Chapter 3

FPGA based Digital Signal Processing / Implementation of 2D-FFT algorithm for OFDMA

3.1 INTRODUCTION

When implementing a digital system like digital filter the one can face two choices for implementation: choosing to use a dedicated Digital Signal Processing (DSP) processor or choosing hardware based approach like Programmable Logic Devices (PLD's) or Field Programmable Gate Array (FPGA) [48] [49]. This chapter attempts to implement the FFT/2D-FFT algorithm in programmable logic devices such as FPGAs. Implementing digital filters in hardware can have various advantages over a dedicated DSP processor approach. As IC processing advances and strives to meet Moore's Law, logic design and programming has improved in both cost and speed, making FPGA a feasible alternative for implementing digital filters [50].

Not only logic of the devices has improved but with the advent of Hardware Description Languages (HDL), the method of designing with programmable logic has become more efficient. HDLs deal with an alternative to the schematic based approach to logic design, in which the logic gates, flip-flop's, registers or counters are connected in schematic fashion. HDLs are coding based or text based design entry method in which the design will be coded with respect to the hardware. This allows the designer to have flexibility and describe the logic function at a higher level of abstraction thereby increasing efficiency. The two primary HDLs used today are Verilog HDL and Very High Speed IC HDL (VHDL), both are IEEE standards. Both of these HDLs (Verilog /VHDL) are separate languages and can be used to implement digital filters.

This chapter discusses how to implement FFT/2D-FFT in programmable logic using VHDL. In the section 3.2 advantages of carrying out Digital signal processing on FPGA's are discussed and followed by basic arithmetic building blocks and radix-2 encoding formats are studied in section 3.3 and 3.4. The section 3.5 describes the implementation of 8-point FFT in VHDL, followed by the Simulation and synthesis results of the same are discussed. Section 3.6 describes implementation of 2D-FFT algorithm, including the concept of DDS-

Core and Ping-Pong Memory architecture used to design the PHY- layer of OFDMA and the last section summarizes the chapter.

3.2 ADVANTAGES OF CARRYING OUT DSP ON PLD'S (FPGA)

Since PLD's are dedicated hardware, they can achieve significantly improved performance over a DSP processor. Other advantages include reduced power consumption. Although dedicated DSP processors offer the most flexibility but they require extra clock cycles compared to a hardware implementation and this can be power inefficient. Since many embedded systems already have some type of programmable logic on the system, the PLD may have the space available for a digital filter. If this extra logic space is not available, the designer may be faced with porting all the firmware over to a dedicated DSP processor. A better alternative might be to increase the size of the programmable logic device to accommodate a digital filter. If the PLD is already in the data path of the embedded system, the latter approach may be easier than initiating a new hardware design with a dedicated DSP processor.

Many embedded systems have both a dedicated DSP processor with a PLD device. In over sampled DSP systems, where the data arrives at a high rate, a PLD can down sample the data prior to the DSP processor. When decimation is required, the PLD can off-load some of these processing tasks by performing both filtering and decimation before data is transferred to the DSP for further processing at a lower sample rate.

3.3 BASIC ARITHMETIC BLOCKS FOR DSP

The three basic building blocks of any digital filter are unit delay, the summing node and the multiplication node (multiplier) which is shown in the Fig.3.1. The combination of a multiplier and summing node is known as MAC block which stands for multiply and accumulate (MAC) block. The unit delay is also called as filter tap and is implemented by recalling an old value from memory, storing the new value into memory and waiting for T seconds, where T

is the sampling period. The Z transform of the unit delay is Z^{-1} which can also be expressed as

$$e^{-j2\pi fT} = \cos(2\pi fT) - j\sin(2\pi fT) \qquad (3.1)$$

where $j^2 = -1$ (Euler's identity).

Simple addition operation is performed by the summing node. The multiplier can be used to multiply a signal by a coefficient or to multiply two signals. These blocks are represented graphically with signal flow diagrams (SFD) or signal flow graphs (SFG) in Fig.3.1.

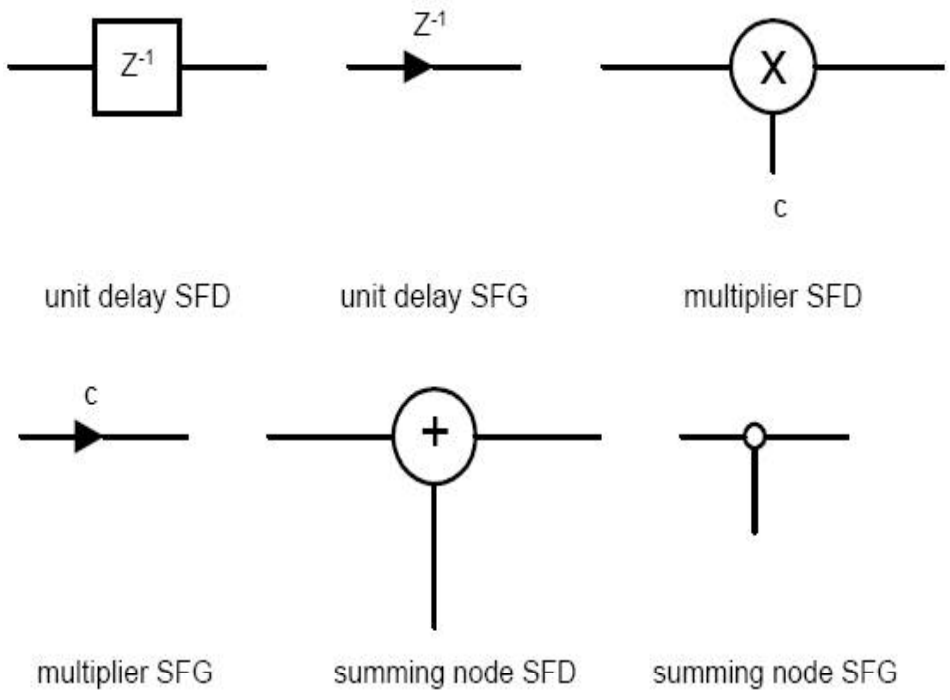

Fig.3.1. Symbols used in Digital filter structures.

3.4 BINARY (RADIX-2) ENCODING FORMAT

The FIR filter specifications contain the filter coefficients to be stored in the filter and are also used to carry out the delay, addition and multiplication operations. These coefficients are signed floating point numbers and such numbers must be converted into another form which can be easily stored and operated in an FPGA. There are so many possible ways to encode numbers using binary, a summary of several possible formats that could be used to represent the above coefficients is described below.

