Reconfigurable Universal SAD-Multiplier Array

Humberto Calderon
H.Calderon@ewi.TUDelft.nl

Stamatis Vassiliadis
S.Vassiliadis@ewi.TUDelft.nl

Computer Engineering Laboratory
Faculty of Electrical Engineering,Mathematics and Computer Science
Delft University of Technology
The Netherlands
P.O. Box 5031, 2600 GA Delft
Phone: +31 15 2787362, Fax: +31 15 2784898.

ABSTRACT

In this paper, we investigate the collapsing of some multi-operand addition related operations into a single array. More specifically we consider multiplication and Sum of Absolute Differences (SAD) and propose an array capable of performing the aforementioned operations for unsigned, signed magnitude, and two's complement notations. The array, called a universal array, is divided into common and controlled logic blocks intended to be reconfigured dynamically. The proposed unit was constructed around three main operational fields, which are feed with the necessary data products or SAD addition terms in order to compute the desired operation. It is estimated that a 66.6 % of the (3:2)counter array is shared by the operations providing an opportunity to reduce reconfiguration times. The synthesis result for a FPGA device, of the new structure, was compared against other multiplier organizations. The obtained results indicate that the proposed unit is capable of processing in 23.9 ns a 16 bit multiplication, and that an 8 input SAD can be computed in 29.8 ns using current FPGA technology. Even though the proposed structure incorporates more operations, the extra delay required over conventional structures is very small (in the order of 1% compared to Baugh&Wooley multiplier).

Categories and Subject Descriptors

B.2 [**High Speed Arithmetic**]

General Terms

Design,Experimentation

Keywords

Binary Multiplication, Sum of Absolute Differences, Reconfigurable Computing, Partial Reconfiguration.

CF'05, May 4–6, 2005, Ischia, Italy.

1. INTRODUCTION

Universal units perform operations on the same hardware for commonly used number notations (in this paper assumed to be sign magnitude, two's complement and unsigned). For an example of such units designed to accommodate multiple instruction types for multiply operations see [11]. Multimedia instruction set architectures (ISA), provide new demands on multi-operand addition related operations. They require square arrays to be added (see sum of absolute values) in the design of a processor rather than multiply arrays. Furthermore, slow dynamic reconfigurations require that when designing multimedia reconfigurable extensions [10] [4], special attention has to be paid to the reconfiguration of arithmetic units dynamically. It is desirable that a part of hardware that is common to numerous operations to be reconfigured in advance and those "differences" rather than the entire unit is loaded for reconfiguration. This work reintroduces universal units, to address additional complexities imposed by multi-operand operations that require rectangular arrays (see SAD implementations e.g.[8],[5],[6],[12]). We consider multimedia reconfigurable units, multiple number integer representations, multiplication and SAD performed by a single multi-operand hardware structure. We assume unsigned, signed magnitude and two's complement number notation, and multiplier design that can incorporate SAD producing a universal array that is targeting partial dynamic reconfigurable units. The unit's features include the following:

- A 16 x 16 multiplier, and an 8 inputs pairs of SAD operands in the same universal array.
- A performance of 23.9 ns for processing a 16 bit multiplication and a 29.88 ns 8 input SAD, for unsigned, signed magnitude and two's complement representations.
- It is estimated that 2/3 of the resources are shared by all functionalities. This sharing could be of benefit when reconfiguring dynamically based of frames differences.

The paper is organized as follows. Section 2, outlines the *Reconfigurable-Universal-Sum-Absolute-Differences-Multiplier-Array* organization. Section 3 presents the experimental results of the mapped unit, as well as other well know multiplier organizations in order to be compared in terms of area used and time delay. Finally, the article is concluded in Section 4.

2. THE RECONFIGURABLE ARRAY

In this section we begin by providing a background for SAD, and then presenting the proposed scheme by describing the array structure.

