PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 7 : G06F 7/52

A1

(11) International Publication Number: **WO 00/49494**

(43) International Publication Date: 24 August 2000 (24.08.00)

(21) International Application Number: PCT/US00/03879

(22) International Filing Date: 15 February 2000 (15.02.00)

(30) Priority Data:
09/250,832 17 February 1999 (17.02.99) US

(71) Applicant: ANALOG DEVICES, INC. [US/US]; One Technology Way, P.O. Box 9106, Norwood, MA 02062–9106 (US).

(72) Inventors: WERTHEIM, Elon; Habariach 7 St., 45291 Hod-Hasharon (IL). OHANA, Eric; Yonah Green Street 7/3, Petah–Tikva (IL).

(74) Agent: HENRY, Steven, J.; Wolf, Greenfield & Sacks, P.C., 600 Atlantic Avenue, Boston, MA 02210 (US).

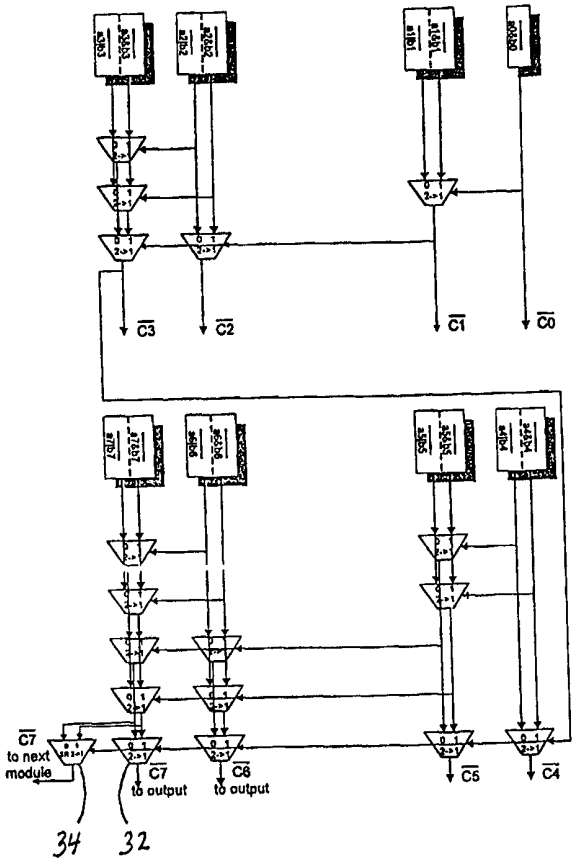
(81) Designated States: CN, European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: FAST MULTI–FORMAT ADDER

(57) Abstract

An adder which accepts operand data in multiple word–size formats and which generates sum outputs at speeds substantially independent of the format or formats of the data. In general, if the maximum word–size of the adder is N bits, it will perform two N/2–bit additions, four N/4–bit additions, etc. A combination of a carry–select algorithm and a forward carry calculation method generate a carry–out value from a given set of input operand bits independently of the sum value of the inputs. In a first aspect, such an adder for generating the sum of two operands comprises a plurality of substantially identical adder modules diposed in pairs, a first adder in each pair having a carry input set to a logical zero value and a second adder in each pair having a carry input set to a logical one value; the adder modules generating substantially all possible sum and carry output values from operand and carry input values; and a multiplexer network selects from the possible sum and carry outputs the correct values to provide the sum of the operands. Circuitry is provided for forced setting the carry–out values from modules to operand–independent logic levels. This allows modules to be operated independently of one another, permitting multi–format use of the modules to form adders of selectable word size.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania
AM	Armenia
AT	Austria
AU	Australia
AZ	Azerbaijan
BA	Bosnia and Herzegovina
BB	Barbados
BE	Belgium
BF	Burkina Faso
BG	Bulgaria
BJ	Benin
BR	Brazil
BY	Belarus
CA	Canada
CF	Central African Republic
CG	Congo
CH	Switzerland
CI	Côte d'Ivoire
CM	Cameroon
CN	China
CU	Cuba
CZ	Czech Republic
DE	Germany
DK	Denmark
EE	Estonia
ES	Spain
FI	Finland
FR	France
GA	Gabon
GB	United Kingdom
GE	Georgia
GH	Ghana
GN	Guinea
GR	Greece
HU	Hungary
IE	Ireland
IL	Israel
IS	Iceland
IT	Italy
JP	Japan
KE	Kenya
KG	Kyrgyzstan
KP	Democratic People's Republic of Korea
KR	Republic of Korea
KZ	Kazakstan
LC	Saint Lucia
LI	Liechtenstein
LK	Sri Lanka
LR	Liberia
LS	Lesotho
LT	Lithuania
LU	Luxembourg
LV	Latvia
MC	Monaco
MD	Republic of Moldova
MG	Madagascar
MK	The former Yugoslav Republic of Macedonia
ML	Mali
MN	Mongolia
MR	Mauritania
MW	Malawi
MX	Mexico
NE	Niger
NL	Netherlands
NO	Norway
NZ	New Zealand
PL	Poland
PT	Portugal
RO	Romania
RU	Russian Federation
SD	Sudan
SE	Sweden
SG	Singapore
SI	Slovenia
SK	Slovakia
SN	Senegal
SZ	Swaziland
TD	Chad
TG	Togo
TJ	Tajikistan
TM	Turkmenistan
TR	Turkey
TT	Trinidad and Tobago
UA	Ukraine
UG	Uganda
US	United States of America
UZ	Uzbekistan
VN	Viet Nam
YU	Yugoslavia
ZW	Zimbabwe

FAST MULTI-FORMAT ADDER

Field of the Invention

This invention relates to the field of digital adders and, in particular, to a fast adder employing a carry-forward calculation method. It has the capability of performing additions and subtractions with operands of various sizes, including multiple such operations in parallel.

Background

In recent years, digital computing has become widespread in a vast number of applications ranging from information systems to signal processing and transmission systems. Digital computing is at the heart of virtually all data communications, audio signal transmission, video recording and transmission, special effects generation, other multimedia systems and medical imaging and instrumentation, among many other data processing situations. All such systems are based on algorithms which perform large numbers of arithmetic operations in microprocessors to process their input data and generate algorithmically processed output data. Among the most common operations executed in these algorithms are additions, subtractions and multiplications. Multiplications actually rely heavily on additions, as well. Thus efficient execution of addition operations is critical to performance, since those operations are typically among the most prevalent operations to be executed. Therefore, speed of operation is a major design consideration for adders in digital signal processing applications.

Speed of operation is sometimes a function of the size of the operands. The degree of precision required of an addition, the size (i.e., number of bits) of its operands and the number of operands may be different for different additions to be performed in executing a given algorithm or algorithms on a processor. Also, data sources may provide input data in various word-size formats ranging, typically, from 8-bit values to 64-bit values. Thus, for example, one addition operation may have two or more 16-bit operands while another may have two 32-bit operands. In a growing number of situations, moreover, 64-bit operands and operations are being used. Therefore, a typical microprocessor designed for general purpose digital signal processing either is designed for a fixed word size or must be provided with multiple adders, each capable of adding (or subtracting) two operands of a fixed size. For example,

there may be one or more each of 64-bit adders, 32-bit adders, 16-bit adders and 8-bit adders. However, the mix of data processing operations required of such a microprocessor usually will differ from one signal processing situation to the next. This makes the hardware design an inefficient use of available chip area.

