

-1-

SPATIALLY PIPELINED ARITHMETIC LOGIC UNIT

Background of the Invention

Technical Field This invention relates to the field of digital circuits and, in particular, to digital circuits for performing arithmetic and logic operations.

Background Art The arithmetic and logic unit (ALU) is a central component of any processor. The ALU performs addition, subtraction, and logical operations, and it is used in every instruction in which an address must be calculated. Because of its central role and complex function, the ALU is usually a critical path for determining the speed at which the processor can operate. For example, the ALU self-bypass path determines how much time is required from the start of an ALU operation until the result of the operation is available for use in a subsequent calculation. Processor designers typically use the ALU self-bypass path to set the cycle time target for the processor.

Currently, much effort is being invested in optimizing the design of 64-bit adders because such adders are often the slowest component in a 64-bit ALU. The various optimization techniques may be understood with reference to the logical equations used to sum two single-bit numbers (A and B):

$$\begin{aligned} \text{(I)} \quad & \text{SUM} = A \oplus B \oplus C_{in} \\ \text{(II)} \quad & C_{out} = G1 + P1 \Sigma C_{in} \end{aligned}$$

Here, G1 is a 1-bit generate signal given by $A \Sigma B$, P1 is a 1-bit propagate signal given by $(A + B)$ or $(A \oplus B)$, " Σ " represents the logical AND operation, "+" represents the logical OR operation, " $\oplus$ " represents the exclusive OR (XOR) operation, C_{out} is the carry bit generated by summing A, B, and C_{in} , and C_{in} is the carry bit generated by a bit of the adder. Because C_{in} is generated by a previous bit sum, straightforward application of (I) and (II) in an N-bit adder would require that the bits be summed in N-sequential stages, with the C_{out} generated by each stage being rippled to the next bit pair. The resulting latency for N-bit adders quickly becomes prohibitive as N (the adder width) increases.

Conventional adders employ various techniques to avoid summing bits sequentially. In particular, carry-lookahead and carry-select techniques are often combined to implement a high degree of parallelism in N-bit adders. Carry-

-2-

lookahead techniques divide the N-bit adder into blocks of L-bits each. Propagate, generate, and carry-out signals (PL, GL, CL) for the L-bit blocks are calculated in parallel. Carry select techniques calculate speculative sums for each L-bit block, for C_{in} to each block equal to 0 and 1. In each case, C_{in} for a block (j) is equal to CL for the preceding block (j-1). The appropriate L-bit sums and carries are selected as the L-bit results from the previous blocks become available.

It can be shown that, under certain conditions, an N-bit adder employing these techniques requires $(2 + \log L [N])$ gates. One gate calculates P and G signals from the L-bit operands, the $\log L [N]$ gates generate the higher order propagate and generate signals and the carry-out, and the remaining gate performs the final XOR, e.g. Equation (I). For a 64-bit adder with $L = 4$, this amounts to a 5 gate delay. While the logarithmic dependence indicates that larger Ls are desirable, a CMOS gate's operation slows markedly if it is driven by more than four inputs (fan-in > 4). Thus, CMOS adders typically employ gates having fan-ins of 4 or less. Moreover, the number of gates in the adder critical path provides only a rough estimate of the adder speed, since the cycle time depends on the complexity of the gates used and the loading on the gates.

Certain adder designs implement a Ling pseudo-carry, which employs a modified generate signal (H) in addition to the propagate signal (P). The modified generate signal allows 4-bit H and P signals (H4, P4) to be calculated directly from the operand bits, using a fan-in of 4 gate. This eliminates one gate from the adder but requires the use of larger, slower gates for calculating the propagate and carry signals. An adder employing this techniques is described in S. Naffziger, "A Sub-Nanosecond 0.5mm 64b Adder Design", Proceedings of the 1996 IEEE International Solid-State Circuits Conference.

Using optimized technology and parallel architectures, these known adders can provide a 64-bit result in approximately 7 fan-out of 4 inverter delays (slightly less than 1 nanosecond with 1996 CMOS technology). In each case, the critical path is limited by the delay of the carry chain. In order to increase the operating frequencies of processors beyond those frequency increases supported by simple process improvements, even faster adder architectures will be required.

Summary of the Invention

The present invention is an N-bit adder that minimizes both the latency of computing the least significant bits of an operation and the cycle time of the adder, at the expense of longer latency for the most significant bits.