Unsigned numbers are easy to encode and Unsigned Radix-2 format is the simplest possible encoding format. All the positive integers are allowed in the range 0 to 2^n-1 to be encoded. In this Unsigned Radix-2 format each bit of the binary number has a weight associated with it, the same has been shown as an example in Table 3.1.

Table 3.1 Unsigned Radix-2 encoding format

Weight:	128	64	32	16	8	4	2	1
Number:	1	0	1	1	0	0	1	0

128 + 32 + 16 + 2 = 178

3.4.1 Sign Magnitude:

Sign magnitude notation allows representing the negative numbers. These are encoded by allocating one bit to specify the sign of the number. MSB bit is used to indicate the negative bit. If the MSB bit is '1' then the number is indicated as negative number, if the MSB bit is '0' then it is positive number. The remaining bits are then encoded by the radix-2 encoding method as shown in the Fig.3.2. This allows the system to store numbers in the range $-(2^n-1)$ to 2^n-1. Example:

Table.3.2 Sign Magnitude encoding's

Weight:	Sign	64	32	16	8	4	2	1
Number:	1	0	1	1	0	0	1	0

(32 + 16 + 2) = -50

Weight:	Sign	64	32	16	8	4	2	1
Number:	0	0	1	1	0	0	1	0

(32 + 16 + 2) = +50

As there are two separate encodings for the number zero, that stands as a disadvantage. In the case of an 8-bit number these are 10000000 and 00000000.

3.4.2 1's Complement:

Compliment of a number plays important role in performing all the arithmetic operations in digital systems. 1's complement also allows to compliment the numbers in the range $-(2^n-1)$ to 2^n-1 to be encoded. To encode a 1's complement all the zeros of a binary number are replaced by one and ones by zeros i.e. bit by bit complement of the binary number is taken. For example:

Table 1.3: 1's complement encoding

1s Complement	Decimal Equivalent
01111111	127
10000000	-127
11111001	-6
00000110	6

3.4.3 2's Complement:

2's complement is the standard method of storing signed binary integers. Representation of numbers in the range $-(2^n)$ to 2^n-1 is allowed in 2's compliment, and has the major advantage of only having one encoding for 0. To perform 2's complement encoding 1's compliment+1 is done i.e. the bits of the binary number are complemented, and then 1 is added. For example:

Table.3.2: 2's complement encoding

2's Complement	Decimal Equivalent
10000000	127
10000001	-128
11111010	-6
00000110	6

3.4.4 Mantissa Exponent Encoding:

The Mantissa exponent representation is the standard format to store the floating point numbers. In this, a number is represented in scientific notation as

$$\text{Mantissa} \times \text{radix}^{\text{sign} \times \text{exponent}}$$

Usually the radix is fixed and only the mantissa, sign and exponent are encoded. up to 2^n numbers can be encoded using this method, obviously the accuracy is limited. The IEEE standard 754 defines a standard for binary floating-point arithmetic using either 32 or 64 bit numbers. The 32-bit format is shown in the Table 3.5.

Table 3.3: Mantissa Exponent encoding

Sign bit	Exponent (8 bits)	Mantissa (23 bits)

The range of numbers which can be stored using this standard is approximately 1.8×10^{-38} to 3.40×10^{38}. However, since the implementation of OFDMA i.e. large FFT calculations requires large number of complex additions and multiplications, floating point arithmetic is not feasible to work out on FPGA.

3.4.5 Chosen format for Present work:

Sign-magnitude representation of the coefficients has been chosen as the number representation to be used to store the twiddle factors and process the inputs points of the FFT. However, Unsigned Radix-2 cannot represent negative numbers and so is not suitable for the implementation. While 1's complement and 2's complement are both signed number representations, the Adders and Multipliers needed to implement operations with these numbers which are more complicated than those required for signed magnitude operations, and so would be harder to make faster on FPGA.

The other advantage of signed magnitude format is that the signed bit can be removed easily before multiplication and then multiplication can be implemented utilizing one less bit for each of the two arguments. This results in reduced logic and time. However, the sign must be computed and restored in the product after multiplication [51], [52], [53].

For a DSP implementation, it is necessary to represent numbers that have both a whole part as well as fractional part. Typically, we view the signal 'S' as a whole number and the coefficient 'C' as having a whole part and a fractional part. A commonly used representation of the coefficients is called the Q_n format. In Q_n format, the number 'n' represents the number of binary digits to the right of the binary point. In other words, it is the size in bits of the fractional part of the number. As an example, the 12-bit number 6.1 in Q_0 format is 0000 0000 0110. The same 12 bit number 6.1 in Q8 format is 0110 0001 1001, which actually equals 6.09765625 due to coefficient quantization. Notice that, for this representation, we simply multiplied $6.1 * 2^8 = 6.1 * 256$. In general, the largest numbers of fractional bits are used to represent the coefficients, while being

careful that the whole part of the number has enough bits to handle the expected range of the coefficients. For example, the 12-bit number 1.918 could be represented best in Q_{10} format. 1.918 * 2^{10} = 011110101100. In fixed-point arithmetic, one trades accuracy and range as shown in Table 3.6.

Table.3.6. Fixed-point Q_n Representation Examples

12bit Format	**Representation**	**Range**	**Approximate resolution**
Q0	Whole numbers	-2047 to 2047	1
Q1	Fractional Numbers	-1023.5 to 1023.5	0.5
Q10	Fractional Numbers	-1.999 to 1.999	0.001

In fixed-point arithmetic, care must be taken such that the numbers are in the same Q format prior to addition or subtraction. For DSP applications, we can think of multiplication as being a Q_0 number (the signal) times a Q_n (a coefficient) number, yielding a Q_0 result (scaled signal). For a 12-bit Q_n multiplication process, we first multiply the signal by the coefficient, yielding a 24 bit result. Only 12 bits of the product are returned. The 12 bits, which are returned, are the ones produced if the product was shifted right by 'n' bits.

As an example, let us take the Q_{10} coefficient 1.414 multiplied by a signal of 46. We expect a result of 65. The coefficient 1.414 in Q_{10} is 1448. 1448*46=66608 = 000000010000010000110000. Shifting right by 10, we have 000001000001 which is 65.

