IOSR Journal of VLSI and Signal Processing (IOSR-JVSP)
Volume 7, Issue 4, Ver. II (Jul. - Aug. 2017), PP 07-18
e-ISSN: 2319 – 4200, p-ISSN No. : 2319 – 4197
www.iosrjournals.org

A High Performance Reconfigurable Data Path Architecture For Flexible Accelerator

[1]D. Naga Divya, [2]M.V. Ganeswara Rao, [3]Rajesh k Panakala, [4]A.M. Prasad
[1](Department of Electronics and Communication Engineering, Shri Vishnu Engineering College for Women (Autonomous), Bhimavaram, INDIA)
[2](Department of Electronics and Communication Engineering, Shri Vishnu Engineering College for Women (Autonomous), Bhimavaram, INDIA)
[3] (Department of Electronics and Communication Engineering, PVPSIT College Vijayawada, INDIA)
[4] (Department of Electronics and Communication Engineering, JNTU College for Engineering Kakinada, INDIA)
Corresponding Author: * D. Naga Divya

***Abstract**: Hardware acceleration in digital signal processing (DSP) domain proved as the best implementation strategy. Overall performance of DSP processor accelerates by using the hardware module named as DSP accelerator by performing certain functions in the accelerator. In some areas such as video processing, flexible DSP accelerator is used to do video encoding and decoding flexibly. The architecture of data path impacts the efficiency of the accelerator. So there is a need to implement flexible data path architecture using Flexible Computational Unit (FCU). This paper solves the problem of developing high-speed and area efficient data path architecture for flexible accelerator, where there is a need to increase computational speed as well as reducing the area to attain efficient architecture. The proposed architecture is compared with that of FCU implemented with Carry Save Adder (CSA) and Modified carry save adder using xor gates in terms of area and delay. The proposed architecture FCU with Modified carry save adder using xor-xnor gates have a better area and delay than FCU with CSA by 2.8% and 7.9% and is also better than FCU with modified carry save adder using xor gates by 0.1% and 0.5%.*

***Keywords:** Flexible Data Path, Flexible Computational unit (FCU), Digital Signal Processor, Carry Save Arithmetic.*

Date of Submission: 02-08-2017 Date of acceptance: 14-08-2017

I. INTRODUCTION

The tremendous development of embedded systems and multimedia increases the demand for Digital Signal Processing [1]. Hardware accelerator invention for DSP systems lead to changes in the digital world. DSP accelerator accelerates the performance of digital signal processor and this digital signal processing algorithm produced delays that affect the computer performance. In embedded systems, some areas such as video processing and communication used this DSP accelerator to reduce power consumption and improve the overall performance. Although, Application specific integrated circuits (ASICs) prove as ideal acceleration in terms of area, due to ASICs inflexibility several ASICs are needed to accelerate various DSP kernels. To overcome this problem, there is need to implement the flexible data path architecture by using operation templates [2] to attain flexible accelerator.

Several researchers have proposed different types of VLSI architectures for the implementation of high performance and area efficient data path architectures. A high performance data path is used to implement digital signal processing kernels. This data path based on Flexible Computational Component (FCC) [3], which is flexible and implements 2x2 template of primitive resources. In this by using a number of FCC's performances of the data path is improved. But this architecture takes more area and time due to chaining operations. In 2009 S. Xydis et al has proposed coarse-grained reconfigurable architecture [4] to introduce flexibility into custom data path architecture by using the canonical interconnection scheme. The canonical interconnection scheme is realized by a transformation, known as uniformity transformation depends on carry save multipliers and carry save chain adders or subtractors. In this the data path architecture based on Reconfigurable Arithmetic units (RAUs) it consists of Reconfigurable Array of UCs. The RAU implements with an array of UCs in canonical form and it introduces flexibility in the data path. The unified cells used in this design require double area and more time.

In 2011 S. Xydis et al has proposed high performance data path based on Flexible Pipeline Stages (FPS) [5]. In this, the data path makes use of horizontal parallelism and vertical parallelism. FPS increases the performance and flexibility of the data path. The chained computational components which are used in this architecture acts as a slow component it led to decrease in computational speed. The architectures in [4] and [5] are more suitable for high performance and flexible data path, but that architectures are not applicable to data paths because of their inefficiency in terms of area and delay. So, there is need for reducing the area and increase the computational speed. This paper solves the problem of reducing the area while increasing the computational speed by implementing an efficient VLSI architecture for flexible accelerator.

In this paper, implementation of flexible data path using Flexible Computational Unit (FCU) for flexible accelerator is presented.

The rest of the paper is structured as follows: In section II, Flexible data path architecture is presented. The following section explains simulation results. In section IV, performance comparisons with other architectures are given. The final conclusion of this paper is shown in section V.

II. VLSI ARCHITECTURES FOR FLEXIBLE ACCELERATOR

2.1 Reconfigurable Flexible Data Path Architecture

The architecture of flexible data path [6] for flexible accelerator is shown in the Fig. 1. In this architecture the main blocks are one Control Unit (CU), one Register Bank, Interconnection Network, which internally consists of 3 Multiplexers and Flexible Computational Unit (FCU).

