PAPA - Packed Arithmetic on a Prefix Adder for Multimedia Applications

Neil Burgess
Division of Electronics,
School of Engineering,
Cardiff University,
CARDIFF CF24 3TF
U.K.
burgessn@cf.ac.uk

Abstract

This paper introduces PAPA: Packed Arithmetic on a Prefix Adder, a new approach to parallel prefix adder design that supports a wide variety of packed arithmetic computations, including packed add and subtract with saturation, packed rounded average, and packed absolute difference. The approach consists of altering the prefix adder cell logic equations to take advantage of a previously unused "don't care" state. Logical Effort is employed to assess the delay of the new adder architecture by establishing the extra effort needed to select and drive the appropriate carry signal to the requisite sum sub-word. This adder will find applications in video processors and other multimedia-orientated processor chips that implement packed arithmetic operations.

1. Motivation

Multimedia processor chips (and others) make much use of "packed" arithmetic operations in order to accelerate a variety of digital signal processing algorithms for consumer applications. In such arithmetic units, long wordlength numbers are optionally treated as several independent shorter wordlength numbers – for example, a 32-bit word may be treated as 2 separate 16-bit words or as 4 8-bit words. The main motivation for this mode of operation is to support SIMD processing with its associated advantages in the context of a conventional pipelined load-store processor architecture [1]. Moreover, a common arithmetic operation used in video processing is "absolute difference", denoted $|A–B|$, and used widely in video motion estimation and prediction algorithms. Hence, a most valuable operation is a "packed absolute difference" operation, which returns the absolute differences of a several independent pairs of 8-bit pixel values simultaneously.

Ordinarily, absolute differences are computed either by performing a subtraction operation followed by a separate "absolute value" operation, which returns the magnitude of a signed number, or by performing a comparison to order the operands followed by a subtraction in which the smaller operand is subtracted from the larger [2]. Instead of such two-step implementations, absolute differences can be obtained by computing both A–B and B–A, and using the signs of the two results to select the positive result [3]. However, this is wasteful and a better technique is sought. Recently, some authors have described how absolute differences can be derived using a single prefix adder [4, 5], but have not extended this insight to packed arithmetic. Previously re-

ported implementations of packed arithmetic prefix adders include [6, 7], but neither of these proposals is able to support packed late increment operations, vital for computing absolute difference and rounded average instructions.

A second valuable arithmetic option for media applications is saturated arithmetic, in which overflows and underflows do not cause exceptions but rather return pre-defined "saturation constants" [8]. Such constants should ideally be incorporated with little or no performance overhead since the integer adder is typically on the critical path that defines a processor's clock rate. This paper describes how a prefix adder can be altered straightforwardly to support packed absolute difference and rounded average instructions as well as packed saturated arithmetic, and further describes how Logical Effort was employed to assess its performance potential.

2. Packed arithmetic on a parallel prefix adder

2.1 Prefix tree cell logic

The parallel prefix carry-lookahead adder is a popular VLSI design technique that accelerates an n-bit addition by means of a parallel prefix tree [9]. A block diagram of a prefix adder is illustrated in Figure 1, where the adder is seen to consist of three blocks: input bit propagate, generate, and not kill cells; the prefix tree; output sum cells. The input cells derive the bit propagate, generate, and not kill signals respectively according to:

$$p(i) = a(i) \oplus b(i) \quad \text{—(1a)}$$

$$g(i) = a(i) \wedge b(i) \quad \text{—(1b)}$$

$$\neg k(i) = a(i) \vee b(i) \quad \text{—(1c)}$$

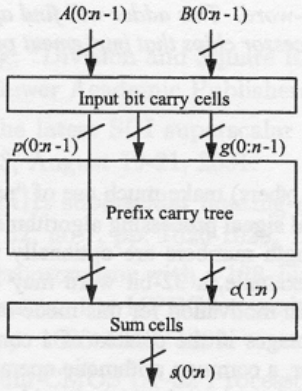

Figure 1 Block diagram of parallel prefix adder

The prefix carry tree expands the input bit generate and bit not kill signals, $g(i)$ and $\neg k(i)$, into "group generate" and "group not kill" signals through a number of levels of logic operations. G_z^w represents a group generate signal across the bits from significance w up to and including significance z, and $\neg K_z^w$ represents a group not kill signal across the same significances. Note that $G_i^i = g(i)$ and $K_i^i = k(i)$. Each level of logic in the tree widens the bit range of the groups at every significance until the lower value of the range covered by the group is 0, and the carry signals are obtained as:

