J. Indian Inst. Sci., Nov.–Dec. 1987, 67, pp. 413–422.

Design of fast pipelined arithmetic units in VLSI*

SHISHPAL RAWAT, PORAS T. BALSARA AND MARY JANE IRWIN
Department of Computer Science, Pennsylvania State University, University Park, Pennsylvania 16802, USA.

Abstract

In this paper we propose a constant time pipelined adder and multiplier. Traditionally, efficient carry-look-ahead adders concentrate on making the design regular so that it can be laid out in VLSI. This paper looks at the construction of constant time adders that are regular and technology independent. However, the constant can be made larger or smaller and depends on the area used. The absolute delay is still $O(\log n)$ and area used is $O(n \log n)$. Actual layout of a processor is shown in *NMOS* following the Mead and Conway design rules.

Key words: Pipelined adder, carry-look ahead, VLSI, layout.

1. Introduction

Adders are fundamental units in any ALU design. Multipliers are found in all special purpose floating point processors. The major delay in any design so far has been due to carry propagation. The proposed implementations are a mixture of pipelining and carry-look-ahead principles. First, we will review the carry-look-ahead scheme and then describe the working and layout of our design. We will show that our design is free of any input/output restrictions to achieve optimal area and time. Our design has been used in the design of a digital image filtering scheme with success. It is extremely useful in situations that require lots of additions or multiplications in real time. Latency is not the major concern, throughput is.

1.1 Pipelining

Pipelines are ensemble of simpler arithmetic units called segments which work together in an assembly line fashion. Each segment takes output of the previous segment, processes it and then sends it to the next segment. The segments are isolated from one another by latches (registers). They are very well suited for iterative or recursive algorithms.

1.2 Carry-look-ahead adder

Let $a_{n-1}a_{n-2}\ldots a_0$ and $b_{n-1}b_{n-2}\ldots b_0$ be two n-bit binary numbers; and $s_{n-1}s_{n-2}\ldots s_0$ be their sum. The carry-look-ahead adder uses the following scheme:

$$c_i = g_i + p_i.c_{i-1} \qquad s_i = a_i \oplus b_i \oplus c_{i-1}$$

*First presented at the Platinum Jubilee Conference on Systems and Signal Processing held at the Indian Institute of Science, Bangalore, India, during December 11–13, 1986.

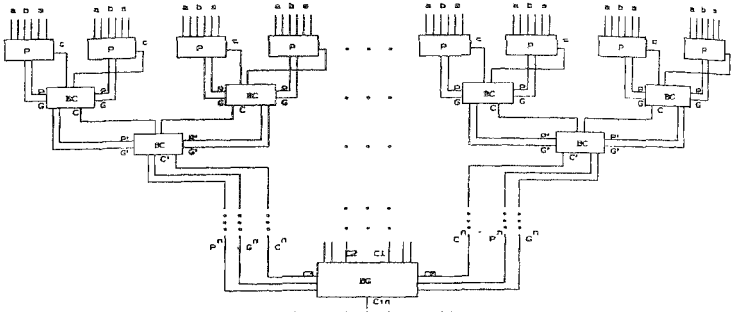

FIG. 1. Block diagram of a conventional carry-look-ahead adder.

where,

$g_i = a_i.b_i$, the generate signal,

$p_i = a_i + b_i$, the propagate signal.

The basic units have been designed as described by Ngai and Irwin[1]. The block diagram in fig. 1 shows a conventional carry-look-ahead adder and Table I summarises the I/O signals for this adder.

Equations for these blocks are formulated in complementary logic due to the nature of basic *NMOS* gates. Different blocking factors result in different speeds and sizes for individual P, BC and BG units[1]. The number of inputs to a primitive block of BC unit defines the blocking factor at that level. It is not necessary to maintain the same blocking factor throughout. Typical size and speed for BC units with different blocking factors are given in Table II.