As is well known, the addition of two binary numbers in a digital computer or processor is a bit-wise operation. An adder must accept multi-bit input data and, for each bit of data, generate two outputs, one representing the sum of the input data and the other representing the carry-out value used in the addition of the next-most-significant bit. The speed with which the carry-out value is generated often controls how fast the adder may be operated to provide a valid sum. Ordinary adders must generate a per-bit carry value for each bit and then use that carry value as an input to the addition of the next-most-significant bit. Such an architecture is referred to as a ripple-carry adder; and the time to complete the addition operation in such a device is thus limited by the time it takes the carry values to propagate (ripple) from the bit-adder which processes the least-significant bit to the bit-adder which processes the most-significant bit.

Another type of adder architecture generates the carry values directly from the input operands without waiting for the ripple effect. Such an adder is referred to as a fast carry, carry look ahead or carry forward adder. In a carry forward adder, the carry output actually may appear before, simultaneously with, or only slightly after the sum output. However, the operating speed often is dependent on operand size and may decrease with increasing operand size.

Summary of the Invention

Advantageously, there is provided, in accordance with the invention, a carry forward adder which accepts operand data in multiple word-size formats and which generates sum outputs at speeds substantially independent of the format or formats of the data. For example, if the adder is designed to perform a single addition on two 64-bit operands (i.e., to have a maximum word size capacity of sixty-four bits), it also will perform two 32-bit additions (i.e., it will accept two sets of two 32-bit operands each), four 16-bit operations, eight 8-bit operations, or any of the foregoing operations in which the input bits sum to 64 or fewer. In

general, if the maximum word size of the adder is N bits, it will perform two N/2-bit additions, four N/4-bit additions, etc.

The architecture of this adder employs a combination of a carry-select algorithm and a forward carry calculation method to generates a carry-out value from a given set of input operand bits independently of the sum value of the inputs. At the same time, it achieves the above-noted multi-format support.

The foregoing combination of features makes this adder particularly useful in high-performance microprocessors which include in their instruction sets provisions for formatting additions in multiple formats (i.e., using different-sized operands).

In an illustrative embodiment, eight 8-bit adder modules are provided. They may be employed as individual modules or may be interconnected to process 16-, 32- or 64-bit operands. In general, however, the modules need not all process the same number of bits. Q modules may be provided, each for processing R_i bits of operand data (where i is an index ranging from 1 through Q). With this arrangement, the critical path, in terms of operating speed, is the carry propagation delay for a maximum word-size addition. However, in the worst case this is only about a nanosecond difference than the time required for an addition of operands half that large.

According to a first aspect of the invention, an adder is provided for generating the sum of two operands, comprising a plurality of substantially identical adder modules disposed in pairs, a first adder in each pair having a carry input set to a logical 0 value and a second adder in each pair having a carry input set to a logical 1 value; the adder modules generating substantially all possible sum and carry output values from operand and carry input values; and a multiplexer network which selects from the possible sum and carry outputs the correct values to provide the sum of the operands.

According to a second aspect of the invention, an adder is provided for performing additions in multiple formats. More particularly, the adder is of the carry forward type and is capable of adding one or more pairs of operands, each pair having a first operand and a second

operand, the total number of bits in all of the first operands being N. The adder comprises Q adder-and-carry-select ahead modules, each for adding a pair of R_i-bit operands (where i is an index which may range from 1 through Q). Each module has a first adder and a second adder, said adders each generate sum and carry outputs. The first adder and second adder receive the same data bits as input while the first adder receives a carry-in value of logical 0 and the second adder receives a carry-in value of logical 1. A multiplexer network is connected to receive as inputs sum outputs from the adders and to receive as control signals carry-out values from the adders; the multiplexer network selecting the correct sum values to provide as outputs from each of the modules.

Circuitry is provided for forced setting the carry-out values form modules to operand-independent logic levels. This allows modules to be operated independently of one another, permitting multi-format use of the modules to form adders of selectable word size.

Other aspects of the invention will appear from, and are discussed in, the Detailed Description, below.

Brief Description of the Drawing

In the drawing,

Fig. 1 is a high-level block diagram of an adder according to an exemplary embodiment of the invention;

Fig. 2 is a more detailed block diagram of an adder according to Fig. 1;

Fig. 3 is a block diagram of suitable circuitry for generating the carry outputs from those adders 16-i-1 of Fig. 2 which have CIN = 0 (i.e., a carry-in set to logical 0);

Fig. 4 is a block diagram of suitable circuitry for generating the carry outputs from those adders 16-i-2 of Fig. 2 which have CIN=1 (i.e., a carry-in value set to logical 1);

Fig. 5 is a block logic diagram of exemplary suitable circuitry for performing a sum bit calculation in the adders 16-i-1 and 16-i-2 of Fig. 2;

Fig. 6 is a diagrammatic illustration of an exemplary 2:1 Set/Reset multiplexer such as may be used in the embodiment of Fig. 2; and

Fig. 7 is a more detailed diagram of a suitable embodiment for the Set/Reset multiplexer of Fig. 6.

Detailed Description

Turning first to Fig. 1, there is shown an adder 10 according to an exemplary embodiment of the invention, and its various input and output signals. In this embodiment, the maximum operand (and operation) size, N, is sixty-four bits. The inputs to the adder 10 are two N-bit operands, A[N-1:0] and B[N-1:0]; a carry-in bit, CIN; and seven pairs of set-reset controls for multi-format operation support, S/R md6[1:0] through S/R md0[1:0]. (The notation [a:b] signifies binary data ordered from a most-significant bit (MSB) labeled "a" to a least-significant bit (LSB) labeled "b".) Internal to the adder and not shown are eight 8-bit adder modules md-0 through md-7. The outputs of the adder are an N-bit sum, Sum[63:0]; and the two most significant, inverted, carry-out signals of each byte-wise sum (i.e., the sums derived from eight-bit segments of the operands), designated $\overline{c7/6}$, for corresponding eight-bit adder modules md-7 to md-0.

Fig. 2 is a block diagram illustrating the structure of an exemplary embodiment for adder 10. The adder is formed from M modules, md-0 through md-(M-1), and a multiplexer (mux) network 14. Each module md-i is built of two adders 16-i-1 and 16-i-2, which receive the same operand data from a data bus 17 and generate bit-wise sum outputs and carry outputs using a carry-select ahead approach. In the example using N = 64 (i.e., 64-bit operand range), there may, for example, be eight modules (i.e., M = 8) and the adders 16-i-1 and 16-i-2 may be 8-bit adders (of course, it is not necessary that all the adders be of 8-bit size or of the same size). The first adder in each module, 16-i-1, assumes (i.e., is provided with) a carry-in value of 0, while the second adder in each module, 16-i-2, assumes a carry-in value of 1. The

outputs of the adders 16 are delivered to the mux network 14. As explained more completely below, the mux network 14 selects one of the sum outputs of adders 16-i-1 and 16-i-2 to produce the sum output of each module, under the control of the carry outputs of those adders. The solid lines output from the adders 16-i deliver 8-bit data and the $\overline{c7/6}$ signals (also written NC7/6); the dashed lines deliver carry-out data.

Each module performs real carry forward calculations, as more fully explained below.

