IOSR Journal of VLSI and Signal Processing (IOSR-JVSP)
Volume 7, Issue 4, Ver. I (Jul. - Aug. 2017), PP 63-72
e-ISSN: 2319 – 4200, p-ISSN No. : 2319 – 4197
www.iosrjournals.org

An Efficient VLSI Architectures for FIR Filter in Fixed and Reconfigurable Applications

[1]D.Bhavani[*], [2]K. Padma Vasavi
[1](M.Tech -VLSID Student, Department of Electronics and Communications Engineering, Shri Vishnu Engineering College for Women (Autonomous), Bhimavaram, India)
[2](Professor, Department of Electronics and Communications Engineering, Shri Vishnu Engineering College for Women (Autonomous), Bhimavaram, India)
Corresponding Author: D.Bhavani

***Abstract**: Recently, with the development of Software Defined Radio (SDR) technology FIR filters have been concentrated on reconfigurable implementation. The filter coefficients in reconfigurable filters change dynamically in run time. In biomedical applications like Electro Cardio Gram (ECG) the coefficients of FIR filters remain fixed. So there is need to implement a Reconfigurable and Fixed FIR filter structure to support multi-standard communications. The need for reducing the area while increasing the computational speed is still felt. This paper addresses the problem of developing a high-speed and area efficient VLSI architectures for FIR filter in Fixed and Reconfigurable applications. The proposed architecture is compared with that of RFIR and FFIR filters using Ripple Carry Adder (RCA) and Carry Select Adder (CSLA) in Pipelined Adder Unit in terms of Area Delay Product(ADP). The proposed architecture has a better ADP than a RFIR and FFIR filter with RCA by 14.8% and 4.51% and is better than RFIR and FFIR filter with CSLA by 19.4% and 17.9%.*

***Keywords:** Finite Impulse Response (FIR) filters, MCM, Transpose-form, VLSI.*

Date of Submission: 10-07-2017 Date of acceptance: 24-07-2017

I. Introduction

The massive advancement in multimedia, computing and mobile communications has led to increase in demand for Digital Signal Processing [1]. In DSP applications, filters are used for two purposes namely signal restoration and signal separation. Digital filters can be categorized into FIR and IIR filters. FIR filters are widely used in mobile communication applications for operations like matched filtering, channel equalization, spectral shaping and interference cancellation because of their high stability and exact linear phase. Recently with the development of Software Defined Radio (SDR) technology [2] FIR filters have been concentrated on reconfigurable implementation. Filter coefficients in reconfigurable filters change dynamically during run time. In biomedical applications like Electro Cardio Gram (ECG) the coefficients of FIR filters remain fixed. So there is need to implement a Reconfigurable and Fixed FIR filter structure to support above mentioned multi-standard communications. Several researchers have proposed different types of VLSI architectures for the implementation of Fixed and Reconfigurable FIR filters. Distributed Arithmetic (DA) based FIR filters use Lookup Tables (LUTs) for storing the filter coefficients and to minimize the system complexity [3] and [4]. Multiple Constant Multiplications (MCM) based architecture is proposed for fixed coefficient implementation in this method, multiplication operation is performed with the help of shift and add unit. Computation sharing programmable FIR filter has been proposed by J. Park [5]. It is used for high speed and low power applications and it minimizes redundant computation in FIR filters by using the Computation Sharing Multiplier (CSHM). In 2006 Chen and Chiueh have proposed a RFIR filter architecture based on Canonic sign digit (CSD). It minimizes the accuracy of filter coefficients, but it occupies more area and does not provide sufficient area delay structure [6]. The architectures in [5] and [6] are more suitable for small order filter lengths and not applicable for channel filters because of their huge complexity. In 2010 R. Mahesh et al proposed a VLSI architecture for RFIR filters based on Programmable Shifts Method (PSM) and Constant Shifts Method (CSM) [7]. PSM provides low area and low power consumption and CSM provides high-speed when compared with PSM. In 2013 and 2014, B.K. Mohanty et al proposed a RFIR filter based on the Block Least Mean Square (BLMS) algorithm. But, it is not suitable for higher order filters and reconfigurable filter coefficients, such as multi channel filters [8] and [9]. Recently, S.Y. Park and P.K. Meher [10] have proposed a DA-based RFIR filter architecture of the area and delay efficiency. The Block-processing method is not used in this architecture. However, the need for reducing the area while increasing the computational speed is still felt. This paper addresses the problem of developing a high speed and area efficient VLSI architectures for FIR filter in Fixed and Reconfigurable applications. In this paper, implementation of transpose form pipelined block Finite Impulse Response (FIR) filters that supports Multiple Constant Multiplications (MCM) is presented. The rest of the paper is organized as follows: In section II, VLSI architectures for Fixed and Reconfigurable applications are presented. Simulation results are presented in section III. In section IV, performance comparisons are discussed. Finally, the conclusion is described in section V.

II. VLSI Architectures For Fixed And Reconfigurable Applications

2.1 Reconfigurable FIR filter architecture

The architecture of block FIR filter for Reconfigurable applications is shown in the Fig.1 for block size *L*=4. The main blocks are one Register Unit (RU), one Coefficient Storage Unit (CSU), one Pipeline Adder Unit (PAU), and M number of Inner Product Units (IPUs).