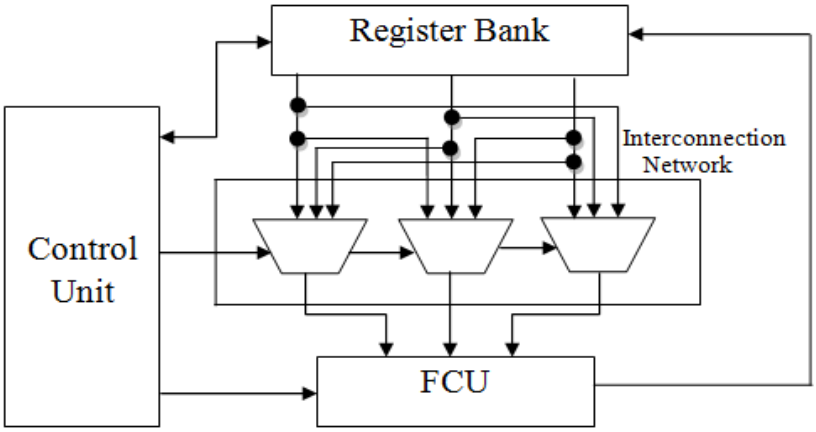

Fig.1. Block diagram of reconfigurable Flexible Data Path.

The Control Unit (CU) is used to provide the control signals to Register Bank and selection signals to interconnection network. It also provides a configuration word to the Flexible Computational Unit (FCU). Control Unit can be divided as a communication control unit and a data path control unit. The communication control unit used to load the data into the register bank. The data path control is used to write the data into register, which is received from FCU and the data path control gives selection signals to multiplexers as well as configuration words to the FCU. Register is mainly used to store the results received from FCU and it takes the signals from the control unit. Depend upon input occurred from the control unit either register unit store the values or pass the input to FCU through interconnection network. Interconnection network is used for communication between the FCU and register bank. The multiplexers in interconnection network take the input from register bank depends upon the selection line occurred from control unit it takes the one of the inputs and execute as output. In this way three multiplexers executes outputs. That outputs from interconnection network are given as input to the FCU. This data path architecture is reconfigurable, so number of FCU's can be changed depending upon the demands made by the designer and if there is a need to implement 32 bit operands then also number of FCU's increased because each FCU supports only 16 bit operands.

2.2 Architecture of Flexible Computational Unit

Architecture of Flexible Computational Unit (FCU) used in the flexible data path shown in Fig.2. It consists of Modified4:2 Carry Save Adder, four multiplexers name as MUX0, MUX1, MUX2, MUX3, CS to MB recoding technique, Partial product generator, carry save adder Wallace tree and carry propagate adder. In this, the FCU can be arranged in template form which is selected from the template library. Template library comprises with different types of operational templates. A template is a combination of different modules like

adders, subtractors and multipliers. FCU support 16 bit operands it is much suitable for implementation of data path architecture. By using FCU, flexible data path performance can be increased. The outputs of interconnection network are 32 bits when that outputs are given as inputs to FCU that each 32 bit operand divided into two 16 bit operands. Consider the output of interconnection network is X*, Y*, K*. X* and Y* is partitioned into X_1, X_2 and Y_1, Y_2. Configurations words received from control unit acts as carry-in for adders and selection lines for multiplexers. Y_1, Y_2 inputs are given to MUX0, if CL_0=0 the output of MUX0 is Y_1, Y_2. If CL_0=1, MUX0 output is 2's complement of Y_1, Y_2.

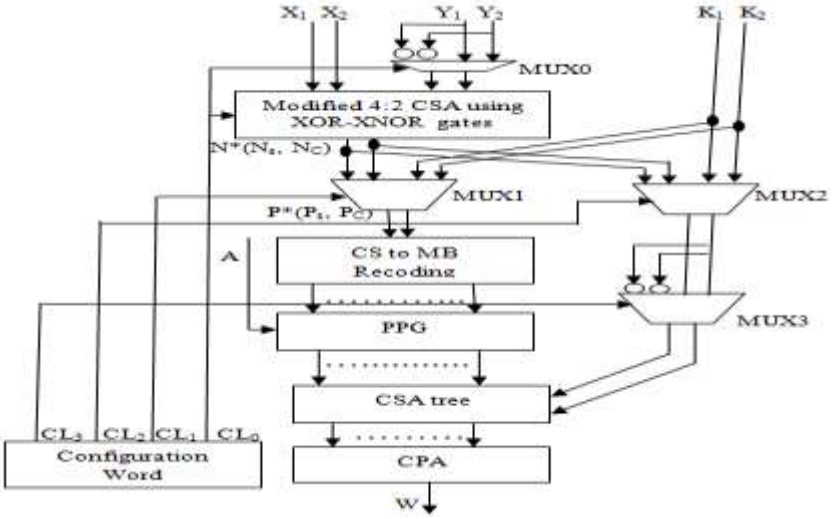

Fig.2. Block diagram of Flexible Computational Unit (FCU)

2.2.1 Modified 4:2 carry save adder

Modified 4:2 carry save adder [7] architecture shown in Fig.3.It acts like a parallel adder. This adder used to perform parallel addition operations without relying on previous columns. 4:2 carry save adder means it takes the 4 inputs and compressed into 2 outputs. Modified 4:2 Carry Save Adder implemented by using xor gates, multiplexers and xor-xnor gates. In this MUX* has used, which is different from normal multiplexer, it gives two outputs instead of one output. Based on selection line MUX* generates one output and other output is complement of first output. XOR - XNOR gates implemented in same block to reduce delay. In FCU, this adder is used to add four operands X_1, X_2, Y_1, Y_2 and it compressed into two outputs N*{N_s, N_c}. The adder output is N*= X*+Y* if carry-in of adder CL_0 =0. If CL_0=1 adder executes N*= X*-Y* (X*={X_1, X_2}, Y*= {Y_1, Y_2}). This output gives as input to MUX1 and MUX2 along with operands K_1, K_2.