3.5 8-POINT FFT IMPLEMENTATION

The basic building blocks of FFT implementation are Adder, Multiplier, Butterfly structure and Twiddle factor multiplier. The basic formula for the FFT calculation, i.e. convert the input samples in to the frequency domain from time domain is given by

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{\frac{-j2\Pi kn}{N}} \quad k = 0,1,....N-1 \tag{3.2}$$

There are two types of Radix-2 FFT algorithms they are Decimation in Time (DIT)-FFT and Decimation in Frequency (DIF)-FFT. Both of transformations rely on the recursive decomposition of an N point transform into two (N/2) point transforms. This decomposition process can be applied to any composite (non-prime) N. The method is easier if N is divisible by 2 and if N is a regular power of 2, the decomposition can be applied repeatedly until the trivial '1 point' transform is reached. The radix-2 decimation-in-frequency FFT is an important algorithm obtained by the divide-and-conquer approach [54], [55]. The Fig. 3.2 reveals the first stage of the 8-point DIF algorithm.

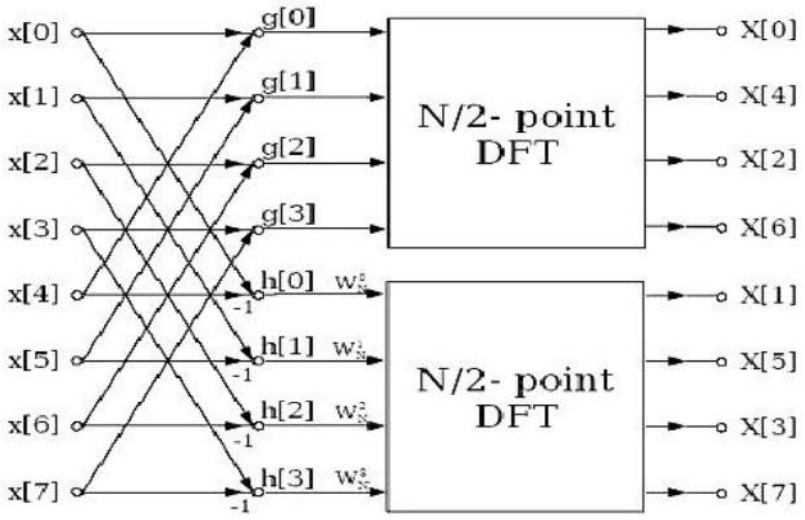

Fig.3.2. First stage of 8-point Decimation in Frequency FFT algorithm

The decimation, however, causes shuffling in data. The entire process involves v = $\log_2 N$ stages of decimation, where each stage involves N/2 butterflies of the type shown in the Fig. 3.3

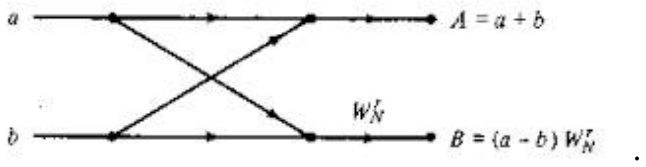

Fig.3.3 Butterfly structure of Decimation in Frequency FFT algorithm

Here a, b are the input points, A, B are the FFT output points in frequency domain, W_N is the twiddle factor and is given by equation 3.3. And few real and imaginary values of twiddle factor are shown in Table 3.7.

$$W_N^n = e^{\frac{-j2\Pi n}{N}} \tag{3.3}$$

Table 3.7. Twiddle factor Values

1	Imag (W_8^0) = 0	0000 0000 0000 0000
2	Real (W_8^1) = 0.707106781186	0000 0000 1011 0100
	Imag (W_8^1) = - 0.707106781186	1000 0000 1011 0100
3	Real (W_8^2) = 0	0000 0000 0000 0000
	Imag (W_8^2) = - 1	1000 0001 0000 0000
4	Real (W_8^3) = - 0.707106781186	1000 0000 1011 0100
	Imag (W_8^3) = - 0.707106781186	1000 0000 1011 0100
5	Real (W_8^4) = -1	1000 0001 0000 0000
	Imag (W_8^4) = 0	0000 0000 0000 0000

Consequently, the computation of N-point DFT via this algorithm requires (N/2) $\log_2 N$ complex multiplications. For illustrative purposes, the eight-point decimation-in frequency algorithm is shown in the Fig.3.4. We observe that the output sequence occurs in bit-reversed order with respect to the input. Furthermore, if we abandon the requirement that the computations occur in place, it is also possible to have both the input and output in normal order.

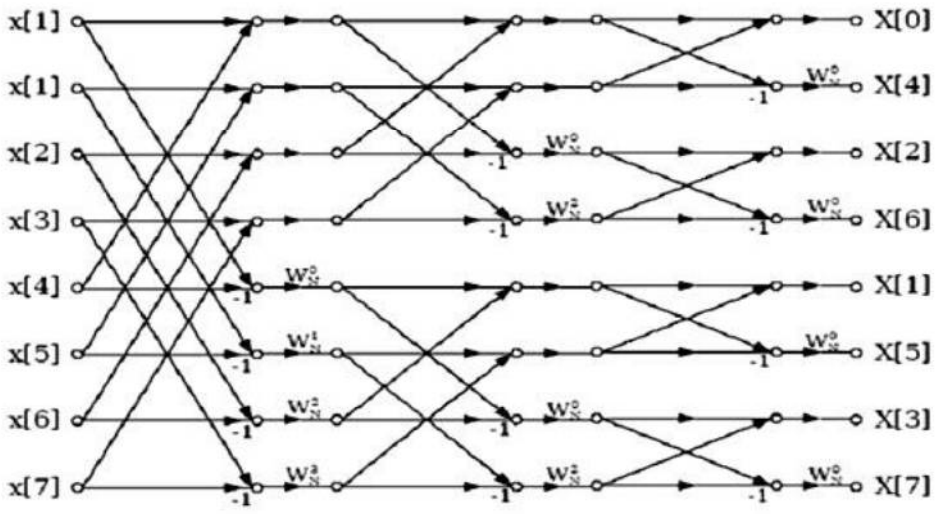

Fig.3.4 Structure of 8-point Decimation in Frequency FFT algorithm

3.5.1 Adder Implementation

The adder module in the FFT implementation is shown in the fig.3.5.The following adder module has been implemented by using signed magnitude data. In DSP, we know that the subtraction also takes place in the form of addition i.e. A-B = A + (-B). The same was simulated in Modelsim and the output wave is shown in the Fig.3.5.

