International Journal For Technological Research In Engineering
Volume 4, Issue 9, May-2017 *ISSN (Online): 2347 - 4718*

A NOVEL DESIGN OF AN AREA EFFICIENT FCU USING CS ADDER AND DADDA MULTIPLIER

Mr. Sudheer Janipalli[1], Mrs. B Satya Sridevi[2]
[1]PG Scholar, [2]Sr. Assistant Professor,
Dept of ECE, ADITYA Engineering College, surampalem, E.G Dist A.P.

Abstract: This paper presents a novel approach to design Flexible computational unit (FCU) exploiting carry save arithmetic and dadda multiplier. Earlier projects emphasized on designing systems with low power and fast operations resulting in increased area. The goal of this project is to design DSP system to make the design more efficient by proposing Dadda tree multiplier. Utilizing Dadda tree multiplier reduces the area required for the advisement. As in these modern times, most of the electronic systems are adapted to perform digital signal processing; the DSP integrated systems have wide range of applications. Kernel which is the core of the DSP system consists of data flow graph that features the corresponding arithmetic operation performed by the system. These DFGs basically consist of arithmetic units such as adders, multipliers etc that are mapped onto the proposed flexible control units which are carry save formatted data. As we know that the basic intension of designing integrated circuits is to minimize the dimensions of a given circuit. This objective is achieved by incorporating dada multiplier as multiplier unit of the DSP data flow graph template. This implementation shows considerable difference in terms of area which further leads to other advantages such as to fabricate more number of components onto the resulted area.

Index Terms: Digital signal processing (DSP), Data flow graph (DFG), Register Transfer logic (RTL), Template(T), Spurious power suppression technique (SPST), Carry save arithmetic (CSA).

I. INTRODUCTION

Modern embedded systems target high-end application domains requiring efficient implementations of computationally intensive digital signal processing (DSP) functions. The incorporation of heterogeneity through specialized hardware accelerators improves performance and reduces energy consumption [1]. Although application-specific integrated circuits (ASICs) form the ideal acceleration solution in terms of performance and power, their inflexibility leads to increased silicon complexity, as multiple instantiated ASICs are needed to accelerate various kernels. Many researchers have proposed the use of domain-specific coarse-grained reconfigurable accelerators in order to increase ASICs' flexibility without significantly compromising their performance. High-performance flexible data paths have been proposed to efficiently map primitive or chained operations found in the initial data-flow graph (DFG) of a kernel. The templates of complex chained operations are either extracted directly from the kernel's DFG or specified in a predefined behavioral template library. Design decisions on the accelerator's data path highly impact its efficiency. Existing works on coarse-grained reconfigurable data paths mainly exploit architecture-level optimizations, e.g., increased instruction-level parallelism (ILP). The domain-specific architecture generation algorithms of [5] and [9] vary the type and number of computation units achieving a customized design structure. The flexible architectures were proposed exploiting ILP and operation chaining. Recently aggressive operation chaining is adopted to enable the computation of entire sub expressions using multiple ALUs with heterogeneous arithmetic features. The aforementioned reconfigurable architectures exclude arithmetic optimizations during the architectural synthesis and consider them only at the internal circuit structure of primitive components, e.g., adders, during the logic synthesis. However, research activities have shown that the arithmetic optimizations at higher abstraction levels than the structural circuit one significantly impact on the datapath performance. In [10], timing-driven optimizations based on carry-save (CS) arithmetic were performed at the post-Register Transfer Level (RTL) design stage. In [11], common subexpression elimination in CS computations is used to optimize linear DSP circuits. Verma et al. [12] developed transformation techniques on the application's DFG to maximize the use of CS arithmetic prior the actual datapath synthesis. The aforementioned CS optimization approaches target inflexible datapath, i.e., ASIC, implementations. Recently, a flexible architecture combining the ILP and pipelining techniques with the CS-aware operation chaining has been proposed. However, all the aforementioned solutions feature an inherent limitation, i.e., CS optimization is bounded to merging only additions/subtractions. A CS to binary conversion is inserted before each operation that differs from addition/subtraction, e.g.,multiplication, thus, allocating multiple CS to binary conversions that heavily degrades performance due to time-consuming carry propagations. In this brief, we propose a high-performance architectural scheme for the synthesis of flexible hardware DSP accelerators by combining optimization techniques from both the architecture and arithmetic levels of abstraction. We introduce a flexible datapath architecture that exploits CS optimized templates of chained operations. The proposed architecture comprises flexible computational units (FCUs), which enable the execution of a large set of operation templates found in DSP kernels. The proposed accelerator architecture delivers average gains in area-delay product and in energy consumption compared to state-of-art flexible datapaths , sustaining efficiency toward scaled technologies.

