International Journal of Innovative Research in Advanced Engineering (IJIRAE) ISSN: 2349-2163
Volume 1 Issue 8 (September 2014) www.ijirae.com

Design & Development of IP-core of FFT for Field Programmable Gate Arrays

Bhawesh Sahu
ME Reserch Scholar ,sem(IV),
VLSI design, SSTC,SSGI(FET),Bhilai,

Anil Kumar Sahu
Assistant Professor,SSGI(FET),Bhlai

Abstract— ***Fourier transform play an important role in many digital signal processing applications including acoustics, optics, telecommunications, speech, signal and image processing. However, their wide use makes their computational requirements a heavy burden in much real world application. Direct computation on discrete Fourier transform requires on the order of N^2 operations, where N is the transform size. The FFT algorithm first explained by Cooly and Turkey[4] opened a new area in digital signal processing, by reducing the order of complexity from N^2 to $N.Log_2N$ for a length $N=2^n$DFT. Most of the research to date for the implementation and bench -marking of FFT algorithm has been performed using general purpose processors, Digital signal processors and dedicated FFT processor IC's. However as FPGA's have grown in capacity ,improved in performance and decreased in cost they have become a viable solution for performing computationally intensive task, with the ability to tackle applications for custom chips and programmable DSP devices. Although there has been intensive research on the hardware implimentacity of FFT algorithm. Reconfigurable hardware, usually in the form of FPGA has been used as a new and better means of performing high performance computing. Reconfigurable computing systems are those computing platforms whose architecture can be modified to suit the application at the hand. Reconfigurable computing involves manipulation of the logic within the FPGA at the runtime. In other words the design of hard ware may change in response to the demands placed upon the system while it is running. This paper discuss about FPGA implementation of radix-4 FFT algorithm, which is simulated ,synthesized, and downloaded on xcv1000 FPGA, which gives the operating speed of 52.3 MHz.***

***Keyword:*FFT,Radix-4 algorithm**

I.INTRODUCTION

The discrete Fourier Transform is widely used DSP algorithm and the key digital filtering operation. It is a transform domain representation which is applicable to only finite length sequences. It can be employed to implement linear convolution of two sequences, and is also used for the frequency domain analysis of signals. Because of wide spread use of the DFT,it is of interest to investigate its efficient implementation-methods.

II. FFT

The fast Fourier transform is a class of efficient algorithms for computing the **DFT**. It always gives the same results (with the possible exception of a different round-off error) as the calculation of the direct form of the DFT .The term "*fast **Fourier transform***" was originally used to describe the fast DFT algorithm popularized by **Cooley and Turkey's** landmark paper (1965).

The basic idea behind the all fast algorithms for computing the discrete Fourier transform (DFT), commonly called FFT algorithms, is to decompose successively the N point DFT computation in to computations of smaller size DFTs and to take advantage of the periodicity and symmetry of the complex number W^{kN}_N.such decompositions, if properly carried out ,can result in a significant savings in the computational complexity.

There are various versions of FFT algorithms As proposed by Cooly Turkey approach leads to auxiliary complex multiplications, initially named twiddle factors, which cannot be avoided in this case. From a theoretical point of view, the complexity issue of discrete Fourier transform has reached a certain maturity. Note that Gauss, in his time, did not even count the number of operations necessary in his algorithm.

In particular ,Winograd's work on DFTs whose lengths have co-prime factors both sets lower bounds (on the number of multiplications) and gives algorithms to achieve these ,although they complexity of the algorithm but also the lack of practical algorithms achieving this minimum(due to the tremendous increase in the number of additions).Considering implementations, the situation of course more involved since many more parameters have to be taken in to account than just the number of operations. A number of subsequent papers presented refinements of the original algorithm, with the aim of increasing its usefulness.

It seems that both the radix4 and split radix algorithm are quite popular for lengths which are power of 2, 4.But *radix-4* and *split radix* has the advantage of having **better structure** and **easier implementation.**

III. Radix-4 Algorithm

When N = 4k, we can employ a radix-4 common-factor FFT algorithm by recursively reorganizing sequences into N_ × N_/4 arrays. The development of a radix-4 algorithm is similar to the development of a radix-2 FFT, and both DIT and DIF versions are possible.