$$c(i) = G_{i-1}^{0} \quad \text{—(2)}$$

Figure 2 shows one of the family of prefix trees proposed by Knowles for $n = 32$ [10]. The black squares in Figure 2 are prefix cells which implement the equation pair:

$$G_z^w = G_z^y \vee \neg K_z^y \wedge G_x^w \quad \text{— (3a)}$$

$$\neg K_z^w = \neg K_z^y \wedge \neg K_x^w \quad \text{— (3b)}$$

and the grey squares are cut-down prefix cells implementing (3a) only. The output sum signals are derived according to:

$$\text{Sum}, A+B: s(i) = G_{i-1}^0 \oplus p(i) \quad \text{— (4)}$$

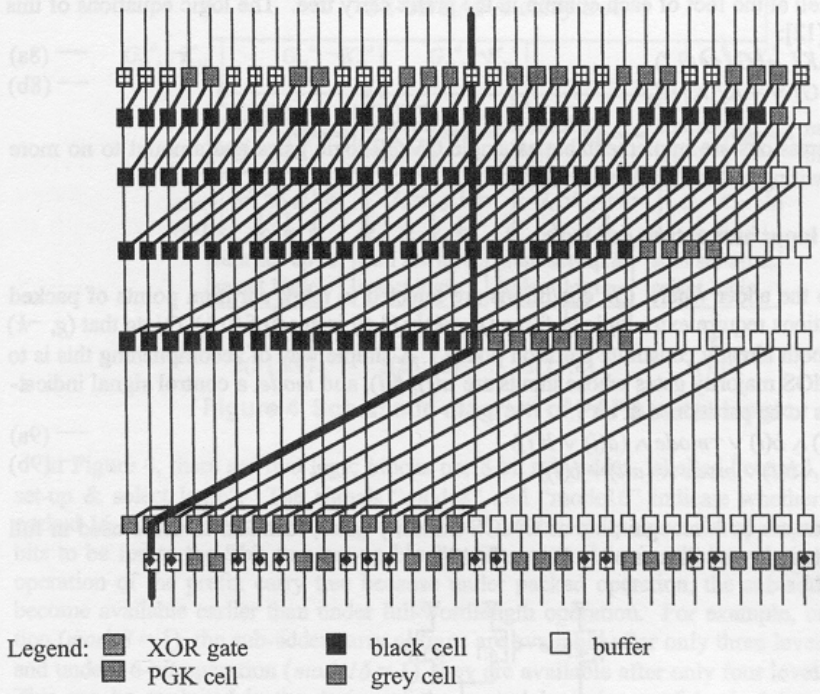

Figure 2 Knowles' [2,2,1,1,1] prefix adder

It has been shown that a trailing string of $(g, \neg k) = (0, 1)$ tuples ("*CP* conditions") identifies the trailing string of sum bits that must change from 1 to 0 if the sum is incremented [11]. Hence, the final group generate and group not kill signals can be combined with control signals, denoted "*inc*" and "*abs*", to derive alternative carry signals, $c_{inc}(i)$ and $c_{abs}(i)$ respectively, that yield results related to the original sum:

$$\text{Incremented Sum}, A+B+1: c_{inc}(i) = G_{i-1}^0 \vee \neg K_{i-1}^0 \wedge inc \quad \text{— (5)}$$

$$\text{Absolute Difference}, |A-B|: c_{abs}(i) = G_{i-1}^0 \equiv abs \vee \neg K_{i-1}^0 \wedge abs \quad \text{— (6)}$$

Packed versions of these two enhanced operations can be supported on a parallel prefix carry tree by defining a fourth symbol, denoted *CB* (*B* for "block"), that prevents carry information from traversing a column in the adder [12]. The *CB* condition can be represented by the "don't care" combination $(G_z^w, \neg K_z^w) = (1,0)$, implying that the *CG* (carry generate) condition must be represented by $(G_z^w, \neg K_z^w) = (1,1)$, and not $(G_z^w, \neg K_z^w) = (1,X)$. This adder is known as PAPA, and the logic equations for the PAPA black cell are [12]:

$$G_z^w = G_z^y \vee \neg K_z^y \wedge G_x^w \quad \text{— (7a)}$$
$$\neg K_z^w = \neg K_z^y \wedge (\neg K_x^w \vee G_z^y) \quad \text{— (7b)}$$

both of which are implementable as single CMOS logic gates.