II. CARRY-SAVE ARITHMETIC: MOTIVATIONAL OBSERVATIONS AND LIMITATIONS

CS representation has been widely used to design fast arithmetic circuits due to its inherent advantage of eliminating the large carry-propagation chains. CS arithmetic optimizations rearrange the application's DFG and reveal multiple input additive operations (i.e., chained additions in the initial DFG), which can be mapped onto CS compressors. The goal is to maximize the range that a CS computation is performed within the DFG. However, whenever a multiplication node is interleaved in the DFG, either a CS to binary conversion is invoked or the DFG is transformed using the distributive property . Thus, the aforementioned CS optimization approaches have limited impact on DFGs dominated by multiplications, e.g., filtering DSP applications. In this brief, we tackle the aforementioned limitation by exploiting the CS to modified Booth (MB) recoding each time a multiplication needs to be performed within a CS-optimized datapath. Thus, the computations throughout the multiplications are processed using CS arithmetic and the operations in the targeted datapath are carried out without using any intermediate carry-propagate adder for CS to binary conversion, thus improving performance.

III. FLEXIBLE COMPUTATIONAL UNIT

Adaptable data paths have been proposed to efficiently scale chained operations found in the initial data-flow graph (DFG) of a kernel of the DSP system. The templates of complex chained operations are either extracted directly from the kernel's DFG or specified in a ready-made behavioral template library. The stated architecture comprises flexible computational units (FCUs), which enable the execution of a large set of operation templates found in DSP kernels. The proposed accelerator architecture delivers average gains in area-delay product and in energy consumption compared to state-of-art flexible data paths, sustaining efficiency toward scaled technologies. The proposed flexible accelerator architecture is shown in Fig. 1. Each FCU operates directly on operands and produces data in the same form for direct reuse of intermediate results. Each FCU operates on 16-bit operands. Such a bit-length is adequate for the most DSP datapaths, but the architectural concept of the FCU can be straightforwardly adapted for smaller or larger bit-lengths. The number of FCUs is determined at design time based on the instruction-level parallelism (ILP) and area constraints imposed by the designer. The register bank consists of scratch registers and is used for storing intermediate results and sharing operands among the FCUs. Different DSP kernels (i.e., different register allocation and data communication patterns per kernel) can be mapped onto the proposed architecture using post-RTL data path interconnection sharing techniques. The control unit drives the entire accelerator architecture (i.e., communication between the data port and the register bank, configuration words of the FCUs and selection signals for the multiplexers) in each clock cycle.

Data Path

The structure of the FCU (Fig. 2) has been designed to enable high performance flexible operation chaining based on a library of operation templates. Each FCU can be configured to any of the T1–T5 operation templates in template library.

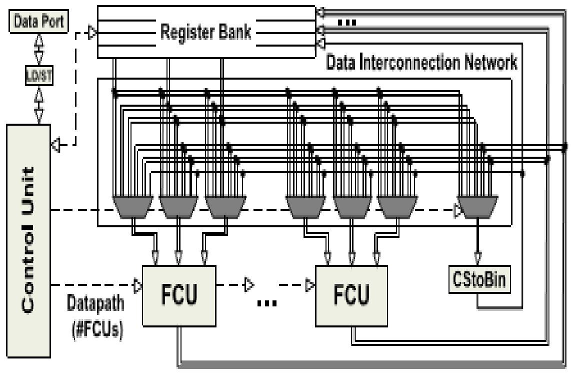


Fig.1: Flexible accelerator architecture.

The proposed FCU enables intra-template operation chaining by fusing the additions performed before/after the multiplication & performs any partial operation template of the following complex operations:

$W^* = A \times (X^* + Y^*) + K^*$ (1)
$W^* = A \times K^* + (X^* + Y^*)$ (2)

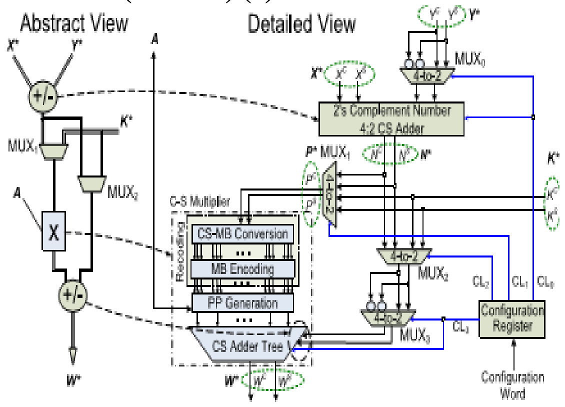


Fig.2: shows a segment of the internal structure of the FCU i.e. operational template of the data

Path specified in equation (1) and (2)