Rabiner and Gold (1975) provide more details on radix-4 algorithms. Figure 1 shows a radix-4 decimation in time butterfly. As with the development of the radix-2 butterfly, the radix-4 butterfly is formed by merging a 4-point DFT with the associated twiddle factors that are normally between DFT stages. The four inputs A, B,C, and D are on the left side of the butterfly diagram and the latter three are multiplied by the complex coefficients Wb, Wc, and Wd respectively. These coefficients are all of the same form as the WN, but are shown with different subscripts here to differentiatethe three since there are more than one in a single butterfly. The four outputs V , W, X, and Y are calculated from,

V = A +BWb +CWc +DWd----------(1)

W = A − iBWb −CWc + iDWd--------(2)

X = A −BWb +CWc −DWd----------(3)

Y = A + iBWb −CWc − iDWd.---------(4)

The equations can be written more compactly by defining three new variables,

B_ = BWb----------(5)

C_ = CWc----------(6)

D_ = DWd, ----------(7)

leading to,

V = A + B_ + C_ + D_ ----------(8)

W = A − iB − C + Id---------- (9)

X = A − B_ + C_ − D_---------- (10)

Y = A + iB_ − C_ − iD_----------. (11)

It is important to note that, in general, the radix-4 butterfly requires only three complex

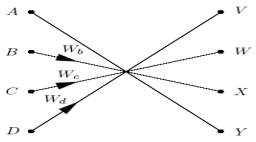


Fig.1 .A radix-4 DIT Butterfly

IV-A SPECIFICATION [6]

• Forward and inverse complex FFT
• Transform sizes $N = 2m$, $m = 3 – 16$
• Data sample precision $bx = 8 – 24$
• Phase factor precision $bw = 8 – 24$

- Un scaled (full-precision) fixed-point
- Scaled fixed-point
- Block floating-point

• Rounding or truncation after the butterfly
• On-chip memory
• Block RAM or Distributed RAM for data or phase factor storage
• Run-time configurable forward or inverse operation

• Optional run-time configurable transforms point size
• Run-time configurable scaling schedule for scaled fixed point
• Bit/digit reversed output order or natural output order
• Three architectures offer an exchange between core sizes and transform time

The FFT core computes an *N*-point forward DFT or inverse DFT (IDFT) where *N* can be 2*m*, *m* = 3–16. The input data is a vector of *N* complex values represented as *bx*-bit two's-complement numbers – *bx* bits for each of the real and imaginary components of the data sample(*bx* = 8 – 24). Similarly, the phase factors *bw* can be 8– 24 bits wide. All memory is on-chip using either Block RAM or Distributed RAM. The *N* element output vector is represented using *by* bits for each of the real and imaginary components of the output data. Input data is presented in natural order, and the output data can be in either natural or bit/digit reversed order.

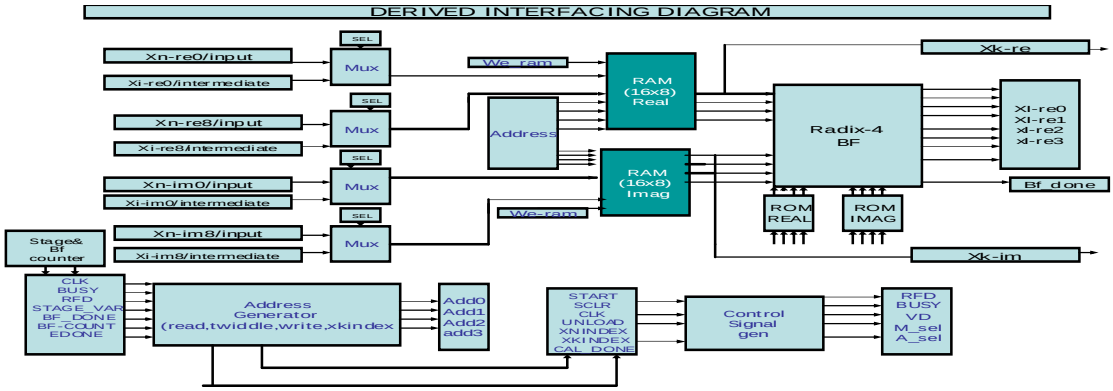
Several parameters can be run-time configurable: the point size *N*, the choice of forward or inverse transform, and the scaling schedule. Both forward/inverse and scaling schedule can be changed frame by frame .Changing the point size resets the core.