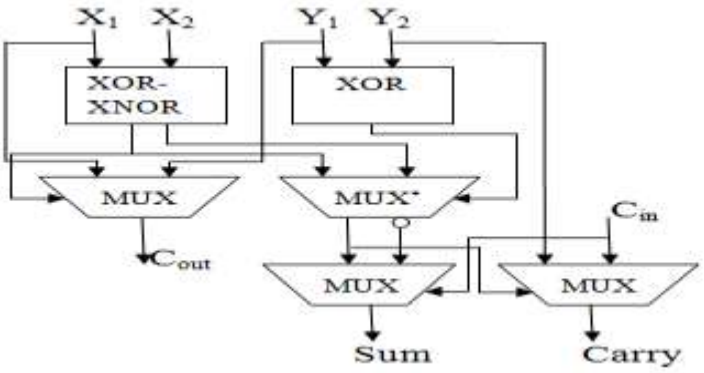

Fig.3. Modified Carry Save Adder using XOR- XNOR gates

2.2.2 Multiplexers

Multiplexers are used to select appropriate data by using selection line. CL_1 and CL_2 act as a selection line for MUX1 and MUX2 respectively. If CL_1=0 and CL_2=0, MUX1 and MUX2 output is N*{N_s, N_c}.MUX1 and MUX2 produce K* {K_1, K_2} as output, if CL_1=1 and CL_2=1. The output of MUX1 is given as input to CS-MB recoding technique. MUX3 accepts the MUX2 output as input and if CL_3=0, output of MUX3 is same as input. If CL_3=1, MUX3 complements the input and produce as output.

2.2.3 CS-MB Recoding technique

CS-MB recoding technique is used to covert carry save form of data into modified booth form and this technique is partitioned into two blocks. One block is CS-MB recoder it is used to recode the data is shown in Fig.4. The outputs obtained from carry save adder are in carry save form is not suitable for multiplication. So, it needs to convert carry save form data into modified booth form. Modified booth algorithm [8] is a prevalent form for multiplication and it also decreases the partial products when compared to normal multiplication. CS-MB recoder consists of FA, FA* and FA**. FA* is different from FA its one of the outputs sum is complemented where as one of the output carry is complemented in FA**. Actually the recoder technique also performs addition operation and place 0 as most significant bit in the output and then makes every three bits as a group. But this grouping can be done by overlapping the last bit in the previous group. The least significant bit of one group became as most significant bit for another group. Another block is Modified Booth encoder used to encode the inputs that accept from CS-MB recoder and multiply with multiplicand A.

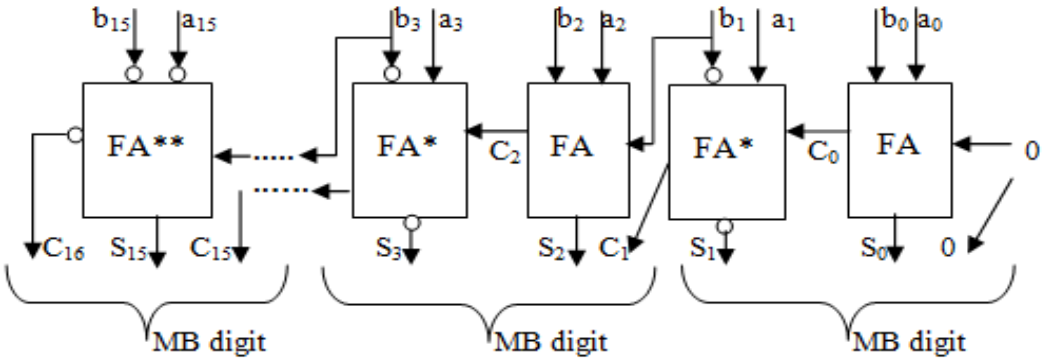

Fig.4. CS - MB recoder

2.2.4 Partial Product Generator (PPG)

Partial product generator produces partial products by multiplying the output of CS-MB recoding technique with multiplicand A. Based on the multiplicand, the number of partial products can be occurred. Each row of partial products is obtained by using complement, add and shift methods. All those partial products are added by using a CSA tree.

2.2.5 CSA tree

CSA tree is used as a Wallace tree. It is used for summing all the outputs obtained from the partial product generator and also it adds the MUX3 output with these partial products. It gives the output sum and carry separately. That sum and carry added by a carry propagate adder. Because of carry save adder tree the speed of FCU architecture increases.

2.2.6 Carry Propagate Adder (CPA)

Carry propagate adder acts as a final block in this architecture. Ripple Carry Adder (RCA) is used in this carry propagate adder. RCA is used for summing the outputs occurred from the CSA tree with carry-in and produce the sum and carry. It is same as full adder. When more than one bit is adding carry can be propagated to next column. Finally the FCU output W was produced.

In this FCU for different configurations, various operations can be done as shown in table 1. The final output of the FCU is stored in the register bank. If required again the stored values can be used as input for FCU or it can be stored in the register bank. By using carry save adder with xor-xnor gates and modified booth multiplier the computational speed of flexible data path increases, which led to flexible and high performance data path.