The following relation holds for all CS data: $X^* = \{X^C, X^S\} = X^C + X^S$. The operand A is a two's complement number. The alternative execution paths in each FCU are specified after properly setting the control signals of the multiplexers MUX1 and MUX2 (Fig. 2). The multiplexer MUX0 outputs Y* when CL0 = 0 (i.e., X* + Y* is carried out) or Y* when X* − Y* is required and CL0 = 1. The two's complement 4:2 CS adder produces the N* = X* + Y* when the input carry equals 0 or the N* = X* − Y* when the input carry equals 1. The MUX1 determines if N* (1) or K* (2) is multiplied with A. The MUX2 specifies if K* (1) or N* (2) is added with the multiplication product. The multiplexer MUX3 accepts the output of MUX2 and its 1's complement and outputs the former one when an addition with the multiplication product is required (i.e., CL3 = 0) or the later one when a subtraction is carried out (i.e., CL3 = 1). The 1-bit ace for the subtraction is added in the CS adder tree. The multiplier comprises a CS-to-MB module, which adopts a recently proposed technique to recode the 17-bit P* in its respective MB digits with minimal carry propagation. The multiplier's product consists of 17 bits. The multiplier includes a compensation method for reducing the error imposed at the product's accuracy by the truncation technique. However, since all the FCU inputs consist of 16 bits and provided that there are no overflows, the 16 most significant bits of the 17-bit W* (i.e., the output of the Carry-Save Adder (CSA) tree, and thus, of the FCU) are inserted in the appropriate FCU when requested.

IV. EXISTING PROJECT

The existing project implemented adders using carry save arithematic as substitute for adders and Modified Booth Encoder for multipliers found in the datapath of the FCUs in the system kernels. Figure 2 shows implementation of Carry Save adders in the internal structure of FCU. But the serious drawback of this project was that there was tradeoff between power and speed resulting in time consumption.

FCU with carry save arithmetic implementation

CS representation has been widely used to design fast arithmetic circuits due to its inherent advantage of eliminating the large carry-propagation chains. CS arithmetic optimizations rearrange the application's DFG and reveal multiple input additive operations (i.e., chained additions in the initial DFG), which can be mapped onto CS compressors. The goal is to maximize the range that a CS computation is performed within the DFG. However, whenever a multiplication node is interleaved in the DFG, either a CS to binary conversion is invoked or the DFG is transformed using the distributive property. Thus, the aforementioned CS optimization approaches have limited impact on DFGs dominated by multiplications, e.g., filtering DSP applications.

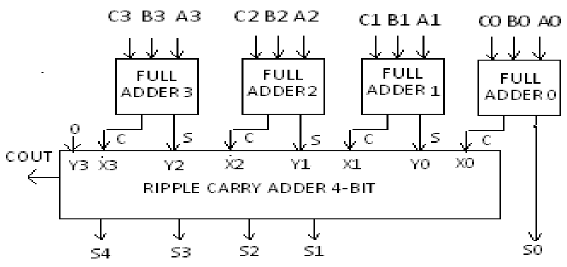


Fig. 3: A 4-bit carry save adder using four full adders and a 4-bit ripple carry adder.

In this brief, we tackle the aforementioned limitation by exploiting the CS to modified Booth (MB) recoding each time a multiplication needs to be performed within a CS-optimized datapath.

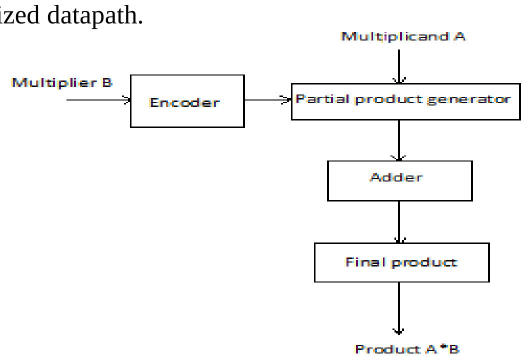


Fig.4: Block diagram of Modified booth multiplier

Thus, the computations throughout the multiplications are processed using CS arithmetic and the operations in the targeted datapath are carried out without using any intermediate carry-propagate adder for CS to binary conversion, thus improving performance.

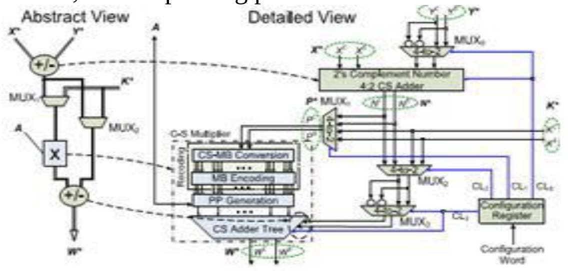


Fig. 5: carry-save adders and modified booth technique for multiplier in FCU.

