is 8-6-th inverse order, i.e. the order is 0,16,32,...240,1,17,33,... . Really the reading address is derived from the writing address by swapping 4 LSB and 4 MSB address bits. The reading waveforms are illustrated by the Fig.10.

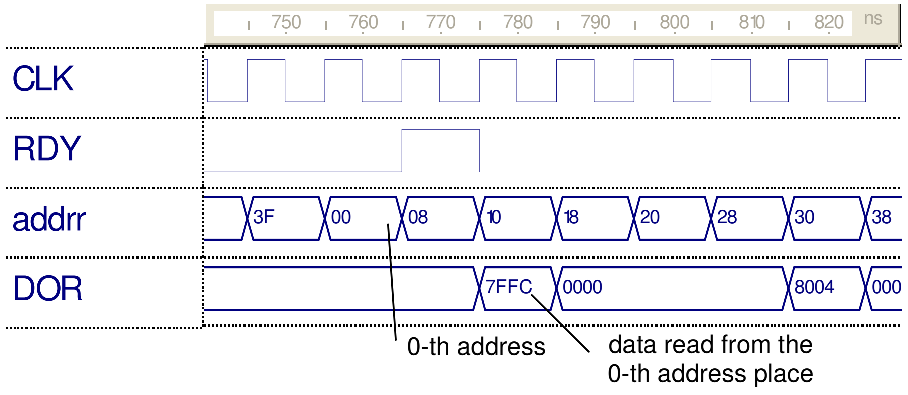


Fig.10. Waveforms of data reading from BUFRAM128

BUFRAM128 unit can be implemented in 2 ways. The first way consists in use of the usual one-port synchronous RAMs. Then BUFRAM128 consists of 2 parts, firstly one data array is stored into one part of the buffer, and another data array is read from the second part of the buffer, Then these parts are substituted by each other. Such a BUFRAM128 is implemented by use of files BUFRAM128C.v – root model of the buffer, RAM2x128C.v - dual ported synchronous RAM, and RAM128.v -single ported synchronous RAM model. This kind of the buffer is implemented when the FFT128bufferports1 parameter is recommented in the FFT128_config.inc file.

The second way consists in use of the usual 2-port synchronous RAM with a single clock input. Such a RAM is usually instantiated as the BlockRAM or the dual ported Distributed RAM in the Xilinx FPGAs. In this situation the FFT128bufferports1 parameter is commented or excluded in the FFT128_config.inc file. Then the file RAM128.v, which describes the simple model of the registered synchronous RAM, is not used. By configuring in Xilinx FPGAs 2 or 3 BlockRAMs are instantiated depending on the parameter FFT128parambuffers3.

FFT16

The datapath FFT16 implements the 16-point FFT algorithm in the pipelined mode. 16 input complex data are calculated for 46 clock cycles, but each new 16 complex results are outputted each 16 clock cycles.

The FFT algorithm of this transform is selected from the book "H.J.Nussbaumer. FFT and convolution algorithms". Due to this algorithm the calculations are:

