e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 05 Issue 15
May 2018

Flexible DSP Accelerator Architecture Exploiting Carry-Save Arithmetic

Dr.Ch.Ravi Kumar & M.Tejaswini

[1] Prof & HOD
Department of ECE
Prakasam Engineering College
Kandukur (m), Prakasam(dt),A.P,India.
kumarece0@gmail.com

[2]M.Tech scholar
Department of ECE
Prakasam Engineering College
Kandukur (m), Prakasam(dt),A.P,India
tejaswini.mutyala@gmail.com

ABSTRACT: *Hardware acceleration has been proved an extremely promising implementation strategy for the digital signal processing (DSP) domain. Rather than adopting a monolithic application-specific integrated circuit design approach, in this brief, we present a novel accelerator architecture comprising Flexible computational units that support the execution of a large set of operation templates found in DSP kernels. We differentiate from previous works on flexible accelerators by enabling computations to be aggressively performed with carry-save (CS) formatted data. Advanced arithmetic design concepts, i.e., recoding techniques, are utilized enabling CS optimizations to be performed in a larger scope than in previous approaches.*

I INTRODUCTION

Modern embedded systems target high-end application domains requiring efficient implementations of computationally intensive digital signal processing (DSP) functions. Digital signal processing (DSP) refers to various techniques for improving the accuracy and reliability of digital communications.

The theory behind DSP is quite complex. Basically, DSP works by clarifying, or standardizing, levels or states of a digital signal. The incorporation of heterogeneity through specialized hardware accelerators improves performance and reduces energy consumption. 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 and vary the type and number of computation units achieving a customized design structure. Flexible architectures were proposed exploiting ILP and operation chaining. Recently, Ansaloni et al. adopted aggressive operation chaining to enable the computation of entire sub expressions using multiple ALUs with heterogeneous arithmetic features.

II EXISTING SYSTEM

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 the existing methods vary the type and number of computation units achieving a customized design structure. Flexible architectures were proposed exploiting ILP and operation chaining. Recently,

Ansaloni adopted aggressive operation chaining to enable the computation of entire sub expressions using multiple ALUs with heterogeneous arithmetic features. 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.

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

III PROPOSED SYSTEM

The Flexible accelerator architecture is shown in the following slide. Each FCU operates directly on CS operands and produces data in the same form1 for direct reuse of intermediate results. Each FCU operates on 16-bit operands. Such a bit length is adequate for the most DSP data paths, 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 ILP and area constraints imposed by the designer. The CS to Bin module is a ripple carry adder and converts the CS form to the two's complement one. 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 overall 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.

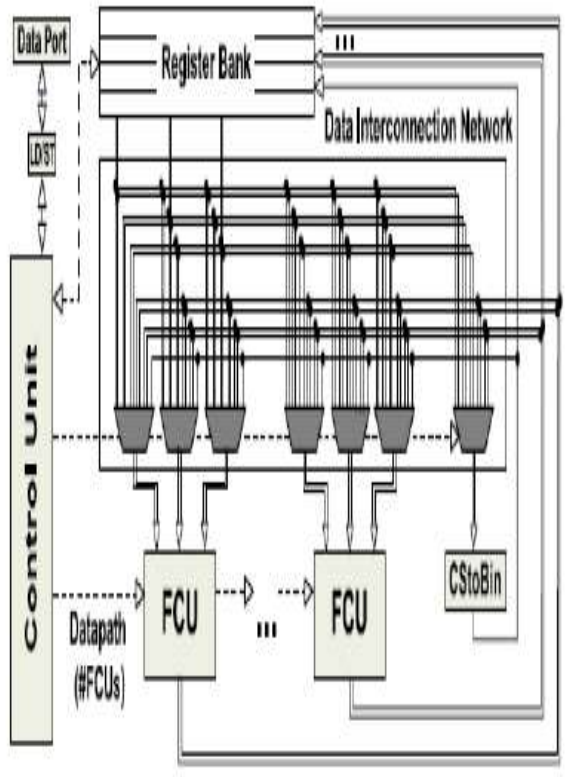


Fig.1: Abstract form of the flexible datapath

Structure of the Proposed Flexible Computational Unit

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 shown in Fig. 3. The proposed FCU enables intra template operation chaining by fusing the additions performed before/after the multiplication and performs any partial operation template of the following complex operations:

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

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

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. TheMUX2 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.