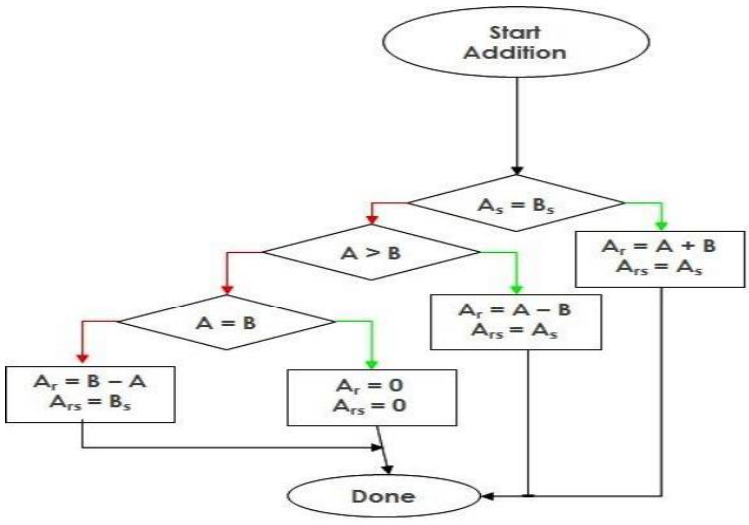

Fig 3.5.Flowchart of Addition and Subtraction with Signed-Magnitude Data

3.5.2 Multiplier Implementation

The sign bit is made positive if both are of same sign, and made negative if both are of opposite sign. Since the product of two Q_8 format numbers will be in Q_{16} format, while during truncation The 22nd bit to 8th bit(total 15 bits) of product are taken as truncated magnitude to make the result also in Q8 format[56].

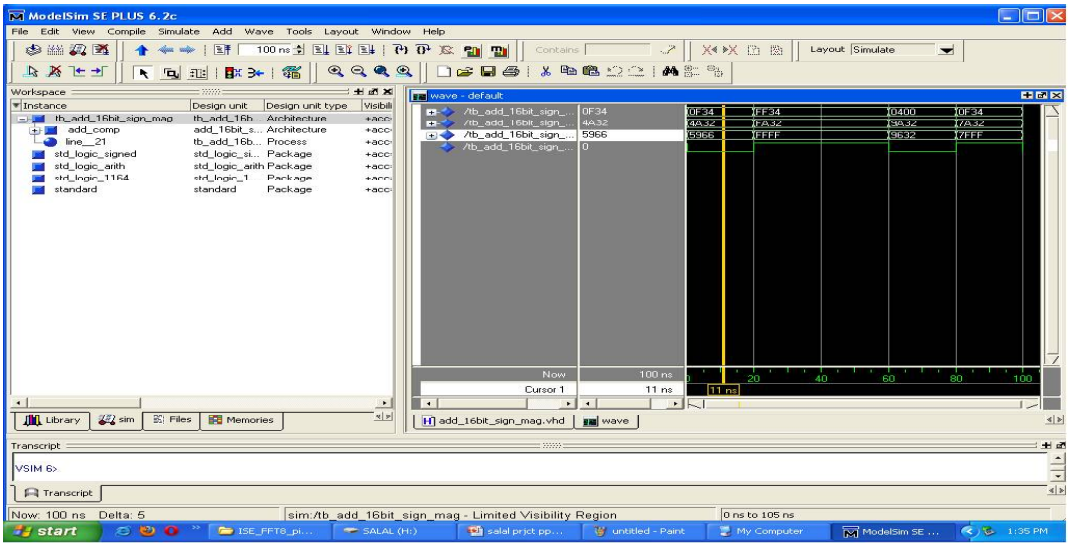

Fig.3.6. Simulation output of 16 bit Signed magnitude addition

In the same way Butterfly structure can be implemented by multiplying with the twiddle factor values where ever necessary in the algorithm. The overall simulation output of 8-point FFT combining the adder module, multiplier module, twiddle factor module and the butterfly structure is shown in the Fig. 3.7.

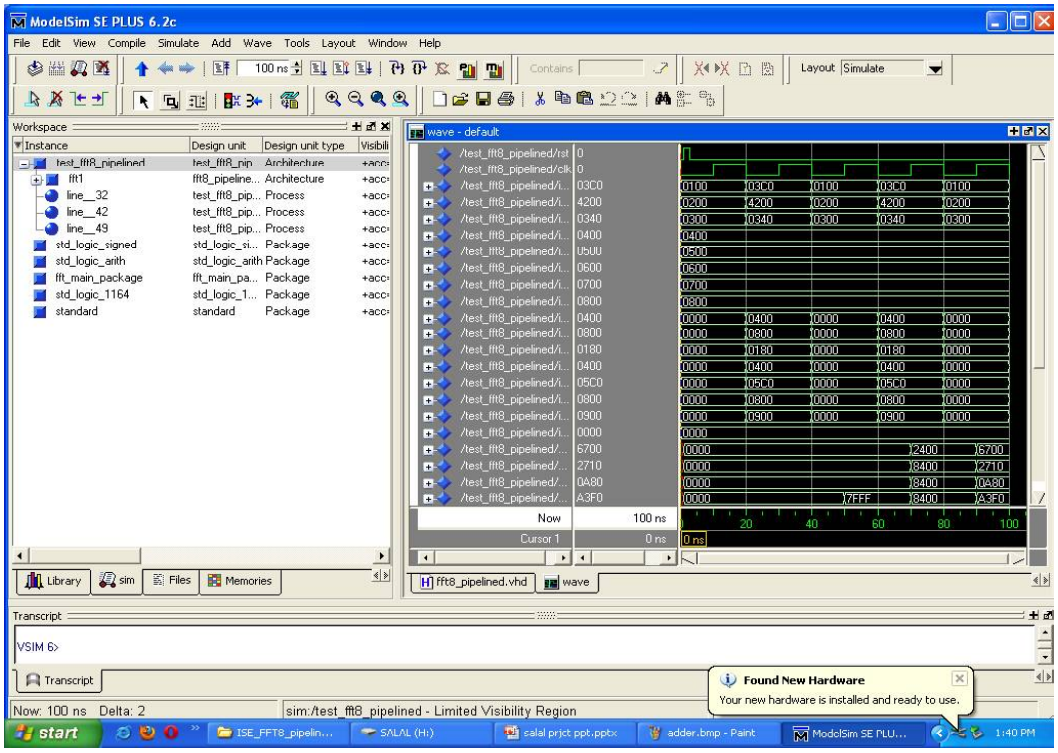

Fig.3.7.The Simulated output of 8-point FFT

3.6. 2D-FFT IMPLEMENTATION FOR OFDMA

2D-FFT algorithm is implemented by using two shorter length FFTs (lengths N1 and N2) to calculate an FFT of length N = NI xN2 [57]. The two-dimensional (2D) FFT of N= N_1X N_2 is defined as follows as shown in the equation 2.7.

$$X[k_1N_2+K_2]=\sum_{n_1=0}^{N_1-1}\left[e^{\frac{-j2\Pi n_1k_2}{N}}\left(\sum_{n_2=0}^{N_2-1}X(n_2N_1+n_1)e^{\frac{-j2\Pi n_2k_2}{N_2}}\right)\right]e^{\frac{-j2\Pi n_1k_1}{N_1}}$$

IEEE STD 802.16-2005 defined clearly: the core module of OFDMA physical layer is the FFT module, which can be used in the calculation of FFT points that are: 2048 points (back compatible with IEEE STD 802.16-2004), 1024 points, 512 points and 128 points.