In accordance with the present invention, k (N/k)-bit adder blocks, each comprising (N/k)-bit sum logic and (N/k)-bit carry-out logic, are coupled together in a pipelined fashion to form the N-bit adder. The sum logic of an adder block provides an (N/k)-bit result to an output of the adder block. The carry-out logic of an adder block determines the (N/k)-bit carry-out signal for the adder block and couples it to an adjacent adder block.

For an N-bit adder in accordance with the present invention, data flows across the k-adder blocks in a pipelined fashion. The N/k least significant bits of the result are available with a delay determined by the (N/k)-bit sum logic, while the higher order bits are available at subsequent intervals, determined by propagation of the carry-out signals between blocks.

Brief Description of the Drawings

The present invention may be best understood with reference to the following detailed description and the accompanying drawings.

Fig. 1 is a timing diagram for a conventional N-bit adder performing a two step computation.

Fig. 2 is a timing diagram of a pipelined N-bit adder in accordance with the present invention, performing a two step computation.

Fig. 3 is a block diagram of one embodiment of a pipelined N-bit adder in accordance with the present invention.

Fig. 4 is a block diagram of one embodiment of a 64-bit pipelined adder in accordance with the present invention.

Fig. 5A is a block diagram of one embodiment of a 16-bit adder block of the 64-bit adder of Fig. 4.

-4-

Fig. 5B is a schematic diagram of one embodiment of a 4-bit adder sub-block of the 16-bit adder block of Fig. 5A.

Fig. 5C is a schematic diagram of one embodiment of the carry-out logic for the 16-bit adder block of Fig. 5A.

Detailed Description

Referring first to Fig. 1, there is shown a timing diagram 100 for a conventional N-bit adder including a self-bypass period 102. The timing diagram shows a two step computation, $C = A + B$ (104) and $E = C + D$ (106), separated by self-bypass period 102 in which the result, C, of the first addition is sent back to the adder as an operand for the second addition. The bracketed expression [N-1:0] indicates that the adder operates on all N-bits (N-1 through 0) simultaneously. For purposes of this discussion, it is assumed that the N-bit adder can be segmented into k-(N/k)-bit blocks, such as a 64-bit adder segmented into four 16-bit blocks.

Also indicated in Fig. 1 are times T1, T2, and T3. T1 is the time required to produce the (N/k) least significant bits of the N-bit result. T2 is the self-bypass cycle time of the adder, i.e. the time required before the adder can begin second addition 106. T3 is the time required to produce the N-bit result. This is the time required before subsequent operations that require all N-bits of the result can begin. Because a conventional adder produces all N-bits at once, T1 is equal to T3. That is, the N/k-bit result is only available when the full N-bit result has been calculated.

Determining the N-bit carry-out represents a significant contribution to T1 and T3, because the carry chain is in the critical path of the adder. For this reason, the carry-out logic is implemented using relatively large gates to speed signal propagation through the carry chain, which is in the critical path of T3. Employing relatively large gates in the carry-out logic increases the load on the sum logic path, which sets T1. However, since minimizing T3 is the objective in these adders, the increase in T1 is considered acceptable.

Many applications of an adder do not require that all N-bits be ready before the next operation begins. For example, adders are used repeatedly for address

calculations in which some subset of the N-bit result is used to initiate a memory read. Consider the case of a processor having a 4 KByte cache configured as 128 cache lines of 32 bytes each. In this configuration, bits 5-11 of an address calculation can be used to begin a row decode as soon as they are available. This is a critical step in cache reads, and anything that speeds up availability of these low order bits will reduce the latency of cache reads. For typical cache organizations, some subset of the sixteen least significant bits will be critical to accessing cache data rapidly, and the speed with which this data is accessed impacts the efficiency with which the processor operates. Therefore, reducing T1, the latency for the N/k least significant bits, is important.

Other operations may likewise commence as soon as a subset of the full N-bit result is determined. For example, the second step of the two step addition shown in Fig. 1 could begin as soon as a subset of the N-bit result from the first step is available, and left shifts are relatively common operations that may be started as soon as convenient sized sub-blocks of data are ready. As noted above, the self-bypass cycle time (T2) characterizing these operations is an important design parameter. Reducing T2 is necessary to increase the frequencies at which ALUs operate.

In summary, T2 ideally dictates the cycle time of the microprocessor and, for this reason, is the most important delay to minimize in an adder. T1 impacts the cache access latency and is also important to minimize. T3 sets the latency of only those operations that require all N-bits of the result (such as right shifts). Minimizing T3 is therefore less important than minimizing either T1 or T2.

