HIGH SPEED AND AREA EFFICIENT VLSI IMPLEMENTATION ON REAL VALUED FFT STRUCTURES

[1] G.SIVA, [2] Mr.M.Adiseshaiah [3]Dr.P.Krishna Murthy,

[1]M.Tech PG Scholar, [2]Asst.Professor,[3] Associate Professor & Head of the Department
[1,2] Dept. of Electronics & Communication Engineering
[1,2,3]Chadalawada Ramanamma Engineering College, Tirupati, Andhra Pradesh, India.

Abstract
Efficient computation of real-valued fast Fourier transform (RFFT) has received significant attention in recent years due to its several applications in conventional digital signal processing and other emerging areas. In-place RFFT architectures are gaining popularity due to their lower hardware complexity compared with pipeline architectures. In previous method, a design approach is presented to develop an area-delay and energy-efficient architecture for in-place RFFT. Generally, an in-place fast Fourier transform (FFT) structure consists of a *butterfly block* which performs a set of butterfly operations in every clock cycle. The resolution of memory access conflict is however more challenging for higher butterfly block sizes. Therefore, we have analyzed the data-flow and memory footprint of in-place RFFT architectures for different throughput requirements, and based on that, we have proposed here a strategy to partition the storage unit into several banks of smaller sizes (without increasing the overall memory size) to resolve the memory access conflicts by concurrent data-swapping between the banks. In this paper ,we have designed a new butterfly unit is proposed in which 8 point butterfly unit is implemented by two 4 point FFT unit with simple butterfly structure and low memory requirement. so when compare to the previous method the proposed method consume less area over ahead and less delay. The synthesis and simulation results are verified by using Xilinx ISE 14.7 tool.

Index Terms—Fast Fourier transform, in-place computation, real-valued FFT.

1.Introduction

Fast Fourier Transform (FFT) is widely used in several applications such as spectral estimation, speech and sound processing, image processing, digital communication, wireless sensor network (VSN), radar signal processing, biomedical signal processing and many others. Due to the growing demand for real-time multimedia services and high data rates, the data rates of wireless communication systems are constantly increasing. In these applications, velocity calculations are performed on sequences of a wide range of lengths and sampling rates. The size of the FFT varies from 64 to 32768, and the sampling rate generally varies from a few mega-samples / s to giga-samples / different applications. Therefore, it is necessary to have a large-scale FFT calculation with adequately high bandwidth in accordance with the requirements of the applications. Realizing larger FFTs in a resource-constrained environment and providing high bandwidth is a challenging task. FFT architectures are mainly two contrasting categories: pipeline architecture (i.e., one-way delay-based feedback (SDF) structures and multi-way delay switch (MDC) -based structures, and assembled processor-based memory (PM) structures. In general, all of these structures consist of three main functional units, namely: (i) butterfly computing units (BCUs), (ii) storage units (SUs), and (iii) tweeddle generating units (TFUs). based structures are generally preferred for FFT calculation due to their regular data flow and higher efficiency of BCU utilization at the cost of marginally higher memory requirements than SDF based structures. the butterfly computational phases are overlapped in the BCU and take advantage of the built-in FFT to realize low complexity. Pipeline and PM-based architecture in place are the two main variants of FFT structures with contrasting characteristics and the choice of one of these two ca Architecture theories largely depends on the space-time constraints of the target application. Therefore, several pipeline and PM-based architectures have been proposed for the efficient application of FFT for different applications. There are several applications such as speech, sound, image and video processing, where FFT signals of real value are used. Today's attention is paid to the efficient realization of FFT of signals with real value due to the appearance of biomedical signal processing and wide applications of time series analysis with real value. Some efforts have also been made in the development of specific architectures for real FFT (RFFT). The data flow of the RFFT algorithm is becoming increasingly disordered for larger FFT sizes, making RFFT design in place very challenging due to the increasing memory access conflict. Some on-site PM-based architectures have been proposed for RFFT using specialized packaging algorithms. An on-premises architecture and conflict-free addressing scheme for RAM-based memory banks have been proposed for continuous RFFT processing. The SU architecture in place with N-point in place is implemented using 2N RAM with one port. Recently, a register-based SU design for the built-in RFFT architecture was introduced to save memory footprint. It has been observed that a pair of two-point butterflies (called a size 2 butterfly block) is used in the existing RFFT design. Due to the nature of on-site computation, SU complexity increases linearly only with FFT length and contributes significantly to total area for lengths greater than 32. During any given clock cycle, the RFFT structure of a size 2 butterfly block uses only 4 N-words stored in SU. This results in low SU use. For small butterfly blocks, combinational logic is a small part of the storage space and therefore the storage is underutilized. On the other hand, if we increase the size of the butterfly block, then with a small increase in overall hardware complexity not only memory can be better utilized, but also bandwidth, product with area delay and power performance can be significantly improved. Scaling the built-in RFFT architecture for larger lengths and larger butterfly block sizes is a challenging issue due to the growing conflict in memory access and high memory bandwidth. Conventional storage structures and SU designs do not support larger butterfly block sizes due to memory access conflicts. We found that SU could be split into a number of banks and an appropriate scheme could be developed to exchange stored data between those banks to avoid a memory access conflict resulting from larger butterfly blocks. We analyzed the complexity of SU along with the memory footprint for