Table.1. FCU Operations

CONFIGURATION	OPERATION
0000	$(X^*+Y^*)*A+(X^*+Y^*)$
0001	$(X^*-Y^*)*A+(X^*-Y^*)$
0010	$K^**A+(X^*+Y^*)$

0011	$K^{*}*A+(X^{*}-Y^{*})$
0100	$(X^{*}+Y^{*})*A+ K^{*}$
0101	$(X^{*}-Y^{*})*A+ K^{*}$
0110	$K^{*}*A+ K^{*}$
0111	$K^{*}*A+ K^{*}$
1000	$(X^{*}+Y^{*})*A-(X^{*}+Y^{*})$
1001	$(X^{*}-Y^{*})*A-(X^{*}-Y^{*})$
1010	$K^{*}*A-(X^{*}+Y^{*})$
1011	$K^{*}*A-(X^{*}-Y^{*})$
1100	$(X^{*}+Y^{*})*A- K^{*}$
1101	$(X^{*}-Y^{*})*A- K^{*}$
1110	$K^{*}*A- K^{*}$
1111	$K^{*}*A- K^{*}$

III. Simulation Results

The hardware architectures for FCU and a flexible data path has been designed. The programming language used in this is Verilog HDL and simulated using Xilinx ISE 14.5 and ISIM simulator. Design properties are Spartan 3E family, FG320 package, XC3S500E device with a speed grade -5.

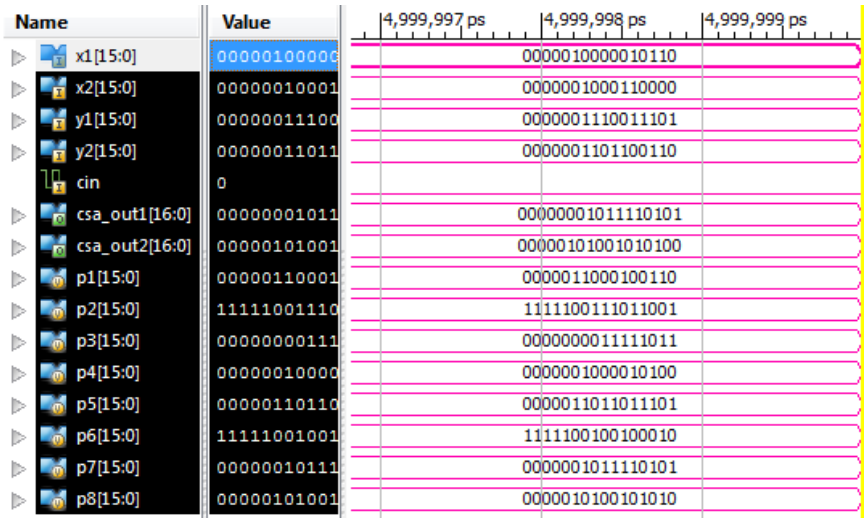

Fig.5. Simulation result of Modified 4:2 CSA using xor- xnor gates

In the FCU, the first block is MUX0 it takes the input Y_1, Y_2 each input consists of 16 bits and depend upon selected line it gives the same input as output or it may be complimented the input and execute as output. The second block is modified 4:2 CSA using xor-xnor gates. Inputs are X_1, X_2 with 16 bit length and Y_1, Y_2 that obtained from MUX0. These 4 inputs with another input C_{in} are given to carry save adder. Summation of X_1, X_2, Y_1, Y_2 executes the results as csa_out1 and csa_out2 with 17 bit length of each output as shown in Fig.5. Here p1, p2, p3, p4, p5, p6, p7, p8 acts as wires in this adder to share the data among xor, xor-xnor gates and multiplexers.

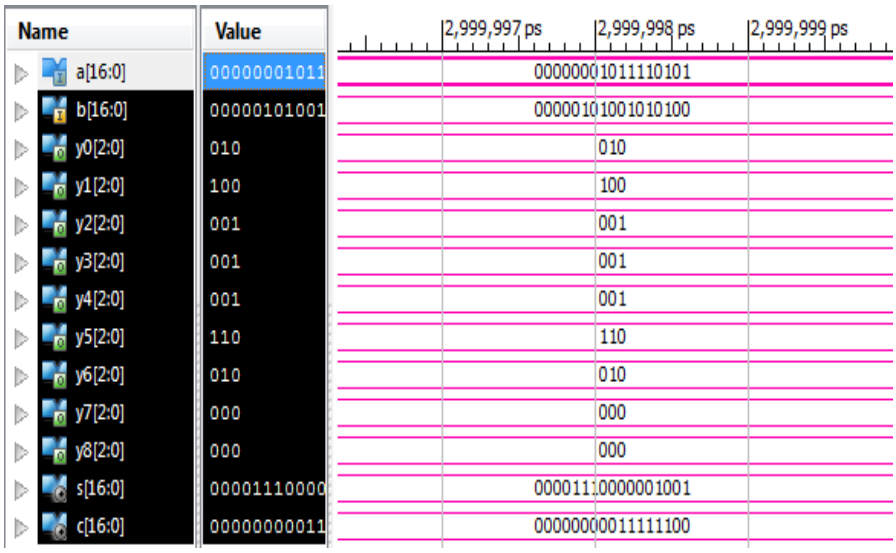

Fig.6. Simulation result of CS-MB recoder

The outputs of adder are given as input to CS-MB recoder. Simulation result of CS- MB recoder shown in Fig.6. First, two inputs are added and produce the output is 17 bits name as S. Place the 0 in the right most significant bit position of output and with that 0 divide each 3 bits as a group of overlapping the last bit from the previous group, it means last bit in previous group became as the first bit in present group and name each group as y0, y1, y2, y3, y4, y5, y6, y7, y8. 17 bits are divided into 9 groups by overlapping the bit from previous group.