Functionally, the whole adder 10 can be viewed as separated into two 32-bit (i.e., N/2-bit) blocks. The first block includes modules md-1 through md-3 and the second block includes modules md-4 through md-7 (assuming M=8). In the naming conventions used herein, the LSB is bit 0 and the MSB is bit N-1 (e.g., bit 63 in a 64-bit system). Correspondingly, bits 0-7 of the output are assigned to and generated by module md-0; bits 8-15 are assigned to and generated by module md-1; and so on until bits 56-63 are assigned to and generated by module md-7.

The selection of the sum value for the output is performed as in the conventional carry-select algorithm. The carry-out value for each module, by contrast, is generated and propagated using a different technique which is the same as for the 8-bit adder described below. Data selection and carry selection for both "blocks" are combined, so that all possible paths for both blocks are calculated in parallel. Note that the difference in time between the results of the lower 32-bit results being available from the first block and the full 64-bit results being available from both blocks is only the delay through a 2:1 multiplexer "layer" 25. Moreover, this scheme can be repeated for larger operands. For example, for a 128-bit maximum operand size, the difference in time to calculate 128-bit results, relative to the time to calculate 64-bit results, is only the propagation delay of a 2:1 multiplexer. Likewise, this relationship continues as the data format (operand size) is widened even further.

The 8-bit adder module structure will now be explained.

According to the standard carry look-ahead algorithm, when two 8-bit operands a[7:0] and b[7:0] are to be added, for stage "j" (where $7 \leq j \leq 1$) the carry-out value, c(j), is given by c(j) =g(j) + c(j-1)*p(j), where g(j) = a(j)*b(j) and p(j) = a(j) + b(j). The asterisk symbol indicates a logical AND operation and the plus sign indicates a logical OR operation.

If a value c(j-1) = 0 is assumed, then c(j) = g(j) = a(j)*b(j) and the inverted carry-out, ~c(j), is given by ~c(j) = a(j) NAND b(j).

Conversely, if the value c(j-1) = 0 is assumed,
c(j) = g(j)+p(j) = a(j)*b(j) + a(j) +b(j) = a(j) + b(j) = p(j); and ~c(j) = a(j) NOR b(j).

Thus, the carry-out value c(j) may be only g(j) or p(j) and when a(j) = b(j), the same carry-out is generated irrespective of the value of c(j-1). However, when a(j) = ~b(j), the carry-out values returned are inverted, irrespective of the value of c(j-1).

According to the exemplary embodiment of Fig. 2, carry-in values for the least significant bit, CIN, must be assumed as there is no carry-in value to be generated and supplied by a module processing lower bits. Therefore, one adder 16-0-2 in the 0^{th} module assumes a carry-in value of CIN = 1 and the other adder 16-0-1 in the module assumes a carry-in value CIN = 0. Thus, for these two adders, respectively, the generated carry-outs are $\sim c_2(0)$ = a(0) NAND b(0) and $\sim c_1(0)$ = a(0) NOR b(0), where the subscripts denote the corresponding third digit in the adder identification (e.g., a 2 refers to adder 16-0-2 and a 1 refers to adder 16-0-1).

The sum outputs from adders 16-0-1 and 16-0-2 are provided to a mux 14A. Responsive to the value of CIN, mux 14A selects as the module md-0 sum output an appropriate one of the inputs from adder 16-0-1 or 16-0-2. The carry outputs from adders 16-0-1 and 16-0-2 are provided in parallel to a mux 14B. Responsive to the value of the CIN signal, the mux 14B generates a control signal to mux 14C which selects which of the adders 16-1-1 or 16-1-2 will supply the module md-1 sum output. That is, there are two parallel paths for the carry-outs from modules md-0 and md-1, and two parallel paths for the carry-outs from modules md-2 and md-3. The correct carry-outs of modules md-0 and md-1 are

selected by CIN and, in their turn, select the correct sum data for the modules md-1 and md-2 in the next-more-significant positions in the block. (Inasmuch as the inverted carry signals are used for circuit purposes, the multiplexer input data states are, as illustrated, also reversed.) More particularly, the carry outputs from adders 16-1-1 and 16-1-2 are supplied in parallel to a pair of muxes, 14D and 14E. Mux 14D is controlled by the carry output from adder 16-0-2 and mux 14E is controlled by the carry output from adder 16-0-1. In turn, the outputs from muxes 14D and 14E are supplied as inputs to mux 14F. The module md-1 carry-out is supplied from mux 14F and, in turn, controls mux 14G to select as the module md-2 sum output either the sum output from adder 16-2-1 or the sum output from adder 16-2-2. As well, the output from mux 14F controls muxes 14H and 14I and thus selects the carry-outs from modules md-2 and md-3. The carry-out from module md-2 (i.e., the output of mux 14H) controls mux 14J to select the proper data to output (from among the sum outputs of adders 16-3-1 and 16-3-2) as the sum data from module md-3. Muxes 14K and 14L are connected and operated analogously to muxes 14D and 14E, and select the carry output values to supply as input to mux 14I, one of which is chosen as the carry output from module md-3; that md-3 carry output controls the md-4 sum data selection by controlling mux 14M.

Two parallel paths are prepared, also, for modules md-4, md-5, md-6, and md-7, carrying every combination of possible data sum and carries. The carry-out from module md-3 (i.e., from mux 14I) provides the selection signal to control not only md-4 output mux 14M, but also md-5-md-7 output muxes 14N-14P. For module md-5, muxes 14Q and 14R receive the sum outputs from adders 16-5-1 and 16-5-2 and, in response to the carry outputs from adders 16-4-2 and 16-4-1, respectively. The carry outputs from adders 16-5-1 and 16-5-2 are supplied, respectively, to muxes 14S and 14T, and are controlled, in turn, by the carry outputs from adders 16-4-2 and 16-4-1, respectively. The carry outputs from muxes 14S and 14T, respectively, are supplied to control muxes 14U and 14V and, in parallel, muxes 14W and 14X. Muxes 14U and 14V receive and select from the sum outputs of adders 16-6-1 and 16-6-2. Mux 14O then selects the module md-6 sum output from the mux 14U and 14V outputs. Muxes 14W and 14X receive and select from the carry outputs of adders 16-6-1 and 16-6-2, supplying their outputs to the control inputs of muxes 14Y and 14Z. The signal inputs of muxes 14Y and 14Z receive the sum outputs of adders 16-7-1 and 16-7-2, with the outputs of those muxes being supplied to the signal inputs of mux 14P.

Note that though the inputs to muxes 14F and 14I are the same, CIN is used as the selector signal to control muxes 14B and 14F. Further, the output of mux 14F is used as the selector signal for controlling muxes 14H and 14I. As a result, the carry out from bit 31 is generated through only three muxes from the eight-bit adders, with no complex logic gates being required.

As will be seen from Fig. 2, the mux network 14 operates as a mechanism to select from all possible paths and outcomes for both carry values and data sum bits. Using this approach, for the most significant bits of a 64-bit adder there is a four-mux propagation delay between the outputs of the adder modules and the results, whereas there is a three-mux delay for 32-bit results. A single mux propagation delay is only about 0.5 to 0.6 ns using typical fabrication technologies (and will decrease in the future). Thus, very little additional time is needed to generate 64-bit results as compared with the time needed to generate 32-bit results. When this architecture is extended to even larger digital word and operand sizes, each doubling of word size carries with it only a single mux propagation delay loss in the speed of operation.