different bandwidth to study memory access by outputand and identify the design constraint for conflict-free memory access. Based on this, a design approach is proposed here for the development of efficient time-delayed architectures

II. COMPLEXITY ANALYSIS AND DESIGN STRATEGY

The SU consists of L register-banks of depth P words each for butterfly block size $(L/2)$, where $P = N/L$, L is the input block size and N is the FFT size. During every clock cycle, a block of L data is read from L banks for parallel computation of $(L/2)$ 2-point butterfly, and stores one block of L outputs back into the same banks. The reordering of intermediate data has no effect when the entire computation of a butterfly stage is performed in one clock cycle (referred to as *full-parallel* structure). Consequently, the SU of the full-parallel structure is comprised of N banks where each bank stores only one sample. When butterfly computations of a particular butterfly stage are folded and performed in different clock cycles then the SU of folded structure need to be split into L banks where each bank stores $(P = N/L)$ intermediate values. We can find that $(P = 2 f)$, where f is the folding factor. The reordering of intermediate data could result in access conflict when a bank stores more than one intermediate data. It is observed that the number of reordering of data increases from the first stage to the final butterfly stages. The bank conflict associated with the butterfly computation of different stages depends on the folding factor (f). The bank conflict starts from $\{S - (f - 1)\}$-th butterfly stage to $\{S\}$-th butterfly stage, where $S = \log2\ N$. Therefore, the access conflict arises early in the butterfly stages for higher folding factor and late in the butterfly stages for smaller folding factor. To avoid these access conflicts, data-swapping between the banks is performed before writing data into the banks. Moreover, it is observed that the data read from the SU are not in the order as required by a particular butterfly stage and they need to be reordered. Therefore, data-swapping need to be performed before and after the memory operations on the banks. Input-output data-flow analysis of the SU for different butterfly stages of RFFT need to be performed to identify the necessary data-swapping instances for resolving bank access conflict.

EXISTING STRUCTURE:

The existing structure for -point in-place DIT RFFT computation is shown in below figure 1. It consists of one arithmetic unit (AU), one data storage-unit (DSU), one twiddle-factor storage unit (TFSU), and one control-unit (CU). During every cycle, the AU receives a block of 8 samples from the DSU and a pair of twiddle factors for every cycle.

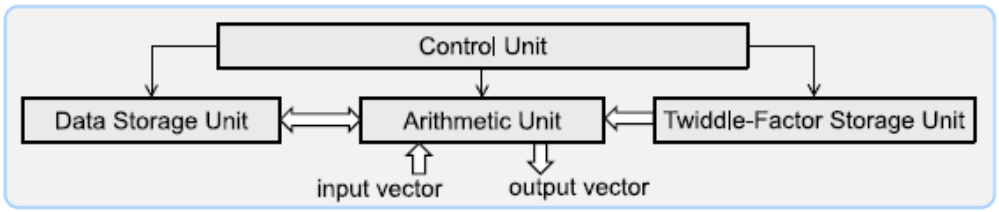


Fig.1. Existing structure for in-place computation of DIT RFFT.