2.1 Background

We begin by introducing for a background purpose the SAD operation. SAD is a common way to compute differences between images to explore temporal redundancies. Commonly the absolute difference is computed by performing a subtraction operation of two pels values, calculating the absolute value of this result, repeating this calculus for the entire pels of the chosen block, and finally adding the obtained differences. The following equation computes the SAD.

$$\sum_{j=1}^{16}\sum_{i=1}^{16}|IN1(x+i,y+j)-IN2((x+r)+i,(y+s)+j)| \quad (1)$$

where, the tuple (x, y) represents the position of the current block, and (r, s) denotes the displacement of $IN2$, relative to reference block $IN1$. Given that SAD's performance is critical, multimedia instructions set have incorporated special SAD instructions and numerous schemes have been proposed to speed up SAD operations (for examples, take a look at [1],[9],[13],[7]).

In this presentation we assume the scheme proposed in [12], because it can be used for merged operations. This is true because it separates the multi-operand addition from the determination of which operand should be subtracted to produce in parallel the sum of absolute values. Details follow in the matrix description.

2.2 The Reconfigurable Array Organization

In our presentation we assume 16 bit integer numbers. With appropriate considerations our approach can be extended to any desired length. We derived a universal unit for a multiplier and a multi-operand addition of non equal lengths as depicted in figure 1.

As indicated early, we consider three number representations, unsigned, signed magnitude and two's complement. We note that when considering the array each symbol represents a (3:2)counter. The array is enumerated from 0 to 31 columns, with 0 to 14 rows, the first row, denoted by the black circles (see figure 1), receives two products corresponding to the first two bits of the multiplier, the rest 14 multiplier bits are accommodated in the remaining 14 rows of the array. The array is logically divided in three sectors denoted by the numbers 1, 2 and 3.

The universal multiplier part of the presented unit is constructed around a hardwired multiplier composed by the following components:

1. Partial product (PC) generation; $PC_{(j,i)} = X_i \cdot Y_j$, for a $X = X_{(15)}...X_{(0)}$, and $Y = Y_{(15)}...Y_{(0)}$.
2. The partial product addition, based on a (3:2)counter organization, and
3. Final adder with some propagation technique.

The accommodation of multiple operation types into the array is accomplished with changes in the sign value of the operands and with the signed extension used to process the two's complement representation. Table 1 presents the various extensions needed to properly perform multiplication for all the three notations. In unsigned and two's complement notations, no changes are introduced in the most significant bits achieved (via sign extension by 0). For signed magnitude, the most significant bit, corresponding to the sign position, is cleared and the sign of the final result is updated with X_{n-1} *XOR* Y_{n-1}. The two's complement notation uses a sign extension along the section 3 as can be seen in figure 1, whereas the unsigned and signed magnitude are extended with a zero.

Table 1: Universal Multiplier Extensions

	X_{n-1}	Y_{n-1}	Extension
Unsigned	X_{n-1}	Y_{n-1}	0
Signed Magnitude	0	0	0
Two's complement	X_{n-1}	Y_{n-1}	$Sign = Y_{(m)} \cdot X_{(m-1)}$

$\forall \;\; 0 \leq m \leq 15$

The SAD calculation can be decomposed into two logical steps. In order to perform a universal operation we need to produce a positive result[1] to achieve such a goal, we proceed as follows:

1. We note that the SAD inputs are positive integers. The first step determine which of the operands is the greatest, so that the smallest is subtracted in the array to produce the absolute values. The logic required to perform this kind of distinction is haded on the carry out of addition of one operand $IN1$ and the inverted operand $IN2$. The carry out indicates if $IN1$ or $IN2$ is the greatest. This is true because of the following:

   a: The sign bit for sign magnitude and two's complement is 0, (inputs to SAD are positive) thus they are equal.