The present invention is an N-bit adder that is formed from k (N/k)-bit adder blocks to take advantage of the lower latency provided by the smaller adder blocks. In particular, each of the k (N/k)-bit adder blocks comprises (N/k)-bit sum and carry-out logic, and the k-adder blocks are connected in a pipelined fashion. This is accomplished by coupling an adder block corresponding to a block of bits to an adder block for the next least significant bits through the carry logic of the first adder block and repeating the coupling with adder blocks for successively more significant bit blocks. The sum logic, which determines the (N/k)-bit result of the adder block, is coupled to an associated output for the adder block. This allows self-bypass to occur as soon as an N/k-bit result for the block

-6-

is available, rather than delaying until the entire N-bit adder finishes. Moreover, since the carry-out of each N/k-bit block is no longer critical to the N/k-bit result, the gates used to implement the carry-out logic can be downsized, reducing the load on the critical N/k-bit sum path. In addition, this topology reduces the length of wires in the adder and hence the loading on the critical path.

Referring now to Fig. 2, there is shown a timing diagram 200 reflecting the spatially pipelined architecture of an N-bit adder in accordance with the present invention. The timing diagram comprises k-timing diagrams 210(1)-210(k). Each diagram 210(1)-210(k) represents the processing by an adder block of an (N/k)-bit block of the N-bit operand to produce an (N/k)-bit result and a carry-out of the N/k-bit block (CN/k). As indicated by arrow 212(1), CN/k of the adder block represented by diagram 210(1) is calculated in parallel with the (N/k)-bit result and coupled to the adder block represented by diagram 210(2) in time for use in determining its (N/k)-bit result. The architecture is repeated for remaining blocks of the adder, as represented by timing diagrams 210(3)-210(k).

The times T1, T2, and T3 for a conventional adder are shown in Fig. 2 beneath the corresponding times T1', T2', and T3' of the spatially pipelined adder. The time to the low order (N/k)-bit result [N/k - 1: 0], T1', is shorter than T1 for the conventional adder. This is because generation of carries into bits [N-1:N/k] no longer loads the critical path to the (N/k)-bit result. Bypass time T2' for diagram 210(1) of adder architecture 200 is shorter than bypass time T2 for conventional adder 110 because only the N/k least significant bits must be known before bypass occurs. In particular, the gate that determines the N-bit result from the speculative (N/k)-bit sums and (N/k)-bit carry signals is eliminated from the path to the (N/k)-bit result. This reduces the number of gates in the adder self-bypass path from nine to eight, allowing the ALU to run approximately $12\frac{1}{2}\%$ faster.

The time required to generate the full N-bit result using a spatially pipelined architecture, T3', is greater than the corresponding time (T3) for a conventional adder. This is due to the fact that pipelining reduces the amount of parallelism that can be imposed on carry generation. In particular, calculation of CN/k for a block cannot complete until CN/k has been determined for the preceding block. The

-7-

extra latency $T_3' - T_3$ increases with k because the block carry must ripple through more blocks.

Referring now to Fig. 3, there is shown a block diagram of an N -bit spatially pipelined adder 300 in accordance with the present invention. The N -bit adder comprises k (N/k) -bit adder blocks 310(1) - 310(k). Throughout the following discussion, references to multiple adder blocks 310(1)-310(k) and their components will be not be indexed unless there is a need to distinguish a particular adder block. Where appropriate, a representative one of adder blocks 310 will be indexed as, for example, adder 310(j). This same indexing convention is employed in all subsequent figures.

As indicated in Fig. 3, each adder block 310 includes (N/k) -bit sum logic 320, (N/k) -bit carry-out logic 340, a first input 312 for receiving an (N/k) -bit block of an N -bit operand, a second input 314 for receiving a carry-in signal from a preceding block, an output 316 for providing an (N/k) -bit result, and an output 318 for providing an (N/k) -bit carry-out signal (CN/k) to an adjacent adder block 310, i.e. the next most significant block of bits. In this discussion, "sum logic" includes gates for calculating lower order P and G signals that are used to determine an (N/k) -bit result, e.g. (N/Lk) -bit P and G signals in an (N/k) -bit block. Carry-out logic 340 includes logic for generating CN/k from precursor propagate (P) and generate (G) signals. In conventional logarithmic adders, for example, the precursor P and G signals for (N/k) -bit carry-out logic 340 will be N/Lk -bit P and G signals, where L is the number of inputs driving the gate (fan-in). In particular, in an $N=64$ bit adder broken into $k=4$ blocks with $L=4$ fan-in, the precursors would be 4-bit generate and propagate signals, G_4 and P_4 , respectively. The present invention is not limited to adders based on logarithmic architectures.