A second cell for the packed arithmetic prefix adder is required that operates as a normal prefix cell if no *CB* conditions are received on the inputs, but which also converts *CB*'s to *CP*'s to enable the enhanced additions and subtractions described by (4) and (5). This cell is akin to the "grey" (G_{i-1}^0 output only) cell in conventional prefix adders in that it replaces the standard "black" prefix cell at the foot of each column in the prefix carry tree. The logic equations of this second cell are [12]:

$$G_z^w = \neg K_z^y \wedge (G_z^y \vee G_x^w) \quad \text{— (8a)}$$
$$\neg K_z^w = G_z^y \vee \neg K_z^y \wedge \neg K_x^w \quad \text{— (8b)}$$

Again, both expressions are implementable as single CMOS logic gates and amount to no more than a simple rewiring of the PAPA black cell.

2.2 Prefix tree input and output cell logic

The inputs to the adder where *CB* conditions are injected to mark partition points of packed arithmetic operations require extra logic to force $(g, \neg k) = (1,0)$ instead of (0,1). (Note that $(g, \neg k) = (1,1)$ or (0,0) both already constitute partition points.) A simple way of accomplishing this is to use a pair of CMOS majority gates whose inputs are $a(i)$, $b(i)$, and *mode*, a control signal indicating if an adder is to be partitioned at bit i:

$$\neg k = a(i) \wedge b(i) \vee \neg mode \wedge (a(i) \vee b(i)) \quad \text{— (9a)}$$
$$g = a(i) \wedge b(i) \vee mode \wedge (a(i) \vee b(i)) \quad \text{— (9b)}$$

These expressions are both recognisable as CMOS minority gates, identical to those used in full adder implementations, and illustrated in Figure 3 [13].

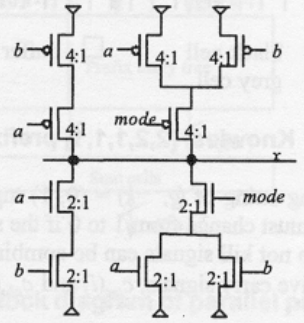

Figure 3 CMOS minority logic gate

Once the carry signals (following either a full wordlength or packed arithmetic operation) emerge from the prefix tree, they must be combined with the bit propagate signals and other control signals so as to return the required results. However, a consequence of recoding the *CG* condition in order to accommodate the *CB* condition is that equations (5) and (6) are no longer correct. Instead, the G_{i-1}^0 prefix tree outputs give the carries for the sum, S, while the $\neg K_{i-1}^0$ outputs give the carries for the sum, $S+1$, so that the output logic equations become:

Incremented Sum, $A+B+1$: $c_{inc}(i) = \neg inc \wedge G_{i-1}^{\,0} \vee inc \wedge \neg K_{i-1}^{\,0}$ — (10)

Absolute Difference, $|A-B|$: $c_{abs}(i) = \neg abs \wedge \neg G_{i-1}^{\,0} \vee abs \wedge \neg K_{i-1}^{\,0}$ — (11)

where *abs* indicates that the difference of the original subtraction, $A-B$, is positive.

Now, the output logic must supply the correct control signals to the appropriate sub-adders so that the desired result is computed using some simple output logic. A block diagram of this approach is presented in Figure 4.