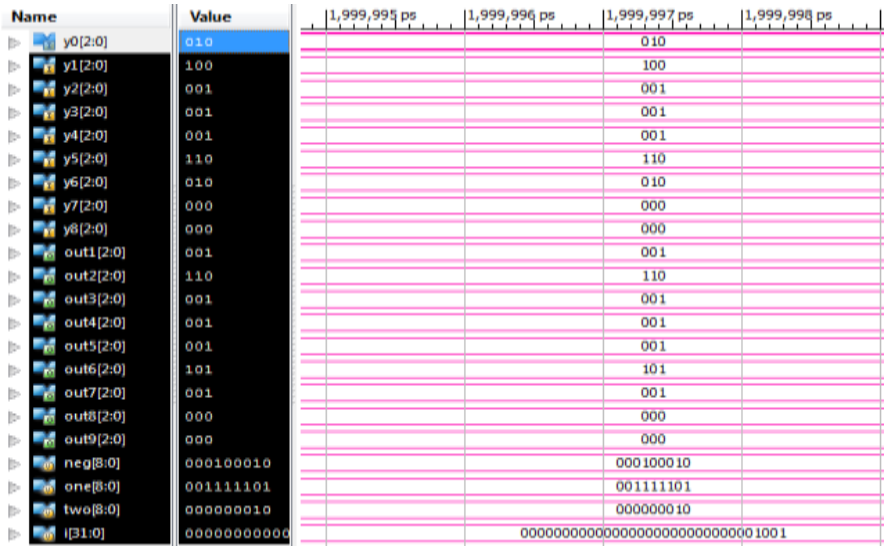

Fig.7. Simulation result of MB encoder

In previous block carry save form data is converted into a modified booth form. But after converting data, it should be encoded by using a Modified Booth encoder. Simulation result of MB encoder shown in Fig.7. It takes the 9 inputs from previous block for each input one output is executed. Based on the inputs encoder executes the output. If the given input is 000 or 111 output is 000, if input is 001 or 010 output is 001 and for 011 the output is 010, for 100 it executes 110 and finally if input is 101 or 110 then the output is 101. Every input consists of 3 bits so 9 combinations are possible in this encoder.

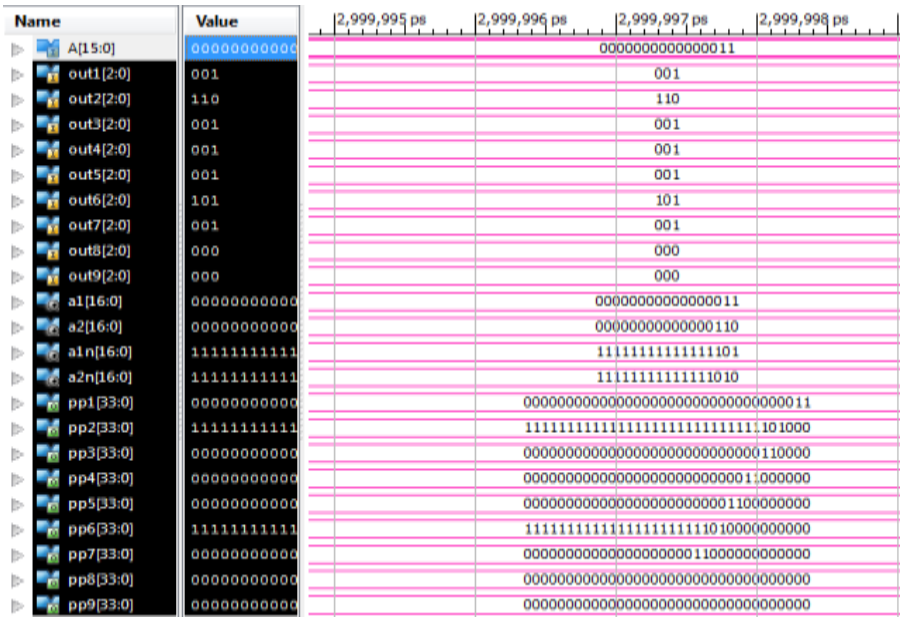

Fig.8. Simulation result of partial Product Generator

Simulation result of the partial product generator shown in Fig.8. The partial product generator takes 9 inputs, which are obtained from MB encoder along with multiplicand A. The 9 inputs bit length is 3 where as multiplicand bit length is 16. Here a1, a2, a1n and a2n act as wires for connection between different blocks present in the partial product generator. By multiplying these 9 inputs with multiplicand, it produces 9 partial products with 34 bit length. If the input is 000 then the output of partial products replaces all the 34 bits with 0. If the input is 001 then it executes 16 bits same as multiplicand remaining bits are filled with 0 and if the input is 010, then left shift the multiplicand one position and place that value as output remaining bits are 0's. If the input is 101 then 2's compliment the multiplicand then execute as output and finally the input is 110 then 2's compliment the multiplicand then shift left and produce as output in these two cases remaining bits are placed with 1's.