The design about variable point FFT processor is just based on FFT module in OFDMA system application. According to the idea of two-dimensional Fourier algorithm, i.e. if N = 128, then N1 = 2, N2 = 64, From 128 = 2 * 64 ,We find that when one wants to achieve 128-point FFT, Firstly the data is arranged in 64 lines and 2 rows, Secondly the input data will transform the 64 points FFT, then the result multiplies with the twiddle factor, Thirdly, let the result do 2 point FFT between 64-pt and 2-pt FFT outputs. Similarly, Calculation of the 512-point FFT firstly do the 64 points FFT, then transform further 8 points FFT [58]; The same way to calculation of the 1024 and 2048 points FFT is implemented. Block diagram of the overall design of the FFT processor is shown in Fig.3.8.

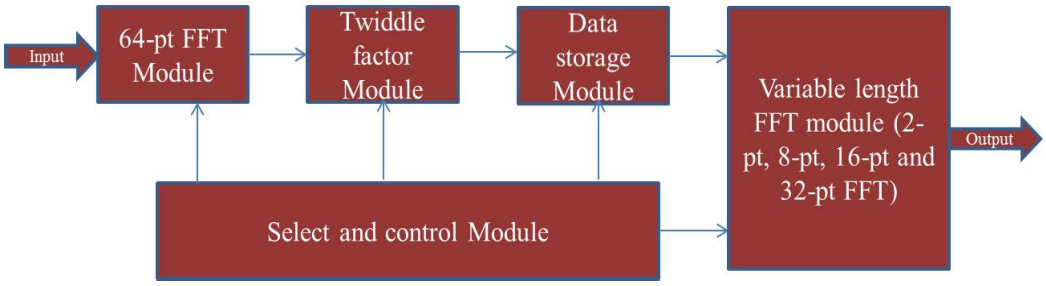

Fig.3.8.Block diagram of the overall design of the Variable length FFT processor

3.6.1. Pipeline structure of the 64-point FFT

The 64 point FFT is also implemented in 2-Dimensional FFT fashion using 8-point FFT modules as the kernel part. This module is the most frequently used in the design. Four kinds of input data length all must first pass through the 64 points FFT module. Block diagram of this part is shown in Fig.3.8. The same idea to implement the 64-point FFT in the module based on 2D Fourier transform algorithm is composed of two 8-point FFT modules [59].

The 8-point FFT processor architecture consists of a single radix-2 butterfly (which is referred as the butterfly processing element), a dual-port FIFO RAM, a coefficient ROM, a controller and an address generation unit.

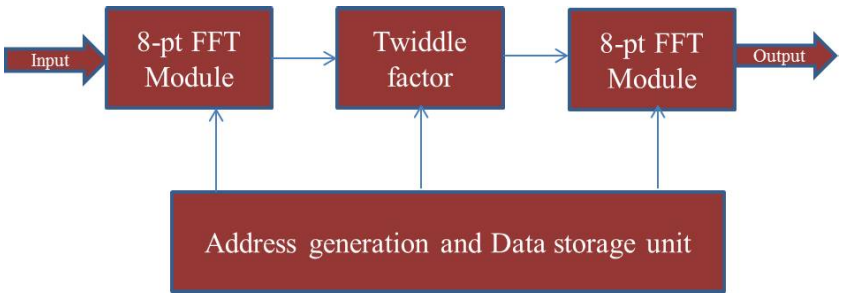

Fig.3.9. Pipeline structure of 64-point FFT Processor

3.6.2 Internal Block Diagram of the 64-point FFT:

The Fig. 3.10 shows the internal blocks of 64 point FFT module. The Input Buffer takes continuous data(x_in) and apply input clock as in_clk as shown in below diagram. This input Buffer performs as a SIPO (serial input and parallel output) i.e. it takes input as serial data and gives output as parallel. It will arrange the input data into rows and columns wise based on $\mathbf{n_2N_1+n_1}$ Where $0 \leq n_2 \leq N_2 - 1$ and $0 \leq n_1 \leq N_1 - 1$.

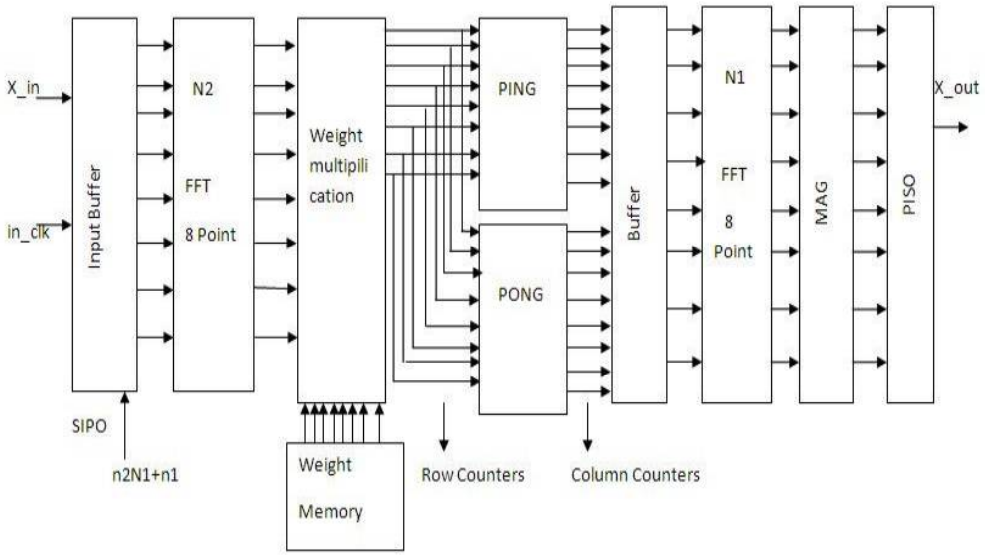

Fig.3.10. Internal Block diagram of the 64-point FFT module

Table.3.8. Data arrangement in Input Buffer based on $n_2N_1+n_1$

n1=0	n1=1	n1=2	n1=3	n1=4	n1=5	n1=6	n1=7
0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