The (N/k) -bit sum at output 316(j) of sum logic 320(j) does not depend on the CN/k carry-out at output 318(j). (N/k) -bit carry-out logic 340(j) operates in parallel with (N/k) -bit sum logic 320(j) beginning at the point in sum logic 320(j) at which the precursor P and G signals are calculated. Except for loading sum logic 310(j), (N/k) -bit carry-out logic 340(j) is effectively removed from the path to the (N/k) -bit result determined by sum logic 310(j), and the loading may be

-8-

reduced by downsizing carry-chain logic 340(j) to speed up the path to the (N/k)-bit result.

Referring now to Fig. 4, there is shown a block level diagram of one embodiment of a 64-bit spatially pipelined adder 400 in accordance with the present invention. Adder 400 comprises four 16-bit adder blocks 410(1)-410(4). Each adder block 410 comprises a fast, 16-bit carry-select adder 420 and 16-bit carry-out logic 440. Also shown in the figure are blocks 470, 412 for routing data to adder 420 and a block 430 for selecting a 16-bit result from adder 420 or from other ALU functional blocks (such as logical operations). Block 470 is an early bypass multiplexer (MUX) that selects non-critical results from the register file or writeback queue to generate ALU source data. Block 412 is a late bypass MUX that selects the ALU operand source from among MUX 470, a data cache read result, or an ALU result from the previous cycle of either its associated adder or another adder in a superscalar execution unit. Block 430 is a late result MUX that selects the 16-bit result of adder 420 or other ALU functional blocks and returns it to bypass MUX 412. Also shown in Fig. 4, to provide context, is the register file access path starting with a decoded register ID. The register file access path includes word line driver 450 and register/read circuitry 460.

For purposes of illustration, only gates in the critical path to the 16-bit result of adder blocks 410 and gates in the critical path to the 64-bit result of adder 400 are shown in Fig. 4. The critical path to the 16-bit result of adder 420 comprises gates 422, 424, 426, and 428. Gate 422 computes 2-bit propagate and generate signals, P2 and G2, respectively, gate 424 computes P4 and G4, gate 426 computes 4-bit carries (C4), and gate 428 (MUX 428) selects a 16-bit result according to the 16-bit carry-in to adder 420 and the 4-bit carries computed by adder 420. A speculative sum path (illustrated further in Fig. 5), which is not in the critical path of adder 420, is configured in parallel with gates 422, 424, and 426, to provide speculative sums to result MUX 428.

Carry-out logic 440 comprises a gate 442 for computing 16-bit generate and propagate signals (P16, G16), an inverting buffer 444, and a gate 446 for determining a 16-bit carry out signal (C16). Carry-out logic 440 determines C16 from P4 and G4 signals (precursor signals) produced by gate 424 in adder 420 and is discussed below in conjunction with Fig. 5C.

As indicated in Fig. 4, a 16-bit result is available at the output of MUX 428. In the disclosed embodiment, the least significant 16-bit result is shown driving the data cache to begin a row decode. The spatially pipelined structure of adder 400 allows the row decode to commence as soon as the 16-bit result is available, e.g. before 16-bit results from adders 420(2)-420(4) have been generated. Thus, the cache access latency (T1) of adder 410 is shorter than it would be if adder 400 first calculated all 64-bits of the result. By downsizing gates 442, 444, and 446 of carry-out logic 440, loading of adders 420 is reduced, further increasing the speed to the lowest order 16-bit result.

Also indicated in Fig. 4 is a self-bypass path formed by late result MUX 430 and late bypass MUX 412 in conjunction with adder block 420. Since the 16-bit result is available at MUX 428(1) with the relatively low latency of 16-bit adder 420(1), i.e. without recourse to any 16-bit carries determined by adder 400, the 16-bit result may be returned to late bypass MUX 412(1) faster. This allows a subsequent addition, subtraction, etc. to begin on the low order bits faster than would be possible if the full 64-bit result had to be generated first. Thus, the self-bypass cycle time (T2) of adder block 410 is reduced, and the operating frequency of adder 400 may be increased accordingly.