The above figure shows the existing structure for in-place computation of DIT RFFT. which consists of arithmetic unit ,data storage unit , twiddle factor storage unit and control unit . the ALU units consists of butterfly operations that is performed in FFT. where as the data storage and twiddle factor storage unit are used to store the internal data and twiddle factor multipliers data respectively. To monitor this operation a control unit is required. the block diagram of control unit is as show in below figure 2. It consists of counter and multiplexers ,counter and AC indicates the AND cell.

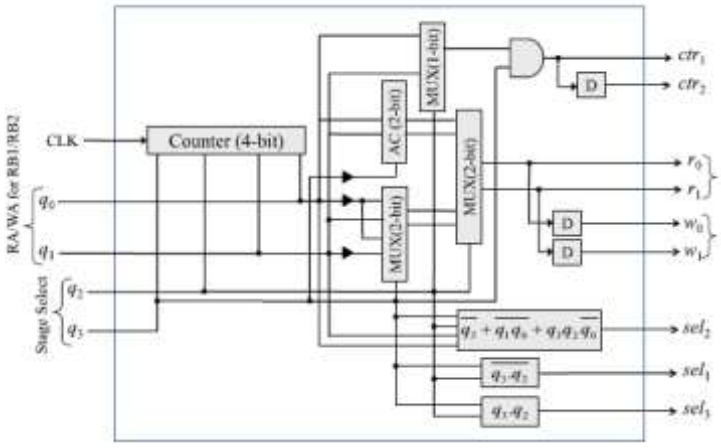


fig 2. Control unit (CU)

A. Data-Flow Analysis of Storage Unit

The BCU for butterfly block size $(L/2)$ takes (N/L) clock cycles to complete the computations of one butterfly stage and takes (N/L) $\log2\ N$ clock cycles to complete all the butterfly operations for N-point FFT. We have taken butterfly block size $(L/2) = 4$ and FFT size $N = 32$ as an example to analyze the input-output data-flow resulting in memory access conflicts and necessary data-swapping instances. The input-output data-flow of other butterfly block sizes can also be under taken in similar manner. Computations of each butterfly stages of 32-point FFT is performed in a set of 4 successive clock cycles and the entire computation is performed in 20 clock cycles. During the first 16 clock cycles of a period of 20 clock cycles, the BCU performs butterfly operations to produce an 8-point intermediate output-data-block which is written back into the SU. During the last 4 clock cycles of a period of 20 clock cycles the FFT components are computed out as output of the BCU while samples of the next input sequence are stored in the SU during the same period.

The SU pertaining to butterfly block size 4 is comprised of 8 memory banks (B1, B2, B3, B4, B5, B6, B7, and B8), so that a block of 8 words can be retrieved/stored from/in the 8 banks in each clock cycle using a common address for reference. The output data-flow from/into the SU for the computations of different stages of two successive 32-point input vectors **x**1 = {*x*(0), *x*(1), · · · , *x*(31)} and **x**2 = {*x*(32), *x*(17), · · · , *x*(63)} is given in Table III for 20 clock cycles.

stage-5 and deliver the output corresponding to the FFT of input sequence **x**1. During the same period, the next input sequence**x**2 is loaded into the storage unit. For in-place FFT computation, the intermediate output values are stored exactly in the same locations from which the corresponding input data were read. Therefore, the intermediate output vectors **r**1, **u**1, **v**1, and **w**1 are stored exactly in the same memory locations of input signals **x**1, **r**1, **u**1, and **v**1, respectively. One can find from Table III that samples of each input data-block of stage-2 and stage-3 computation are sourced from 8 different banks and no conflict arises while reading input data-blocks from different banks. However, during stage-4 of butterfly computation, 8 samples of each input-block are read from 4 banks. For example: sample of first input-block {*v*1(0), *v*1(4), *v*1(2), *v*1(6), *v*1(16), *v*1(24), *v*1(18), *v*1(26)} are read from register-banks, (B1, B3, B5, B7) while samples of the second input-block {*v*1(8), *v*1(12), *v*1(10), *v*1(14), *v*1(20), *v*1(28), *v*1(22), *v*1(30)} are read from register-banks (B2, B4, B6, and B8). Similar situation also arises for other input blocks of stage-4 as well as stage-5 computations. When more than one data are required from a particular bank during a clock cycle then an access conflict occurs. The access conflict can be resolved by swapping samples of each data-block in pairs before/after they are stored/read in/from the register-banks simultaneously. The storage unit uses two data-swapping circuits (DS1 and DS2) to perform the required data-swapping on the intermediate data-blocks produced by the BCU to resolve the bank conflicts and reorder the samples of input-blocks