The data arrangement in the input buffer is based on $\mathbf{n_2N_1+n_1}$ and the same is shown in the table 3.8. In Input Buffer it will select first column because it is serial input parallel output, so it will select first column and send to the next block i.e. N2 point FFT, this process is continuous up to end of the data. For example if input data can be 64 samples it will arrange the data as shown in above Fig. Next block is N_2 point FFT i.e. 8 point FFT so it will take 8 samples at a time so Input Buffer select first column it contain 8 samples an send to the N_2 point FFT. This perform computation is fast as compared with input buffer i.e. 8 times faster than input it is required because it will complete computation until the next input is reached. So as compared with clock FFT clock is 8 times faster than input clock. After completion of FFT computation it will send the data to Ping Pong Buffer before it will multiply with twiddle factors .This Twiddle factors are stored in weight memory and it will multiply with according to coming data and send the data to Ping Pong Buffer.

Fig.3.11. shows the implementation of the 64-point FFT module by using two 8-point FFTs using MATLAB software. The Figure 1 indicates the input signal where all the input points will be covered and figure two indicates two 8-point FFT's implemented and the peaks indicate the Real and Imaginary parts of the processed FFT.

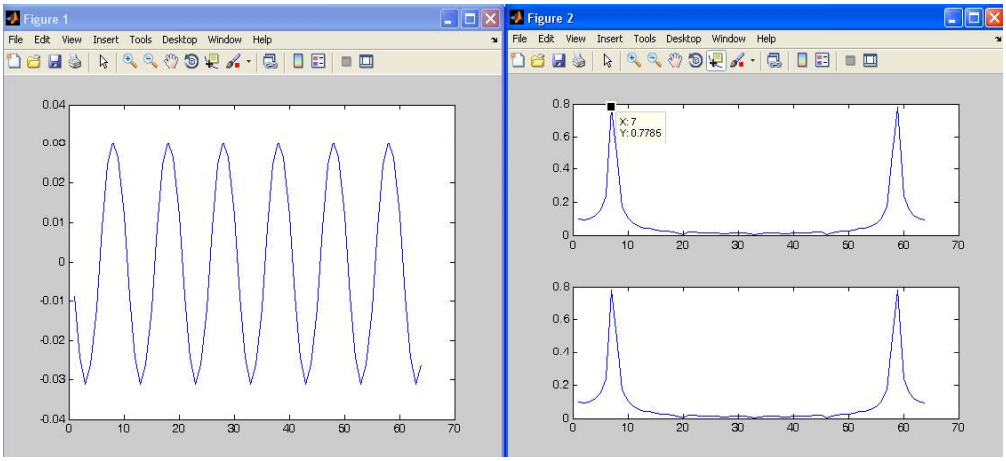

Fig.3.11. FFT 64 with two 8 point FFTs MATLAB results

3.6.3 Ping-Pong Memory architecture

The Ping Pong Buffer contains two memory blocks, these two memory blocks perform Simultaneously i.e. one memory block writes the first FFT output and Second memory block reads the data from first FFT output and in the next clock it will perform reverse operation i.e. first memory block reads, and another performs the write operation and again in next clock operations vice versa [60], [61].

Ping-Pong buffering increases memory bandwidth by a factor of two. Ping-pong buffering halves the number of memory operations per unit time, allowing faster buffers to be built from a given type of memory. Alternatively, for a buffer of given bandwidth, ping-pong buffering allows the use of slower, lower-cost memory devices. But ping-pong buffers have disadvantage that they waste a fraction of the memory. The overflow rate is increased until the additional memory is used half of the memory is wasted in the worst case. This can be compensated by doubling the size of the memory.

Fig.3.12 shows a ping-pong buffer of total capacity M cells, with the arrival and the departure processes denoted as A and B, respectively. The buffer consists of

two physically separate memory devices, each of size M/2. The two memories are arranged in such a way that from the outside, they appear to be a single buffer. Read and Write operations can take place simultaneously in a ping-pong buffer, but only in physically separate memory devices. When a cell arrives and finds that one memory is being read, as shown in Fig.3.13, the arriving cell is directed into the other memory device. We call this type of write operation a 'constrained write'; we have no choice into which memory to write the arriving cell.

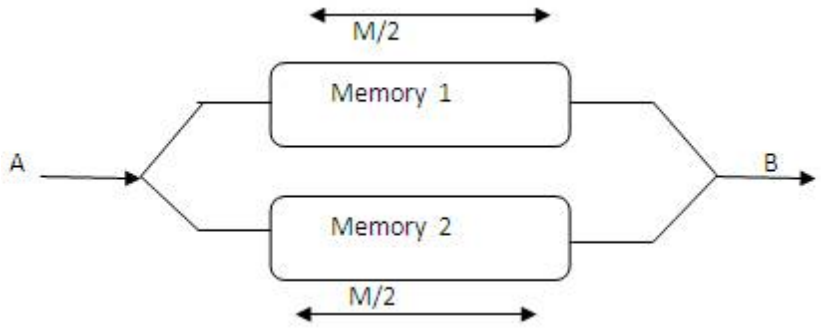

Fig.3.12. PING PONG memory

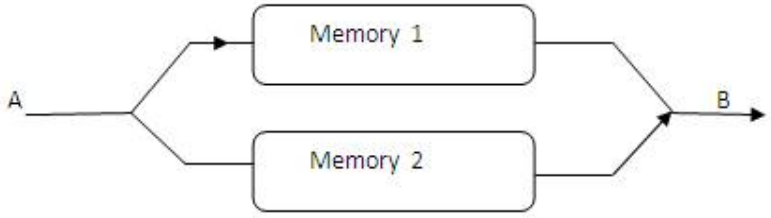

Fig.3.13 PING PONG memory Read and Write operations

If Ping Pong memory writes the output data in to the buffer as column wise means it will send the data as parallel to the next buffer. This buffer arranges complete data as rows and columns wise. It will send the first column to the N_1 point FFT i.e. 8 point FFT which is the kernel module. This process continuous up to last column .The output of N_1 point FFT can be perform magnitude operation i.e. Real ^{2}+imaginary^2= magnitude in the Magnitude block and send output to output Buffer which is PISO (parallel input and serial output). Since it works on high frequency, it will send the output as serially. This can be achieved by applying

clock whose clock rate is 64 times faster than input clock. Finally the data is captured at the output serially.