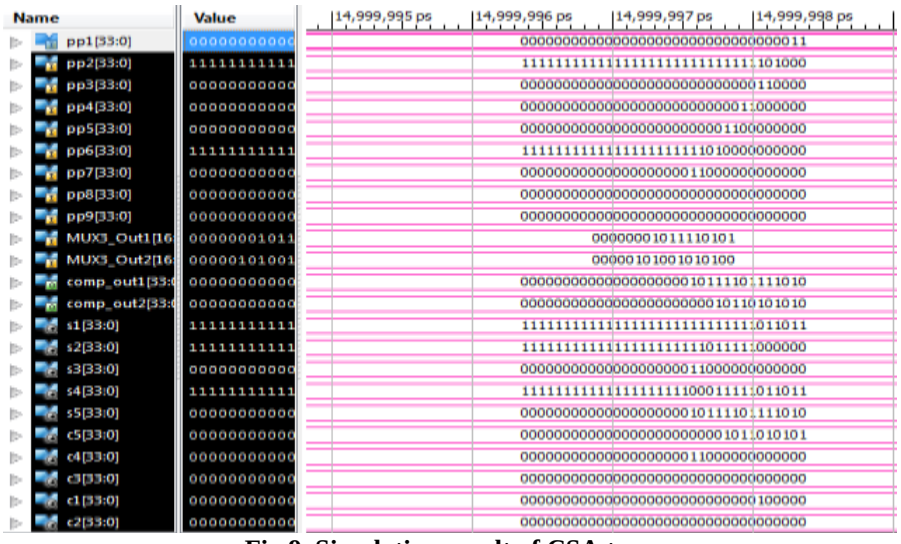

Fig.9. Simulation result of CSA tree

All the 9 partial products obtained from the partial product generator along with other operands occurred from MUX3 are added in the CSA tree. Simulation result of CSA tree shown in Fig.9. This CSA tree consists of full adders where it's taken three inputs and executed two outputs, sum and carry. In this way all the inputs are added by using wires s1, s2, s3, s4, s5, c1, c2, c3, c4, c5 and finally it executes two outputs comp_out1 and comp_out2 and its bit length is 34 bits.

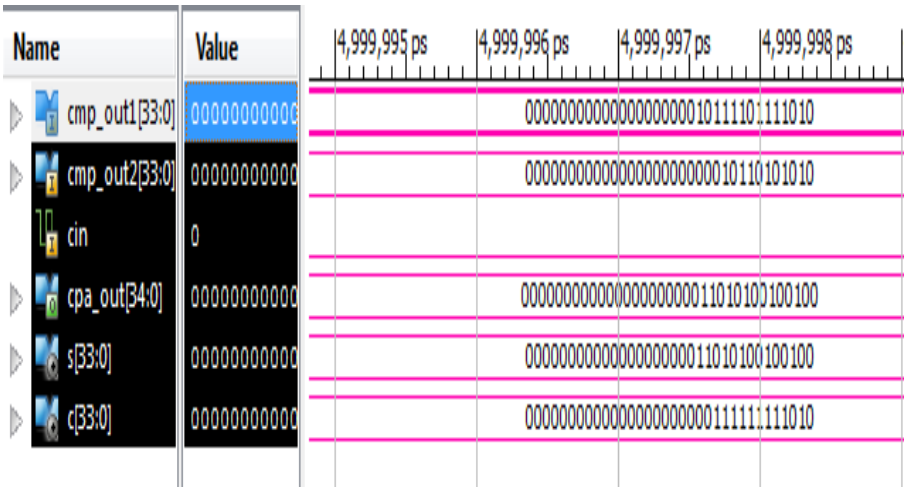

Fig.10. Simulation result of CPA

The carry propagate adder is a final block in the FCU. Simulation result of CPA shown in Fig.10. CPA uses Ripple Carry Adder (RCA) in this work. It takes the two inputs from CSA tree with carry in and RCA summation three inputs and executes the final output cpa_out. Its bit length is 35 bits.

Fig.11. Simulation result of FCU

Simulation result of FCU shown in Fig.11. All the simulation results discussed above are part of this flexible computational unit. So, all the individual operations are combined in this and form FCU. In this adder takes four inputs X_1, X_2, Y_1, Y_2 of bit length 16 and produce output by summation of these inputs. These outputs are given as input to CS - MB recoding technique. In this two inputs are added and partitioned into 9 outputs. Partial product generator receives input from CS – MB technique and executes 9 partial products. All the partial products along with two inputs occurred from MUX3 is added by using carry save adder tree, which acts as Wallace tree. The CSA tree gives two outputs and these two outputs are added by using Ripple Carry Adder (RCA) that used as carry propagate adder. This adder gives the final output of 35 bits. As seen in the fig. 11 for different selected lines various operations can be done and depend upon the operations different operations occurred.

This FCU has designed to obtain flexible accelerator by placing FCU in the flexible data path. Simulation result of the flexible data path as shown in Fig.12. For flexible data path operand A with 16 bits and operand X, Y, K with 32 bits take an inputs along with along with clk, reset, sel and ld. All the inputs are applied to control unit if rst is 1 then no operation can be done the output became 0. If the rst is 0 and ld =1 then control unit load the inputs into a register bank with control signal cs=1. When cs=1, register bank passes the outputs to FCU, through interconnection network. Else, register bank stores the values in it. Control unit also gives control signals as selection lines to the multiplexers in interconnection network. Based on the selection line multiplexers selects the one of the inputs and give as input to the FCU. The interconnection network outputs given as input to FCU then each 32 bit operands divided as two 16 bit operands. FCU receives configuration words from the control unit.

Fig.12. Simulation result of Flexible Data Path

In FCU, these configuration word act as carry-in for adders and selected lines for multiplexers and same the FCU operation discussed above can be done in this. Finally, the output which occurred with 35 bit length can stored in the register bank.

IV. COMPARISONS

In section III simulation results are presented. Comparisons of the FCU with different Carry Save Adders like 4:2 Carry Save Adder (CSA), modified 4:2 CSA using xor gates and modified 4:2 CSA using xor-xnor gates are discussed in this section. For synthesis results, device properties Spartan 3E family, FG320 package, XC3S500E device with a speed grade of -5 is used.