SIMULATION RESULTS

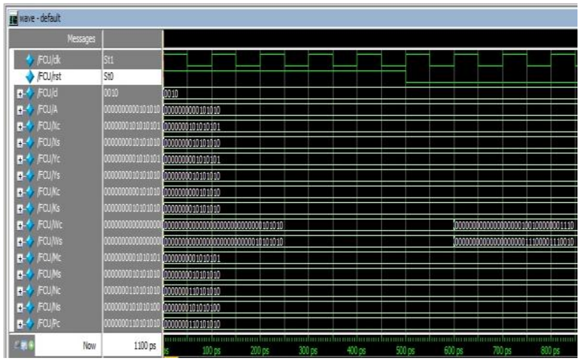


Fig. 6: output waveforms

Synthesis Results:

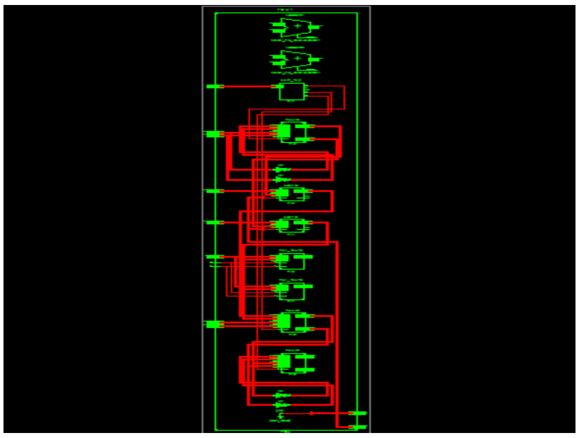
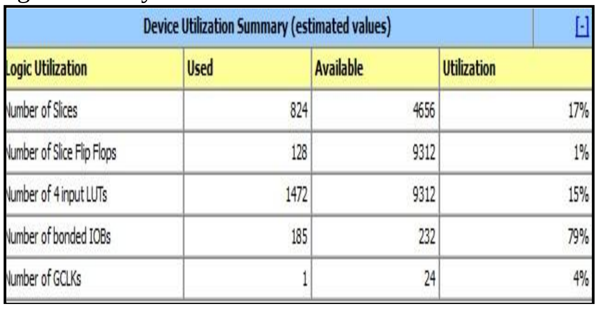


Fig. 7: RTL Schematic

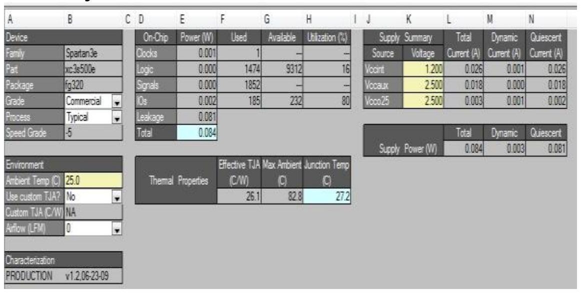
Design Summary:



Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slices	824	4656	17%
Number of Slice Flip Flops	128	9312	1%
Number of 4 input LUTs	1472	9312	15%
Number of bonded IOBs	185	232	79%
Number of GCLKs	1	24	4%

Fig. 8: CS adder and MB Multiplier based FCU device summary

Power Analysis



Device	
Family	Spartan3e
Part	xc3s500e
Package	fg320
Grade	Commercial
Process	Typical
Speed Grade	-5

On-Chip	Power (W)	Used	Available	Utilization (%)
Clocks	0.001	1	--	--
Logic	0.000	1474	9312	16
Signals	0.000	1852	--	--
IOs	0.002	185	232	80
Leakage	0.081			
Total	0.084			

Supply Summary		Total	Dynamic	Quiescent
Source	Voltage	Current (A)	Current (A)	Current (A)
Vccint	1.200	0.026	0.001	0.026
Vccaux	2.500	0.018	0.000	0.018
Vcco25	2.500	0.003	0.001	0.002

	Total	Dynamic	Quiescent
Supply Power (W)	0.084	0.003	0.081

Environment	
Ambient Temp (C)	25.0
Use custom TJA?	No
Custom TJA (C/W)	NA
Airflow (LFM)	0

Thermal Properties	Effective TJA (C/W)	Max Ambient (C)	Junction Temp (C)
	26.1	82.8	27.2

Characterization	
PRODUCTION	v1.2,06-23-09

Fig .9: CS adder and MB multiplier based FCU

V. PROPOSED PROJECT

Since area parameter is a major concern in the integrated circuits, so our extended project exploits dadda multiplier for multiplier in FCU. The internal operation in the dadda multiplier includes adders exploiting carry save arithmetic. In our project as stated earlier, the dadda multiplier is implemented in the multiplier unit of the operational template found in the data flow graph of the kernel of the DSP system.

FCU with Carry save Arithmetic and Dadda Multiplier Implementation

The Dadda multiplier is a hardware multiplier design invented by computer scientist Luigi Dadda in 1965. It is similar to the Wallace multiplier, but it is slightly faster (for all operand sizes) and requires fewer gates (for all but the smallest operand sizes). In fact, Dadda and Wallace multipliers have the same 3 steps: Firstly, multiply (logical AND) each bit of one of the arguments, by each bit of the other, yielding (n*n) results. Depending on the position of the multiplied bits, the wires carry different weights. Secondly, reduce the number of partial product to two by layers of full and half adders. Thirdly, group the wires in two numbers and add them with a conventional adder. Fig 11 demonstrates the reduction of partial products in a dadda multiplier.