IV -B ARCHITECTURE OPTIONS

Three architecture are available:

• **Pipelined, Streaming I/O.** Allows continuous data processing.
• **Radix-4, Burst I/O.** Offers a load/unload phase and a processing phase; it is smaller in size but has a longer transform time.
• **Radix-2,** Minimum Resources. Uses a minimum of logic resources and is also a two-phase solution.

The FFT core provides three architecture options to offer a trade-off between core sizes and transform time



V-A FINITE WORD LENGTH CONSIDERATIONS

The radix-4 and radix-2 FFT algorithms process an array of data by successive passes over the input data array. On each pass, the algorithm performs radix-4 or radix-2 butterflies, where each butterfly picks up four or two complex numbers and returns four or two complex numbers to the same memory. The numbers returned to memory by the processor are potentially larger than the numbers picked up from memory. A strategy must be employed to accommodate this dynamic range expansion.

For a radix-4 DIT FFT, the values computed in a butterfly stage (except the second) can experience a growth to $4\sqrt{2}=5.657$. .For radix-2, the growth can be up to $1+\sqrt{2}=2.414$. This bit growth can be handled in three ways:

• Performing the calculations with no scaling and carrying all significant integer bits to the end of the computation
• Scaling at each stage using a fixed-scaling schedule
• Scaling automatically using block-floating point

All significant integer bits are retained when doing full-precision unscaled arithmetic. The width of the data path increases to accommodate the bit growth through the butterfly. The growth of the fractional bits created from the multiplication are truncated (or rounded) after the multiplication. The width of the output will be the (input width + number of stages + 1). This will accommodate the worst case scenario for bit growth. For example, a 1024-pt transform with an input of 16 bits consisting of 1 integer bit and 15 fractional bits, will have an output of 27 bits with 12 integer bits and 15 fractional bits. The core does not have a specific location for the binary point. The output will simply maintain the same binary point location as the input. For the above example, a 16 bit input with 3 integer bits and 13 fractional bits would have an unscaled output of 27 bits with 14 integer bits and 13 fractional bits.

When using scaling, a scaling schedule is used to scale by a factor of 1, 2, 4, or 8 in each stage. If scaling is insufficient, a butterfly output may grow beyond the dynamic range and cause an overflow. As a result of the scaling applied in the FFT implementation, the transform computed is a scaled transform. The scale factor *s* is defined as,

$$S = 2^{\sum_{i=0}^{LogN-1} b_i}$$

V-B RADIX-4, BURST I/O

With the Radix-4, Burst I/O solution, the FFT core uses one radix-4 butterfly processing engine and has two processes (Figure 2). One process is loading and/or unloading the data, and the second process is calculating the transform. Data I/O and processing are not simultaneous. When the FFT is started, the data is loaded. After a full frame has been loaded, the core computes the FFT. When the computation has finished, the data can be unloaded, but cannot be loaded or unloaded during the calculation process. The data loading and unloading processes can be overlapped if the data is unloaded in digit reversed order.
This architecture has less resource usage than the Pipelined Streaming I/O architecture but a longer transform time, and covers point sizes from 64 to 65536. All three arithmetic types are supported:unscaled, scaled, and block floating point. Data and phase factors can be stored in Block
RAM or in Distributed RAM(for point sizes less than or equal to 1024.