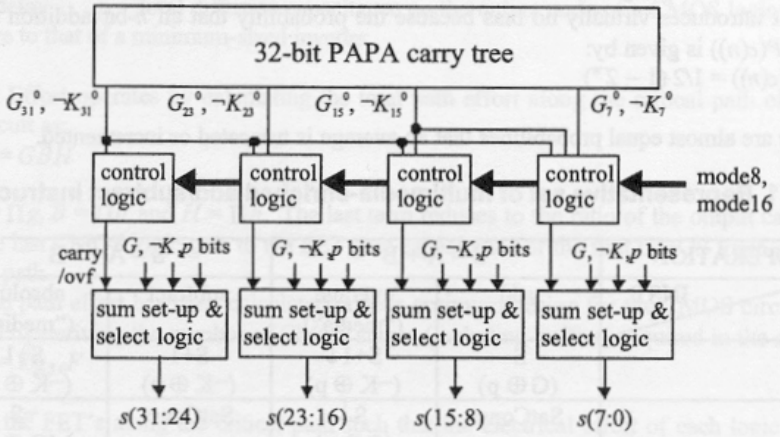

Figure 4 Schematic diagram of PAPA output logic

In Figure 4, there are two logic blocks per 8-bit sub-adder, labelled "control logic" and "sum set-up & select logic". The signals "mode8" and "mode16" indicate whether packed 8-bit or packed 16-bit arithmetic is to be performed and conditionally select the appropriate $G_x^{\,0}$ and $\neg K_x^{\,0}$ bits to be fed to the 8-bit sum set-up blocks. The control logic runs largely in parallel with the operation of the prefix carry tree because under packed operation, the sub-adder carry outputs become available earlier than under full-wordlength operation. For example, under 8-bit operation (*mode8* = 1), the sub-adder carry outputs are available after only three levels of prefix cells, and under 16-bit operation (*mode16* = 1), they are available after only four levels of prefix cells. This can be exploited in the design of the control logic by careful scheduling of the logic, as shown in Figure 5, which illustrates the control logic for *carry/ovf* for bits (7:0) of the adder.

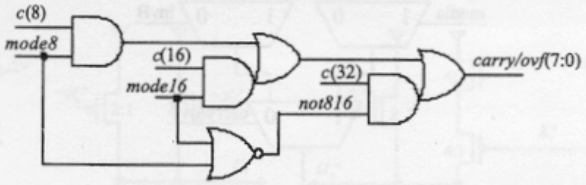

Figure 5 Control logic for sub-adder computing s(7:0)

The logic equations for the control logic blocks for the other sub-groups of output bits are all simpler to realise than Figure 5:

$carry/ovf(15:8) = mode8 \wedge c(16) \vee mode16 \wedge c(16) \vee not816 \wedge c(32)$ — (12a)

$carry/ovf(23:16) = mode8 \wedge c(24) \vee mode16 \wedge c(32) \vee not816 \wedge c(32)$ — (12b)

$carry/ovf(31:24) = mode8 \wedge c(32) \vee mode16 \wedge c(32) \vee not816 \wedge c(32)$ — (12c)

The second logic block, labelled "sum set-up & select logic", prepares the prospective sum outputs based on the instruction being executed. It does such that when the *carry/ovf* signal does become available, it needs to select one of only two possible results. A representative set of multimedia-oriented instructions is presented in Table 1, and has been derived from recent multimedia instruction sets [14-16]. The Table also indicates, where appropriate, which of G_{n-1}^{0} and $\neg K_{n-1}^{0}$ is to be combined with $p(n)$ to form the required sum, as discussed earlier. The rounded average function has been implemented as shown to balance the fan-out loads on the G_{n-1}^{0} and $\neg K_{n-1}^{0}$ outputs. It introduces virtually no bias because the probability that an n-bit addition causes an overflow, $P(c(n))$ is given by:

$$P(c(n)) = 1/2 \cdot (1 - 2^{-n}) \qquad \text{— (13)}$$

Thus there are almost equal probabilities that an average is truncated or incremented.

Table 1 Representative set of multimedia-enriched add/subtract instructions

OPERATION / INST / OVF	S = A + B		S = A + ¬B	
	add	average ("media")	subtract	absolute ("media")
no	S (G ⊕ p)	S+1↓ (¬K ⊕ p)	S+1 (¬K ⊕ p)	S+1 (¬K ⊕ p)
yes	SatConst	S↓ (G ⊕ p)	SatConst	¬S ¬(G ⊕ p)

Figure 6 presents a possible implementation of Table 1. The multiplexer control signals, *invB* and *media* pick out sub-sets of results in the Table (i.e. columns of Table 10) and the logic has been organised so as to minimise the number of multiplexers needed. The saturation constant, *SatConst*, is also selected while the prefix tree is evaluating and fed to the sum output logic blocks.