2. Computational model

Our model assumes the existence of complex *NMOS* gates with a constant fan-in (4) and a constant fan-out (max. 4). This can be reaffirmed by looking at the design of all our

Table I
I/O signals for adder shown in fig. 1

Unit	Input	Output
P	a_is, b_is and carry c_{i-1} from BC	P_i and G_i to a BC, and sum bits with internal look ahead.
BC	P_is, G_is from Ps or other BCs and c_{i-1} from BG or other BCs	P_i, G_i to the next BC or BG and carry out signals to BCs or the Ps
BG	c_0 and P_is, G_is from BCs	Carry out signals to BCs

Table II
Typical blocking factors, size and speed

Blocking factor (B)	Area $\lambda \times \lambda$	Time nsec
2	88×68	5
3	152×127	9
4	196×141	12.5

basic units, described later in this paper. We assume that the wires do not involve any computational delay. This is justifiable for most of our interconnections since the major delay is encouraged in the gate rise and fall times. For extremely long wires (although in our design no wire is longer than $O(\log n)$ length) we can assume the existence of drivers that take up less than 10% of the area[2]. We shall concentrate on measuring the area, rather than the number of gates in order to account for interconnections as well. Gates are assumed to have a constant area and wires constant width. Thus, we can assume that our P, BC and BG units occupy constant area since the interconnections within the units are straightforward and of constant length.

3. Pipelined adder

We were concerned that the adder designed using the general blocking scheme (fig. 1) might not satisfy the timing requirements for our application. This led to the pipelining of various computational stages (fig. 2). The main concern as stated earlier is the throughput; latency is not of much importance.

A block diagram of a typical pipelined adder unit is shown in fig. 2. Equations for each of the above stages are well known[3–7].

However, one problem that remains to be handled is that of interconnecting these units. One should note that the P and G signals produced in the PG stage are needed at

Table III
I/O signals for pipelined adder

Unit	Input	Output
PG	a_is b_is and carry input c_0	P_is and G_is to a BPG unit
BPG	P_is and G_is from previous BPG/PG units	Block P_is and G_is signals to other BPG or BG units
BG	Block P_is and G_is from BPG units	Block carry signals to CP units
CP	Block C_is, P_is and G_is	Block C_is to S units
SUM	Block C_is, P_is and G_is	Generate individual c_is and S_is using internal carry look ahead