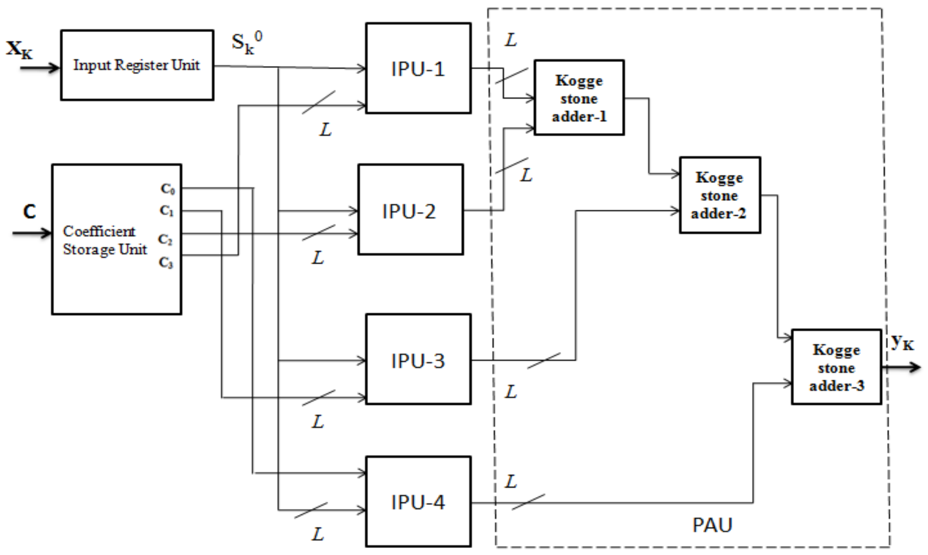


Fig.1. Block FIR filter for reconfigurable applications.

The Coefficient Storage Unit (CSU) is used to store the coefficients of all the filters. These coefficients are used in the reconfigurable applications. It has N Read Only Memory (ROM) Lookup Tables (LUTs) where N is the length of the filter (N=ML). The Register Unit (RU) is used for storing the input samples is shown in Fig.2. It contains (L-1) registers. During the K^{th} cycle, the register unit accepts input sample X_K and computes *L* rows of S_K^0 in parallel. The outputs from the RU are given as inputs to *M* Inner Product Units.

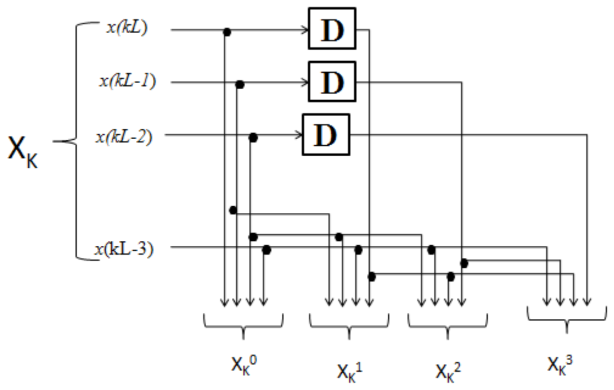


Fig.2. Internal structure of Register Unit (RU) for block size L=4.

The Inner Product Unit (IPU) is used to perform a multiplication operation of S_K^0 with the small weight vector c_m is shown in Fig.3. The *M* Inner Products Units accepts *L* rows of S_K^0 from the RU and *M* small weight vectors from the CSU. Each Inner Product Unit contains *L* number of Inner Product Cells (IPCs) which performs *L* inner product computations of *L* rows of S_K^0 with coefficient vector c_m and produces a block of *L* number of partial inner products. All the four IPUs work simultaneously and *M* blocks of a result are obtained.

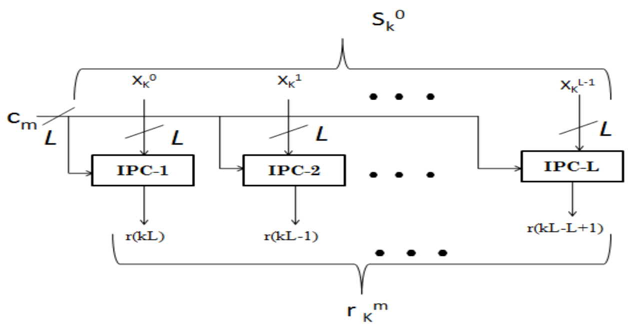


Fig.3. Internal structure of (m+1)$_{th}$ IPU.

The internal structure of (l +1) $_{th}$ IPC is shown in Fig.4. The Inner Product Cell (IPC) accepts *(l+1)$_{th}$* row of $\mathbf{S}_k^0$ and small weight vector c_m and produce a partial result of inner product *$r(kL - l)$*, for $0 \leq l \leq L - 1$. Each IPC consists of *L* multipliers and *(L-1)* number of adders.

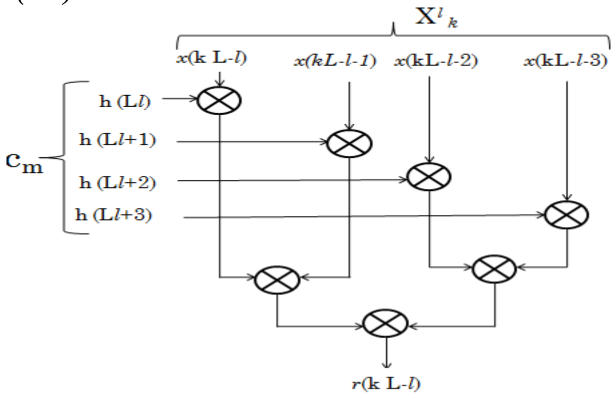


Fig.4. Internal structure of (l + 1)$_{th}$ IPC.

The Pipelined Adder Unit (PAU) receives partial products from all the M IPUs. Array of Kogge Stone Adder is used in PAU to add all the partial products is shown in Fig.5. KSA is one of the Carry Tree Adders or Parallel Prefix Adders. Kogge Stone Adders gains more importance among all the adders because of its high performance.