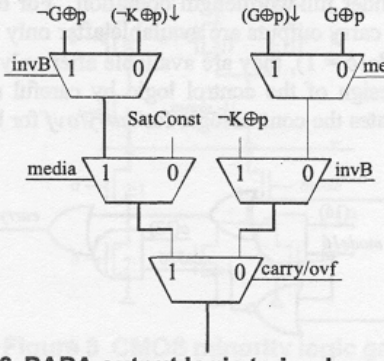

Figure 6 PAPA output logic to implement Table 1

3. Logical Effort applied to a prefix adder

In this section, the principle of *Logical Effort* [17,18] is applied to a parallel prefix adder design so as to permit an assessment of the relative performance of PAPA. Logical Effort is a design methodology for estimating the number of CMOS stages (including buffers) required to implement a given logic function. The principle uses a small number of basic concepts, which are:

logical effort g total FET gate capacitance of a CMOS logic gate relative to that of a minimum-sized inverter

electrical effort h ratio of output capacitance to input capacitance for each CMOS logic gate along a critical path

branching effort b ratio of total capacitative load on one CMOS logic gate's output along the critical path to the FET gate capacitance of the next CMOS gate

parasitic delay p total diffusion capacitance on the output node of a CMOS logic gate relative to that of a minimum-sized inverter

Logical Effort operates by calculating the total path effort along the critical path of a digital CMOS circuit as:

$$F = GBH \qquad (14)$$

where $G = \Pi g$, $B = \Pi b$, and $H = \Pi h$. The last term reduces to the ratio of the output capacitance loading the last CMOS logic gate to the FET gate capacitance of the first CMOS logic gate along the critical path.

Once the path effort has been calculated, a near-optimum design for the CMOS circuit can be determined by deriving the number of CMOS stages (including buffers) required in the circuit as:

$$N = \log_{3.6} F \qquad (15)$$

and sizing the FET's along the critical path such that the electrical effort of each logic gate (i.e. the ratio of the total output load capacitance to the input FET gate capacitance) $h = F^{1/N}/g$. Then the total delay of the CMOS circuit may be written as:

$$D = NF^{1/N} + \sum p \qquad (16)$$

in arbitrary delay units. Dividing this expression by 5 will yield an approximation to the delay in terms of fan-out = 4 ("FO4") inverter delays.

By way of illustration, we shall now apply the tenets of Logical Effort to the prefix adder presented in Figure 2. The critical path of the adder runs vertically through the black cells down bit 15 to the grey cell in 4th row, then out through the bit 30 grey cell, and is highlighted by a thick black line in Figure 2. This path has larger cell fan-outs than the path from $a(0)$ and $b(0)$ to $c(31)$. Also, the black cell $\neg K$ outputs have higher fan-outs than the corresponding G outputs so that the following analysis will consider the $\neg K$ outputs of the black cells (grey cells have G outputs only).

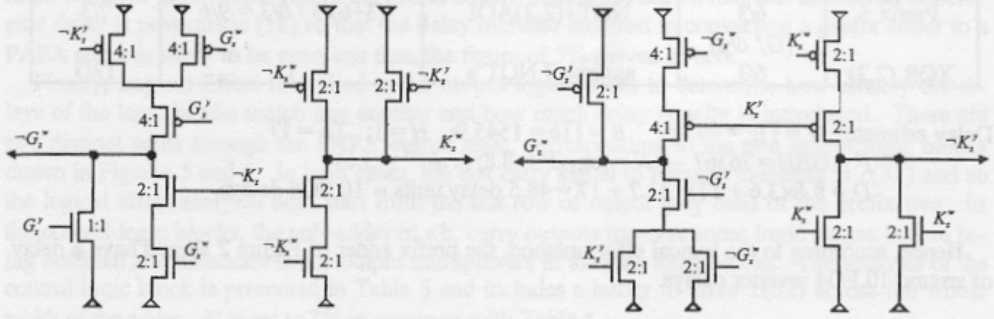

Figure 7 Transistor diagrams of black prefix cell

Figure 7 shows the transistor diagram of the two versions of the black prefix cell: one with true inputs, and the other with inverted inputs. The XOR gates are assumed to be implemented as

CMOS complex (2,2) AOI gates, with a logical effort on each input of 6/3. Table 2 gives the logical effort and parasitic delay of all the different logic gates used in the prefix adder.

Table 2 Logical effort and parasitic delays of CMOS cells in prefix adder

Cell type	CMOS gate	logical effort, g	parasitic delay, p
PGK cell	NOR	5/3	6/3
Black cell (inv^d i/p)	NOR	5/3	6/3
Black cell (true i/p)	NAND	4/3	6/3
Grey cell (inv^d i/p)	(2,1) OAI	6/3	8/3
Grey cell (true i/p)	(2,1) AOI	6/3	7/3
XOR cell	(2,2) AOI	6/3	12/3

The path effort of the adder can now be calculated by taking into account the fan-out loads at each node along the critical path in order to derive the branching effort. In computing the branching effort, track capacitance has been calculated as one minimum-geometry n-FET per lateral cell traversed, equivalent to 1/3 of the gate capacitance of a minimum-size inverter. Hence, a track that travels a lateral distance of four cells has been allocated a track fan-out of 4/3 minimum size inverters. The full calculation of the path effort, F, for the adder is presented in Table 3.