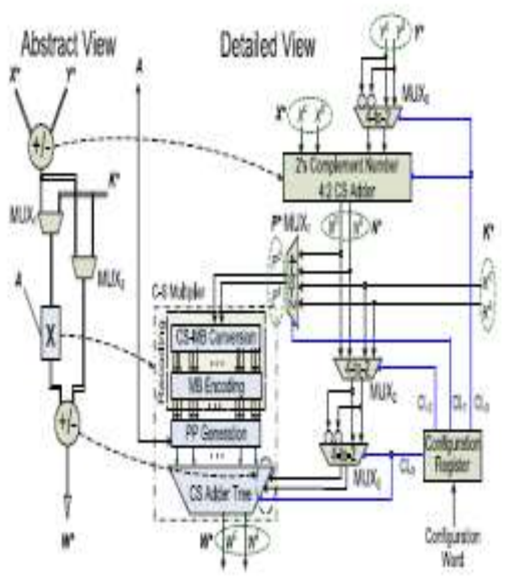


Fig.2 FCU.

Each FCU can be configured to any of the T1–T5 operation templates shown in fig.2. The proposed FCU enables intra template operation chaining by fusing the additions performed before/after the multiplication and performs any partial operation. The multiplier comprises a CS-to- MB module, which adopts are 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.

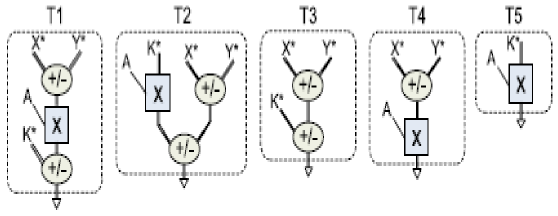
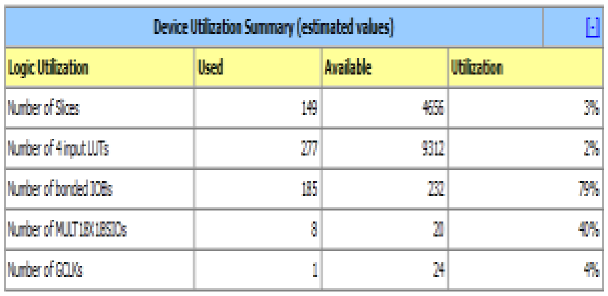


Fig.3. FCU template library.

IV RESULTS

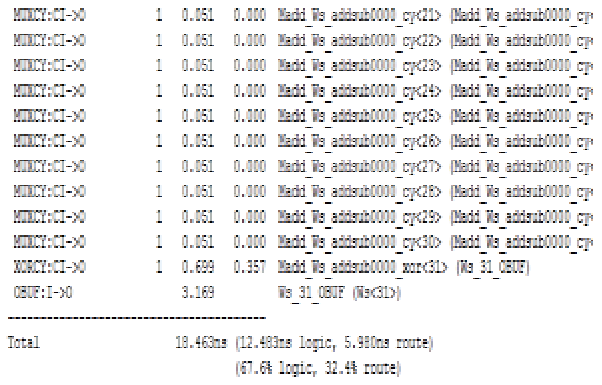
SIMULATION RESULTS:

DESIGN SUMMARY:



Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slices	149	4656	3%
Number of 4 input LUTs	277	9312	2%
Number of bonded IOBs	185	232	79%
Number of MULT18X18SIOs	8	20	40%
Number of GCLKs	1	24	4%