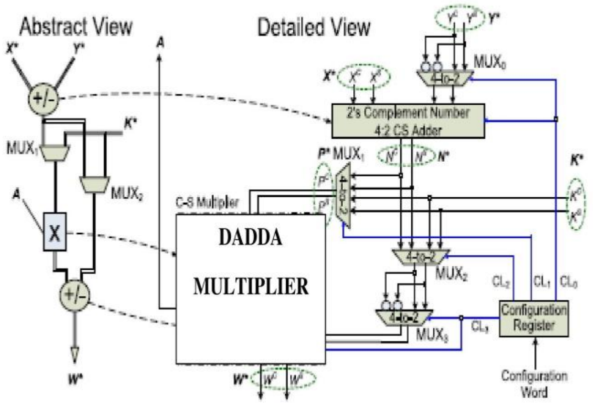


Fig.10: Implementation of carry-save technique for adders and dadda technique for multiplier in FCU.

Algorithm

- Multiply (that is - AND) each bit of one of the arguments, by each bit of the other, yielding N2 results.
- Reduce the number of partial products to two layers of full and half adders. For this, Dadda reduction scheme uses the following algorithm.
- Let d1 = 2 and dj+1 = [2.dj / 2], where dj is the matrix height for the j-th stage from the end. Find the largest j such that at least one column of the matrix has more than dj bits.
- Employ (3, 2) and (2, 2) counters to obtain a reduced matrix with no more than dj elements in any column. c)Until a matrix with only two rows is generated. Let j = j-1 and repeat step b
- Group the wires in two numbers, and add them with a conventional adder.

Table. 1 Number of reduction stages for DADDA multiplier

Bits in Multiplier(N)	Number of Stages
3	1
4	2
5 ≤ N≤ 6	3
7 ≤ N ≤ 9	4
10 ≤ N ≤ 13	5
14 ≤ N ≤ 19	6
20 ≤ N ≤ 28	7
29 ≤ N ≤ 42	8
43 ≤ N ≤ 63	9
63 ≤ N ≤ 94	10

The reduction rules however are as follows: Take any 3 wires with the same weights and input them into a full adder. The result will be an output wire of the same weight and an output wire with a higher weight for each 3 input wires. If there are 2 wires of the same weight left, and the current number of output wires with that weight is equal to 2 (modulo 3), input them into a half adder. Otherwise, pass them through to the next layer.

Flow Chart

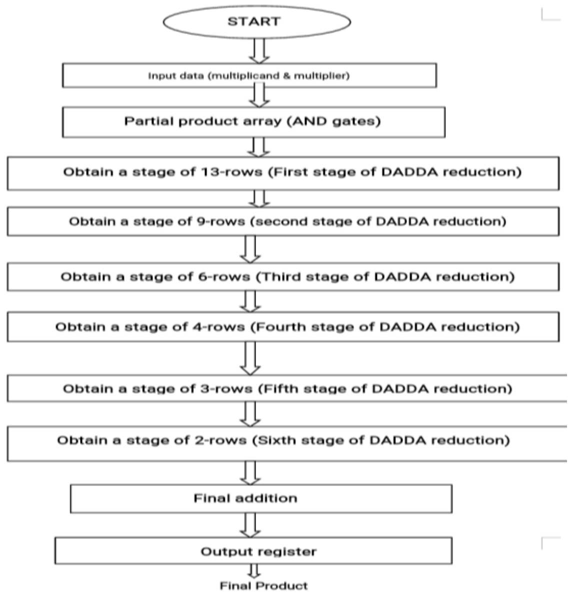

START
Input data (multiplicand & multiplier)
Partial product array (AND gates)
Obtain a stage of 13-rows (First stage of DADDA reduction)
Obtain a stage of 9-rows (second stage of DADDA reduction)
Obtain a stage of 6-rows (Third stage of DADDA reduction)
Obtain a stage of 4-rows (Fourth stage of DADDA reduction)
Obtain a stage of 3-rows (Fifth stage of DADDA reduction)
Obtain a stage of 2-rows (Sixth stage of DADDA reduction)
Final addition
Output register
Final Product