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of 4 input LUTs	951	9,312	10%	
Number of occupied Slices	537	4,656	11%	
Number of Slices containing only related logic	537	537	100%	
Number of Slices containing unrelated logic	0	537	0%	
Total Number of 4 input LUTs	959	9,312	10%	
Number used as logic	951			
Number used as a route-thru	8			
Number of bonded IOBs	151	232	65%	
Average Fanout of Non-Clock Nets	3.46			

Fig.13. Area report of FCU implemented with 4:2 CSA

```
Timing Summary:
---------------
Speed Grade: -5

   Minimum period: No path found
   Minimum input arrival time before clock: No path found
   Maximum output required time after clock: No path found
   Maximum combinational path delay: 57.478ns
```

Fig.14. Delay report of FCU implemented with 4:2 CSA

Area and delay reports of the FCU implemented with 4:2 CSA shown in Fig.13 and Fig.14. From device utilization summary, it is noticed that, 4,656 slices are available in this device, but only 537 slices are used in this design and number of 4 input LUTs available are 9,312 but this design utilizes only 959 4 input LUTS and delay observed in this method is 57.478ns.

Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of 4 input LUTs	930	9,312	9%	
Number of occupied Slices	520	4,656	11%	
Number of Slices containing only related logic	520	520	100%	
Number of Slices containing unrelated logic	0	520	0%	
Total Number of 4 input LUTs	938	9,312	10%	
Number used as logic	930			
Number used as a route-thru	8			
Number of bonded IOBs	151	232	65%	
Average Fanout of Non-Clock Nets	3.54			

Fig.15. Area report of FCU implemented with Modified 4:2 CSA using xor gates

```
Timing Summary:
---------------
Speed Grade: -5

   Minimum period: No path found
   Minimum input arrival time before clock: No path found
   Maximum output required time after clock: No path found
   Maximum combinational path delay: 51.720ns
```

Fig.16. Delay report of FCU implemented with Modified 4:2 CSA using xor gates.

Area and delay report of the FCU implemented with Modified 4:2 CSA using xor gates shown in Fig.15 and Fig.16. From device utilization summary, it is noticed that, 4,656 slices are available in this device, but only 520 slices are used in this design and number of 4 input LUTs available are 9,312 but this design utilizes only 938 4 input LUTS and the delay observed in this method is 51.720ns.

Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of 4 input LUTs	929	9,312	9%	
Number of occupied Slices	519	4,656	11%	
Number of Slices containing only related logic	519	519	100%	
Number of Slices containing unrelated logic	0	519	0%	
Total Number of 4 input LUTs	937	9,312	10%	
Number used as logic	929			
Number used as a route-thru	8			
Number of bonded IOBs	151	232	65%	
Average Fanout of Non-Clock Nets	3.56			

Fig.17. Area report of FCU implemented with Modified 4:2 CSA using xor-xnor gates

```
Timing Summary:
---------------
Speed Grade: -5

   Minimum period: No path found
   Minimum input arrival time before clock: No path found
   Maximum output required time after clock: No path found
   Maximum combinational path delay: 51.279ns
```

Fig.18. Delay report of the FCU implemented with Modified 4:2 CSA using xor-xnor gates

Area and delay report of the FCU implemented with Modified 4:2 CSA using xor-xnor gates shown in Fig.17 and Fig.18. From device utilization summary, it is noticed that 4,656 slices are available in this device, but only 519 slices are used in this design and number of 4 input LUTs available are 9,312 but this design utilizes only 937 4 input LUTs and the delay observed in this method is 51.279ns. This shows that FCU implemented with Modified 4:2 CSA using xor-xnor gates proved as best implementation strategy for flexible data path in terms of area and delay.

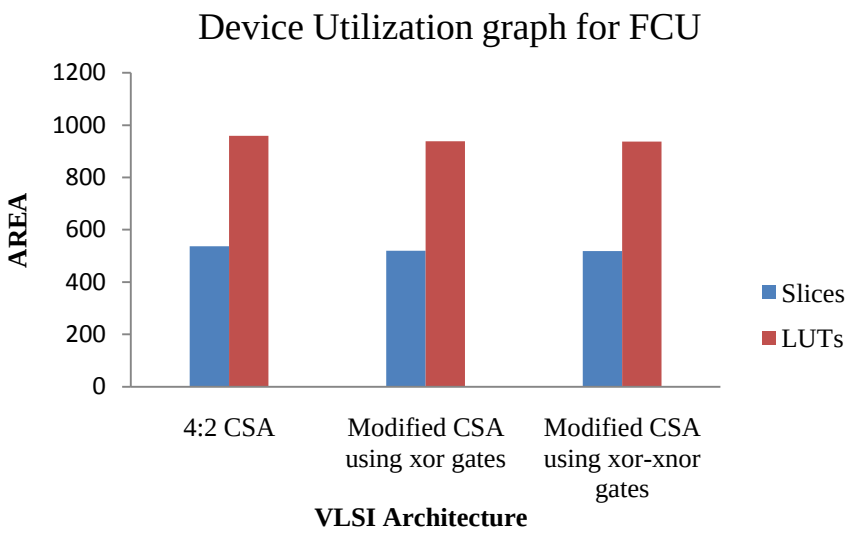

Fig.19. Comparison of area for VLSI architectures of FCU using different adders

By using device utilization summary, area utilized for the FCU implemented with 3 different adders is noticed and to show the comparisons clearly among the FCU's implemented with 3 adders like 4:2 CSA, modified 4:2 CSA using xor gates and modified 4:2 CSA using xor-xnor gates area drawn in graph form as shown in Fig.19. Modified 4:2 CSA using xor-xnor gates occupied less area than other two adders.