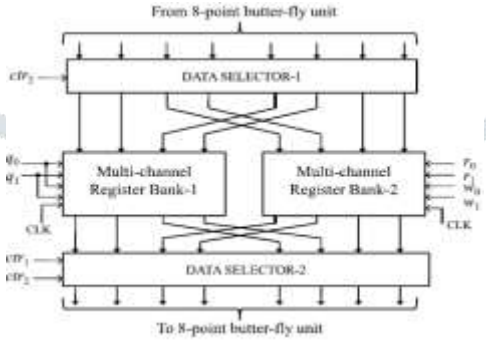


Fig. 3. Storage unit design of existing RFFT structure for butterfly

The data selectors and multi-channel bank are shown in below figure 6 and 7 . This are used to store the internal data and each stage and data selectors are used to shuffle the data between them and inputs scheduling.

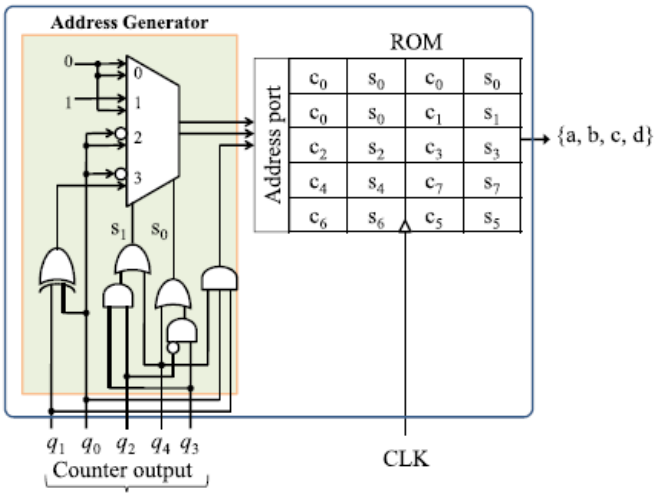


Fig. 4. Structure twiddle factor storage unit for FFT size N = 32 and butterfly block-size 4.

The design of twiddle factor storage unit (TSU) for the 32-point RFFT structure and butterfly block-size 4 based on the approach of in previous papers requires a ROM look-up-table (LUT)of 8-words. However, we find that the TSU stores a few redundant values and that can be avoided to reduce the ROM size. An optimized TSU design for 32-point RFFT with block-size 8 is shown in Fig.4. It involves one ROM LUT of size (5 × 4*w*), where *w* is the word length of real twiddle factor constants {*cl* , *sl* }. The five BF stages of 32-point FFT are encoded by a 3-bit word {*q*4, *q*3, *q*2} to generate the control bits {*s*1, *s*0} of the MUX in order to select the lower 2-bit address of particular BF stage.

III. PROPOSED ARCHITECTURE

In existing work a real-valued fast Fourier transform (RFFT) has designed ,so in this design approach is presented to develop an area-delay and energy-efficient architecture for in-place RFFT. Generally, an in-place fast Fourier transform (FFT) structure consists of a *butterfly block* which performs a set of butterfly operations in every clock cycle. From complexity analysis we find that in-place FFT structures with larger butterfly blocks are more efficient in terms of area-time complexity and energy consumption. Therefore, we have analyzed the data-flow and memory footprint of in-place RFFT architectures for different throughput requirements, and based on that, we have proposed here a strategy to partition the storage unit into several banks of smaller sizes (without increasing the overall memory size) to resolve the memory access conflicts by concurrent data-swapping between the banks. so in this paper they have designed taking from 8 point input FFT unit of which is shown in this paper. we use this design to implement 32 point FFT. By using these manner the no of twiddle factors and design complexity will be little high.