Fig. 11: Flow chart of 16x 16 DADDA Multiplier

If there is just 1 wire left, connect it to the next layer. This step does only as many adds as necessary, so that the number of output weights stays close to a multiple of 3, which is the ideal number of weights when using full adders as (3, 2) counters. However, when a layer carries at most 3 input wires for any weight, that layer will be the last one. In this case, the Dadda tree will use half adder more aggressively to ensure that there are only two outputs for any weight. Then, the second rule is above changes as follows If there are 2 wires of the same weight left, and the current number of output wires with that weight is equal to 1 or 2 (modulo 3), input them into a half adder. Otherwise, pass them through to the next layer. III Implementation of multiplier In order to make the most effective use of the processing elements, the multiplier was implemented as a linear pipeline. It was important to ensure that the delay of each processing stage in the pipeline was approximately equal so that a ‘bottleneck’ was not introduced by any individual processing stage. The multiplication of an M-bit multiplicand by an N-bit multiplier yields an N by M matrix of partial products. The reduction of this partial product matrix through the parallel application of (3, 2) and (2, 2) counters results in a matrix with a height of two. Each (3, 2) counter (full adder) accepts three inputs from a given column and produces a sum bit which remains in that column and a carry bit which goes into the next more significant column. A (2, 2) counter (half adder) accepts two inputs from a column and produces a sum bit in the same column and a carry bit in the next more significant column. The implemented 16 × 16 Dadda multiplier with the help of dot diagram is shown in Fig 12 (The notation is taken from in which the outputs from a full adder are joined by a solid line, and those from half adders are joined by a line with a dash through the centre). The Dadda scheme essentially minimizes the number of adder stages required to perform the summation of the partial products. This is achieved by using full and half adders to reduce the number of rows in the matrix of bits at each summation stage by a factor of 3/2. This results in a final matrix consisting of two rows of bits which must be summed using a multiple-bit adder (e.g. a ripple-carry or carry look ahead adder). By way of contrast, in a popular multiplication scheme the array, the summation proceeds in a more regular, but slower manner, to obtaining the summation of the partial products .Using this scheme only one row of bits in the matrix is eliminated at each stage of the summation. The process of Dadda multiplication is as follows: The entire 16 × 16 multiplication requires six stages. Always the first stage is partial products stage, which is obtained by simple multiplication of multiplicand with multiplier. The number of rows (height) present at this stage is 16. Now reduce the number of rows further in such a way that final stage contains only two rows. For this, Dadda introduces a sequence of intermediate matrix heights that provides the minimum number of reduction stages for a given size multiplier. This sequence determined by working back from the final two row matrix, limit the height of each intermediate matrix to the largest integer that is no more than 1.5 times the height of its successor. The proposed multiplier 16x16 Dadda multiplier requires six reduction stages with intermediate matrix heights of 13, 9,6,4,3 and finally 2. The single bit in 1st column of the first stage represents the least significant bit of the product. From the dot diagram, 2 – row stage can be derived from 3 – row stage, and 3 – row stage can be derived from 4 – row stage with the help of (3, 2) and (2, 2) counters. This is (S-1)th stage, where S is the number of stages to implement the multiplier. The 4 – row stage can be derived from 6 – row stage. This is (S-2)th stage. The 6 – row stage can be derived from 9 – row stage. This can be (S-3)th stage. The 9 – row stage can be derived from 13 – row stage.

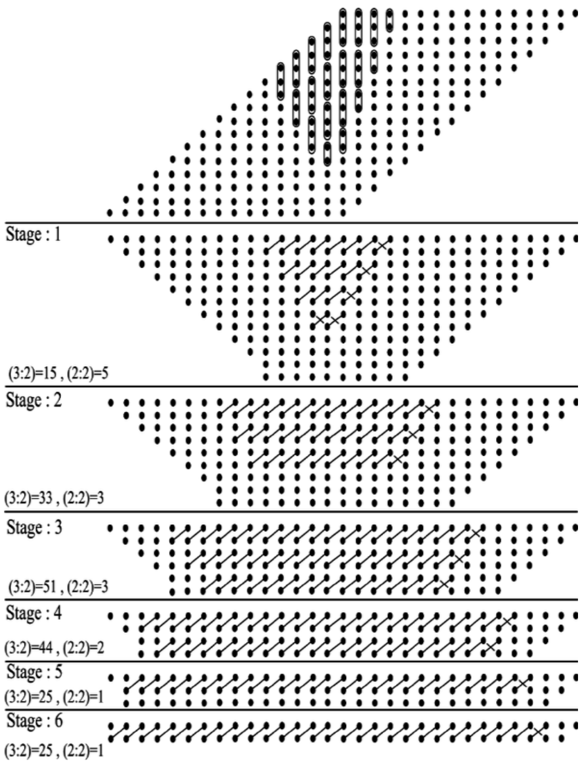


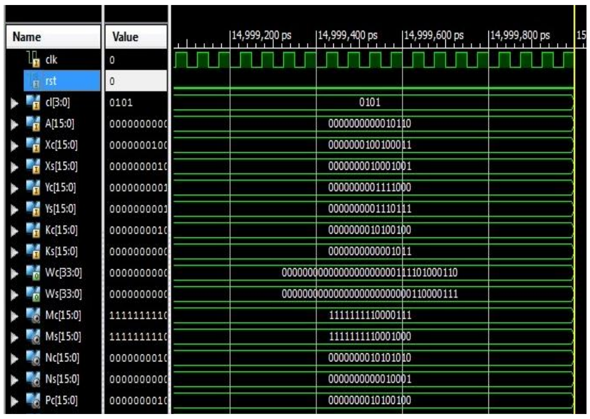
Fig.12: Example of Dadda reduction on 16x16 multiplier. Bits with lower weight are rightmost.