As mentioned above, the result multiplies with twiddle factor, and performs 2 point-FFT with the result, Finally 128 point FFT is achieved. Similarly, Calculation of the 512-point FFT firstly do the 64 points FFT, then transform further 8 points FFT; The same way is used to calculate the 1024 and 2048 points FFT.

Select and control module is also the kernel part to complete the alterable data length in the current dissertation. It is based on the input data points to select the results stored in data memory and choose the next flow. A two bits signal 'mode' is chosen as the mode signal simply as 4X1 Multiplexer to which mode 00, mode 01, mode 10, mode 11 will be the inputs. When mode = 00, means to choose 2 points FFT module, to complete the 128-point FFT; when mode = 01, means to choose 8 FFT module, to complete the 512-point FFT; Equally: When the mode = 10, means to completed 1024 point FFT; when mode = 11, means to complete the 2048 point FFT.

3.6.4 Direct Digital Synthesis Core

Direct Digital Synthesis (DDS) provides remarkable frequency resolution and allows direct implementation of frequency, phase and amplitude modulation instead of using function generators. DDS is a method of producing an analog waveform usually a sine wave by generating a time-varying signal in digital form and then performing a digital-to-analog conversion. Because operations within a DDS device are primarily digital, it can offer fast switching between output frequencies, fine frequency resolution, and operation over a broad spectrum of frequencies. With advances in design and process technology, today's DDS devices are very compact and draw little power [62], [63], [64], [65].

DDS devices like the AD9833 are programmed through a high speed serial peripheral- interface (SPI), and need only an external clock to generate simple

sine waves. DDS devices are now available that can generate frequencies from less than 1 Hz up to 400 MHz (based on a 1-GHz clock). Because a DDS is digitally programmable, the phase and frequency of a waveform can be easily adjusted without the need to change the external components that would normally need to be changed when using traditional analog-programmed waveform generators. DDS permits simple adjustments of frequency in real time to locate resonant frequencies or compensate for temperature drift.

The core consists of two main parts, a Phase Generator and SIN/COS LUT, which can be used independently or together with an optional dither generator to create a DDS capability. A time-division multi-channel capability is supported, with independently configurable phase increment and offset parameters. Fig.3.14 provides the block diagram of the DDS Compiler core.

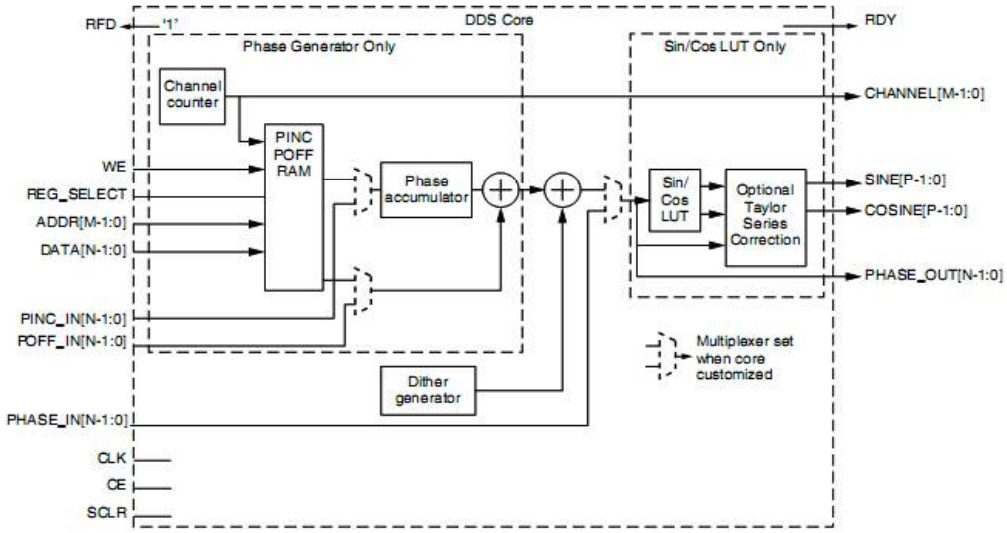

Fig.3.14. DDS Compiler Core

Fig.3.15 and Fig.3.16 shows the simulation result i.e. Modelsim result of variable length FFT processor for OFDMA system which is implemented by using 2D-FFT algorithm. A 50 MHz cosine signal was generated from the DDS core in which all the input points will be covered for the processing of variable FFT sizes.

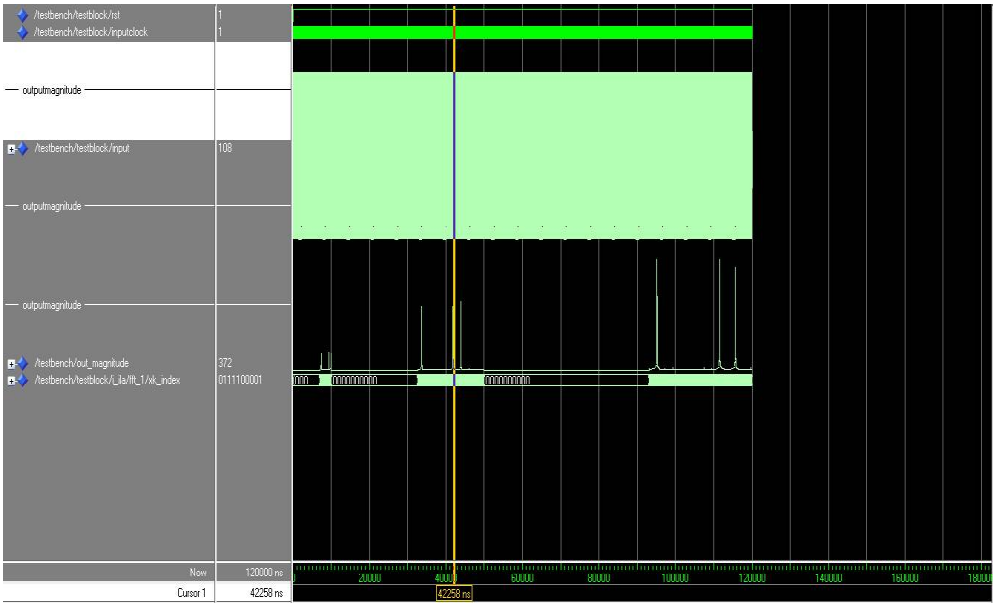

Fig.3.15 Simulation result of Variable length FFT processor for OFDMA

In the Fig.3.15.depending on the index that is mode of the processor, if it is 00 then the processor performs 128-pt FFT, and if the input mode is 01 then it performs 512-pt FFT similarly for 1024-pt and 2048-pt FFT's. The peaks in the Fig.3.15.with respect to the distance of the out_magnitude indicate that the processing of four modes have been completed. The first peak occurred indicates the 128-pt FFT has been processed and two peaks are used to show the processing of Real as well as Imaginary parts of the input. Similarly the second peak indicates processing of 512-pt FFT and is same for the 1024-pt and 2048-pt FFT's also.