   b: For all the bits for the unsigned numbers and the magnitude of the signed magnitude and two's complement numbers for the first operand $IN1$ to be greater, then the second operand $IN2$ must be that all most significant bits are equal (sign included, see a:, this is to make equal length operands and to have the same carry circuitry for all notations) and that there is a bit position on i such that the bit value of $IN1$ is 1 and $IN2$ is 0. The reason is the following: if the bit position is i, then starting at position i and ending at position 0, $IN1 = 2^i$ is the worst case (the rest of the bit starting at i-1 are all 0) and $IN2 = 2^i - 1$ is the best case (all remaining least significant bits are 1). Consequently, when inverting $IN2$ all most significant bits starting from $i + 1$ have opposite values for both $IN1$ and $IN2$ and at the bit position i, $IN1_i = 1$ and $IN2_i = 1$, implying that at position i a carry will be generated and it will be transmitted out, consequently $carryout = 1$. If $IN2$ is greater, then $IN1_i = 0$ and $IN2_i = 1$. Thus $\overline{IN2_i} = 0$ and a potential carry from the

[1]All representations have the same representation for positive numbers and unsigned notation numbers and can be viewed as positive numbers with an implicit 0 sign-bit.

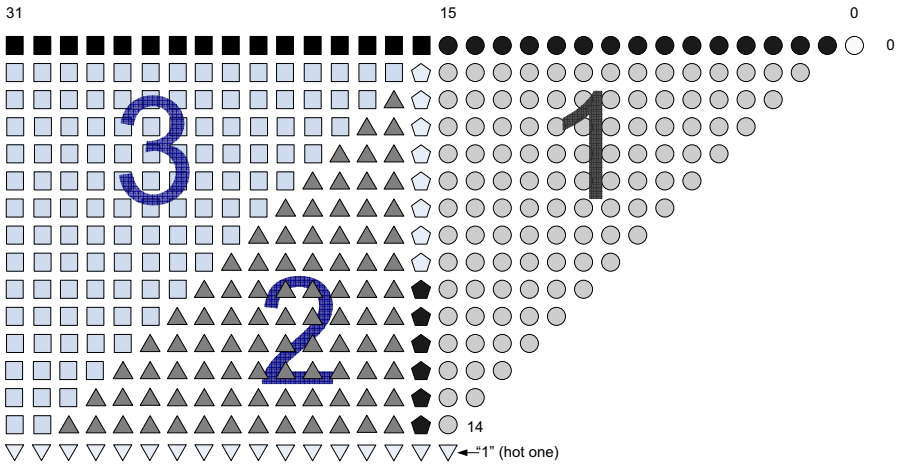

Figure 1: The universal multiply SAD array

least significant bit is killed. Also because the most significant bits starting at position $i+1$ have opposite values there is no generated carry thus ($carryout = 0$). If $IN1$ and $IN2$ are equal then the carry out is zero.

2. The second step creates the array for multi-operand additions starting from bit enumerated 16 to bit 31 inclusive, the $IN1$ and $IN2$ are placed from the carry unit described above, corresponding to the sections 2 and 3 of figure 1. This part of the array receives 16 inputs of 16 bits width each one, in order to process concurrently the half-block of a 4x4 SAD operations. Equation 2, states with the index j the 16 inputs, and i is used to denote the positional weight of the data bits in each input.

$$IN_{(j,i)} = IN_{(j,15)} \cdots\cdots IN_{(j,1)} IN_{(j,0)} \quad \forall\ 1 \leq j \leq 16 \quad (2)$$

Figure 2, details the different kinds of logical blocks used in the implementation, some of those blocks utilize multiplexors to control the operation related data. As indicated earlier each block in figure 1 is a (3:2)counter and the bar presented in figure 2 represents a multiplexor. Furthermore, it has to be noticed that the clear circle presented in the first row of figure 1, represents the product $X_{(0)} \cdot Y_{(0)}$

A concise description of logic block index ranges of the elements presented in figure 2 are depicted on table 2; this table presents a nested loop of $\triangle i$ for a $\triangle j$ variation, using an index j for a row position, and index i is used to reference the column variations.