In contrast with other designs for adders, such as the binary carry look ahead algorithm (bcla), the mux network 14 generates the carry-out values for each bit as a by-product of sum-bit generation. It supplies not only the carry-out value from the MSB, but also all intermediate bit carry out values, without the need for additional complex logic (to be distinguished from the bcla needing such logic). The network 14 provides a double-prediction approach to generating certain carry out values. That is, muxes 14D and 14E both select between two possible prediction values, the outputs of adders 16-1-1 and 16-1-2. As noted above, muxes 14D and 14E are controlled by two different selector signals. Next, mux 14F selects between the outputs of muxes 14D and 14E. It is this approach which avoids the need for complex logic and provides high speed of operation.

Turning to Fig. 3, there is shown the circuitry (in block diagram form) for generating the carry outputs from the adders 16-i-1, which have CIN = 0 (i.e., a carry-in set to logical 0). Correspondingly, Fig. 4 shows the circuitry for generating the carry outputs from the adders 16-i-2, which have CIN=1 (i.e., a carry-in value set to logical 1). In these two figures, the

nomenclature used specifies bit 0 of a first operand "a" as a0, bit 1 of that operand as a1, bit 7 of a second operand "b" as b7, etc. The ampersand indicates a logical AND operation while the vertical bar indicates a logical OR.

For bits 1 through 7, the 8-bit adder circuit 16-i handles three groups of bits in parallel paths. The first group of bits is a single bit, bit 1; the second group comprises bits 2 and 3, while the third group comprises bits 4 - 7. This partitioning equalizes the number of "vertical" and "horizontal" delay stages through with carry signals have to propagate so that the carry propagation path for 8-bit addition is, at most, one NOR or NAND gate plus three 2:1 multiplexers. Using modern manufacturing systems, the differential delay should only be about .5 - .6 nanoseconds.

Sum bits are calculated differently. In the general case, sum(j) = S(j) = a(j) $\oplus$ b(j) $\oplus$ c(j), where $\oplus$ signifies an exclusive-OR or XOR operation. It will be seen, however, that only one XOR operation is necessary. That is, if c(j) = 0, S(j) = a(j) $\oplus$ b(j); and if c(j) = 1, S(j) = a(j) XNOR b(j), where XNOR signifies an exclusive-NOR operation. Turning to Fig. 5, XOR gate 52 generates the function a(j) $\oplus$ b(j). When that function is negated by inverter 54, the result is a(j) XNOR b(j). The outputs of XOR gate 52 and inverter 54 are supplied to the inputs of a 2:1 mux 56 which receives as a control input the signal ~c(j) = ~C(j-1), where the lowercase "c" represents a carry-in value and the upper case "C" represents a carry-out value. The output signal from the mux, s(j), is therefore a(j) $\oplus$ b(j) when c(j) = 0 and a(j) XNOR b(j) when c(j) = 1. Thus, when the carry-in value of a given bit position is ready, the only delay for the sum calculation is a 2:1 mux delay.

The general rule in other adder designs is that support for multiple bit formats comes at the expense of speed. By contrast, an adder constructed in accordance with the teaching of the present invention can perform 64-bit additions at virtually the same speed as 32-bit, 16-bit and 8-bit additions. Speed may potentially be lost at 8-bit module boundaries where carry signals should or should not propagate to adjacent modules, depending on the operation to be performed. For example, in 8-bit additions, the carry value should not propagate from one 8-bit module to another, whereas in 64-bit additions the carries must, of course, propagate from

one 8-bit module to another.

In order to avoid the use of circuits to gate carry-out signals between stages and effect the aforementioned propagate/do not propagate carry-gating stages, a logical value of 1 or 0 is forced on inverted carry-out signals of each 8-bit module by using for each of muxes 14B, 14F, 14H, 14I, 14S, 14T, 14W and 14X a Set/Reset 2:1 mux 60, depicted in Figs. 6 and 7. Thus every operation is, to the hardware, a 64-bit addition, albeit in some cases with a forced carry for some 8-bit modules. Of course, the module in the most significant position does not require such a multiplexer.

The SR muxes are activated by control lines derived from the opcode so they are stable when the operation starts and are not part of the critical paths. Looking at the SR mux circuitry in Fig. 7, one can see that there are no additional transistors in the carry path, only diffusion capacitance of the pull-up and pull-down transistors, 72, 76 and 82, 86, respectively.

Referring again to Figs. 3 and 4, in each module, the inverted carry ~c(7) is duplicated. One path goes through a regular mux 32 or 42 to influence the result. The other path goes through the SR mux 34 or 44 to the next 8-bit module, as described above.

The value of the inverted carry-out of each byte, ~c(7), is constrained through the use of a Set/Reset 2:1 mux 60, except that module md-7, as the module which processes the MSB, does not require such a mux. This approach avoids introducing an additional transistor in the critical path (i.e., the path of maximum delay) and the major additional impact on speed is only the diffusion capacitance of the pull-up and pull-down transistors.

Assume, as a first example, that it is desired to perform four 16-bit subtractions. Operand B, the second operand, is logically inverted before entering the adder and the carry-in value is set to 1. The seven pairs of inputs to the set/reset controls of the SR muxes are set as follows:

SR md-0 = 00 (i.e., no set, no reset)

SR md-1 = 01 (i.e., no set, reset)

SR md-2 = 00 (i.e., no set, no reset)

SR md-3 = 01 (i.e., no set, reset)

SR md-4 = 00 (i.e., no set, no reset)

SR md-5 = 01 (i.e., no set, reset)

SR md-6 = 00 (i.e., no set, no reset)

(Where SR md-x identifies the SR mux on the carry output of module md-x, "set" means the S or Set input of the mux is a 1, "no set" means the S input is a 0, "reset" means the R or Reset input is a 1 and "no reset" means the R input is supplied a 0 value.) Because the circuit operates on the basis of inverted carry values propagating from module to module, it is necessary to "reset" every carry-out byte output and not "set" it (i.e., assert the Reset input of the mux, not the Set input). In this way, the "inner" carry-out for path selection (i.e., the outputs of the SR muxes of Fig. 6) of every half word is forced to 0 and, obviously, the byte adder 16-i-2 with a carry-in of 1 will be selected as the sum output.

In this example, the inverted c7 carry-out bit of modules md-0, md-2, md-4 and md-6 is not needed. That will not be true for some other operations or formats.

Format selection of the data to be processed is thus achieved through the selection of correct values of the SR input pairs to the Set/Reset muxes.

As a second example, assume that it is desired to perform four 8-bit additions and one 32-bit subtraction. The seven pairs of inputs to the set/reset controls of the SR muxes are set as follows:

SR md-0 = 10 (i.e., set, no reset)

SR md-1 = 10 (i.e., set, no reset)

SR md-2 = 10 (i.e., set, no reset)

SR md-3 = 01 (i.e., no set, reset)

SR md-4 = 00 (i.e., no set, no reset)

SR md-5 = 00 (i.e., no set, no reset)

SR md-6 = 00 (i.e., no set, no reset)

As a third example, assume that it is desired to perform one 64-bit addition. CIN is assigned a value of 0 and the seven pairs of inputs to the set/reset controls of the SR muxes are set as follows:

SR md-0 = 00 (i.e., no set, no reset)

SR md-1 = 00 (i.e., no set, no reset)

SR md-2 = 00 (i.e., no set, no reset)

SR md-3 = 00 (i.e., no set, no reset)

SR md-4 = 00 (i.e., no set, no reset)

SR md-5 = 00 (i.e., no set, no reset)

SR md-6 = 00 (i.e., no set, no reset)

64-bit subtraction is achieved with the same mux control settings but a CIN value of 1 is used and the second operand is inverted.