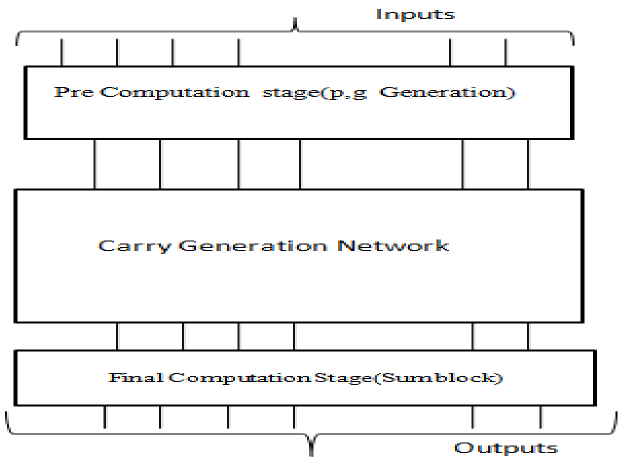

Inputs
Pre Computation stage(p,g Generation)
Carry Generation Network
Final Computation Stage(Sumblock)
Outputs

Fig.5. Different Stages in Kogge Stone Adder.

KSA can be implemented in 3 stages, namely Pre-Computation Stage, Carry generation network and final computation stage. Generate and Propagate signals are computed in Pre-Computation Stage, corresponding to each pair of input bits A and B. The second stage compute carries corresponding to each bit. Execution of these operations is performed in parallel form, and they are partitioned into smaller pieces. Group generate and propagate bits which are computed in the first stage are used as intermediate signals in carry generation network. The final computation stage is common for all the adders of this family which gives the summation of input bits.

2.2 Fixed FIR filter architecture

The architecture of block FIR filter for fixed application is shown in the Fig.6. For Fixed FIR filter implementation, the CSU is not necessary here filter coefficients are fixed. Similarly, IPUs are not used because multiplication operation is performed with Multiple Constant Multiplication (MCM) units to reduce the huge complexity of the architecture.

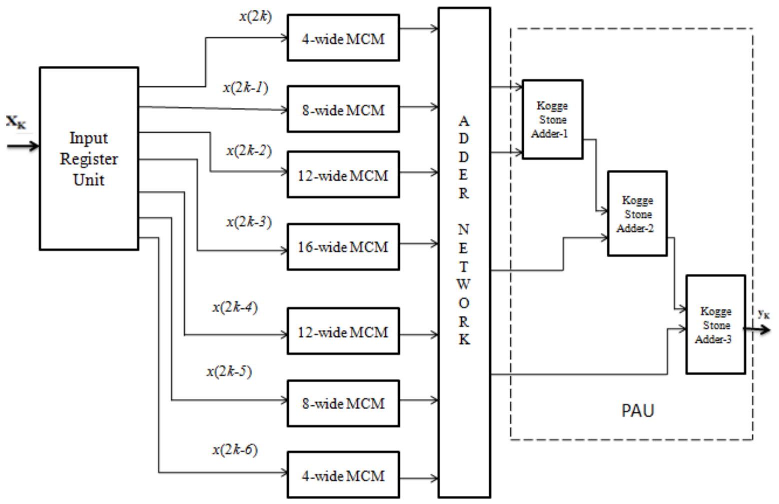


Fig.6. Fixed FIR filter using MCM.

The MCM based method is more efficient when a given input variable is multiplied with more number of fixed constants using shift and add method is shown in Fig.7. It can be implemented by using adders/sutractors and shifters. Initially, the constants are expressed in binary form. Then for every non-zero digits in the binary format of the constant, based on its digit position the input variable is shifted and adds up the shifted variable to obtain a result. MCM is employed in many applications like error correcting codes, frequency multiplication and Multiple Input and Multiple Output systems (MIMO).

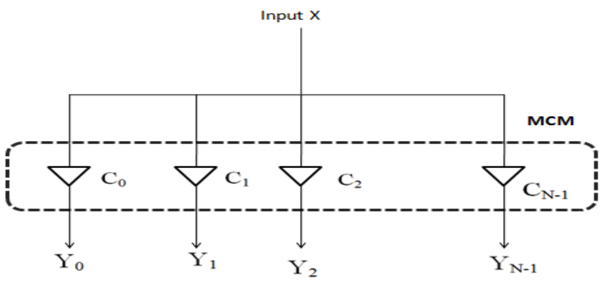
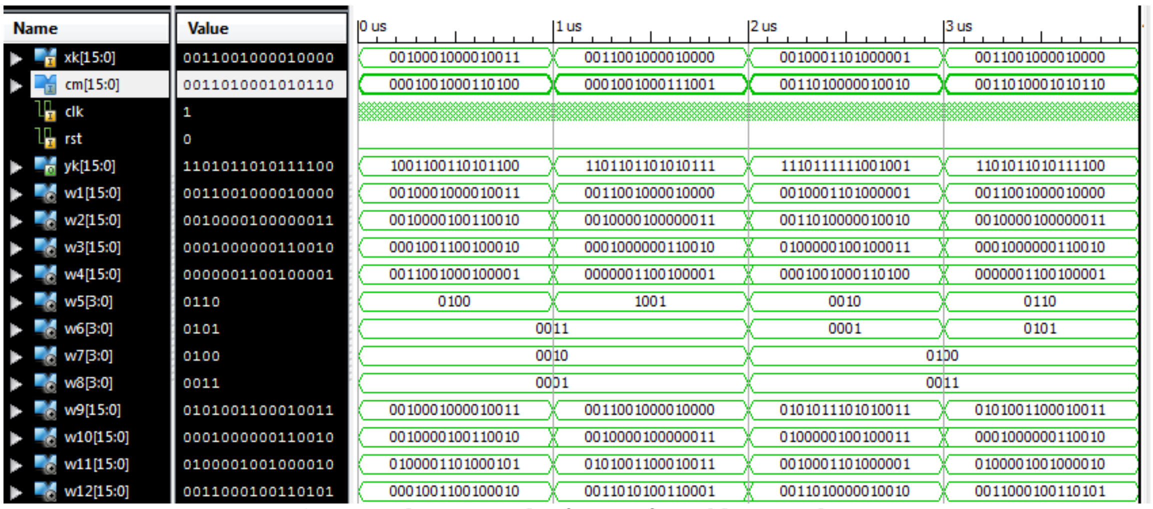


Fig.7. Block Diagram of MCM.