Adder 400 may be implemented in any of the available semiconductor technologies. A preferred embodiment of pipelined adder 400 is implemented in CMOS using opportunistic time borrowing domino logic to eliminate latch delays from the critical path. Dual rail domino techniques are used in order to implement non-monotonic functions. Opportunistic time borrowing domino logic is discussed in greater detail, for example, in U.S. Patent No. 5,517,436, which is hereby incorporated by reference.

Referring still to Fig. 4, gates of block 410(4) corresponding to gates 414, 416, 422, 424, 426, 428, 432, 434, 442, 444, and 446 are marked with a D or an S, followed by a number that indicates the gate structure in a domino implementation of adder 400. Here, D signifies a domino gate, S signifies a static gate, and the number indicates the number of transistor in the pull-down/pull-up chain of the D/S gate, respectively. In the domino embodiment, early bypass MUX 470 comprises a domino MUX 472 and a static inverter 474. Word-line driver 450 comprises a domino inverter 452 and a static inverter 454, and bit-line

-10-

sense module 460 comprises an SRAM cell 462 and a static inverter 464. In addition, block 480 represents a staging network necessary to coordinate clock signals among the domino gates of adder blocks 410(1)-410(4).

In the remaining discussion, dual-rail domino circuits are used for illustration, with a polarity of each gate and signal disclosed. The complementary signals and the gates for generating these complementary signals may be readily determined from the illustrated embodiments using known methods.

Referring now to Fig. 5A, there is shown a block diagram of one embodiment of fast 16-bit adder 420. In this embodiment, 16-bit adder 420 is partitioned into four 4-bit adders 510(1)-510(4) and carry-out logic 520. Each of 4-bit adders 510 has inputs 512 for receiving 4-bit sub-blocks of the 64-bit operand, an input 514 for receiving a 16-bit carry-in from the preceding stage, and inputs 515 for receiving 4-bit carry-ins ($C4_{in}$). Adder 510(1), which handles the lowest order 4-bits of the 16-bit block, has its inputs 515 coupled to 0 and 1, whereas $C4_{ins}$ for adders 510(2)-510(4) are determined from 16-bit block 520 using P4 and G4 signals calculated from adders 510(1)-510(3). Adders 510 are discussed in greater detail below in conjunction with Fig. 5B.

Carry-out logic 520 includes lookahead circuitry for propagating 4-bit carries between 4-bit adders 510 for the cases where $C4_{in}$ to adder 510(1) equals 0 and 1. In particular, 16-bit block 520 implements logic equation (III):

$$(III) \quad C4(j)_{out} = G4(j) + P4(j)\Sigma C4(j-1)_{in}$$

Here, the index, j, specifies one of adders 510(1)-510(4). For adder 510(1), $C4(0)_{in}$ is 0 and 1 as discussed above. For example, $C4(1)_{out} = G4(1)$, when $C4(0)_{in} = 0$, and $C4(1)_{out} = G4(1) + P4(1)$ when $C4(0)_{in} = 1$. These functions are implemented by buffer 522 and OR gate 524, respectively, in 16-bit block 520.

The calculation and corresponding logic gets more complex, further down the carry-chain. For example, the carry-in to adder 510(4), $C4(4)_{in}$, is given by logic equation (IV):

$$(IV) \quad C4(3)_{out} = G4(3) + P4(3)\Sigma G4(2) + P4(3)\Sigma P4(2)\Sigma G4(1) + P4(3)\Sigma P4(2)\Sigma P4(1)\Sigma C4(0)_{in}.$$

This is implemented by complex gates 526 and 528 for the cases $C4(j-1)_{in}$ equals 0 and 1, respectively. Here, "complex gates" refers to gates formed by combining component AND and OR gates as can be done efficiently in CMOS logic.

Complex gates are indicated by placing component gates in contact with each other. In particular, complex gate 528 is in the critical path to the 16-bit result of adder 420 due to its size and complexity, and corresponds to gate 426 in Fig. 4. Gates 422, 424, and 428, which are complex gates that complete the critical path of adder 420, are components of adders 510 and are discussed in greater detail below.