Finally to describe the logical equations for the blocks in figure 2, we consider that the counters have the followings inputs $I1(j,i)$, $I2(j,i)$ and $I3(j,i)$ for the ranges shown in figure 1, and produce two outputs $S(j,i)$ that correspond to the sum, and $C(j,i)$ for the carry output of the (3:2)counter, signal e_i controls the multiplexer in order to feed with correct data the (3:2)counters. The logic equations describing the behavior of the logic blocks are as follows:

(a): $\forall\ \ 1 \leq i \leq 15$:
$I1_{(0,i)} = PC_{(0,i)}$,
$I2_{(0,i)} = PC_{(1,i-1)}$,
$I3_{(0,i)} = 0$

(b): Let $n = 2$; $\forall\ \ 1 \leq j \leq 14$ and $\forall\ \ n \leq i \leq 15$:
$I1_{(j,i)} = PC_{(j+1,i)}$,
$I2_{(j,i)} = S_{(j,i)}$,
$I3_{(j,i)} = C_{(j,i)}$;
with $n = n + 1$ for each $\Delta\ j$

(c): $\forall\ \ 1 \leq j \leq 8$:
$I1_{(j,16)} = IN_{(j+3,0)} \cdot e_0 + PC_{(j+1,15-j)} \cdot e_1$,
$I2_{(j,16)} = S_{(j,16)}$,
$I3_{(j,16)} = 1 \cdot e_0 + C_{(j,16)} \cdot e_1$

(d): $\forall\ \ 9 \leq j \leq 14$:
$I1_{(j,16)} = IN_{(j+3,0)} \cdot e_0 + PC_{(j+1,15-j)} \cdot e_1$,
$I2_{(j,16)} = S_{(j,16)}$,
$I3_{(j,16)} = 0 \cdot e_0 + C_{(j,16)} \cdot e_1$

(e): Let $n = 1$; $\forall\ \ 2 \leq j \leq 14$, and $\forall\ \ 1 \leq i \leq n$:

$$I1_{(j,i+16)} = IN_{(j+3,i)} \cdot e_0 + PC_{(j+1,15-j+i)} \cdot e_1 + PC_{(j+1,15-j+i)} \cdot e_2 + PC_{(j+1,15-j+i)} \cdot e_3$$

$I2_{(j,i+16)} = S_{(j,i+16)}$,
$I3_{(j,i+16)} = C_{(j,i+16)}$
with $n = n + 1$ for each $\Delta\ j$

(f): $\forall\ \ 0 \leq i \leq 15$:

$$I1_{(0,i+16)} = IN_{(1,i)} \cdot e_0 + PC_{(i+1,15)} \cdot e_1 + 0 \cdot e_2 + PC_{(0,15)} \cdot e_3,$$

$$I2_{(0,i+16)} = IN_{(2,i)} \cdot e_0 + 0 \cdot e_1 + PC_{(i+1,15)} \cdot e_2 + PC_{(1,15)} \cdot e_3$$

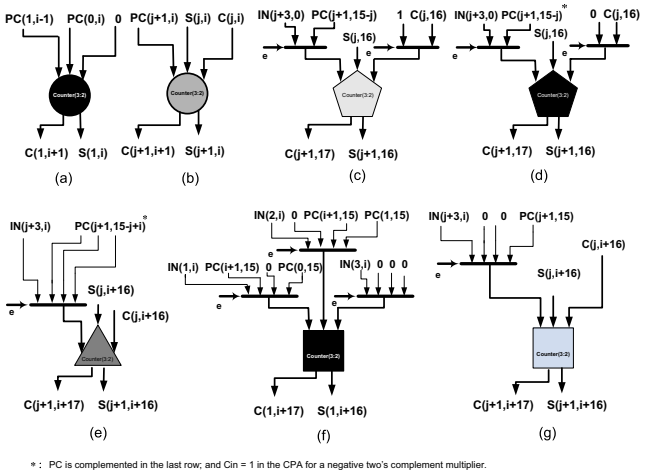

Figure 2: (counters(3:2) of the universal array)

Table 2: Index ranges for Fig 2. (3:2)counters of the universal array

Counter Type	Init Value	$\triangle i$	$\triangle j$	Operation	# CSA
(a)	-	1 to 15	-	MUL	15
(b)	n = 2	n to 15	1 to 14	MUL	105
(c)	-	-	1 to 8	MUL-SAD	8
(d)	-	-	9 to 14	MUL-SAD	6
(e)	n = 1	1 to n	2 to 14	MUL-SAD	91
(f)	-	0 to 15	-	MUL-SAD	16
(g)	n = 1	n to 15	1 to 14	MUL-SAD	119