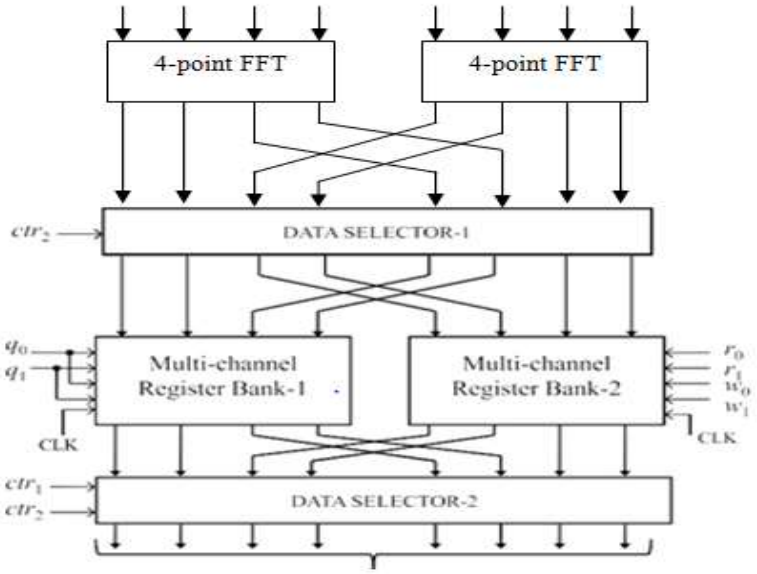


fig 5. Proposed FFT structure for butter fly unit and storage device

In existing method, a design approach is presented to develop an area-delay and energy-efficient architecture for in-place RFFT . so in this design they use 8 point FFT to design the circuit where there is increase the complexity.In proposed method we divided the 8 point FFT into two 4 point FFT multiple delay commutators as shown in the figure 5. The proposed structure consists of simple butterfly structure and less memory requirement which leads to less area and less delay. In this twiddle factor unit a little modification are done in accordance to 4 point FFT ,which will consume less memory requirement.

Storage Unit Design:

The proposed structure for *N*-point in-place DIT RFFT computation is similar to the structure of at the block level. However, the design of SU as well as the BCU are different from those designs. The design of SU of the proposed structure for 32-point RFFT and butterfly block-size 4 is shown in Fig.5. It consists of two multi-channel register-banks (MCRBs) and two data-selectors. The MCRB implements multiple memory banks in one unit which uses a common address decoding logic for all the banks. The MCRB-1 implements register-banks (B1, B2, B5, B6) while MCRB-2 implements register-banks (B3, B4, B7, B8). The internal structure of MCRB for four bank channels of depth 4 each is shown in Fig.6. It consists of 16 registers which are arranged in 4 rows and 4 columns. The four registers of each column form one bank channel, such that registers (R11, R12, R13, R14) form Bank-1 and registers (R41, R42, R43, R44) form Bank-4.

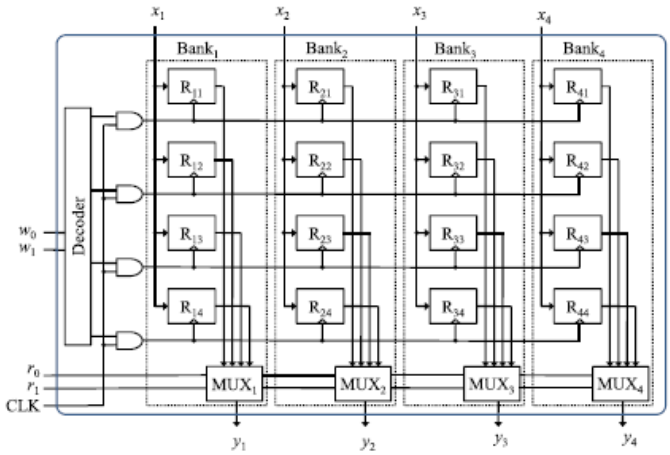


fig 6. Internal structure of multi-channel register bank for butterfly-size 4 and N=32

A 2:4 decoder is used to decode the 2-bit write-address *(w*0,*w*1*)* and enable the clock signal *CLK* for a particular row of registers belonging to four different banks to write one block of input-data {*x*1, *x*2, *x*3, *x*4}. The content of all four registers of each bank are accessed through the multiplexer that selects one of the registers as specified by 2-bit read address (*r*0, *r*1). During each clock cycle, four data values are read from four register-banks of one MCRB such that one block of 8 data are retrieved from MCRB-1 and MCRB-2 to perform the butterfly operation of the *i* -th BF stage and the intermediate outputs are written back into the same locations (registers) during the next clock transition for the *(i* + 1*)*-th stage of butterfly computations (for $1 \le i \le \log 2\ N$).