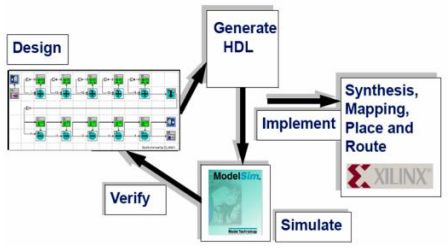


fig:3 Design flow diagrams

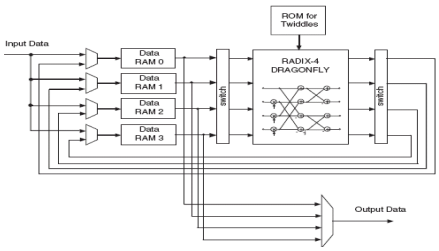


Fig.: Radix-4 burst mode

Now First task according to design flow graph of fig[3] is to study and understand the specification with all detail timing diagram and architecture option. There I took the help of Mat lab .I implemented the generic core using radix2 and 4 in malt with variable point size and variable input bit width and twiddle bit width. I have calculated R.M.S errors and Percentage errors for variable bit width and variable point size. this shows that with increasing the point size from 8 to 64k the error increases gradually ,also increasing input bit width from 8 bit to 24 bit the errors decreases.

Then have considered take input and twiddle bit precision of 8 bit and output bit precision also of 8 bit with variable point size. First session I have developed a radix-2 butterfly with my own multipliers and adders, but then my guide told me to take available speed optimized multipliers and adder-sub tractors.

Then in the second session I have integrated all blocks using timing diagrams given in the specification .

1.radix-4 butterfly

2.contol signal generators

3 address generators for twiddle and data with stage and bf counters.

4 ram for real and Imaginary (16x8)

5 rom real and Imaginary (8x8)

6 mux for data and address

Then after Integrating the above modules I have simulated and synthesized the core for fix point (N=16) and
Compared the results with the same modeling in mat lab. this gives 5 to 10% errors. Finally I have downloaded the core on FPGA.

VI-A DESCRIPTION OF FUNCTIONAL MODULES:

1.Control signal generator:
Control signal generator is reset when the signal SCLR is '1'.actually SCLR is a global reset pin which is asynchronous.

when SCLR =’0’ and busy=’0’ then if START =’1’ then RFD=’1’ which is an acknowledgement for the loading the data in to the RAM. when N-1 the data is being loaded the RFD signal Goes low.
When RFD=’0’ initialize the calculation phase by generating busy=’1’.when all butterfly (N/4*M-1) calculation is over then cal_done signal goes high, which reset the busy signal at the same edge.
Then when unload=’1’ then VD signal is generated which starts unloading which is indexed by Xk-index pin, when xkindex=’1111’ then VD=’0’ is generated after this phase is over again start signal can start the next data loading phase.

2.Address generator:

There are four different address generators are used in my core.

2.1.Write address generator

1.first is used to index the input data,i.e address the ram from 0000 to 1111.when sclr=’1’ then this address generator is reset, and when sclr=’0’ then it starts generating the address from 0 to 15.it increments the address every clock. here for I am writing two real and two imaginary data simultaniously,so that only 8 clock cycles are needed to load the data.

2.2Read address generator

After Load phase is over busy signal goes high, and hence it requires reading the two real and two imaginary data from Real and Imaginary RAM which is passed to the Radix-4 Butterfly. so this address generator is in reset mode when sclr=’1’ and starts generating four addresses when Busy=’1’,this address generator is incremented every time when the signal EDONE=’1’,this is required in order to restore the intermediate results from the butterfly at the same place from where this data were read(In place calculation).

2.3.Twiddle address generator

When calculation phase starts with Busy=’1’ ,The twiddles are also required for each butterfly calculation ,so this address generator generates correct addresses of corresponding twiddle required ,again 4 addresses are read after Busy=’1’from the ROM at each butterfly operation.

2.4.Unloading address generator

cal_done signal is=’1’ When all m stages of N/4 butterflies are calculated. Now it is time to display the result on xk_re, and xk_im pins ,Now it is always required that this results must be indexed in order to take the correct output to the users, for that pin xkindex gives the corresponding point index. This address generator is initialized when vd=’1’.