$n = n + 1$ for each $\triangle j$

$I3_{(0,i+16)} = IN_{(3,i)}$

(g): Let $n = 1$; $\forall \quad n \leq i \leq 15$ and $\forall \quad 1 \leq j \leq 14$:
$I1_{(j,i+16)} = IN_{(j+3,i)} \cdot e_0 + 0 \cdot e_1 + 0 \cdot e_2 + PC_{(j+1,15)} \cdot e_3$,
$I2_{(j,i+16)} = S_{(j,i+16)}$,
$I3_{(j,i+16)} = C_{(j,i+16)}$;
with $n = n + 1$ for each Δj

The following must also be consider for the correct processing of two's complement representation:

$$PC(15, i) = PC(15, i)\ XOR\ PC(15, 15)$$

and $Cin = 0\ XOR\ PC(15, 15)$; this is necessary to to produce "hot one" addition effect needed for two's complement correction.

3. EXPERIMENTAL RESULTS

The universal array and the carry unit were implemented using VHDL, synthesized, functionally tested, and proved with the ISE 5.2 Xilinx environment [2], for the VIRTEX II PRO FPGA device. Furthermore, an unsigned array multiplier, and a Baugh and Wooley (BW) signed two's complement multiplier [3] were described and synthesized also with the same tools for comparison reasons. For all implementations we use a carry ripple adder. We have also implemented and synthesized parallel additions and the use of the fast carry support of the Xilinx technology. Table 3, summarizes the performance in time of these structures.

Table 3: Universal array and other multiply Units. (Time Delay)

Unit	Logic	Wire	Total
Unsigned M. ‡	14.589 ns 49.8%	14.639 ns 50.2%	29.282 ns 100%
Baugh Wooley ‡	15.555 ns 49.6%	15.826 ns 50.4%	31.381 ns 100%
our proposal ‡	15.877 ns 50.2%	15.741 ns 49.8%	31.618 ns 100%
our proposal§ with CLA	16.112 ns 54.2%	13.603 ns 45.8%	29,715 ns 100%
our proposal (RCA-Xilinx)	14.311 ns 59.9%	9.568 ns 40.1%	23.879 ns 100%
carry unit	2.576 ns 43.6 %	3.338 ns 56.4 %	5.914 ns 100 %

‡ : RCA as a final adder; LUT based implementation
§ CLA: Carry Lookahead Adder as final adder; LUT implementation

It can be noticed that the proposed array incorporates additional logic and it is expected to perform somehow slower than the other multiplier units. It is observed than both, our proposal and Baugh&Wooley's proposal for two's complement numbers, is a bit slower than the unsigned multiplier. There are negligible differences in timing between our universal array and the Baugh&Wooley's two's complement multiplier.

The time delay introduced by the multiplexors used to feed the (3:2)counters in order to perform the four operations has a constant delay for all the block of the array; therefore, from the obtained results it is evident that the routing delay is still significant in reconfigurable devices. Additionally, regarding the carry logic required for SAD, it is observed that 5.914 ns are needed for the processing.

Concerning the silicon used by the proposed unit and the other structures depicted on table 4,as expected, the added functionalities of the proposed unit are larger.

Table 4: Universal array and other multiply Units. (Hardware Use)

Unit	# Slices	# LUTs	# IOBs
Unsigned M. ‡	300	524	64
Baugh & Wooley ‡	330	574	65
our proposal ‡	686	1198	322
our proposal§ with CLA	711	1244	322
our proposal with RCA-Xilinx	658	1170	322
carry unit	35	61	64

‡: RCA as a final adder; LUT based implementation
§ CLA: Carry Lookahead Adder as final adder; LUT implementation

The carry unit consumes for routing the correct operands into the universal array a considerable hardware as is depicted on table 4; the 16 data entry will consume 560 slices, which represent an 81 % of the slices used in the proposed unit. In spite of that, this constitutes only 1 % of the VIRTEX II PRO resources device.
Regarding dynamic reconfigurations it is of interest to partially set the reconfiguration and then to reconfigure the differences rather than the entire structure. Our calculations indicate that a 66.6 % of the array is common. We estimated the reconfiguration frames difference between the functionalities could be around 50 % using partially reconfiguration, improving in this way the over complete reconfigurations.