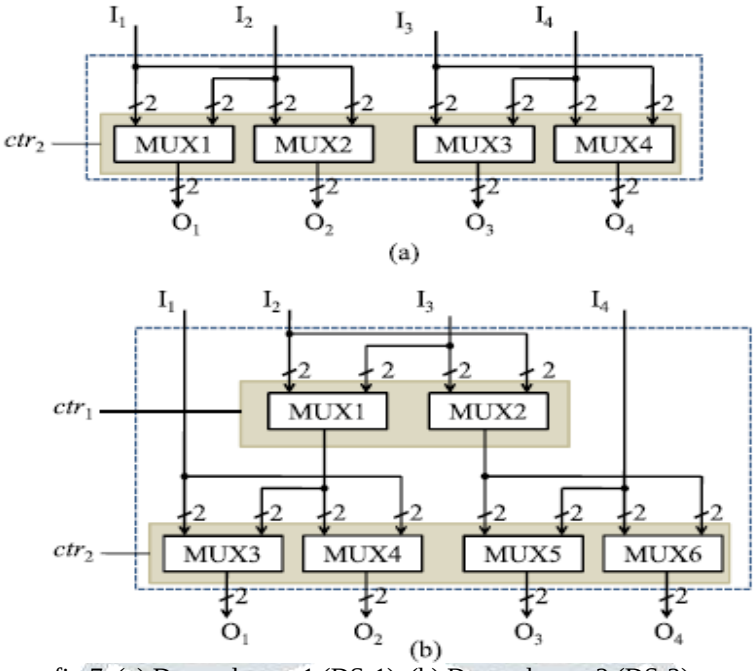


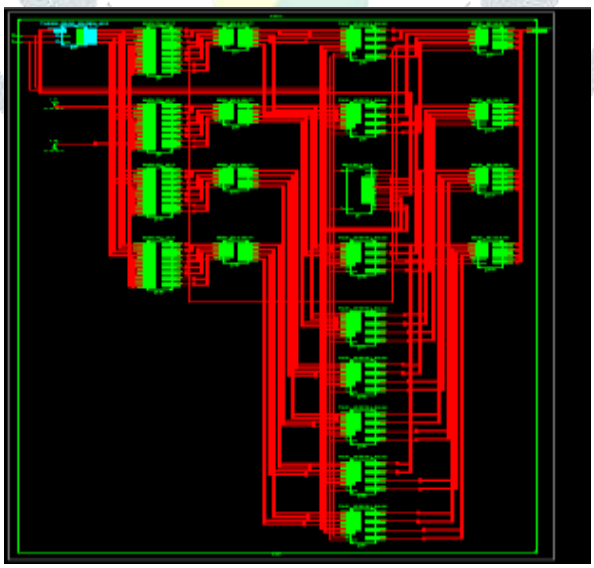
fig 7. (a) Data selector-1 (DS-1). (b) Data selector-2 (DS-2).

The internal structure of DS-1 and DS-2 is shown in Fig.7. Each input-line (I*i*) or output-line (O*i*) of DS-1 and DS-2 receives/sends a pair of data samples from/to the BCU, for *(*$0 \leq i \leq 3$*)*. The data-swapping operation of DS-1 and DS-2 are controlled by the signal *ctr*2. The signal *ctr*1 is set to '1' during swapping operation of DS-1. As shown in Fig.7(b), {MUX1, MUX2} swap the data among them for reordering the input-block retrieved from MCRB while {MUX3, MUX4} and {MUX5, MUX6} swap the data to annul the swapping operation of DS-1. The design of BCU for butterfly block size 4 is identical to arithmetic unit design of [12] except that it comprises of two complex multipliers and two pairs of line-changers {LC1 and LC2}.