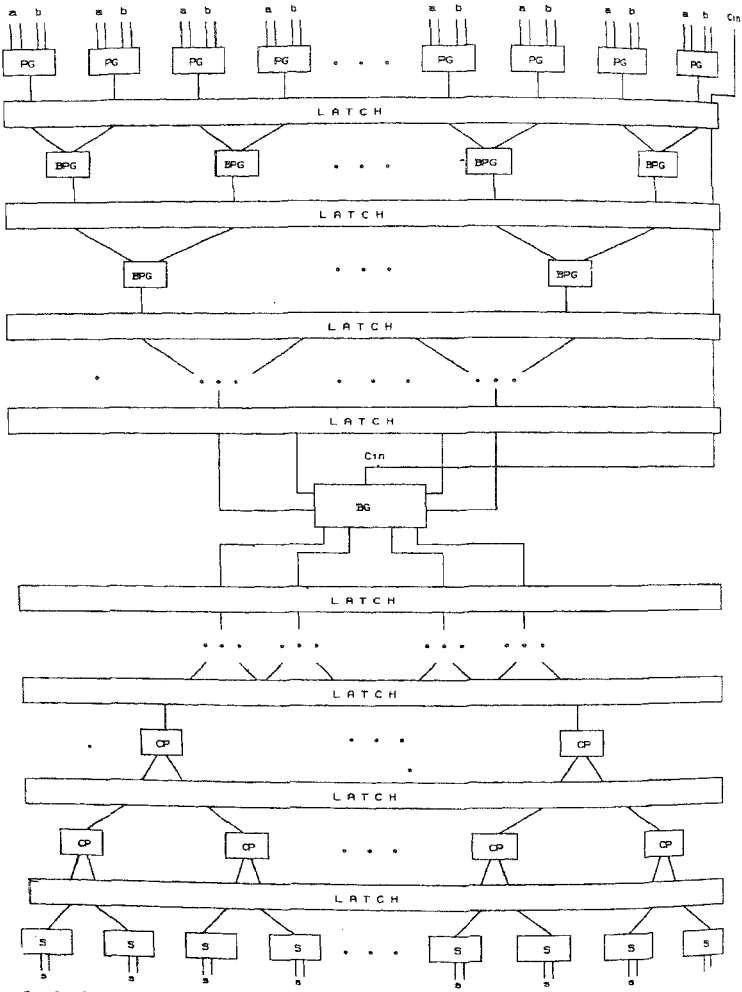

FIG. 2. Block diagram of the pipelined adder.

the very end to produce the carries for S_is. We also have to carry the input operands a and b through the pipe to perform other necessary computations. Our filtering application required that min, max and pass operations be also implemented. Clearly the total data path is $O(n)$.

Let B be the blocking factor and n be the number of bits per operand.

Total number of input signals = $2n$,

Number of signals carried from stage 1

$$= \left(\frac{n}{B}\right)Ps + \left(\frac{n}{B}\right)Gs.$$

Total number of signals carried until the BG unit

$$= 2n + \frac{2n}{B}\left(1 + \frac{1}{B} + \frac{1}{B^2} + \ldots\right) \leq 2n + \frac{2n}{B}\frac{1}{\left(1 - \frac{1}{B}\right)} \leq 2n\left(1 + \frac{1}{B-1}\right)$$

i.e., the larger the blocking factor, the smaller the number of signals that are to be carried through. For $B = 3$ the data path width is $3n$.

During the bottom half of the adder pipe, *i.e.* when the carries are generated we can see that for every P and G used, only one carry signal is produced, since,

$$C_i = P_i C_{i-1} + G_i.$$

This implies that the width of the data path can only shrink.

4. Layout

If one is not careful, interconnections can really consume a lot of area on silicon. In the following paragraphs we suggest a method to layout the above mentioned pipelined adder (for $B = 2$) in area $n \log n$. Clearly, the number of stages in the above design is approximately $2 \log n$. The layout algorithm described below, specifies the horizontal spacing required.

Algorithm 4.1 Designadder (n)

(Assume that n is a power of two; 2 is chosen as the blocking factor)

If $(n > 2)$ *then*
begin

Designadder $\left(\frac{n}{2}\right)$;

Provide horizontal spacing of $2 \log(2n)$

Designadder $\left(\frac{n}{2}\right)$;

end

else

Design the basic 2-bit Carry-Look-Ahead Adder

End {Designadder}

The above algorithm provides for the extra space required by P and G signals exactly from the stage where they are generated. No global signals other than *Vdd*, *Gnd* and *Clock* are required at any stage. Wires need to be routed to the nearest neighbors only. A layout of 64-bit pipelined adder with a blocking factor of 2 is shown in fig. 3.

4.1 Area and time measures

We now show that algorithm 4.1 causes the horizontal dimension of the adder to stay $O(n)$. Let, the extra length for expansion needed to carry additional signals at each stage be $L(n)$, then,

$$L(n) = 2L\left(\frac{n}{2}\right) + 2\log 2n$$

$L(n) = kn - 2\log n - 6$ can be shown by elementary summation techniques.

Thus, we have (for the adder),

Total length	:	$O(n)$
Depth	:	$O(\log n)$
Area	:	$O(n \log n)$
Delay	:	$O(\log n)$
Pipelined delay	:	$O(1)$

Figure 4 depicts a partial layout of a video processor[8] which uses a pipelined CLA adder.

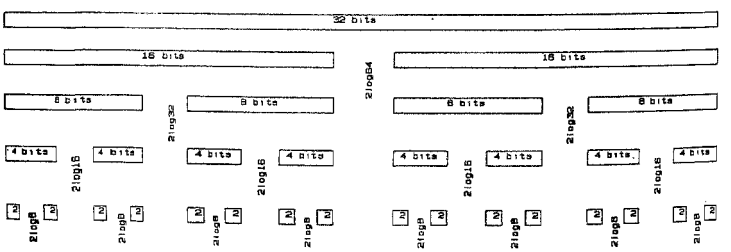

FIG. 3. Layout scheme for a 32-bit pipelined adder ($B = 2$).