MCM based technique for a Fixed FIR filter with block size *L*=4, make utilize of symmetry in input matrix S_K^0 to execute vertical and horizontal common subexpression elimination and to reduce the number of shift and add functions in the MCM units. MCM can be employed in both vertical and horizontal order of the coefficient matrix. The MCM based method consists of six input samples similar to six MCM blocks. All MCM blocks compute the required product terms using shift and add method. The outputs of all MCM blocks are given to the adder network to produce the inner product terms. In the Pipelined Adder Unit (PAU) array of KSA is used to add inner product values and produce a block of the filter output.

III. Simulation Results

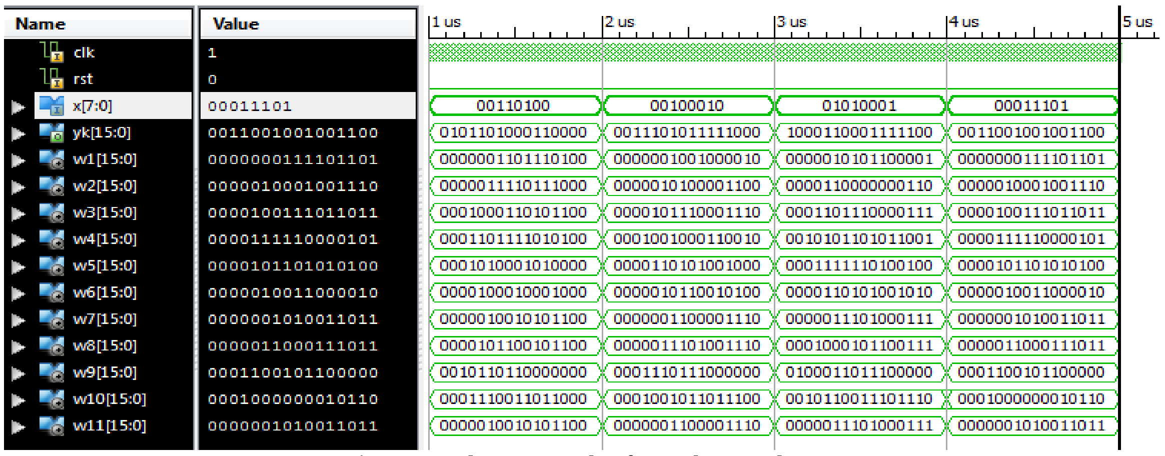
The proposed VLSI architectures for fixed and reconfigurable applications are written in a VerilogHDL, synthesized and simulated using Xilinx ISE 12.2 design tool and an ISIM simulator. The design properties used for simulation results are Spartan 3E family, XC3S500E device, FG320 package with a speed grade of -5.



```
Name      Value              |0 us              |1 us              |2 us              |3 us
xk[15:0]  0011001000010000   0010001000010011   0011001000010000   0010001101000001   0011001000010000
cm[15:0]  0011010001010110   0001001000110100   0001001000111001   0011010000010010   0011010001010110
clk       1
rst       0
yk[15:0]  1101011010111100   1001100110101100   1101101101010111   1110111111001001   1101011010111100
w1[15:0]  0011001000010000   0010001000010011   0011001000010000   0010001101000001   0011001000010000
w2[15:0]  0010000100000011   0010000100110010   0010000100000011   0011010000010010   0010000100000011
w3[15:0]  0001000000110010   0001001100100010   0001000000110010   0100000100100011   0001000000110010
w4[15:0]  0000001100100001   0011001000100001   0000001100100001   0001001000110100   0000001100100001
w5[3:0]   0110               0100               1001               0010               0110
w6[3:0]   0101               0011                                  0001               0101
w7[3:0]   0100               0010                                  0100
w8[3:0]   0011               0001                                  0011
w9[15:0]  0101001100010011   0010001000010011   0011001000010000   0101011101010011   0101001100010011
w10[15:0] 0001000000110010   0010000100110010   0010000100000011   0100000100100011   0001000000110010
w11[15:0] 0100001001000010   0100001101000101   0101001100010011   0010001101000001   0100001001000010
w12[15:0] 0011000100110101   0001001100100010   0011010100110001   0011010000010010   0011000100110101
```

Fig.8. Simulation Result of Reconfigurable FIR Filter

The simulation result of Reconfigurable FIR filter is shown in the Fig.8. The input sample X_K is a 16-bit binary value and filter length C_m is also a 16-bit binary value. Input values are applied to RU it produces four rows of 16-bit input samples by performing a shift operation. The 16-bit filter coefficients are partitioned into a group of four bits. Multiplication operations are performed between filter coefficients and the input samples. These partial products are finally added in the PAU by using an array of Kogge Stone Adders (KSA). Finally, from KSA filter output is obtained. To illustrate the functionality of the proposed architecture a 16-bit input sample Xk=0011001000010000 and filter coefficient cm=0011010001010110 with clock=1 and reset =0 are considered, then shift operation is performed on input sample, which gives four rows of input. The first row of input is the same as the input sample w1. For the second row of input, the first four bits of the MSB (0011) of input sample become LSB and this process continues w2, w3 and w4 are obtained. Filter coefficient cm is partitioned into a group of four bits which gives w5, w6, w7, and w8. The four rows of input w1, w2, w3, and w4 are multiplied with filter coefficients w8,w7,w6, and w5 then the output w9,w10,w11,w12 are obtained respectively. After that by using KSA the summation of w9, w10, w11 and w12 are performed. Finally, the filter output yk=1101011010111100 is obtained.