Referring now to Fig. 5B, there is shown a schematic diagram of one embodiment of a 4-bit adder 510. In the disclosed embodiment, adder 510 is divided into four sets of gates 552, 554, 556, 558 reflecting the separation of gates 422, 424, 426, 428 of Fig. 4. Dashed vertical lines indicate the boundaries between the different gate sets 552, 554, 556, 558. In the domino embodiment of adder 510, the dashed lines also represent the boundaries between domino and static gate sets, as indicated in the figure.

Each of gate sets 552, 554, 556, 558 is also labeled according to the signal it generates. For example, complex gates 532, 534, 536, 538 of set 552 generate $G2<0>$, $P2<0>$ and $G2<1>$, $P2<1>$ signals from input bits $a<3:0>$ and $b<3:0>$. The remaining gates of set 552 determine 1-bit generate and propagate signals $P1<0>-P1<3>$ and $G1<0>$ and $G1<2>$. Where, as in this case, $P1<n>$ is defined as the XOR of operand bits $a<n>$, $b<n>$, $P1<n>$ is also equal to the bit sum for $Cin<n> = 0$. Due to its relative size and complexity, complex gate 532 is in the critical path of adder 510, and corresponds to gate 422 in Fig. 4.

In gate set 554, complex gate 542 and gate 544 determine the 4-bit generate and propagate signals $G4$ and $P4$, respectively, from $P2<0>$ $G2<0>$, $P2<1>$, $G2<1>$. The lower gates of set 554 determine 1-bit carry-outs from 1-bit propagate and generate signals $P1<0>$, $P1<1>$ $P1<2>$, $P1<3>$ and $G1<0>$, $G1<2>$, respectively. Because of its relative size and complexity, complex gate 542 is in the critical path of adder 510 and corresponds to gate 424 in Fig. 4.

For gate set 556, lower gates determine speculative sum pairs $s0<0>/s1<0>$, $s0<1>/s1<1>$, $s0<2>/s1<2>$, and $s0<3>/s1<3>$ for bits 0-3

- 12 -

assuming carries in of 0/1, respectively. Upper gates of set 556, which are included in 16-bit block 520, determine speculative 4-bit carries, i.e. 4-bit carries for $C16_{in} = 0$ and 1. As noted in conjunction with Fig. 5A, gate 528 is in the critical path of adder 510 and corresponds to gate 426 in Fig. 4.

Gate set 558 comprises MUXs 559, which select one of the speculative sums $s0/s1$, according to the 4-bit carry-outs of adder 510 and the 16-bit carry-in to the stage comprising adders 510. Since it is the only gate present in gate set 558, MUX 559 is in the critical path of adder 510 and corresponds to gate 428 in Fig. 4.

Referring now to Fig. 5C, there are shown block diagrams of embodiments of gate sets 560, 570, 580 for determining $C16$ from $P4$ and $G4$ signals produced in adders 510(1)-510(4). In particular, gate set 560 includes complex gate 562 which determines $G16$ from $G4$ and $P4$ signals calculated in 16-bit block 420, and gate 564, which determines $P16$ from the $P4$ signals. Due to its size and complexity, complex gate 562 is in the critical path and corresponds to gate 442 in Fig. 4. Gate set 570 includes buffer 572, which corresponds to buffer 444 in Fig. 4, and gate set 580 includes complex gate 582 to compute $C16$, which corresponds to gate 446 of Fig. 4. Persons skilled in the art will recognize that there are alternative circuits for generating these signals that fall within the scope and spirit of the present invention.

There has thus been provided an N -bit adder having a spatially pipelined configuration to reduce the time to the lowest order bit result ($T1$) and the bypass cycle time ($T2$). This is accomplished by segmenting the adder into k (N/k)-bit adder blocks and coupling the adder blocks in a spatially pipelined fashion. The invention enables self-bypass after the shorter latency of an (N/k)-bit adder block. It also enables downsizing the carry logic to reduce loading on the critical path to the low order (N/k)-bit sum.

The present invention has been described in terms of several embodiments solely for the purpose of illustration. Persons skilled in the art will recognize from this description that the invention is not limited to the embodiments described, but may be practiced with modifications and alterations limited only by the spirit and scope of the appended claims.