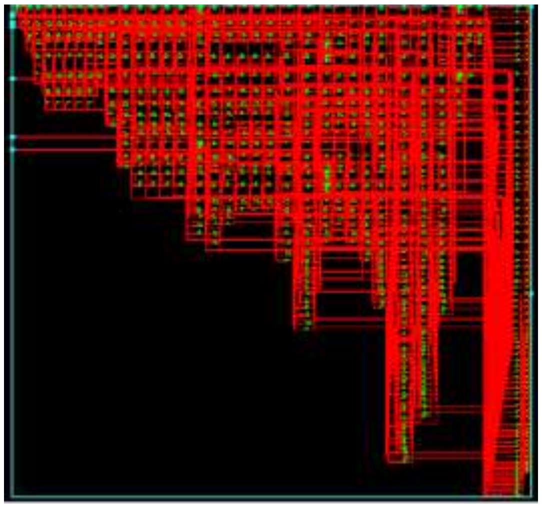
TIMING REPORT:



```
MUXCY:CI->O          1   0.051   0.000  Madd_Ws_addsub0000_cy<21> (Madd_Ws_addsub0000_cy
MUXCY:CI->O          1   0.051   0.000  Madd_Ws_addsub0000_cy<22> (Madd_Ws_addsub0000_cy
MUXCY:CI->O          1   0.051   0.000  Madd_Ws_addsub0000_cy<23> (Madd_Ws_addsub0000_cy
MUXCY:CI->O          1   0.051   0.000  Madd_Ws_addsub0000_cy<24> (Madd_Ws_addsub0000_cy
MUXCY:CI->O          1   0.051   0.000  Madd_Ws_addsub0000_cy<25> (Madd_Ws_addsub0000_cy
MUXCY:CI->O          1   0.051   0.000  Madd_Ws_addsub0000_cy<26> (Madd_Ws_addsub0000_cy
MUXCY:CI->O          1   0.051   0.000  Madd_Ws_addsub0000_cy<27> (Madd_Ws_addsub0000_cy
MUXCY:CI->O          1   0.051   0.000  Madd_Ws_addsub0000_cy<28> (Madd_Ws_addsub0000_cy
MUXCY:CI->O          1   0.051   0.000  Madd_Ws_addsub0000_cy<29> (Madd_Ws_addsub0000_cy
MUXCY:CI->O          1   0.051   0.000  Madd_Ws_addsub0000_cy<30> (Madd_Ws_addsub0000_cy
XORCY:CI->O          1   0.699   0.357  Madd_Ws_addsub0000_xor<31> (Ws_31_OBUF)
OBUF:I->O                3.169          Ws_31_OBUF (Ws<31>)
----------------------------------------
Total                    18.463ns (12.483ns logic, 5.980ns route)
                                  (67.6% logic, 32.4% route)
```

RTL SCHEMATIC:

TECHNOLOGY SCHEMATIC:



V CONCLUSION

In this brief, we introduced a Flexible accelerator architecture that exploits the incorporation of CS arithmetic optimizations to enable fast chaining of additive and multiplicative operations. The proposed Flexible accelerator architecture is able 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 proposed solution forms an efficient design tradeoff point delivering optimized latency/area and energy implementations.

REFERENCES

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

[2] 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.

[3] 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.

[4] 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. Circuits Syst., vol. 25, no. 6, pp. 1154–1162, Jun. 2006.

[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.

e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 05 Issue 15
May 2018

[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] R. Kastner, A. Kaplan, S. O. Memik, and E. Bozorgzadeh, "Instruction generation for hybrid reconfigurable systems," ACM Trans. Design Autom. Electron. Syst., vol. 7, no. 4, pp. 605–627, Oct. 2002.

[11] [Online]. Available: http://www.synopsys.com, accessed 2013.

[12] 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.