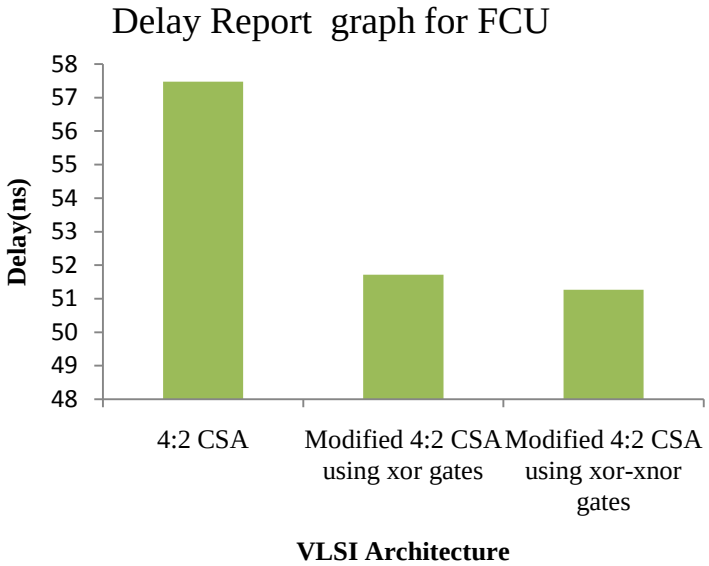

Fig.20. Comparison of delay for VLSI architectures of FCU using different adders

Comparison of delay for VLSI architecture of FCU with different adders like 4:2 CSA, modified 4:2 CSA using xor gates and modified 4:2 CSA using xor-xnor gates drawn in graph form as shown in Fig.20. Modified 4:2 CSA using xor-xnor gates has less delay when compared with two other adders.

Finally, from the above comparisons it is clear that FCU implemented with Modified 4:2 CSA using xor-xnor gates has less area and less delay when compared with FCU implemented with 4:2 CSA and FCU implemented with modified 4:2 CSA using xor gates.

V. CONCLUSION

In this paper, area efficient and high computational speed flexible data path is implemented by using FCU architecture for flexible accelerator. The proposed VLSI architecture of the FCU implemented with modified 4:2 CSA using xor-xnor gates compared with FCU architecture implemented with 4:2 CSA and modified 4:2 CSA using xor gates. The parameters consider for implemented architectures are area and delay.

The proposed FCU architecture has a less area and less delay than FCU implemented with 4:2 CSA by 2.8% and 7.9% and FCU implemented with modified 4:2 CSA using xor gates by 0.1% and 0.5%.

ACKNOWLEDGEMENTS

The authors would like to thank the our beloved Principal Dr. G. Srinivasa Rao and Vice Principal Dr. P. Srinivasa Raju of Shri Vishnu Engineering College for Women for guidance and encouragement to do research by providing facilities and also like to acknowledge the anonymous reviewers for their suggestions that helps to improve the presentation of this paper.

REFERENCES

[1]. J.G. Proakis and D.G. Manolakis, *Digital Signal Processing*: *Principles, Algorithms and Applications*. Upper Saddle River, NJ, Upper Saddle River, NJ, USA:Prentice-Hall, 1996.

[2]. Andrea Lodi, Mario Toma, Fabio Campi, Andrea Cappelli, Roberto Canegallo, and Roberto Guerrieri, "A VLIW Processor with Reconfigurable Instruction Set for Embedded Applications," Ieee Journal of Solid- state Circuits, Vol. 38, No. 11, November 2003.

[3]. M.D. Galanis, G. Theodoridis, S. Tragoudas, and C.E. Goutis, "A high performance data path for synthesizing DSP Kernels, *" IEEE Trans. Comput- Aided Design Integr. Circuit Syst*, vol. 25, no.6, pp.1154-1162, June 2006.

[4]. S. Xydis, G. Economakos, and K. Pekmestzi, "Designing Coarse-grain reconfigurable architectures by inlining flexibility into custom arithmetic data paths," *Integr, VLSI J*, vol. 42, no. 4, pp. 486-503, Sep. 2009.

[5]. S. Xydis, G. Economakos, D. Soudris, and K. Pekmestzi, "High Performance and area efficient flexible DSP data path synthesis," *IEEE Trans. Very Large Scale Integr.(VLSI) Syst.*, vol. 19, no. 3, pp.429-442, Mar.2011.

[6]. K. Tsoumanis, S. Xydis, G. Zervakis and K. P3kmestzi : "Flexible DSP Accelerator Architecture Exploiting Carry Save arithmetic." *IEEE Trans on very Large Scale Integration (VLSI) systems*. 2015.

[7]. K. Pitambar Patra, Janmejaya Samal, and Sambit Patnaik, " High Speed and Area Efficient Discrete Hartley Transform using Urdhwa Multiplier," *Inter J*, vol. 6, no. 2, February 2017.

[8]. K. Tsoumanis, S. Xydis, C. Efstathiou, N. Moschopoulos and K. Pekmestzi. "An Optimized Modified Booth Recoder for Efficient design of the Add- Multiply Operator." In circuits and systems, IEEE Trans, vol. 61, no.4, (2014).

D. Naga Divya. "A High Performance Reconfigurable Data Path Architecture For Flexible Accelerator." IOSR Journal of VLSI and Signal Processing (IOSR-JVSP) , vol. 7, no. 4, 2017, pp. 07–18.