- 13 -

CLAIMS

What is claimed is:

1. An N-bit adder comprising k (N/k)-bit adder blocks, each (N/k)-bit adder block including sum logic for providing an (N/k)-bit result at an associated output and carry-out logic for determining a carry-out signal for the adder block, the k (N/k)-bit adder blocks being coupled in a spatially pipelined configuration through the carry-out logic of the adder blocks.
2. The N-bit adder of claim 1, wherein the (N/k)-bit carry-out logic determines the carry signal for the adder block according to precursor propagate and generate signals provided by the sum logic.
3. The N-bit adder of claim 1, wherein the carry-out logic is sized to reduce loading the sum logic.
4. The N-bit adder of claim 2, wherein the (N/k)-bit carry-out logic of each adder block is configured in parallel with the (N/k)-bit sum logic of the adder block.
5. The N-bit adder of claim 3, wherein the spatially pipelined configuration comprises coupling adder blocks 1 - (k-1) to adder blocks 2 - k, respectively, through the carry-out logic of adder blocks 1 - (k-1).
6. The N-bit adder of claim 1, wherein the sum logic comprises gates for determining speculative sums and low order generate and propagate signals from (N/k)-bit operands applied to the adder block.



- 14 -

7. The N-bit adder of claim 1, wherein the carry-out logic comprises gates for determining an (N/k)-bit carry signal from precursor propagate and generate signals.
8. An N-bit adder comprising first through kth (N/k)-bit adder blocks, each of the first through kth adder blocks including sum logic for providing an (N/k)-bit result to an associated output according to (N/k)-bit operands and a carry-in signal provided to the adder block, each of the first through kth adder blocks also including carry-out logic for determining an (N/k)-bit carry signal, the (N/k)-bit carry-out logic of the first through (k-1)th adder blocks being coupled to provide the carry-in signals to the second through kth adder blocks, respectively.
9. The N-bit adder of claim 8, wherein the sum logic of an adder block includes gates for determining (N/k)-bit speculative sums and selecting one of the speculative sums according to a carry-in signal.
10. The N-bit adder of claim 8, wherein the carry-out logic of an adder block comprises gates for determining the (N/k)-bit carry signal of the adder block according to the precursor propagate and generate signals provided by the sum logic of the adder block and coupling the carry signal to an adjacent adder block.
11. An N-bit adder comprising:
 - a first (N/k)-bit adder having an operand input, a carry input, a result output, and logic for determining generate and propagate signals and providing an (N/k)-bit result to the result output according to operand and carry-in signals provided to the operand and carry inputs, respectively;
 - a first (N/k)-bit carry generator coupled to the first adder, the first carry generator having a carry output and logic for generating an (N/k)-bit carry signal at the carry output from the propagator and generator signals of the first adder; and



-15-

second through k^{th} (N/k) -bit adders and (N/k) -bit carry generators, each adder having an operand input, a carry input, and a result output, and each carry generator having a carry output, the second through k^{th} adders being coupled to second through k^{th} carry generators substantially as described for the first adder and carry generator, the first through k^{th} adders and carry generators being further configured in a spatially pipelined manner that couples the carry-out outputs of the first through $(k-1)^{\text{th}}$ carry generators to the carry inputs of the second through k^{th} adders, respectively.

12. The N-bit adder of claim 11, wherein the first through k^{th} adders and carry generators are first through fourth 16-bit adders and carry generators, respectively.

13. The N-bit adder of claim 12, wherein each of the 16-bit adders comprises four 4-bit adders.

14. An N-bit adder comprising:

first through k^{th} (N/k) -bit adder blocks, each adder block having first and second inputs and first and second outputs, and including (N/k) -bit sum logic coupled to (N/k) -bit carry-out logic, for generating an (N/k) -bit result at the first output and an (N/k) -bit carry at the second output, respectively, from operand and carry-in signals provided at the first and second inputs, the second outputs of the first through $(k-1)^{\text{th}}$ units being coupled in a pipelined manner to the second inputs of the second through k^{th} units, respectively.



-16-

15. A method for determining an N-bit result using k (N/k) -bit adders each comprising (N/k) -bit sum logic and (N/k) -bit carry-out logic, the method comprising the steps of:

providing (N/k) -bit operands and a carry-in signal to the sum and carry-out logic of a first of the k (N/k) -bit adders;

applying the sum logic to the operands and carry-in signal to provide an (N/k) -bit result at a first output of the first of the (N/k) -bit adders and precursor signals to the carry-out logic;

applying the carry-out logic to the precursor signals and carry-in signal to provide an (N/k) -bit carry signal at a second output of the first of the (N/k) -bit adders; and

repeating the providing, applying, and applying steps with the second through k^{th} (N/k) -bit adders to provide the N-bit result in sequential (N/k) -bit blocks.