Thus the invention provides a method and apparatus which generates all possible combinations of data sum and carry-out paths for each byte, half-word and word data format (assuming N-bit word size), yielding very simple and predictable carry propagation paths and delays. The carry-out used for path selection is established independently of the calculation carry-out (i.e., the carry-out from an addition or subtraction of operand values), thus facilitating the kind of segmentation needed in order to support operation on data of varied formats (i.e., word sizes). Every sum calculation requires the same calculation delay, regardless of data format. The difference in calculation delay from one format to another is thus a result only of carry generation and is, in any event, only the delay through one 2:1 mux when comparing the generation of a result for 64-bit words with that for 32-bit words.

The speed of operation of adders according to the invention will now be understood. Consider as a point of reference one of the known fastest methods to propagate a carry value, the binary carry-look-ahead (bcla) scheme using the Kogge-Stone algorithm. As an example,

assume two 8-bit bytes x and y (i.e., x[[7:0] and y[7:0]) are to be added. The carry-out of MSB 7, c7, is calculated in three levels. At the first level, Gj(1) = x(j) * y(j); Pj(1) = x(j) + y(j), for $7 \leq j \leq 0$. At the second level, Gj(2) = Gj(1) + Pj(1)*G_{j-1} and Pj(2) = Pj(1)*P_{j-1}(1), for j = 7, 5, 3, 1. At the third level, the calculations are the same aas for the second level for j = 7 and 3. At the fourth level, the calculation is done for only j = 7. Also, c7 = G7(4) + cin P7(4). This means there are five logical levels in the path, four for G and P calculations and one for C7 output; and four of those levels involve complex gates such as calculate a + (b*c).

In the present invention, by contrast, c7 has only one logical level in its path and four pass transistor muxes (three of the varieties illustrated in Figs. 3 and 4 plus one to select between the adders having cin = 0 and cin = 1). These are much simpler and faster than complex gates.

In the bcla scheme, moreover, extending carry propagation from an 8-bit operation to a 16-bit operation requires an additional complex gate whereas the instant invention requires only a 2:1 pass transistor mux.

Further, in the bcla scheme, the carry calculation for every bit position requires additional logic. In the worst case, this means two complex logica gates are needed for, for example, bit 5. By contrast, the approach taken by the present invention requires at most a single 2:1 pass transistor mux.

Thus the structure of an adder according to this invention facilitates operation in a multi-format, multiple combination of operations arrangement without significant speed penalties. The structure may be optimized, as illustrated, for the largest word format to be accommodated and it then will support smaller formats, as well, by taking advantage of the partitioning afforded by the use of the 2:1 SR muxes which control the carry values between modules.

Such operation is afforded by a combination of features. First, the carry in value CIN can be used to select the output of mux 14F even though the outputs of adders 16-1-1 and 16-

1-2 have no direct relationship with CIN. Second, the output of mux 14F (i.e., the carry out of bit 15) can be used to select the carry out of bit 31, the output of mux 14I. Third, the carry out of bit 31, the output of mux 14I, can be used to select the carry out of bit 63, the MSB from mux 14P. In other words, CIN selects the carry out of bit 15, which selects the carry out of bit 31, which selects the carry out of bit 63. That is, a real binary selection tree is created out of muxes.

The foregoing illustrative embodiment is presented by way of example only. It will be appreciated by those skilled in the art that various alterations and improvements may be made without departing from the spirit of the invention. For example, an adder according to the invention may be designed to process operands of different sizes than those used in the examples. Also, though the examples depict uniformly-sized 8-bit adder modules, adder modules of other sizes may be employed and the adder modules need not all be the same size. Thus the invention is to be limited not by the foregoing examples, but only by the following claims and equivalents thereto.

What is claimed is:

CLAIMS

1. A carry forward adder capable of adding one or more pairs of operands, each pair having a first operand and a second operand, the total number of bits in all of the first operands being N, comprising:

Q adder-and-carry select ahead modules, each for adding a pair of R_i operands (where i is an index ranging from 1 through Q), each module having a first adder and a second adder, said first adder and said second adder receiving the same data bits as input while the first adder receives a carry-in value of logical 0 and the second adder receives a carry-in value of logical 1;

a multiplexer network connected to receive as inputs sum outputs from the adders and to receive as control signals carry-out values from the adders; and

each adder-and-carry-select ahead module including means for generating a sum output representing the sum of a first operand and a second operand, and a carry-out value, the means for generating a carry-out value including means for setting said carry-out value to a desired logical value,

whereby the word sizes of operands which may be added may be selected by setting module carry-out values to appropriate logical values.

2. An adder for generating the sum of two operands, comprising:

a plurality of substantially identical adder modules disposed in pairs;

a first adder in each pair having a carry input set to a logical zero value and a second adder in each pair having a carry input set to a logical one value;

the adder modules generating substantially all possible sum and carry output values from operand and carry input values; and

a multiplexer network which selects from the possible sum and carry outputs correct values to provide the sum of the operands.

3. An adder capable of adding one or more pairs of operands, each pair having a first operand and a second operand, the total number of bits in all of the first operands being N, comprising:

a plurality of substantially identical adder modules (md-0 through md-(M-1)); and

a multiplexer network (14) connected to receive outputs from the adder modules and to select among such outputs;

each module having two adders (16-i-1 and 16-i-2) which receive the same operand data and generate bit-wise sum outputs and carry outputs;

a first adder in each module being provided with a carry-in value of zero and a second adder in each module being provided with a carry-in value of one; and

the multiplexer network selecting one of the sum outputs of each of the first and second adders to produce the sum output of each module, under control of the carry outputs of those adders.

4. The adder of claim 3 wherein the adders in the modules are carry-select-ahead type adders.

5. The adder of any of claims 1-4 wherein the difference in time to calculate N-bit results, relative to the time to calculate N/2-bit results, is only a propagation delay of a multiplexer.

6. The adder of any of claims 1-4 wherein the multiplexer network operates as a mechanism to select from all possible paths and outcomes for both carry values and data sum bits from the adder modules.