$t1:=x(0) + x(8);$	$m4:=x(0) - x(8);$
$t2:=x(4) + x(12);$	$m12:= -j*(x(4)-x(12));$
$t3:=x(2) + x(10);$	$t4:=x(2) - x(10);$
$t5:=x(6) + x(14);$	$t6:=x(6) - x(14);$
$t7:=x(1) + x(9);$	$t8:=x(1) - x(9);$
$t9:=x(3) + x(11);$	$t10:=x(3) - x(11);$
$t11:=x(5) + x(13);$	$t12:=x(5) - x(13);$
$t13:=x(7) + x(15);$	$t14:=x(7) - x(15);$
$t15:=t1 + t2;$	$m3:= t1 - t2;$
$t16:=t3 + t5;$	$m11:= -j*(t3 - t5);$
$t17:=t15 + t16;$	$m2:= t15 - t16;$
$t18:=t7 + t11;$	$t19:= t7 - t11;$
$t20:=t9 + t13;$	$t21:= t9 - t13;$
$t22:=t18 + t20;$	$m10:= -j*(t18 - t20);$
$t23:=t8 + t14;$	$t24:= t8 - t14;$
$t25:=t12 + t10;$	$t26:= t12 - t10;$
$m0:=t17 + t22;$	$m1:=t17 - t22;$
$m13:= -j* \sin(\pi/4)*(t19 + t21);$	$m5:= \cos(\pi/4)*(t19 - t21);$
$m6:= \cos(\pi/4)*(t4 - t6);$	$m14:=-j* \sin(\pi/4)*(t4 + t6);$
$m7:= \cos(3\pi/8)*(m24+m26);$	$m15:= -j* \sin(3\pi/8)*(t23 + t25);$
$m8:= (\cos(\pi/8) + \cos(3\pi/8))*t24;$	$m16:= -j* (\sin(\pi/8) - \sin(3\pi/8))*t23;$
$m9:=- (\cos(\pi/8) - \cos(3\pi/8))*t26;$	$m17:= -j*(\sin(\pi/8) + \sin(3\pi/8))*t25;$
$s7:= m8 - m7;$	$s15:= m15 - m16;$
$s8:= m9 - m7;$	$s16:= m15 - m17;$
$s1:=m3 + m5;$	$s2:=m3 - m5;$
$s3:=m13 + m11;$	$s4:=m13 - m11;$
$s5:=m4 + m6;$	$s6:=m4 - m6;$
$s9:=s5 + s7;$	$s10:=s5 - s7;$
$s11:=s6 + s8;$	$s12:=s6 - s8;$
$s13:=m12 + m14;$	$s14:=m12 - m14;$
$s17:=s13 + s15;$	$s18:=s13 - s15;$

$s19{:=}s14 + s16;$	$s20{:=}s14 - s16;$
$y(0){:=}m0;$	$y(8){:=}m1;$
$y(1){:=}s9 + s17;$	$y(15){:=}s9 - s17;$
$y(2){:=}s1 + s3;$	$y(14){:=}s1 - s3;$
$y(3){:=}s12 - s20;$	$y(13){:=}s12 + s20;$
$y(4){:=}m2 + m10;$	$y(12){:=}m2 - m10;$
$y(5){:=}s11 + s19;$	$y(11){:=}s11 - s19;$
$y(6){:=}s2 + s4;$	$y(10){:=}s2 - s4;$
$y(7){:=}s10 - s18;$	$y(9){:=}s10 + s18;$

where x and y are input and output arrays of the complex data, $t1,\ldots,t26$, $m1,\ldots, m17$, $s1,\ldots,s20$ are the intermediate complex results, $j = \sqrt{(-1)}$. As we see the algorithm contains only 20 real multiplications to the untrivial coefficients
$\sin(\pi/4) = 0.7071$; $\sin(3\pi/8) = 0.9239$; $\cos(3\pi/8) = 0.3827$;
$(\cos(\pi/8) + \cos(3\pi/8)) = 1.3066$; $(\sin(\pi/8) - \sin(3\pi/8)) = 0.5412$;
and 156 real additions and subtractions.

The datapath is described in the files FFT16.v, MPUC707.v, MPUC924_383.v, MPUC1307.v, MPUC541.v widely using the resource sharing, and pipelining techniques. The counter ct counts the working clock cycles from 0 to 15. So a single inferred adder adds $x(0) + x(8)$ in one cycle, $x(1) + x(9)$ in the next cycle, $D(1) + D(5)$ in another cycle and so on, and $x(7) + x(15)$ in the final cycle of the sequence of cycles deriving the results $t1, t7, t9,\ldots, t13$ respectively.