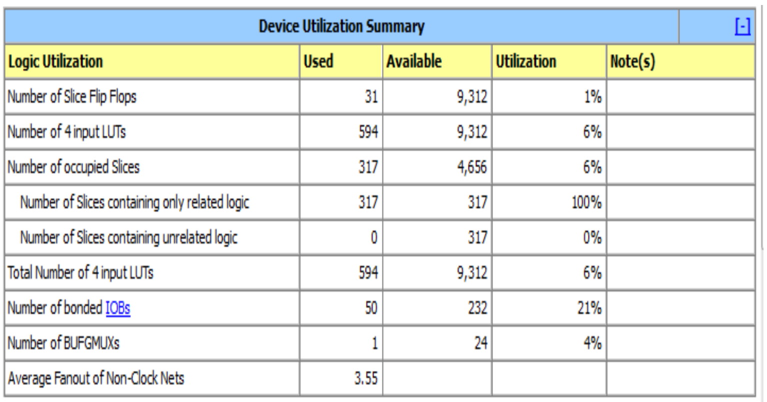
```
Name      Value              |1 us              |2 us              |3 us              |4 us              |5 us
clk       1
rst       0
x[7:0]    00011101           00110100           00100010           01010001           00011101
yk[15:0]  0011001001001100   0101101000110000   0011101011111000   1000110001111100   0011001001001100
w1[15:0]  0000000111101101   0000001101110100   0000001001000010   0000010101100001   0000000111101101
w2[15:0]  0000010001001110   0000011110111000   0000010100001100   0000110000000110   0000010001001110
w3[15:0]  0000100111011011   0001000110101100   0000101110001110   0001101110000111   0000100111011011
w4[15:0]  0000111110000101   0001101111010100   0001001000110010   0010101101011001   0000111110000101
w5[15:0]  0000101101010100   0001010001010000   0000110101001000   0001111110100100   0000101101010100
w6[15:0]  0000010011000010   0000100010001000   0000010110010100   0000110101001010   0000010011000010
w7[15:0]  0000001010011011   0000010010101100   0000001100001110   0000011101000111   0000001010011011
w8[15:0]  0000011000111011   0000101100101100   0000011101001110   0001000101100111   0000011000111011
w9[15:0]  0001100101100000   0010110110000000   0001110111000000   0100011011100000   0001100101100000
w10[15:0] 0001000000010110   0001110011011000   0001001011011100   0010110011101110   0001000000010110
w11[15:0] 0000001010011011   0000010010101100   0000001100001110   0000011101000111   0000001010011011
```

Fig.9. Simulation Result of Fixed FIR Filter

The simulation result of Fixed FIR filter is shown in the Fig.9. The input sample X is 8-bit binary value and having the filter length N=16. Input values are applied to RU it produces six input samples of 8-bit binary value. These input samples are multiplied by several constant coefficients using the MCM method. The outputs of all the MCM blocks are given to adder network it produce inner product values. The array of KSA is used in PAU for summation of all the inner product values. Finally, from KSA filter output is obtained. To illustrate the functionality of the proposed architecture a 8-bit input sample Xk=00011101 and fixed coefficient are considered. MCM operation is performed between the input sample and 4- wide MCM (h0, h1, h2, h3) having a width of 8-bit binary value then w1 is obtained. This process repeats for 8,12 and 16 wide MCM and the output obtained are w2, w3, w4, w5, w6, and w7. In adder network, these partial inner products are added which gives w8, w9, w10 and w11. After that by using KSA the summation of w8, w9, w10 and w11 are performed. Finally, the filter output yk is obtained.

IV. Comparisons

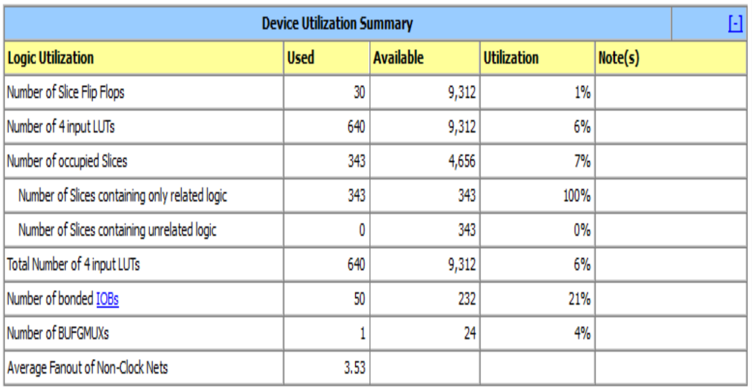
The above section deals with the simulation results. In this section comparisons of Fixed and Reconfigurable VLSI architectures using different types of adders like Ripple Carry Adder, Carry Select Adder and Kogge Stone Adder are examined. The design properties used for synthesis results are Spartan 3E family, XC3S500E device, FG320 package with a speed grade of -5.



Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	31	9,312	1%	
Number of 4 input LUTs	594	9,312	6%	
Number of occupied Slices	317	4,656	6%	
Number of Slices containing only related logic	317	317	100%	
Number of Slices containing unrelated logic	0	317	0%	
Total Number of 4 input LUTs	594	9,312	6%	
Number of bonded IOBs	50	232	21%	
Number of BUFGMUXs	1	24	4%	
Average Fanout of Non-Clock Nets	3.55			

Fig.10. Area report of RFIR filter implemented with Ripple Carry Adder

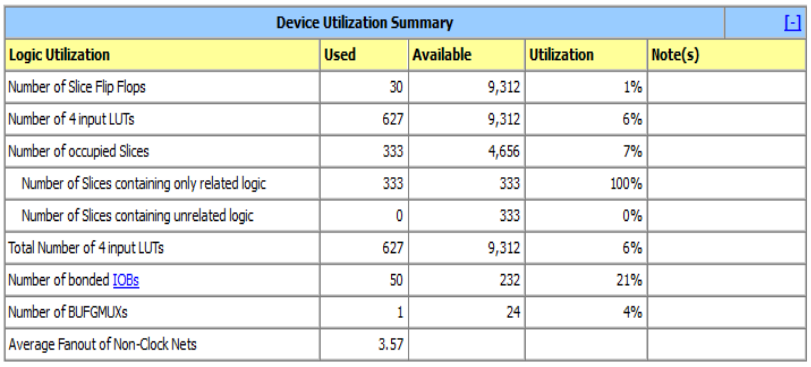
Device utilization summary of the RFIR filter implemented with Ripple Carry Adder is shown in Fig.10. From the figure, it is observed that, out of 4,656 available slices 317 slices are occupied by the logic and the system utilizes 594 four input LUT's out of 9312 available 4 input LUT's.



Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	30	9,312	1%	
Number of 4 input LUTs	640	9,312	6%	
Number of occupied Slices	343	4,656	7%	
Number of Slices containing only related logic	343	343	100%	
Number of Slices containing unrelated logic	0	343	0%	
Total Number of 4 input LUTs	640	9,312	6%	
Number of bonded IOBs	50	232	21%	
Number of BUFGMUXs	1	24	4%	
Average Fanout of Non-Clock Nets	3.53			

Fig.11. Area report of RFIR filter implemented with Carry select adder

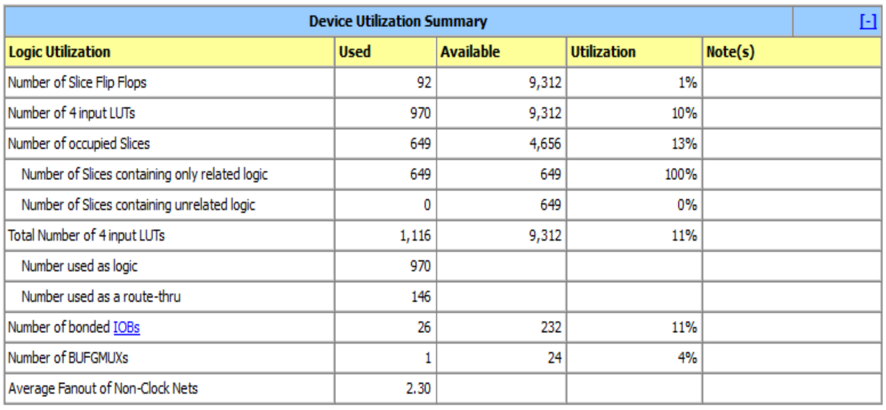
Device utilization summary of the RFIR filter implemented with Carry Select Adder is shown in Fig.11.From the figure, it is observed that, out of 4,656 available slices 343 slices are occupied by the logic and the system utilizes 640 four input LUT's out of 9,312 available 4 input LUT's.



Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	30	9,312	1%	
Number of 4 input LUTs	627	9,312	6%	
Number of occupied Slices	333	4,656	7%	
Number of Slices containing only related logic	333	333	100%	
Number of Slices containing unrelated logic	0	333	0%	
Total Number of 4 input LUTs	627	9,312	6%	
Number of bonded IOBs	50	232	21%	
Number of BUFGMUXs	1	24	4%	
Average Fanout of Non-Clock Nets	3.57			

Fig.12.Area report of RFIR filter implemented with Kogge Stone adder

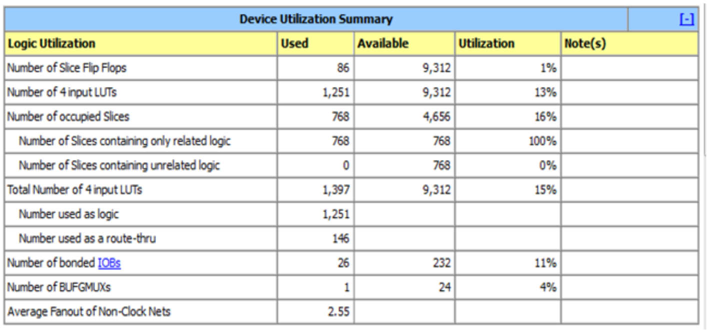
Device utilization summary of the RFIR filter implemented with Kogee Stone Adder is shown in Fig.12. From the figure, it is observed that, out of 4,656 available slices 333 slices are occupied by the logic and the system utilizes 627 four input LUT's out of 9,312 available 4 input LUT's.



Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	92	9,312	1%	
Number of 4 input LUTs	970	9,312	10%	
Number of occupied Slices	649	4,656	13%	
Number of Slices containing only related logic	649	649	100%	
Number of Slices containing unrelated logic	0	649	0%	
Total Number of 4 input LUTs	1,116	9,312	11%	
Number used as logic	970			
Number used as a route-thru	146			
Number of bonded IOBs	26	232	11%	
Number of BUFGMUXs	1	24	4%	
Average Fanout of Non-Clock Nets	2.30			

Fig.13. Area report of Fixed FIR filter implemented with Ripple Carry Adder

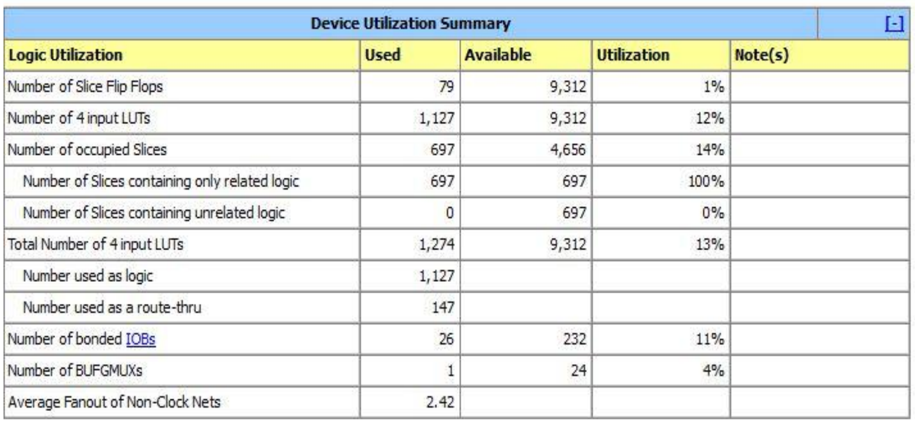
Device utilization summary of the Fixed FIR filter implemented with Ripple Carry Adder is shown in Fig.13. From the figure, it is observed that, out of 4,656 available slices 649 slices are occupied by the logic and the system utilizes 970 four input LUT's out of 9,312 available 4 input LUT's.



Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	86	9,312	1%	
Number of 4 input LUTs	1,251	9,312	13%	
Number of occupied Slices	768	4,656	16%	
Number of Slices containing only related logic	768	768	100%	
Number of Slices containing unrelated logic	0	768	0%	
Total Number of 4 input LUTs	1,397	9,312	15%	
Number used as logic	1,251			
Number used as a route-thru	146			
Number of bonded IOBs	26	232	11%	
Number of BUFGMUXs	1	24	4%	
Average Fanout of Non-Clock Nets	2.55			

Fig.14. Area report of Fixed FIR filter implemented with Carry Select Adder

Device utilization summary of the Fixed FIR filter implemented with Carry Select Adder is shown in Fig.14. From the figure, it is observed that, out of 4,656 available slices 768 slices are occupied by the logic and the system utilizes 1,251 four input LUT's out of 9,312 available 4 input LUT's.



Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	79	9,312	1%	
Number of 4 input LUTs	1,127	9,312	12%	
Number of occupied Slices	697	4,656	14%	
Number of Slices containing only related logic	697	697	100%	
Number of Slices containing unrelated logic	0	697	0%	
Total Number of 4 input LUTs	1,274	9,312	13%	
Number used as logic	1,127			
Number used as a route-thru	147			
Number of bonded IOBs	26	232	11%	
Number of BUFGMUXs	1	24	4%	
Average Fanout of Non-Clock Nets	2.42			

Fig.15. Area report of Fixed FIR filter implemented with Kogge Stone Adder

Device utilization summary of the Fixed FIR filter implemented with Kogee Stone Adder is shown in Fig.15. From the figure, it is observed that, out of 4,656 available slices 697 slices are occupied by the logic and the system utilizes 1,127 four input LUT's out of 9,312 available 4 input LUT's.

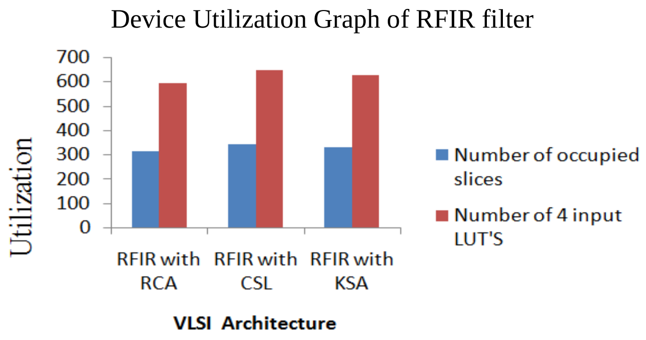


Fig.16. Comparison of area of VLSI architectures for RFIR filters using different adders

From the above statistics the device utilization graph is drawn for RFIR architectures using three different adders like Ripple Carry Adder, Carry Select Adder and Kogge Stone Adder are shown in Fig.16. RCA occupies less area when compared with KSA and CSLA.

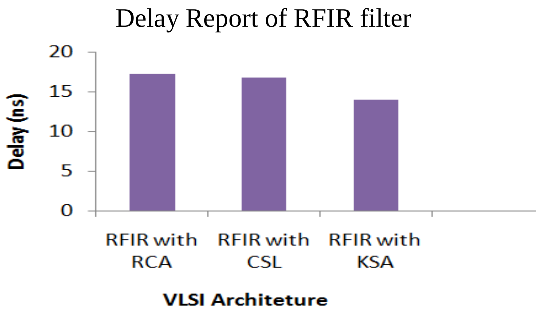


Fig.17. Comparison of delay of VLSI architectures for RFIR filters using different adders

The delay report is drawn for RFIR architectures using three different adders like Ripple Carry Adder, Carry Select Adder and Kogge Stone Adder are shown in Fig.17. KSA has less delay when compared with RCA and CSLA.

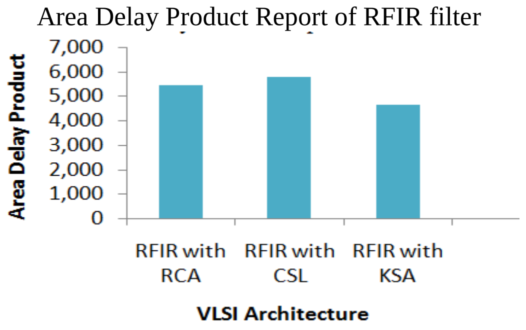


Fig.18. Comparison of Area Delay Product of VLSI architectures for RFIR filters using different adders

From the above area and delay reports, Area Delay Product (ADP) graph is drawn for RFIR architectures using three different adders like Ripple Carry Adder, Carry Select Adder and Kogge Stone Adder are shown in Fig.18. KSA has a less Area Delay Product (ADP) when compared with RCA and CSLA.

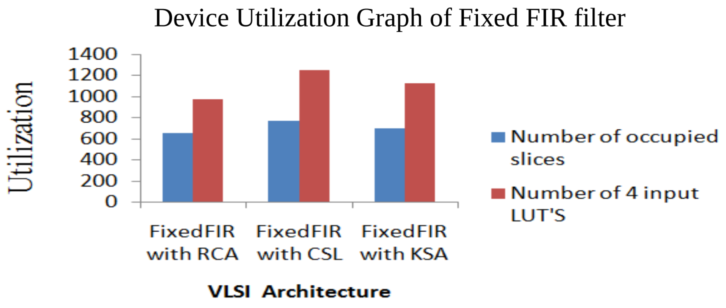


Fig.19. Comparison of area of VLSI architectures for Fixed FIR filters using different adders

From the above statistics the device utilization graph is drawn for Fixed FIR architectures using three different adders like Ripple Carry Adder, Carry Select Adder and Kogge Stone Adder are shown in Fig.19. RCA occupies less area when compared with KSA and CSLA.