3.Rom:

In cooly Turkey Radix-4 algorithm when the splited factors are not coprime,twiddles can not be avoided, and hence at every butterfly calculation we need the twiddle factors. which are constants must be stored prior to calculation. and we must read 4 twiddles at each calculation time. This generator is reset when sclr=’1’ and starts generating 4 addresses when busy=’1’ and address is incremented when edone=’1’.

4.Ram

The data form the External World comes serially, and here we use butterfly for all intermediate calculations. As all calculations done one by one we must store this data in the memory ,so that at each calculation of butterfly the input can be read from there. In the loading phase when the RFD=’1’ we write the data in to the ram when we_ram=’1’ and read the data when we_ram=’0’.

5.Radix-4 Butterfly

This module implements radix-4 butterfly. It takes three clock cycles to complete the radix-4 calculation. It is in reset mode when the input signal reset=’1’.In the first clock cycle it does the first step calculation with add subtract opration.In the second clock cycle it implements the intermediate stage data and twiddle multipliers. then in the third clock it does necessary correction in the data. The correction means shifting right the results by 7 bit ,this is required because the twiddles were multiplied by 128 for precise values. so the 15 down to 0 results is being converted to 15 down to 7.now I scale down the least significant bit by one bit. so the result is 15 down to 8.(8 bit) at the third clock only signal bf_done goes high, which indicate that now its time to restore the intermediate results to the same addresses and read the next four real-and four imaginary data to the butterfly. when the data are restored the edone goes high which actually increments the read address generator.

6.Reset and wr_inmr generator

These signals are control signals. Reset signal set and reset the butterfly in order to perform m stages with N/4 butterflies in each stage. the butterfly must read correct inputs at each calculation time ,and hence this signal controls the working of it.
The wr_inmr signal is necessary for writing the intermediate data to the correct place in the Ram. this signal becomes a select signal for the mux.because we have four real and four imaginary outputs of butterfly to write each time but our Ram can write on two addresses simultaniously,so two clock cycles are required to write four data. so this signal will change from '1' to '0' during this write operation to change two addresses on address pins of the Ram through the Mix.

7.Wr_en generator

This signal is very important because it generates the we-ram signal which is required to perform write operation in RAM. We-ram must be '1' during the loading phase and also during the calculation phase when we write the intermediate data to the ram .