Four constant multipliers are used to derive the multiplication to 5 different coefficients. So the unit in MPUC707.v implements the multiplication to the coefficient 0.7071 in the pipelined manner. Note that the unit MPUC924_383.v implements the multiplication both to 0.9239 and to 0.3827. The multipliers use the adder tree, which adds the multiplicand shifted to different bit numbers. For example, for short input bit width the coefficient 0.7071 is approximated as 0.10110101_2, for long input bit width it is approximated as 0.10110101000000101_2. The long coefficient bit width is set by the parameter FFT128bitwidth_coef_high. The first kind of the constant multiplier occupies 3 adders, and the second one occupies 4 adders.

The importance of the long coefficient selection is seen from the following fact. When the input bit width is 16 and higher, the selection of the long coefficient bit width decreases the FFT128 result error in two times.

The FFT16 unit implements both FFT and inverse FFT depending on the parameter FFT128paramifft. Practically the inverse FFT is implemented on the base of the direct FFT by the inversion of operations in the final stage of computations for all the results except $y(0)$, $y(8)$. For example, $y(1){:=}s9 + s17;$ is substituted to $y(1){:=}s9 - s17;$

The FFT16 unit starts its operation by the START impulse. The first result is preceded by the RDY impulse which is delayed from the START impulse to 30 clock impulses. The output results have the bit width which is in 4 higher than the input data bit width. That means that all the calculations except multiplication by coefficients like 0.7071 are implemented without truncations, and therefore, the FFT128 results have the minimized errors comparing to other FFT processors.

FFT8

The datapath FFT8 implements the 8-point FFT algorithm in the pipelined mode. 8 input complex data are calculated for 22 clock cycles, but each new 8 complex results are outputted each 8 clock cycles.
The FFT algorithm of this transform is selected from the book "H.J.Nussbaumer. FFT and convolution algorithms". Due to this algorithm the calculations are:

$t1=D(0) + D(4);$ $\quad$ $m3=D(0) - D(4);$
$t2=D(6) + D(2);$ $\quad$ $m6=j*(D(6)-D(2));$
$t3=D(1) + D(5);$ $\quad$ $t4=D(1) - D(5);$
$t5=D(3) + D(7);$ $\quad$ $t6=D(3) - D(7);$
$t8=t5 + t3;$ $\quad$ $m5=j*(t5-t3);$
$t7=t1 + t2;$ $\quad$ $m2=t1 - t2;$
$m0=t7 + t8;$ $\quad$ $m1=t7 - t8;$
$m4=\sin(\pi/4)*(t4 - t6);$ $\quad$ $m7=-j* \sin(\pi/4)*(t4 + t6);$
$s1=m3 + m4;$ $\quad$ $s2=m3 - m4;$
$s3=m6 + m7;$ $\quad$ $s4=m6 - m7;$
$DO(0)=m0;$ $\quad$ $DO(4)=m1;$
$DO(1)=s1 + s3;$ $\quad$ $DO(7)=s1 - s3;$
$DO(2)=m2 + m5;$ $\quad$ $DO(6)=m2 - m5;$
$DO(5)=s2 + s4;$ $\quad$ $DO(3)=s2 - s4;$

where D and DO are input and output arrays of the complex data, $j = \sqrt{(-1)}$, $t1,\ldots,t8$, $m1,\ldots, m7$, $s1,\ldots,s4$ are the intermediate complex results. As we see the algorithm contains only 4 multiplications to the untrivial coefficient $\sin(\pi/4) = 0.7071$, and 26*2 real additions and subtractions. The multiplication to a coefficient *j* means the negation the imaginary part and swapping real and imaginary parts.

The datapath is described in the files FFT8.v, MPU707.v widely using the resource sharing technique.
The FFT8 unit starts its operation by the START impulse. The first result is preceded by the RDY impulse which is delayed from the START impulse to 17 clock impulses.

CNORM

During computations in FFT8 and FFT16 the data magnitude increases up to 8 and 16 times, respectively, and the FFT128 result can increase up to 128 times depending on the spectrum properties of the input signal. Therefore, to prevent the signal dynamic bandwidth loose, the output signal bit width must be at least in 8 bits higher than the input signal bit width. To prevent this bit width increase, to provide the proper signal dynamic bandwidth, and to ease the next computation of the derived spectrum, the CNORM units are attached to the outputs of the FFT16 units.