IV.RESULTS

The schematic of proposed simple butterfly structure is shown figure RTL Schematic and the simulated waveforms are obtained by assigning input parameters like area and delay. Generated results with parameters area and delay shows that the proposed design unit is proposed in which 8 point butterfly unit is implemented by two 4 point FFT consumes less area and less delay when compare to existing design by using Xilinx ISE 14.7 tool. The outputs obtained are complementary with respect to the corresponding complementary inputs.

RTL Schematic:



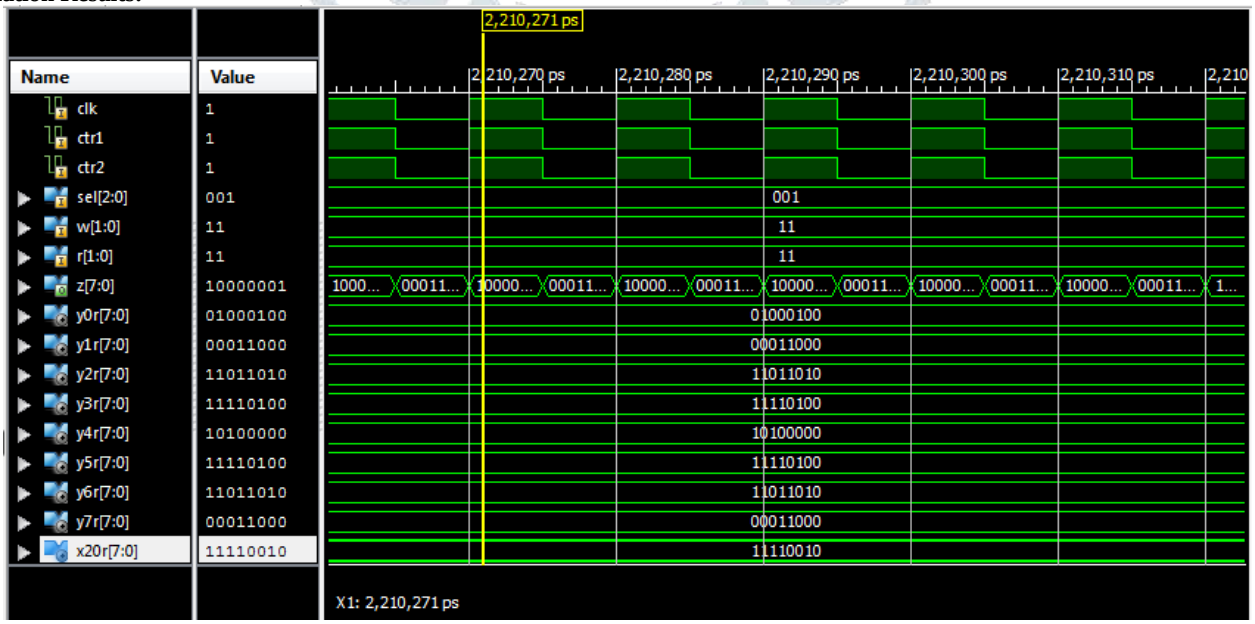
Area:

```
Selected Device : 7v2000tflg1925-2


Slice Logic Utilization:
 Number of Slice Registers:                7  out of  2443200      0%
 Number of Slice LUTs:                  4489  out of  1221600      0%
    Number used as Logic:               4489  out of  1221600      0%
```