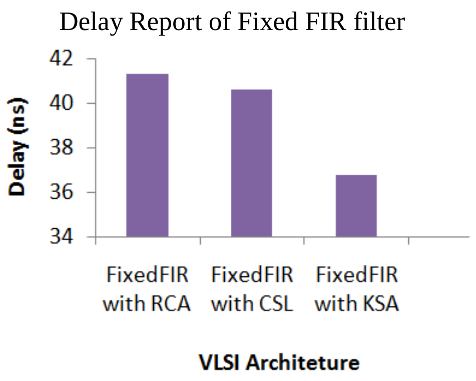


Fig.20. Comparison of delay of VLSI architectures for Fixed FIR filters using different adders

The delay report is drawn for Fixed FIR architectures using three different adders like Ripple Carry Adder, Carry Select Adder and Kogge Stone Adder are shown in Fig.20. KSA has less delay when compared with RCA and CSLA.

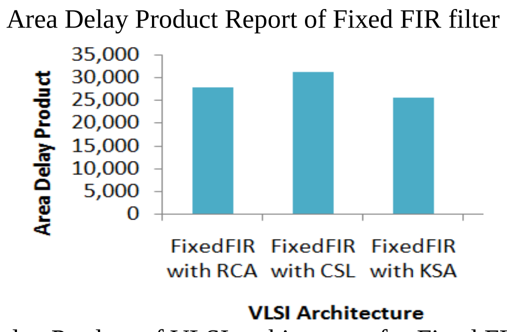


Fig.21. Comparison of Area Delay Product of VLSI architectures for Fixed FIR filters using different adders

From the above area and delay reports, Area Delay Product (ADP) graph is drawn for Fixed FIR architectures using three different adders like Ripple Carry Adder, Carry Select Adder and Kogge Stone Adder are shown in Fig.21. KSA has less Area Delay Product (ADP) when compared with RCA and CSLA. Thus, from the above simulation results and comparisons it is clear that VLSI architectures for Fixed and Reconfigurable applications using Kogge Stone Adder obtains less Area Delay Product when compared with other two adders.

V. Conclusion

In this paper, a high-speed and area efficient transpose form block FIR filter is implemented for both Fixed and Reconfigurable applications. The proposed architecture is compared with that of RFIR and FFIR filter using Ripple Carry Adders (RCA) and Carry Select Adder (CSLA) in pipelined adder unit in terms of Area Delay Product (ADP). The proposed architecture has a better ADP than a RFIR and FFIR filter with RCA by 14.8% and 4.51% and is better than RFIR and FFIR filter with CSL by 19.4% and 17.9%. In the future, the area and delay can further be reduced in transpose form FIR filter by using Dadda multiplier in the Inner Product Unit.

Acknowledgements

The authors would like to express their sincere gratitude to our beloved Principal Dr. G. Srinivasa Rao and Vice-Principal Dr. P. Srinivasa Raju of Shri Vishnu Engineering College for Women (SVECW), Bhimavaram for their support in providing facilities to pursue the research. The authors would like to acknowledge the anonymous reviewers for their suggestions and detailed comments which greatly helps in enhancing the presentation of this work.

References

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

[2] J. Mitola, Software Radio Architecture: Object-Oriented Approaches to Wireless Systems Engineering. New York, NY, USA: Wiley, 2000.

[3] P. K. Meher, S. Chandrasekaran, and A. Amira, “FPGA realization of FIR filters by efficient and flexible systolization using distributed arithmetic,” IEEE Trans. Signal Process., vol. 56, no. 7, pp. 3009–3017,Jul. 2008.

[4] P. K. Meher, “New approach to look-up-table design and memory based realization of FIR digital filter,” IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 57, no. 3, pp. 592–603, Mar. 2010.

[5] J. Park, W. Jeong, H. Mahmoodi-Meimand, Y. Wang, H. Choo, and K. Roy, “Computation sharing programmable FIR filter for low- power and high-performance applications,” IEEE J. Solid State Circuits, vol. 39, no. 2, pp. 348–357, Feb. 2004.

[6] K.-H. Chen and T.-D. Chiueh, “A low-power digit-based reconfigurable FIR filter,” IEEE Trans. Circuits Syst. II, Exp. Briefs, vol. 53, no. 8, pp. 617–621, Aug. 2006.

[7] R. Mahesh and A. P. Vinod, “New reconfigurable architectures for implementing FIR filters with low complexity,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 29, no. 2, pp. 275–288, Feb. 2010.

[8] B. K. Mohanty and P. K. Meher, “A high-performance energy-efficient architecture for FIR adaptive filter based on new distributed arithmetic formulation of block LMS algorithm,” IEEE Trans. Signal Process.,vol. 61, no. 4, pp. 921–932, Feb. 2013.

[9] B. K. Mohanty, P. K. Meher, S. Al-Maadeed, and A. Amira, “Memory footprint reduction for power-efficient realization of 2-D finiteimpulse response filters,” IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 61, no. 1,pp. 120–133, Jan. 2014.

[10] S. Y. Park and P. K. Meher, “Efficient FPGA and ASIC realizations of a DA-based reconfigurable FIR digital filter,” IEEE Trans. Circuits Syst. II, Exp. Briefs, vol. 61, no. 7, pp. 511–515, Jul. 2014.

D.Bhavani. "An Efficient VLSI Architectures for FIR Filter in Fixed and Reconfigurable Applications." IOSR Journal of VLSI and Signal Processing (IOSR-JVSP) 7.4 (2017): 63-72.