CNORM unit provides the data shift left to 0,1,2, and 3 bits depending on the code SHIFT. The input data width is nb+3 and the output data width is nb+2, where nb is the given processor input bit width.
The overflow occurs in CNORM unit when the SHIFT code is given too high. The SHIFT code must be set by the customer to prevent the data overflow and to provide the proper dynamic bandwidth. The CNORM unit contains the overflow detector with the output OVF. When FFT128 core in operation, a 1 at the output OVF signals that for some input data an overflow occurred. OVF flag is resetted by the RST or START signal.

The SHIFT inputs of two CNORM stages are concatenated to the 4-bit input SHIFT of the FFT128 core, 2 LSB bits control the first stage, and 2 MSB bits do the second stage.
The selection of the proper SHIFT code depends on the spectrum property of the input signal. When the input signal is the sinusoidal one or contains a few of sinusoids, and the noise level is small then SHIFT =0000, or 0001, or 0010. When the input signal is a noisy signal then SHIFT can be 1100 and higher.
When the input signal has the stable statistic properties then the code SHIFT can be set as a constant. Then the OVF outputs can be not in use, and the CNORM units will be removed from the project by the hardware optimization when the core is synthesized.

ROTATOR128

The unit ROTATOR implements the complex vector rotating to the angles W_{128}^{ms}. The complex twiddle factors are stored in the unit WROM128. Here the ROM contains the following table of coefficients

(w0, w0, w0, w0, w0, w0, w0, w0, w0, w0, w0, w0, w0, w0, w0, w0,
w0, w1, w2, w3, w4, w5, w6, w7, w8, w9, w10, w11, w12, w13, w14, w15,
w0, w3, w6, w9, w12,w15,w18,w21, w24, w27, w30, w33, w36, w39, w42, w45,
…
w0,w7,w15,w23,w31,w39,w47,w55,w63,w71,w79,w97,w103,w111,w119,w127),

where $wi = W_{128}^{i}$. Here the row and column indexes are *m* and *s* respectively. These coefficients are read in the natural order addressing by the 7-bit counter addrw. The complex vector rotating is implemented by the usual schema of the complex number multiplier which contains 4 multiply units and 2 adders.

Testbench

Block Diagram

The block diagram of the testbench is shown in the Fig. 11.

