DEPARTMENT OF THE NAVY
OFFICE OF COUNSEL
NAVAL UNDERSEA WARFARE CENTER DIVISION
1176 HOWELL STREET NEWPORT RI 02841-1708

IN REPLY REFER TO

Attorney Docket No. 102484
21 April 2021

The below identified patent application is available for licensing. Requests for information should be addressed to:

TECHNOLOGY PARTNERSHIP OFFICE
NAVAL UNDERSEA WARFARE CENTER
1176 HOWELL ST.
CODE 00T2, BLDG. 102T
NEWPORT, RI 02841

Serial Number 17/123,194
Filing Date 16 December 2020
Inventor Makia S. Powell

Address any questions concerning this matter to the Technology Partnership Office at (401) 832-3339.

DISTRIBUTION STATEMENT
Approved for Public Release
Distribution is unlimited

PARALLEL HYBRID ADDER

STATEMENT OF GOVERNMENT INTEREST

[0001] The invention described herein may be manufactured and used by or for the Government of the United States of America for governmental purposes without the payment of any royalties thereon or therefor.

CROSS REFERENCE TO OTHER PATENT APPLICATONS

[0002] None.

BACKGROUND OF THE INVENTION

(1) Field of the Invention

[0003] The present invention is directed to a adder for digital circuitry. More particularly the invention is directed toward an efficient, multistage adder.

(2) Description of the Prior Art

[0004] Kogge-Stone addition is different from traditional ripple-carry based addition since it computes the sum all at once in parallel versus waiting for a carry to propagate from right to left. Traditionally, the problem with addition is waiting for the carry to propagate from the least significant bit, to the most significant bit of the answer. This is because the sum is actually computed digit-by-digit, or in the case of

binary, bit-by-bit from right to left. This is actually the same way that addition is done by hand.

[0005] For example, in base 10 to compute 21+ 19, 1+9 is first computed, which leaves 0 with a carry of 1. In the next stage, 2 and 1 are added which yields 3. The carry out of the previous stage is added which yields 3+1=4. Writing the results of each stage from most-significant to least significant digit, results in 40. The same is true for binary based addition, in order to compute X, where X =A+B, one would need to wait for the carry out to propagate from the least significant bits of A+B to the most significant bits of A+B.

[0006] FIG. 1 is a diagram of a prior art combination of ripple carry adders that is being used to make an **N**-bit adder 10. First addend **A** has **N** bits with each bit denoted as **n**. Second addend **B** has **N** bits with each bit denoted as **n**. By combining a number **M** of smaller adders 12_0, 12_{M-1}, and 12_M, it is possible to compute an **N**-bit adder having as many bits as desired. First addend **A** is separated into segments of bits $\mathbf{A_0}$, ..., $\mathbf{A_{M-1}}$, $\mathbf{A_M}$. Each segment should have the same number of bits as the corresponding smaller adder 12_m. Likewise, addend **B** is separated into segments of bits $\mathbf{B_0}$, ... $\mathbf{B_{M-1}}$, $\mathbf{B_M}$ with each segment having the same number of bits as the corresponding segment adder 12_m. Each adder 12_m produces a partial sum result $\mathbf{X_m}$ and a carry $\mathbf{CX_m}$. Carry $\mathbf{CX_m}$ is one bit. Carry $\mathbf{CX_m}$ is provided to the

next higher order adder 12_{m+1} as a carry in. The lowest order adder 12_0 can be a half adder because it does not receive a carry in. Highest order adder 12_M produces the most significant bit for the adder $\mathbf{CX_M}$. The final sum **S** is the combination of $\mathbf{CX_M}$, $\mathbf{X_M}$, $\mathbf{X_{M-1}}$, ... $\mathbf{X_0}$.

[0007] In order to compute **S=A+B**, obtaining a result must wait for the carry out of the least significant adder to make it to the most significant segment adder. This means that the most significant segment adder cannot compute until **M-1** full adder delays after **A** and **B** are entered into the combined adder. The more bits and adder segments, the more stages and more resulting area, power and delays within the adder.

[0008] The Kogge-Stone adder eliminates this delay by passing the carry out bits to all stages. Since all of the carry out bits are available in all stages, the result can be computed in parallel. This parallel computation requires dedicated carry propagation hardware connected to each bit of the answer. Each stage in the Kogge-Stone adder is one bit wide resulting in 32 stages for a 32-bit adder. Dedicated carry propagate logic is connected to all 32 stages. Since each stage is made of basic AND and XOR logic gates, is very interconnected with the dedicated carry propagation logic. Implementation of the Kogge-Stone adder causes a great increase in area and loss of

performance due to the increase in routing between the logical elements, and a greater number of required logical elements.

[0009] U.S. Patent No. 5,701,504 to Timko teaches an "Apparatus and Method for Addition Based Upon Kogge-Stone Parallel Algorithm." This uses the traditional Kogge-Stone Addition method which has the same number of stages as an adder having the same number of bits. The embodiment is particularly tailored for CMOS integrated circuits. The carry propagation routing necessary to make the carry of one stage available to all stages creates a larger area, slower speed, and increased power when applied to programmable logic solid state devices such as field programmable gate arrays (FPGAs). For many different size adders (up to 64 bits), Kogge-Stone adders are actually one half as fast as ripple carry adders when implemented on FPGAs.