Table 3 Calculation of Path Effort for standard prefix adder

CMOS cell	logical effort, g	fanout load	branching effort, b	parasitic delay, p
PGK block	5/3 (K_z^y o/p)	(trk+2NOR+OAI)	(1+2×5+6)/3 / (5/3) = 17/5	6/3
Prefix 1	5/3 ($\neg K_z^y$ o/p)	(trk+2NAND+AOI)	(2+2×4+6)/3 / (4/3) = 16/4	6/3
Prefix 2	4/3 (K_z^y o/p)	(trk+2NOR+OAI)	(4+2×5+6)/3 /(5/3) = 20/5	6/3
Prefix 3	5/3 ($\neg K_z^y$ o/p)	(trk+2NAND+AOI)	(8+2×4+6)/3 / (6/3) = 22/6	6/3
Prefix 4	6/3 ($\neg G_z^y$ o/p)	(trk+2OAI+NOT)	(16+2×6+3)/3 / (6/3) = 31/6	8/3
Prefix 5	6/3 (G_z^y o/p)	NOT+(2,2)AOI	(3+6)/3 / 6/3 = 9/6	7/3
XOR (2,2)	6/3	assume 3 NOT's	1	12/3

Delay estimate: $G = \prod g = 49.4$; $B = \prod b = 1545.9$; $H = 1$; $\Sigma p = 17$

$F = GBH = 76367 \rightarrow N = \log_{3.6} F = 8.8$; ($\therefore F^{1/N} = 3.6$)

$D = 8.8 \times 3.6 + 17 = 31.7 + 17 = 48.5$ delay units $\approx$ 10 FO4 delays

Hence, according to the logical effort method, the prefix adder of Figure 2 should have a delay of around 10 FO4 inverter delays.

4. Logical Effort applied to the PAPA architecture

Next, Logical Effort is applied to the same prefix adder topology, but now modified to implement PAPA, in order to assess the performance overhead in modifying the cells of the standard

prefix tree. The transistor diagrams for the PAPA black cells are given in Figure 8 and the calculation of the new path effort in Table 4.

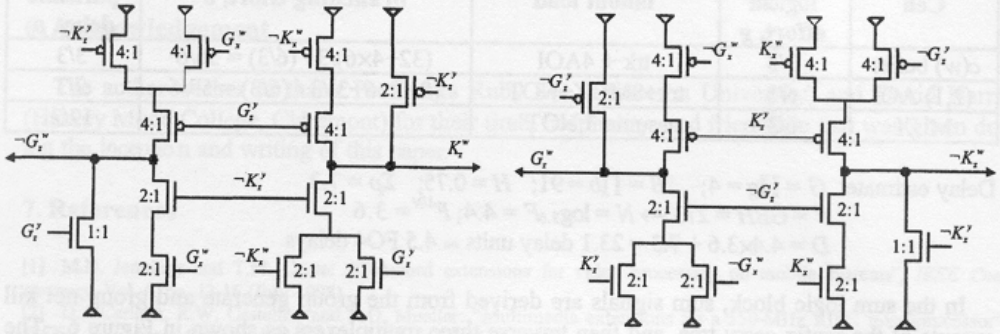

Figure 8 Transistor diagrams of PAPA black and grey cells

Table 4 Calculation of Path Effort for PAPA

CMOS cell	logical effort, g	fanout load	branching effort, b	parasitic delay, p
PGKB	6/3	(trk+2AOI+OAI)	(1+2×6+5)/3 / (5/3) = 18/5	12/3
Prefix 1	5/3	(trk+2OAI+AOI)	(2+2×6+4)/3 / (6/3) = 18/6	7/3
Prefix 2	6/3	(trk+2AOI+OAI)	(4+2×6+5)/3 /(5/3) = 21/5	8/3
Prefix 3	5/3	(trk+3OAI+AOI)	(8+3×6+4)/3 / (6/3) = 30/6	7/3
Prefix 4	6/3	(trk+2AOI+NOT)	(16+2×6+3)/3 / (6/3) =31/6	8/3
Prefix 5	6/3	NOT+(2,2)AOI	(3+6)/3 / (6/3) = 9/6	7/3
XOR (2,2)	6/3	assume 3 NOT's	1	12/3