4. CONCLUSIONS

A detailed algorithmic description for the creation of a universal unit for the processing of the Sum of Absolute Differences and a multiplier operating with unsigned, signed magnitude and two's complement representation has been presented. A brief analysis of the time cost associated to the implementation indicates that the proposed approach reveals a similar performance in terms of time delay when compared with multiplication schemes.

The implementation of the proposed approach indicates that a 49.8 % of delay is wire delay, showing that current technology wires inflict a considerable amount of delay. The proposed unit is capable to process an 8 pairs input SAD operation in 29.8 ns; and takes 23.9 ns for a 16 bit multiplication using current FPGA technology. Our approach, when compared to multiplier for two's complement notation, shows that it requires negligible extra delays for the universal array that computes the summation of absolute values and multiplication for unsigned, two's complement and sign magnitude integer numbers. Our preliminary experimental results, presents that instead of reconfigure the hole unit into the FPGA device, we only need 50 % of the stream data, using a difference based partial reconfiguration, diminishing in this way the set up time of the functionalities between a multiplier and an 8 input SAD operation.

5. REFERENCES

[1] 3dNow Technology Manual. *Euromicro Conference, Proceedings. 24th*, 2:559–566, August 2000.

[2] The xilinx software manuals, xilinx 5.1i. *http://toolbox.xilinx.com/docsan/xilinx5/manuals.htm*, 2000.

[3] C. Baugh and B. Wooley. A Two's Complement Parallel Array Multiplication Algorithm. *IEEE, Transactions on Computers*, pages 1045–1047, Dicember 1973.

[4] K. Compton and S. Hauck. Reconfigurable Computing: a Survey of Systems and Software. *ACM Computing Surveys (CSUR)*, 24(2), June 2002.

[5] D. Guevorkian, A. Launiainn, P. Liuha and V. Lappalainen, Architectures for the Sum of Absolute Differences Operation. *IEEE Workshop on Signal Processing Systems (SPIS'02)*, October 2002.

[6] P. Kuhn. Fast MPEG-4 Motion Estimation: Processor based and Flexible VLSI Implementations. *Journal of VLSI Signal Processing*, 23:67–92, June 1999.

[7] M. Tremblay, J.M. O'Connor, V. Narayanan and He Liang VIS Speeds New Media Processing. *IEEE Micro*, 14(4):10–20, August 1996.

[8] P. Pirsch, N. Demassieux and W. Gehrke VLSI Architectures for Video Compression. *Proceedings of the IEEE*, 83(2):67–92, February 1995.

[9] S.K. Raman, V Pentkovski and J. Keshava Implementing Streaming SIMD Extensions on the Pentium 3 Processor. *IEEE Micro*, 2:47–57, August 2000.

[10] S. Vassiliadis, S. Wong, G. Gaydadjiev, K. Bertels, G. Kuzmanov and E.M. Panainte. The MOLEN Polymorphic Processor. *Computers, IEEE Transactions on*, 53(11):1363 – 1375, November 2004.

[11] S. Vassiliadis, E. Schwarz and M. Putrino Quasi-Universal VLSI Multiplier with Signed Digit Arithmetic. *Southern Tier Technical Conference,Proceedings of the 1987 IEEE*, pages 1–10, April 1987.

[12] S. Vassiliadis, E. Hakkennes, S. Wong and G. Pechanek, The Sum-Absolute-Difference Motion Estimation Accelerator. *Euromicro Conference, Proceedings. 24th*, 2:559–566, August 1998.

[13] S. Thakkar and T. Huff. The Internet Streaming SIMD Extensions. *Intel Technology Journasl*, 32(12):1–8, 1999.