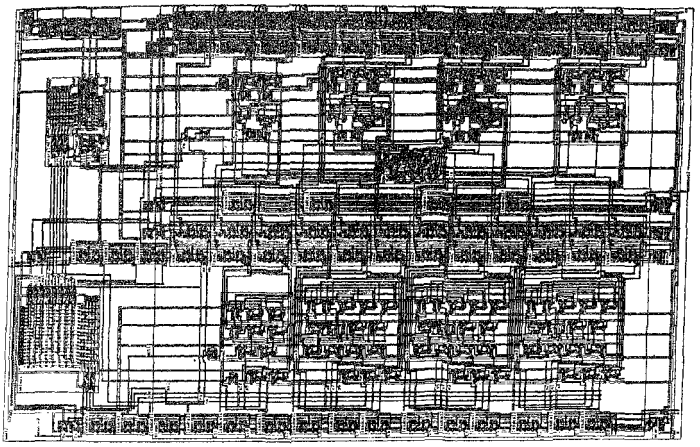

FIG. 4. Layout of a processor which uses a pipelined CLA adder.

4.2 Tradeoffs for speed

1. Blocking factor can be varied for various combinations of areas and times.
2. The as and bs need not be carried all the way through the pipeline. However, if we carry $(a \oplus b)$ instead of a and b, initial Ps and Gs have to be carried through so that individual sum bits can be generated. This will speed up the last stage but increase the dimension, namely, the length of the adder by n and seems unnecessary. We can achieve some latency speed up by generating Ps and Gs or *exored* as and bs at any stage before coming to the final stage. However, each of these must be considered with respect to the area that will be occupied.

5. Pipelined multiplier

The above mentioned concept of pipelined adder can be further extended to design an $O(1)$ pipelined multiplier. Figure 5 shows a block diagram of such a pipelined multiplier. The adders shown in this figure have some front-end logic in order to enable them to generate partial products.