Delay estimate: $G = \prod g = 88.9$; $B = \prod b = 1757.7$; $H = 3 / (6+6)/3 = 0.75$; $\Sigma p = 20.3$

$F = GBH = 117195 \rightarrow N = \log_{3.6} F = 9.1$; $(\therefore F^{1/N} = 3.6)$

$D = 9.1 \times 3.6 + 20.3 = 32.8 + 20.3 = 53.1$ delay units $\approx$ 10.5 FO4 delays

PAPA (excluding the output logic blocks) runs approximately 5% slower than the standard prefix adder. This increase in delay is mostly due to the increase in parasitic delays, p, of the more complex gates employed in the prefix adder. Harris has shown that this definition of parasitic delay is pessimistic [18] so that the delay increase incurred in converting a prefix adder to a PAPA adder is likely to be even less than the figure of 5% arrived at here.

Finally, Logical Effort is applied to the output logic blocks to determine how closely the delays of the logic blocks match one another and how much delay penalty is introduced. There are two distinct paths through the PAPA output logic, corresponding to the two output logic blocks drawn in Figures 5 and 6. In both cases, the last carry signal to become available is $c(32)$ and so the logical effort analyses both start from the last row of output grey cells of the prefix tree. In the control logic blocks, the sub-adder m.s.b. carry outputs traverse some logic stages before being buffered and broadcast to the output multiplexers in the sum logic blocks. The analysis of the control logic block is presented in Table 5 and includes a buffer to drive $c(32)$ across the whole width of the adder. H is set to 3/4 in common with Table 4.

Table 5 Calculation of Path Effort for control logic

Cell	logical effort, g	fanout load	branching effort, b	parasitic delay, p
$c(w)$ buffer	3/3	trk + 4AOI	(32+4×6)/3 / (6/3) = 56/6	3/3
(2,1) AOI	6/3	trk+8MUX+NOT	(8+8×6+3)/3 / (6/3) = 59/6	7/3
MUX	6/3	assume 3 NOT's	1	12/3

Delay estimate: $G = \prod g = 4$; $B = \prod b = 91$; $H = 0.75$; $\Sigma p = 7.3$
$F = GBH = 273 \rightarrow N = \log_{3.6} F = 4.4$; $F^{1/N} = 3.6$
$D = 4.4 \times 3.6 + 7.3 = 23.1$ delay units $\approx$ 4.5 FO4 delays

In the sum logic block, sum signals are derived from the group generate and group not kill outputs of the prefix carry tree, and then traverse three multiplexers as shown in Figure 6. The Logical Effort analysis is presented in Table 6, where, as before, the MUX's are assumed to be constructed from (2,2) AOI gates and that $H = 0.75$.

Tables 5 and 6 show that both output logic blocks each add around 5 FO4 inverter delays to the delay of the prefix carry tree and are thus well matched for speed, given the pessimistic values for parasitic delay, p. The delay due to the output XOR gates is around 1 FO4 inverter delay so that the output logic adds 4 FO4 delays to the 10.5 FO4 delays of the PAPA adder, for a total of 14.5 FO4 delays.

Table 6 Calculation of Path Effort for control logic

CMOS cell	logical effort, g	fanout load	branching effort, b	parasitic delay, p
XOR	6/3	2 MUX	(2 × 6/3) / (6/3)	12/3
MUX	6/3	1 MUX	1	12/3
MUX	6/3	1 MUX	1	12/3
MUX	6/3	3 NOT's	1	12/3

Delay estimate: $G = \prod g = 16$; $B = \prod b = 2$; $H = 0.75$; $\Sigma p = 16$
$F = GBH = 24 \rightarrow N = \log_{3.6} F = 2.5$; $F^{1/N} = 3.6$
$D = 2.5 \times 3.6 + 16 = 25$ delay units $\approx$ 5 FO4 delays

5. Summary and Future Work

This paper has introduced PAPA - a means of implementing Packed Arithmetic on a Prefix Adder [19]. The key insight has been a new definition of the prefix tree's function, taking advantage of a previously unused don't care state. This adder should be of use in the efficient design of VLSI processor chips implementing media enhanced instruction sets because of its support of SIMD operation of instructions such as packed addition and subtraction with saturation, and packed absolute difference and rounded average. The PAPA concept can be extended to cover other media-oriented instructions not considered in this paper: for example, Galois field arithmetic for Reed-Solomon coding can be supported by using the sub-word carry signals to select between S and S+1. Similarly, min/max instructions should be straightforward to implement. The wide variety of possible overflow conditions that depend on which, if either, of the operands were signed require further study to assess the impact on the adder's performance of supporting all

mechanisms. Finally, detailed VLSI layout and simulation would give a more accurate assessment of PAPA's performance relative to a standard prefix adder.