Delay:

```
                                        Gate    Net
   Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
   ----------------------------------------  ------------
    FDR:C->Q             9   0.236   0.384  g1/g1/count_0 (g1/g1/count_0)
    INV:I->O             1   0.064   0.339  g1/g1/Mcount_count_xor<0>11_INV_0 (g1/Result<0>)
    FDR:D               -0.000          g1/g1/count_0
   ----------------------------------------
   Total                    1.018ns (0.290ns logic, 0.728ns route)
                                    (28.4% logic, 71.4% route)

--------------------------------------------------------------------------------
iming constraint: Default OFFSET IN BEFORE for Clock 'clk'
 Total number of paths / destination ports: 7 / 7
--------------------------------------------------------------------------------
Offset:              0.807ns (Levels of Logic = 1)
 Source:             rst (PAD)
 Destination:        g2/g8/out_55 (FF)
 Destination Clock: clk rising

 Data Path: rst to g2/g8/out_55
                                        Gate    Net
   Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
   ----------------------------------------  ------------
    IBUF:I->O           428   0.000   0.543  rst_IBUF (rst_IBUF)
    FDR:R                     0.264          g2/g8/out_55
   ----------------------------------------
   Total                     0.807ns (0.264ns logic, 0.543ns route)
Timing constraint: Default OFFSET OUT AFTER for Clock 'clk'
  Total number of paths / destination ports: 13036177933284 / 848
--------------------------------------------------------------------------------
Offset:              16.087ns (Levels of Logic = 70)
  Source:            g2/g8/out_55 (FF)
  Destination:       Out<952> (PAD)
  Source Clock:      clk rising

  Timing constraint: Default path analysis
    Total number of paths / destination ports: 928 / 896
  --------------------------------------------------------------------------------
  Delay:               1.713ns (Levels of Logic = 4)
    Source:            rst (PAD)
    Destination:       Out<952> (PAD)

    Data Path: rst to Out<952>
                                          Gate     Net
      Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
      ----------------------------------------  ------------
       IBUF:I->O           428   0.000   0.609  rst_IBUF (rst_IBUF)
       LUT3:I1->O           12   0.043   0.674  Out<120>21 (Out<120>2)
       LUT6:I0->O            2   0.043   0.344  Out<88>1 (Out_88_OBUF)
       OBUF:I->O                 0.000          Out_600_OBUF (Out<600>)
      ----------------------------------------
      Total                     1.713ns (0.086ns logic, 1.627ns route)
                                        (5.0% logic, 95.0% route)
```

Simulation Results:



V.CONCLUSION

We have proposed a design approach to develop efficient architecture for in-place RFFT which could be scaled for higher throughput and larger FFT sizes. In existing design the 8 point butterfly unit is proposed and given as input to storage device to perform storing and data swapping operations which complexity is more. In proposed design a novel 8 point FFT architecture implemented by using two 4 point FFT and design is implemented. The above results shows the proposed design consumes less area and less delay when compare to existing design. The synthesis and simulation results have been verified by using Xilinx ISE 14.7 tool.

VI.REFERENCES

[1] E. H. Wold and A. M. Despain, "Pipeline and parallel-pipeline FFT processors for VLSI implementations," *IEEE Trans. Comput.*, vol. C-33, no. 5, pp. 414–426, May 1984.

[2] Y.-N. Chang and K. K. Parhi, "An efficient pipelined FFT architecture," *IEEE Trans. Circuits Syst.II, Analog Digit. Signal Process.*, vol. 50, no. 6, pp. 322–325, Jun. 2003.

[3] H. Sorensen, D. Jones, M. Heideman, and C. Burrus, "Real-valued fast Fourier transform algorithms," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. ASSP-35, no. 6, pp. 849–863, Jun. 1987.

[4] S.-F. Hsiao and W.-R. Shiue, "Design of low-cost and high-throughput linear arrays for DFT computations: Algorithms, architectures, and implementations," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 47, no. 11, pp. 1188–1203, Nov. 2000.

[5] M. Garrido, K. K. Parhi, and J. Grajal, "A pipelined FFT architecture for real-valued signals," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 56, no. 12, pp. 2634–2643, Dec. 2009.

[6] M. Ayinala, M. Brown, and K. K. Parhi, "Pipelined parallel FFT architectures via folding transformation," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 6, pp. 1068–1081, Jun. 2012.

[7] A.Wang and A. P. Chandrakasan, "Energy-aware architectures for a real valued FFT implementation," in *Proc. Int. Symp. Low Power Electron. Design*, Aug. 2003, pp. 360–365.

[8] H.-F. Chi and Z.-H. Lai, "A cost-effective memory-based real-valued FFT and Hermitian symmetric IFFT processor for DMT-based wire-line transmission systems," in *Proc. IEEE Int. Symp. Circuits Syst.*, vol. 6, May 2005, pp. 6006–6009.

[9] L. G. Johnson, "Conflict free memory addressing for dedicated FFT hardware," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 39, no. 5, pp. 312–316, May 1992.

[10] M. Ayinala, Y. Lao, and K. K. Parhi, "An in-place FFT architecture for real-valued signals," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 60, no. 10, pp. 652–656, Oct. 2013.