Let the ith multiplicand (n bit) be

$$A^i = A_{in}A_{in-1}\ldots A_{i1}$$

and the ith multiplier (m bit) be

$$B^i = B_{im}B_{im-1}\ldots B_{i1}$$

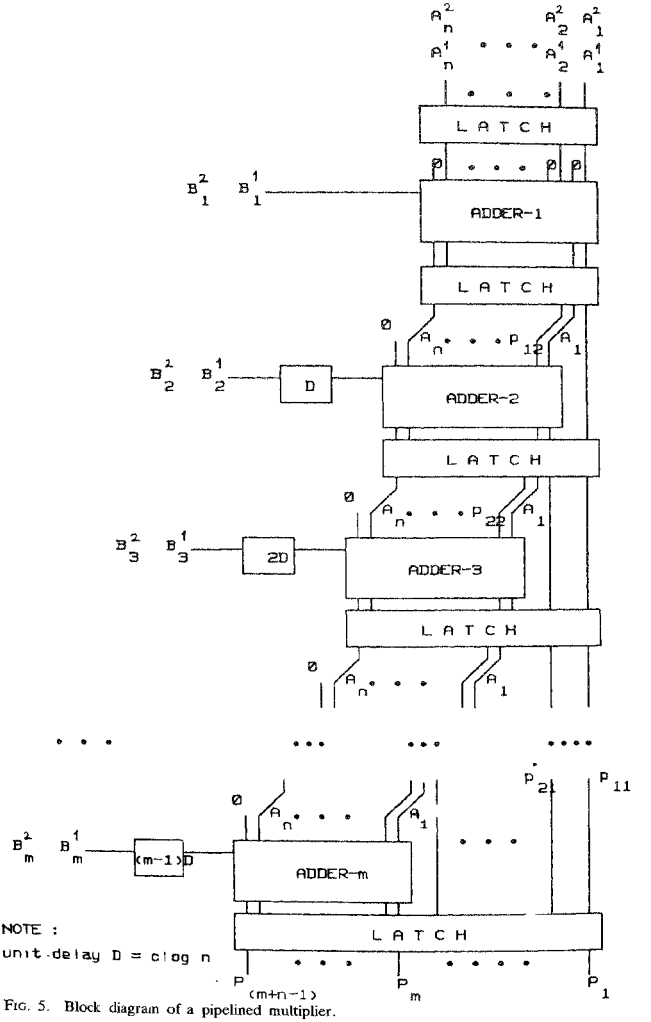

FIG. 5. Block diagram of a pipelined multiplier.

then the partial product at stage $k-1$ of A^i and B^i is

$$P^i_{k-1} = P_{il}P_{il-1}\ldots P_{i1} \qquad (l = n+k-1).$$

In this discussion a stage refers to 1 adder stage. At stage k the partial product must be increased by an amount equal to $A^i * B_{ik}$. If B_{ik} happens to be zero, the partial product need only be shifted (left shift).

From the block diagram in fig. 5 one can observe that the partial product at every stage is available after an $O(\log n)$ delay. This puts a restriction on the time at which the next multiplier bit has to arrive and so, the multiplier bit at the ith stage is skewed in time $((i-1)*\log n)$. A large number of buffers will be needed if the external logic supplies the operand data bits in parallel.

A large number of multiplications can be done in average $O(1)$ time if the latency can be tolerated. The first product, $A^1 * B^1$ is available after a time delay of $O(m \log n)$, after which, all the subsequent products are available after a delay of one clock pulse.

Multiplier recoding can be used to reduce the number of stages by half with minimal addition of logic.

6. Conclusions

Once the pipeline is full, the sum/product bits from our arithmetic units are obtained in constant time irrespective of the length of the input operands. This makes it very appropriate for applications in which a stream of data is to be processed continuously at a high speed. In the pipelined adder the constant delay is the delay of the slowest stage in the pipeline and that is much smaller than the delay of an entire carry-look-ahead adder.

The AT^2 product for our adder is $O(n \log^3 n)$ which compares favorably with the Brent and Kung adder[5] which is $O(n \log^2 n)$. However, the word width in their adder has to be $n/\log n$ to achieve the mentioned AT^2 product. Our design is free of such restrictions. For any general n our adder does significantly better than serial adders mentioned by Mead and Conway[2], and is nearly optimal. Its pipelined delay of $O(1)$ though, can be really put to good use in a pipelined architecture.

This design will be quite useful in data flow type architecture where a scheduler can keep these units busy. If at any point the pipe is broken because of insufficient data, the pipeline does not have to be flushed in order to be restarted.

References

1. Ngai, T. F. and Irwin, M. J. — *Area-time efficient carry look ahead adders*, Proc. of the Arith-7 Conf., Urbana-Champagne, June 1985.
2. Mead, C. A. and Conway, L. A. — *Introduction to VLSI systems*, Addison-Wesley, 1980.
3. Waser, S. and Flynn, M. J. — *Introduction to arithmetic for digital systems designers*, Holt, Rinehart & Winston, 1983.

4. HAMACHER, V., VRANESIC, Z. AND ZAKY, S. *Computer organization*, McGraw-Hill, 1984.

5. BRENT, R. P. AND KUNG, H. T. A regular layout for parallel adders, *IEEE Trans.*, 1982, **C-31**, 260–264.

6. NGAI, T. F., IRWIN, M. J. AND RAWAT, S. Area time efficient carry lookahead adder, *J. Parallel Distributed Computing* 3 Nov. 1985, 92–105.

7. BAYOUMI, M., JULLIEN, G. AND MILLER, W. An area time efficient NMOS adder integration, *VLSI J.*, 1983, 1.

8. IRWIN, M. J., RAWAT, S., BALSARA, P. T. AND MCKOWIAK, T. *Design and implementation of a video processor*, Technical Report 1985, Deptt of Computer Science, Penn. State University, University Park, PA 16802.