II VHDL SIMULATION AND SYNTHESIS RESULTS

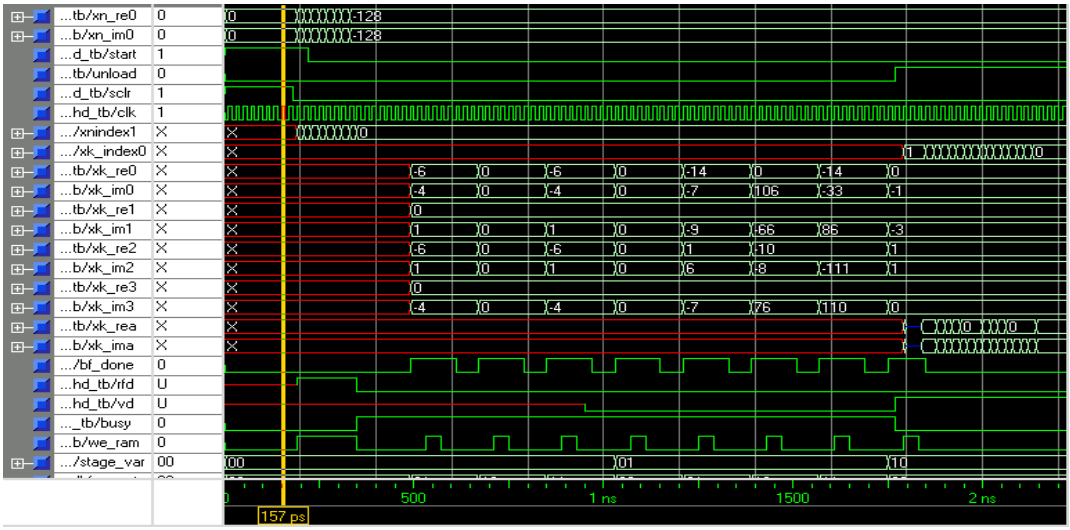


Fig.6 Simulation of FFT Processor

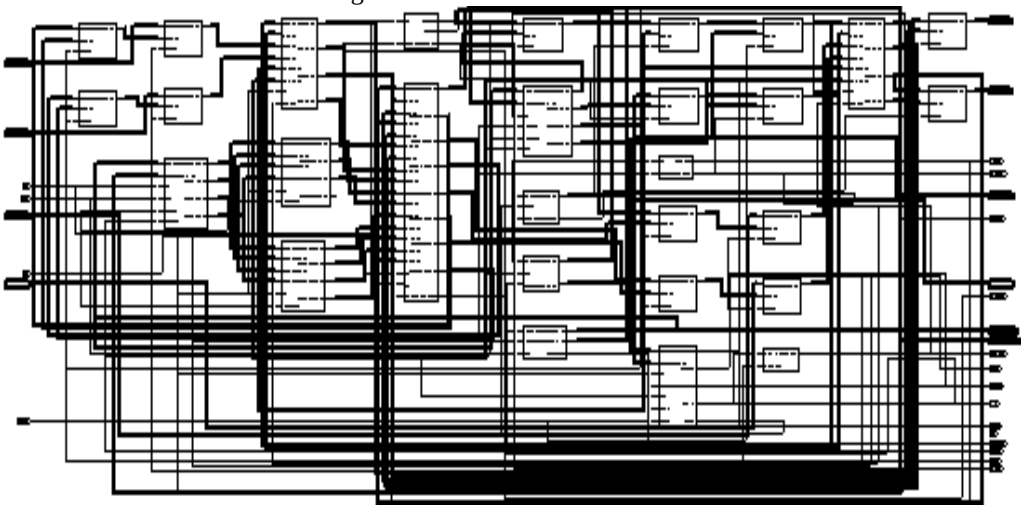


Fig.7.RTL of FFT Core

IX CONCLUSION

The synthesis results of my core is following.

Timing Summary:

Speed Grade: -4
Minimumperiod:19.080ns(MaximumFreq52.411MHz Minimum input arrival time before clock: 17.635ns Maximum output required time after clock: 8.498ns Maximum combinational path delay: 12.012ns

(1) Synthesis Result of My core

Device utilization summary:
Selected Device : v1000bg560-4
Number of Slices: : 282 out of 12288 2%
Number of Slice Flip Flops: : 304 out of 24576 %
Number of 4 input LUTs: : 361 out of 24576 1%
Number of bonded IOBs: : 77 out of 408 18%
Number of GCLKs : 1 out of 4 25%

.

REFRENCES

[1]. FPGA implementation of FFT for real time signal &image processing by I.S.Uzun,A.Amira&A.Bouridone,IEEE proceedings on vision,image &signal processing ,volume152no:3may2005
[2]. VLSI Implementation of FFT by .M.Chau,W.H.Ku, Ieee proceedings on real time signal processingIX, volume-6,1986
[3]. Fault Taulerent FFT Networksby Jing-young jou,jacob abraham,Ieee transaction on computer MAY1988
[4]. J.W. Cooley and J. W. Tukey, An Algorithm for the Machine Computation ofComplex Fourier series, Mathematics of Computation, Vol. 19, pp. 297-301, April 1965.
[5]. J. G. Proakis and D. G. Manolakis, Digital Signal Processing Principles,Algorithms and ApplicationsSecond Edition, Maxwell Macmillan International, New York, 1992.
[6]. Xilinx Logicore – Fast Fourier Transform version 3.2 (xfft320) – Product Specification
[7]. Application notes from altera.
[8]. Discrete-Time Signal Processing – Alan V. Oppenheim, Ronal W. Schafer & John R. Buck
[9] Digital signal processing with field programable gate arrays by Uwe meyer-Baese,springer International edition.
[10]. Digital signal processing by sanjit.k.mitra
[11]Digital systems design using VHDLby charles H.roth,jr.,Thomsan learning edition.
[12]. Digital logic and computer design by M.Moris Mano,prentice Hall India.
[13]. VHDL Primer, J. Bhaskar
[14]. VHDL , Douglas Perry.