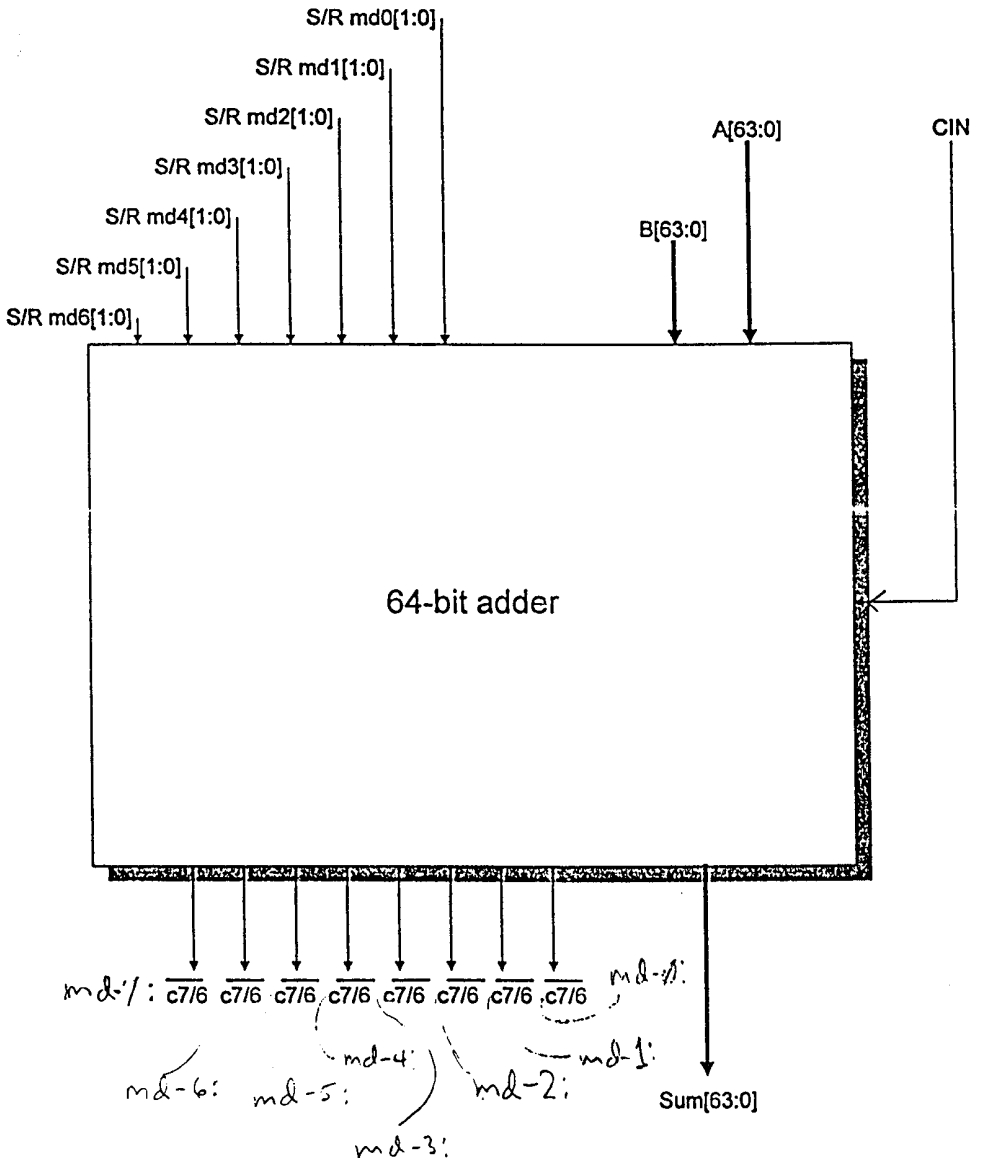
7. The adder of claim 6 wherein the multiplexer network generates the carry-out values for each bit as a by-product of sum-bit generation.

8. The adder of any of claims 1-4 wherein each adder module is an 8-bit adder which processes three groups of bits in parallel paths, a first group of bits being a single bit, a second group of bits being two bits in a next-most significant position, and a third group of bits being the four most significant bits, such that a number of vertical and horizontal delay stages through which carry signals propagate are substantially equalized and the carry propagation path for 8-bit addition is at most a single gate and three multiplexers.

9. The adder of the claims 1-4 wherein the multiplexer network comprises a plurality of SR multiplexers activated by control lines derived from an opcode so as to be stable when an addition operation starts.

10. The adder of claim 9 wherein the inputs to the set/reset controls of the SR multiplexers are presented with logic values to format the size or sizes of operand data to be processed.

11. An adder formed of multiple modules operable and interconnectable to accommodate multi-format operand data of selectable word size, as substantially shown and described herein.


S/R md0[1:0]
S/R md1[1:0]
S/R md2[1:0]
S/R md3[1:0]
S/R md4[1:0]
S/R md5[1:0]
S/R md6[1:0]
A[63:0]
CIN
B[63:0]
64-bit adder
md-7: c7/6 c7/6 c7/6 c7/6 c7/6 c7/6 c7/6 c7/6
md-0:
md-6:
md-5:
md-4:
md-3:
md-2:
md-1:
Sum[63:0]

10

FIG. 1

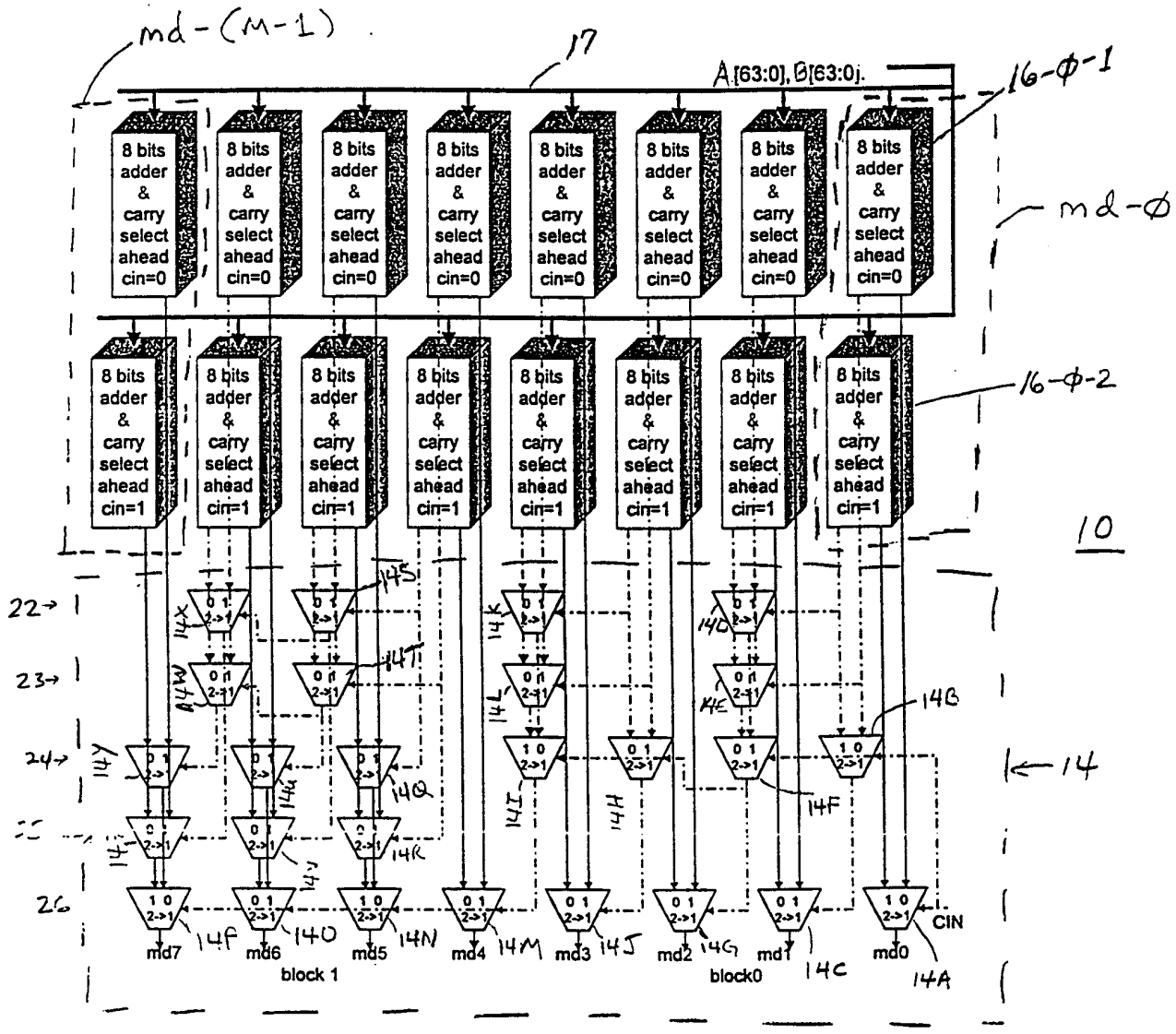

md-(M-1)
17
A[63:0], B[63:0]
16-Φ-1
md-Φ
16-Φ-2
10
14
8 bits adder & carry select ahead cin=0
8 bits adder & carry select ahead cin=1
22
23
24
26
md7
md6
md5
md4
md3
md2
md1
md0
block 1
block0
CIN