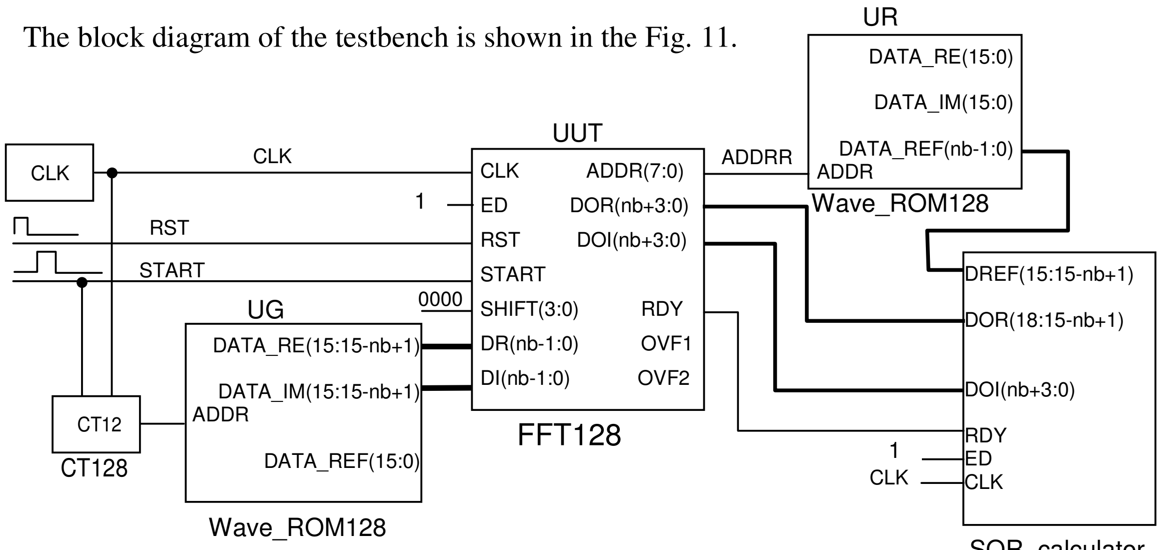


Fig.11. Testbench structure

The units UG and UR are implemented as ROMs which contain the generating waveforms (UG) and the reference waveform (UR). They are instantiated as a component Wave_ROM128 which is described in the file Wave_ROM128.v. This file can be generated by the PERL script sinerom128_gen.pl. In this script the tables of sums of up to 4 sine and cosine waves are generated which frequencies are set by the parameters $f1, $f2, $f3, and $f4. The table of the respective frequency bins is generated too. The table length is set as $n = 128. The samples of these tables are outputted to the outputs DATA_IM, DATA_RE, and DARA_REF of the component Wave_ROM128, respectively.

The counter process CT128 generates the address sequence to the UG unit starting after the START impulse. The UG unit outputs the testing complex signal to the UUT unit (FFT128) with the period of 128 clock cycles.

When the FFT result is ready then UUT generates the RDY signal after that it generates the address sequence ADDR of the results. This sequence is the input data for the UR unit which outputs the correct real samples (bins) of the spectrum. Note that because the input data is the complex sine wave sum then the imaginary part of the spectrum must be a sequence of zeros. The process SQR_calculator calculates the sum of square differences between spectrum results and reference samples. It starts after the impulse RDY and finishes after 128 clock cycles. Then the result is divided to 128 and outputted in the message in the console of the simulator. For example, the message: "rms error is 1 lsb" means that the square of the residue mean square error is equal to 1 LSB of the spectrum result.

When the model FFT128 is correct and its bit widths are selected correctly then the rms error not succeeded 1 or 2. When this model is not correct then the message will be a huge positive or negative integer, or 'X'.

The model correctness can be proven or investigated by looking at the input and output waveforms. Fig.12 illustrates the waveforms of the input signals, and Fig.13 shows the output waveforms. Not that the scale of the waveform DOI is in thousand times higher than one of the waveform DOR