Fig.3.16 also shows the same processing with extended input. And Fig.3.17 shows the synthesis result i.e. the chipscope result on the vertex FPGA kit. The same has been programmed in VHDL language for the hardware implementation.

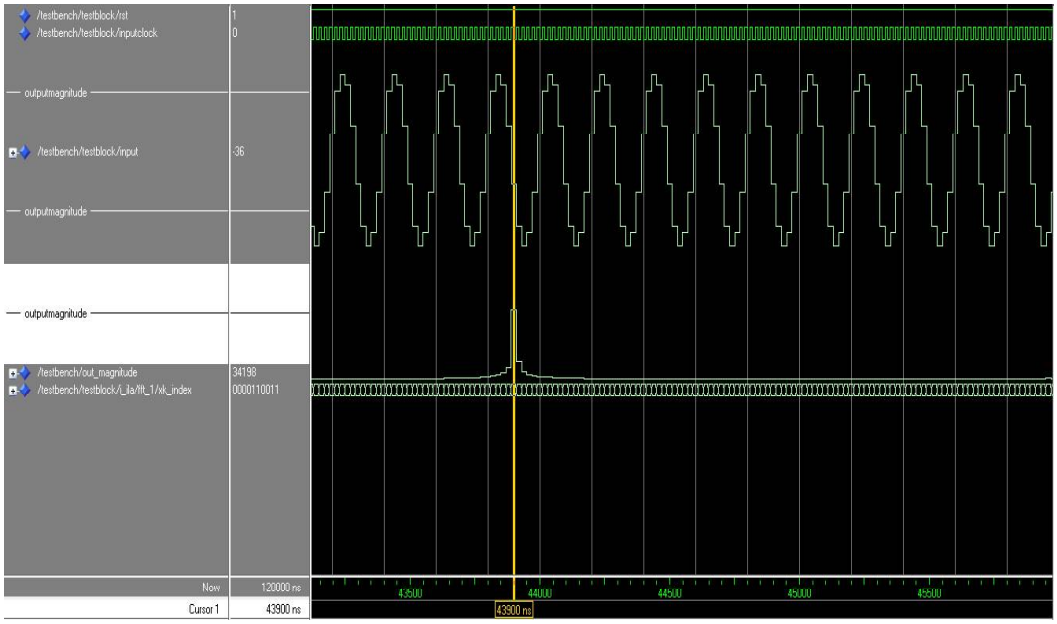

Fig.3.16 Expanded Simulation result of Variable length FFT processor for OFDMA

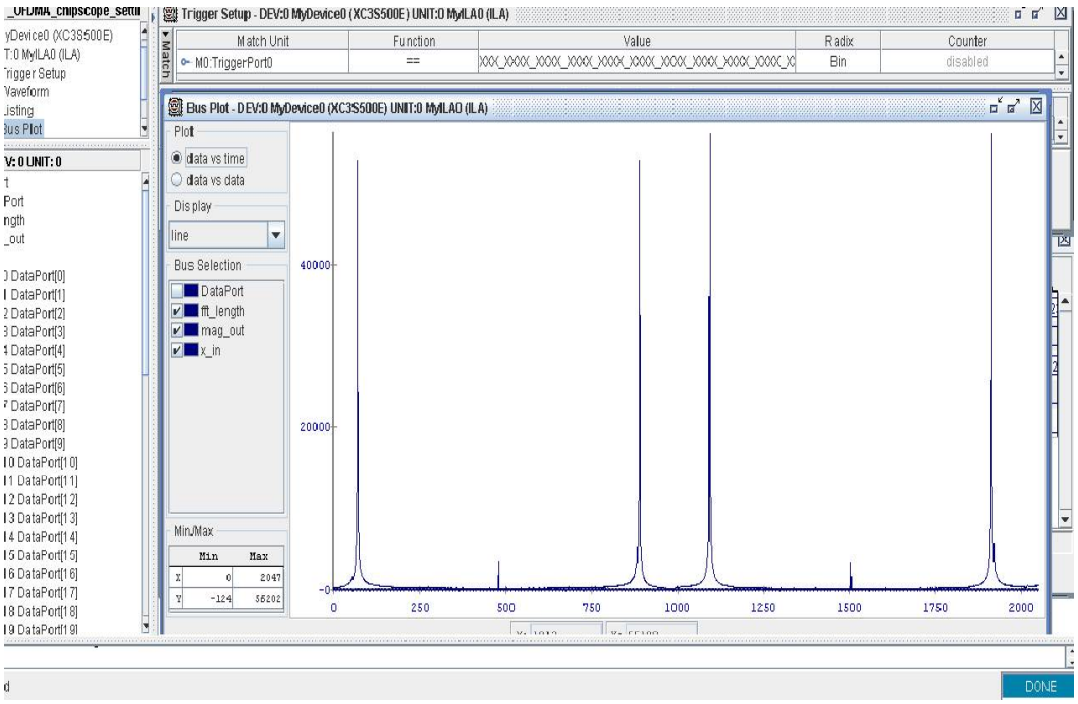

Fig.3.17. Chipscope Synthesis result of Variable length FFT processor for OFDMA

3.7 SUMMARY

In this chapter starting from the advantages of carrying out DSP's on FPGA, binary encoding format, number representation and signed magnitude number representation is also shown. In the next section 8-point FFT algorithm has been shown which is the kernel module of the OFDMA processing using 2D-FFT. Simulation result for the Adder module has been shown. Twiddle factor coefficients have been calculated and also simulation 8-point FFT has been implemented. In the next section the important part of the dissertation i.e. variable length FFT processor using 2D-FFT algorithm for OFDMA system has been implemented. The overall block diagram, and the internal processing along with the ping-pong memory architecture and DDS-core has been studied and at the last implementation part simulation and synthesis results of variable length FFT processor using 2D-FFT algorithm were shown.