FIG. 2

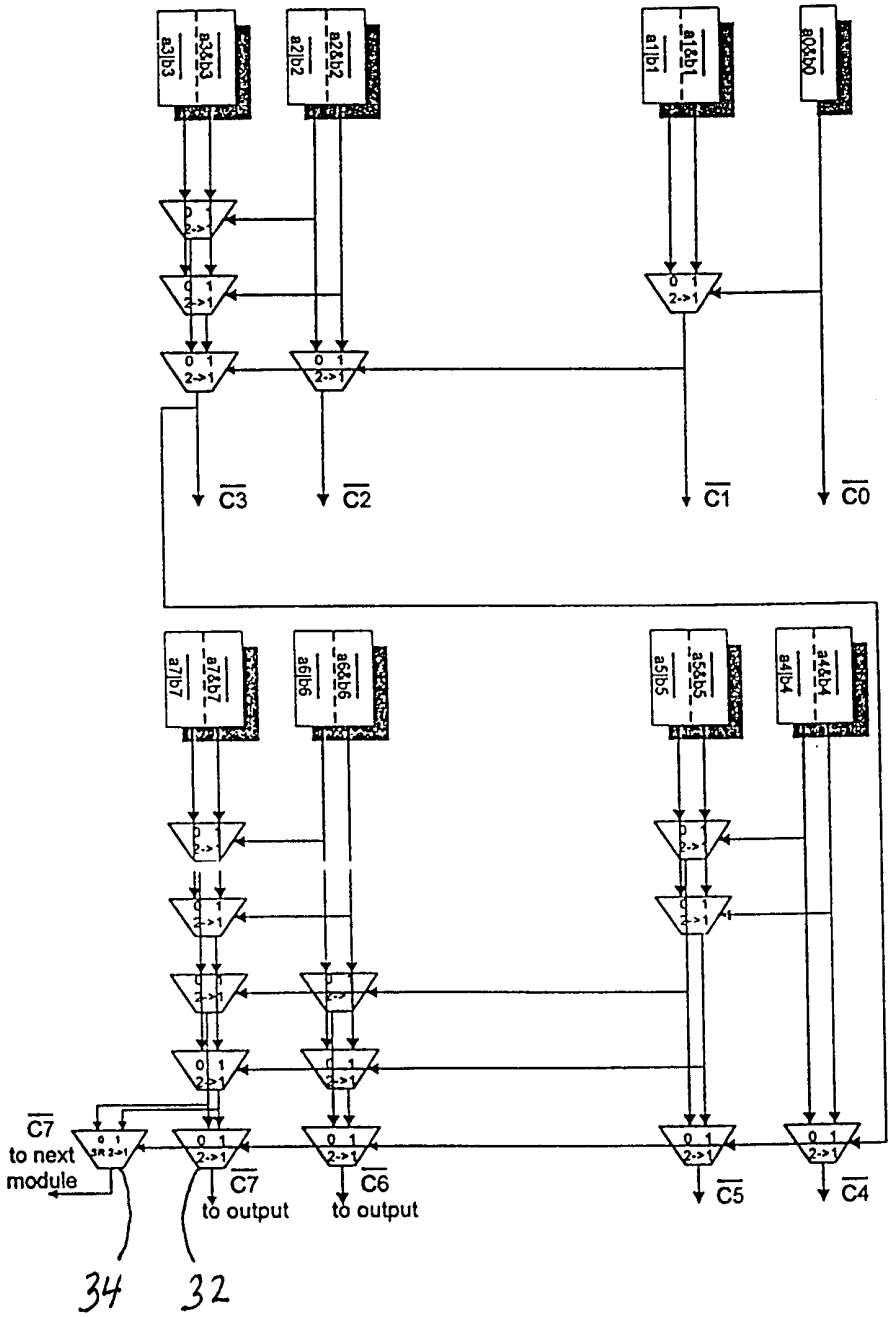

a3|b3
a2|b2
a1|b1
a0|b0
C3
C2
C1
C0
a7|b7
a6|b6
a5|b5
a4|b4
C7 to next module
C7 to output
C6 to output
C5
C4
34
32