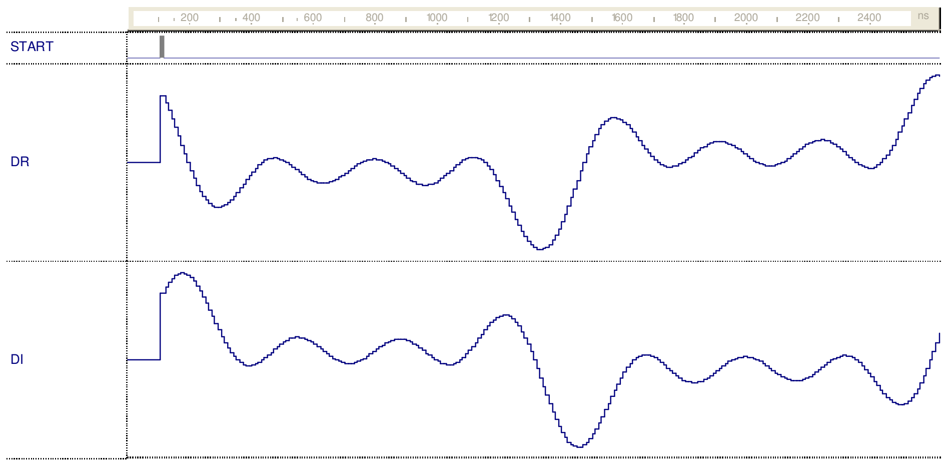


Fig.12. Input waveforms

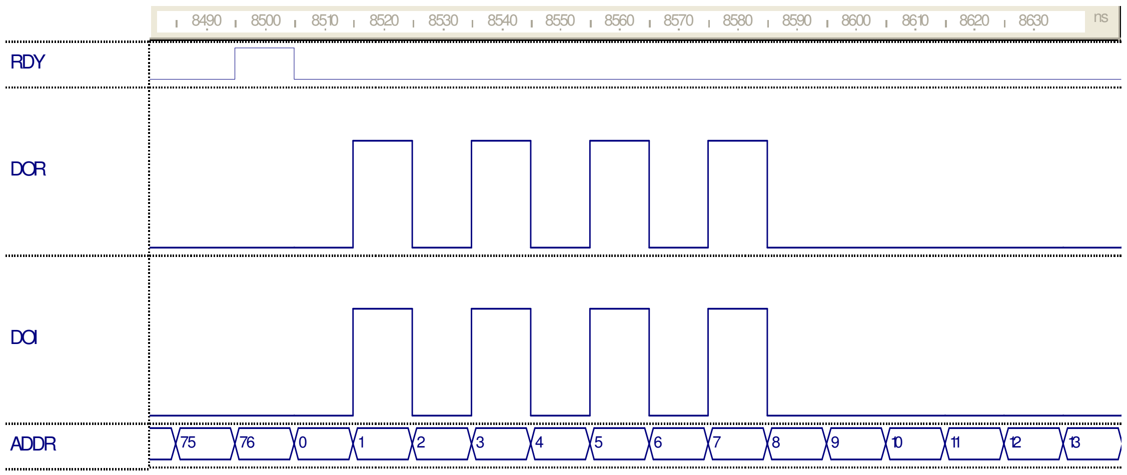


Fig.13. Output waveforms

Implementation Data

The following table 2 illustrates the performance of the FFT128 core with two data buffers based on BlockRAMs in Xilinx Virtex device when implementing 128-point FFT for 10 and 16-bit data and coefficients. Note that 4 DSP48 units in all projects are used. The results are derived using the Xilinx ISE 9.1 tool.

Target device and its data bit width	**XC 4VSX35-12**		**XC 5VLX30-3**	
	10	16	10	16
Area, Slices	4147 (19%),	4573 (29%),	1254 (26%),	1746 (35%)
RAMBs	3	4	3	3
Maximum system clock	215 MHz	203 MHz	263 MHz	210 MHz

Table 2. Implementation Data – Xilinx Virtex FPGA

Deliverables:

Deliverables are the following files:

FFT128_2B.v - root unit,
    FFT128_CONFIG.inc - core configuration file
        BUFRAM128C.v - 1-st data buffer,
        BUFRAM128C_1.v - 2-nd data buffer,
        BUFRAM128C_2.v - 3-d data buffer, contains:
        RAM2x128C_1.v - dual ported synchronous RAM, contains:
        RAM128.v -single ported synchronous RAM
    FFT8.v - 1-st stage implementing 8-point FFTs, contains
        MPUC707.v - multiplier to the factor 0.7071;
    FFT16.v - 2-nd stage implementing 16-point FFTs, contains
        MPUC707.v - multiplier to the factor 0.7071,
MPUC924_383.v - multiplier to the factors 0.924, 0.383,
MPUC1307.v - multiplier to the factor 1.307,
MPUC541.v - multiplier to the factor 0.541;
    ROTATOR128.v - unit for rotating complex vectors, contains
        WROM128.v - ROM of twiddle factors.
    CNORM.v, CNORM_1.v, CNORM_2.v, - 1,2,and 3-d normalization stages
    UNFFT128_TB.v - testbench file, includes:
        Wave_ROM128.v - ROM with input data and result reference data
        SineROM128_gen.pl - PERL script to generate the Wave_ROM128.v file