This is (S-4)th stage and then finally 13 – row stage can be derived from partial product stage. In passing from partial products stage to stage 1, columns are partially reduced, so that no more than 13 rows are obtained. From the dot diagram, column 14(14th bit) of partial products stage will be transformed in a 13 –bits column in stage 1 by reproducing 12 bits without transformation and transforming only 2 bits by (2, 2) counter.

Consequently, column 15 (15th bit and 14th bit) of the partial products stage will be transformed in a 13 – bits column in stage 1 by reproducing 12 bits without transformation and transforming only 2 bits by a (3, 2) counter with the help of the carry generated from the previous column. Consequently, only some columns in the central portion of partial products stage are actually transformed. In passing from stage 1 to stage 2, columns having no more than 9 bits are obtained by means of applying (2, 2) and (3,2) counters. In succeeding transformations, columns with no more than 6, 4, 3 and 2 bits respectively are obtained.

VI. RESULTS

Simulation Results:



```
Name        Value         |14,999,200 ps |14,999,400 ps |14,999,600 ps |14,999,800 ps |15
clk         0
rst         0
cl[3:0]     0101          0101
A[15:0]     000000000C    0000000000010110
Xc[15:0]    000000010C    0000000100100011
Xs[15:0]    000000001C    0000000010001001
Yc[15:0]    0000000001    0000000001111000
Ys[15:0]    0000000001    0000000001110111
Kc[15:0]    000000001C    0000000010100100
Ks[15:0]    000000000C    0000000000001011
Wc[33:0]    000000000C    0000000000000000000000111101000110
Ws[33:0]    000000000C    0000000000000000000000000110000111
Mc[15:0]    111111111C    1111111110000111
Ms[15:0]    111111111C    1111111110001000
Nc[15:0]    000000001C    0000000010101010
Ns[15:0]    000000000C    0000000000010001
Pc[15:0]    000000001C    0000000010100100
```

Fig. 13: Simulation of FCU using DADDA

Synthesis Results:

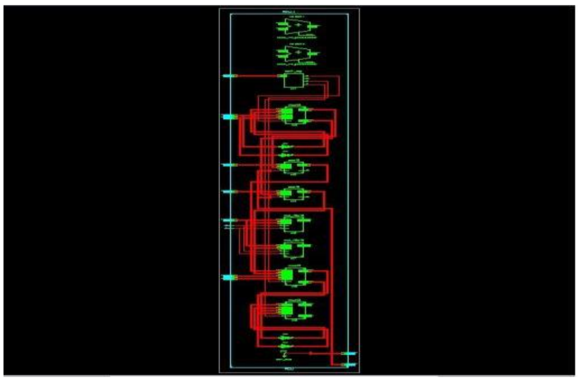
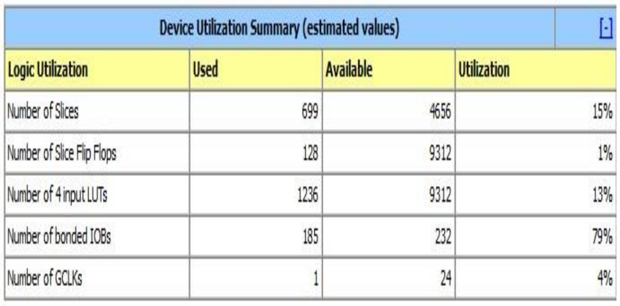


Fig. 14 RTL Schematic

Design Summary:

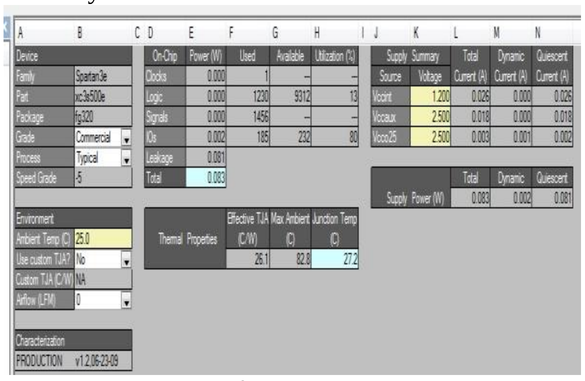


Device Utilization Summary (estimated values)

Logic Utilization	Used	Available	Utilization
Number of Slices	699	4656	15%
Number of Slice Flip Flops	128	9312	1%
Number of 4 input LUTs	1236	9312	13%
Number of bonded IOBs	185	232	79%
Number of GCLKs	1	24	4%

Fig. 15: CS adder and DADDA Multiplier based FCU device summary

Power Analysis:



Device	
Family	Spartan3e
Part	xc3s500e
Package	fg320
Grade	Commercial
Process	Typical
Speed Grade	-5

On-Chip	Power (W)	Used	Available	Utilization (%)
Clocks	0.000	1	--	--
Logic	0.000	1230	9312	13
Signals	0.000	1456	--	--
IOs	0.002	185	232	80
Leakage	0.081			
Total	0.083			

Supply Source	Summary Voltage	Total Current (A)	Dynamic Current (A)	Quiescent Current (A)
Vccint	1.200	0.026	0.000	0.026
Vccaux	2.500	0.018	0.000	0.018
Vcco25	2.500	0.003	0.001	0.002

	Total	Dynamic	Quiescent
Supply Power (W)	0.083	0.002	0.081

Environment	
Ambient Temp (C)	25.0
Use custom TJA?	No
Custom TJA (C/W)	NA
Airflow (LFM)	0

Thermal Properties	Effective TJA (C/W)	Max Ambient (C)	Junction Temp (C)
	26.1	82.8	27.2

Characterization	
PRODUCTION	v1.2,06-23-09

Fig. 16 Power analysis of CS adder and DADDA multiplier based FCU

Comparisons between MB multiplier and DADDA multiplier FCU

Table. 2 Comparison Table for FCU

FCU unit	No. of LUT'S	Power (in mw)
Existing FCU unit	1472	84
Proposed FCU unit	1236	83

VII. CONCLUSION

Concisely, an FCU architecture was introduced that exploits the incorporation of CS arithmetic and Dadda algorithmic optimizations to enable fast chaining of additive and multiplicative operations. This flexible accelerator architecture allows to operate on both conventional two's complement and CS formatted data operands, thus enabling high degrees of computational density to be achieved. Theoretical and experimental analyses have shown that the extended solution forms an efficient design delivering considerable amount in terms of area and minute reduction in power. Even if the area required is less it would not have its affect on speed when compared to the MB multiplier, these advantages can be efficiently utilized in applications such as ALUs for designing advanced high speed microprocessors and DSP systems as well.

VIII. FUTURE ASPECT

Flexible accelerator architecture is able to operate on both conventional two's complement and CS-formatted data. As it is the source for performing all ALU operations in kernels of the DSP system, as efficient it is the better is the performance of the DSP system. Though these systems are widely used for a limited range of 16 bit data in many applications, these can further increase to more number of bits i.e., up to 64-bits. Further, the improved multiplier can also be used for designing the low power multi-tap FIR filters in DSP applications with suitable tools.

REFERENCES

[1] P. Ienne and R. Leupers, Customizable Embedded Processors: Design Technologies and Applications. San Francisco, CA, USA: Morgan Kaufmann, 2007.

[2] B. Mei, S. Vernalde, D. Verkest, H. D. Man, and R. Lauwereins, "ADRES: An architecture with tightly coupled VLIW processor and coarse-grained reconfigurable matrix," in Proc. 13th Int. Conf. Field Program. Logic Appl., vol. 2778. 2003, pp. 61–70.

[3] P. M. Heysters, G. J. M. Smit, and E.Molenkamp,

"A flexible and energy-efficient coarse-grained reconfigurable architecture for mobile systems," J. Supercomput., vol. 26, no. 3, pp. 283–308, 2003.
[4] A. Hosangadi, F. Fallah, and R. Kastner, "Optimizing high speed arithmetic circuits using three-term extraction," in Proc. Design, Autom. Test Eur. (DATE), vol. 1. Mar. 2006, pp. 1–6.
[5] K. Compton and S. Hauck, "Automatic design of reconfigurable domainspecific flexible cores," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 16, no. 5, pp. 493–503, May 2008.
[6] 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.
[7] S. Xydis, G. Economakos, D. Soudris, and K. Pekmestzi, "High performance and area efficient flexible DSP datapath synthesis," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 19, no. 3, pp. 429–442, Mar. 2011.
[8] G. Ansaloni, P. Bonzini, and L. Pozzi, "EGRA: A coarse grained reconfigurable architectural template," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 19, no. 6, pp. 1062–1074, Jun. 2011.
[9] M. Stojilovic, D. Novo, L. Saranovac, P. Brisk, and P. Ienne, "Selective flexibility: Creating domain-specific reconfigurable arrays," IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 32, no. 5, pp. 681–694, May 2013.
[10] T. Kim and J. Um, "A practical approach to the synthesis of arithmetic circuits using carry-save-adders," IEEE Trans. Comput.- Aided Design Integr. Circuits Syst., vol. 19, no. 5, pp. 615–624, May 2000.
[11] A. Hosangadi, F. Fallah, and R. Kastner, "Optimizing high speed arithmetic circuits using three-term extraction," in Proc. Design, Autom. Test Eur. (DATE), vol. 1. Mar. 2006, pp. 1–6.