FIG. 3

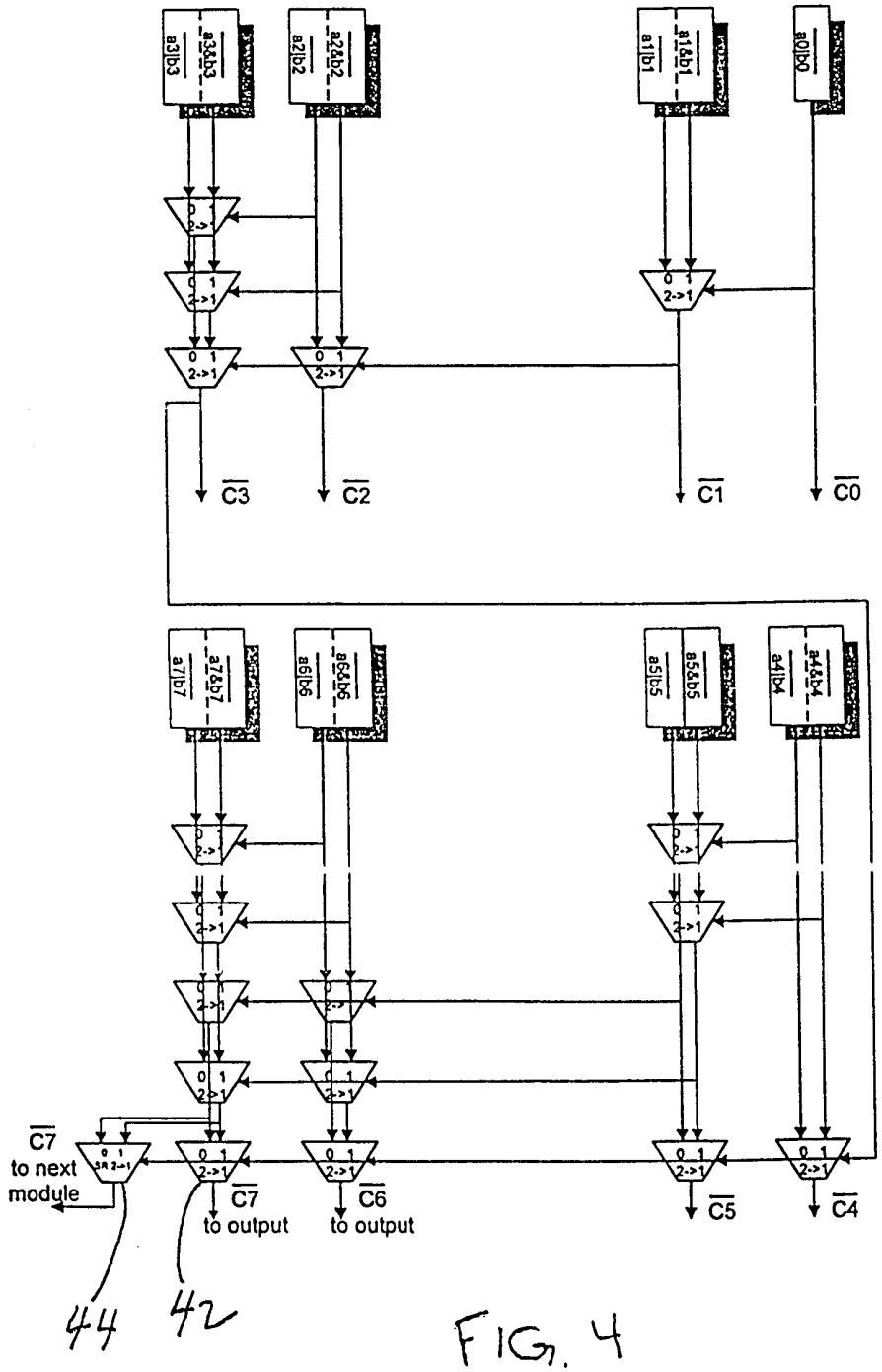

a3|b3
a2|b2
a1|b1
a0|b0
a7|b7
a6|b6
a5|b5
a4|b4
C3
C2
C1
C0
C7
to next
module
C7
to output
C6
to output
C5
C4
44
42

FIG. 4

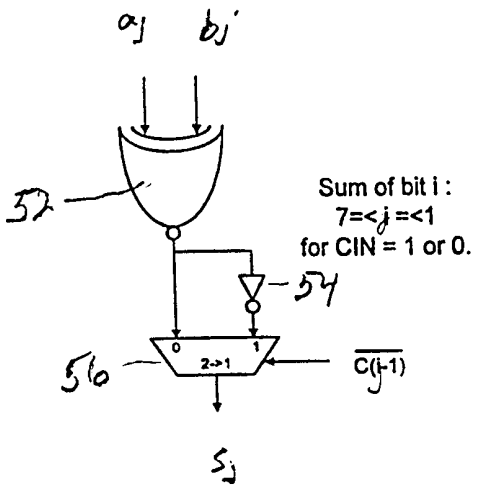


FIG. 5

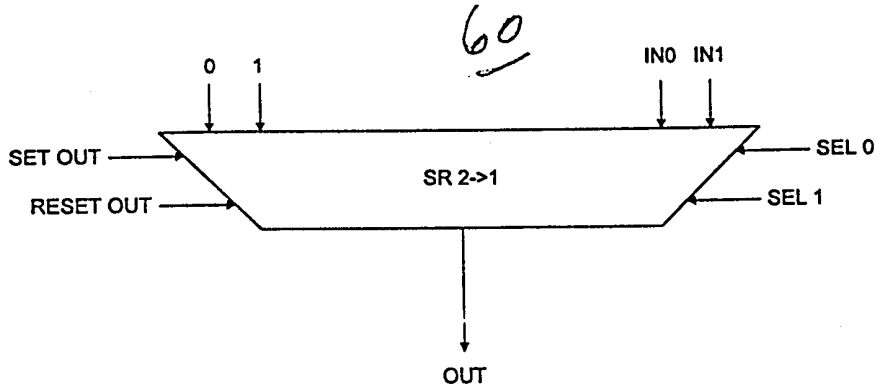

60
0
1
IN0
IN1
SET OUT
RESET OUT
SR 2->1
SEL 0
SEL 1
OUT

FIG. 6

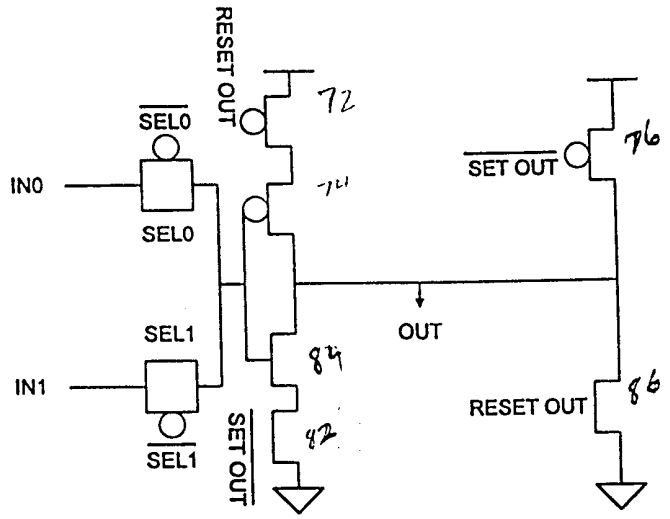

RESET OUT
72
SEL0
IN0
SEL0
74
SET OUT
76
SEL1
OUT
IN1
84
SEL1
SET OUT
82
RESET OUT
86

FIG. 7

INTERNATIONAL SEARCH REPORT

Inte. onal Application No
PCT/US 00/03879

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F7/52

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	KONDO Y ET AL: "SP 25.5: AN EARLY-COMPLETION-DETECTING ALU FOR A 1GHZ 64B DATAPATH" IEEE INTERNATIONAL SOLID STATE CIRCUITS CONFERENCE,US,IEEE INC. NEW YORK, vol. 40, 1 February 1997 (1997-02-01), pages 418-419,497, XP000753159 ISSN: 0193-6530	2-7,9
Y	figure 3	1,10
A		8

	-/--	

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
13 June 2000	19/06/2000

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL – 2280 HV Rijswijk Tel. (+31–70) 340–2040, Tx. 31 651 epo nl, Fax: (+31–70) 340–3016	Cohen, B

1

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Inte .onal Application No
PCT/US 00/03879

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
Y	"HIGH-SPEED LOGIC TO IMPLEMENT BYTE ADDITION WITHOUT IMPACTING THE PERFORMANCE ASSOCIATED WITH THE ADDITION OF HALF- OR FULL-WORD ENTITIES" IBM TECHNICAL DISCLOSURE BULLETIN,US,IBM CORP. NEW YORK, vol. 33, no. 7, 1 December 1990 (1990-12-01), pages 9-11, XP000210755 ISSN: 0018-8689	1,10
A	the whole document	2-9
X	US 5 579 254 A (KUMAR SUDARSHAN ET AL) 26 November 1996 (1996-11-26) figures 5,10,17	2-9
X	US 5 396 445 A (LAL SHIANG J) 7 March 1995 (1995-03-07)	2
A	abstract	5
A	EP 0 654 733 A (HEWLETT PACKARD CO) 24 May 1995 (1995-05-24) page 4, line 23 - line 57; figure 4	1,10

1

INTERNATIONAL SEARCH REPORT

information on patent family members

Inte onal Application No
PCT/US 00/03879

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 5579254 A	26-11-1996	US 5471414 A	28-11-1995
US 5396445 A	07-03-1995	NONE	
EP 0654733 A	24-05-1995	EP 0924601 A	23-06-1999
		JP 7200260 A	04-08-1995
		US 5636351 A	03-06-1997