6. Acknowledgement

The author wishes to thank Professors Ruby Lee (Princeton University) and David Harris (Harvey Mudd College, Claremont) for their time, hospitality, and friendship that was given during the inception and writing of this paper.

7. References

[1] M.D. Jennings and T.M. Coate: "Subword extensions for video processing on mobile systems", *IEEE Concurrency*, Vol. 6, pp. 13-16 (July 1998)
[2] D.A.Carlson, R.W. Castelino and R.O. Mueller: "Multimedia extensions for a 550-MHz RISC microprocessor", *IEEE J. Solid-State Circuits*, Vol. 32, pp. 1618 -1624 (Nov. 1997)
[3] S.C. Knowles, "Arithmetic processor design for the T9000 Transputer", Proc. SPIE, vol. 1566, ASPAAI-2, San Diego, July 1991, pp. 230-243
[4] A Beaumont-Smith et al, "Reduced latency IEEE floating-point adder architectures", Proc. 14th IEEE Symp. Computer Arithmetic, Adelaide, April 1999, pp. 35-42
[5] S.C. Knowles, "Simultaneous Arithmetic", British Patent Application, no. 9813328.3, June 1998
[6] A. Farooqui, V. G. Oklobdzija, and F. Chehrazi, "Multiplexer Based Adder for Media Signal Processing", Proc. IEEE Int. Symp. on VLSI Technology, Systems, and Applications, Taipei, June 1999, pp. 100-103
[7] Suzuki, K. et al, "A 2000-MOPS embedded RISC processor with a Rambus DRAM controller", *IEEE J. Solid-State Circuits*, Vol. 34, pp. 1010 -1021 (July 1999)
[8] R.B. Lee, "Multimedia extensions for general-purpose processors", IEEE Workshop on Design and Implementation of Signal Processing Systems, SIPS 97, Leicester, U.K., Nov. 1997, pp. 9-23
[9] P.M. Kogge and H.S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence relations", *IEEE Transactions on Computers*, vol. 22, pp. 786-793 (August 1973)
[10] S.C. Knowles, "A Family of Adders", Proc. 14th IEEE Symp. Computer Arithmetic, Adelaide, April 1999, pp. 30-34 (reprinted in full in Proc. 15th IEEE Symp. Computer Arithmetic, Vail, CO, June 2001, pp. 277-284)
[11] N. Burgess, "The flagged prefix adder and its applications in integer arithmetic", *J. VLSI Signal Processing*, Vol. 31, pp. 259-267 (June 2002)
[12] N. Burgess, "Packed arithmetic on a prefix adder (PAPA)", Proc. SPIE vol. 4791, ASPAAI-12, Seattle, WA, July 2002.
[13] J.M. Rabaey, *Digital Integrated Circuits: A Design Perspective* (Upper Saddle River, NJ: Prentice Hall) 1996
[14] R.B. Lee, "Subword parallelism with MAX-II", *IEEE Micro*, vol. 16, pp. 51-59 (August 1996)
[15] K. Diefendorff, P.K. Dubey, R. Hochsprung, and H. Scale: "AltiVec extension to PowerPC accelerates media processing', *IEEE Micro*, Vol. 20, pp. 85-95 (March 2000)
[16] S.K. Raman, V. Pentkovski, and J. Keshava: "Implementing streaming SIMD extensions on the Pentium III processor", *IEEE Micro*, Vol. 20, no. 4, pp. 47-57 (July 2000)
[17] R.F. Sproull and I.E. Sutherland, "Logical Effort: designing for speed on the back of an envelope", *IEEE Advanced Research in VLSI*, Ed. C. Sequin, (Boston, MA: MIT Press) 1991
[18] I.E. Sutherland, R.F. Sproull and D. Harris, *Logical Effort: Designing Fast CMOS Circuits* (San Francisco, CA: Morgan-Kaufman) 1999
[19] N. Burgess, "PAPA Addition Circuitry", International Patent Application, no. PCT/GB 01/05358 December 2001