[0010] There is thus a need to provide a digital adder that operates more efficiently than current digital adder designs when applied in programmable logic devices.

SUMMARY OF THE INVENTION

[0011] It is a first object to provide an adder utilizing fewer computational cycles.

[0012] Another object is to provide an adder particularly adapted for implementation with field programmable gate arrays.

[0013] Accordingly, there is provided a combined adder for **N** logical bits to produce a sum from a first addend having **N** first addend bits and a second addend having **N** second addend bits. A least significant adder produces a segment sum of the least significant bits and a carry out. Segment adder pairs are used for each higher order of significant sums. One segment adder produces a segment sum portion, and the other produces an incremented segment sum portion. Carry logic associated with each segment is utilized with a multiplexer to select the incremented segment sum portion or the segment sum portion. The selected segment sum portions are assembled with a most significant carry out to produce the sum.

BRIEF DESCRIPTION OF THE DRAWINGS

[0014] Reference is made to the accompanying drawings in which are shown an illustrative embodiment of the invention, wherein corresponding reference characters indicate corresponding parts, and wherein:

[0015] FIG. 1 is diagram of a prior art ripple carry adder.

[0016] FIG. 2 is a diagram of an adder according to a first embodiment.

[0017] FIG. 3 is a detail of carry logic utilized in the adder of FIGS. 2 and 4.

[0018] FIG. 4 is a diagram of an adder according to a second embodiment.

DETAILED DESCRIPTION OF THE INVENTION

[0019] In FIG. 2 there is shown an adder embodiment 20 that is tailored for use in programmable logic components such as field programmable gate arrays. This adder 20 provides a hybrid between ripple-carry adder logic and carry look-ahead logic. As in the prior art, adder 20 provides the sum **S** when a first addend **A** is added to a second addend **B**. First addend and second addend are **N** bits wide. Each individual bit is shown as **n**. Each addend is separated groups of adjacent bits or segments for provision to segmented adders 22_0, ... , 22_{M-1}, and 22_M and to segmented incremented adders 24_0, ... , 24_{M-1}, and 24_M . The segments **m** can be different bit widths, but they should have the same width as the associated segment adder 22_m and incremented segment adder 24_m.

[0020] Each segmented adder 22_m produces a segmented sum portion $\mathbf{X_m}$ and a segment carry out $\mathbf{CX_m}$. Likewise, each segmented incremented adder 24_m produces an incremented segmented sum portion $\mathbf{X'_m}$ and an incremented segment carry out $\mathbf{CX'_m}$. Segmented sum portion $\mathbf{X_m}$ and incremented segmented sum portion $\mathbf{X'_m}$ are the **K-1** least significant bits of a segmented adder having **K** bits. $\mathbf{CX_m}$ and $\mathbf{CX'_m}$ are the most significant bits. The incremented

segmented sum portion $\mathbf{X'_m}$ is equal to the segmented sum portion $\mathbf{X_m}$ plus 1. The incremented segment carry out $\mathbf{CX'_m}$ reflects any carry that results from adding 1 to the segmented sum portion $\mathbf{X_m}$. Thus, the output of the segment adder 22m is the segmented sum portion if no carry is received from the lower order segment **m-1**, and the incremented segment adder 24m output is the segmented sum portion if a carry in is received from the lower order segment **m-1**. There is no incremented segment adder for the lowest segment m=0 because this segment doesn't receive a carry in.

[0021] Carry logic 26_m is associated with segments 1 to **m** to select either the segmented sum portion $\mathbf{X_m}$ or the incremented segmented sum portion $\mathbf{X'_m}$ as the final segmented sum portion $\mathbf{S_m}$. For this purpose, segmented sum portion $\mathbf{X_m}$ and incremented segmented sum portion $\mathbf{X'_m}$ are provided to a segment multiplexer 28_m. In the embodiment shown, segment multiplexer 28_m provides incremented segmented sum portion $\mathbf{X'_m}$ as the final segment sum portion $\mathbf{S_m}$ if carry logic 26_m provides a 1 as the segment carry factor $\mathbf{Cf_m}$. If carry logic 26m provides a 0 as the segment carry factor $\mathbf{Cf_m}$, the final segment sum portion $\mathbf{S_m}$ is the segmented sum portion $\mathbf{X_m}$. Carry logic 26_m is shown in further detail in FIG. 3.

[0022] As detailed in FIG. 3, carry logic 26m receives a segment carry out $\mathbf{CX_{m-1}}$ and an incremented segment carry out,

$\mathbf{CX'_{m-1}}$ from the previous, lower order segment **m-1** at an XOR component 30_m. XOR component 30_m output along with the previous segment carry factor $\mathbf{Cf_{m-1}}$ are provided to an OR component 32_m. An OR logical function is performed on these components to give the current segment carry factor $\mathbf{Cf_m}$. Referring back to FIG. 2, current segment carry factor $\mathbf{Cf_m}$ is used by multiplexer 28_m to select the segmented sum portion $\mathbf{X_m}$ or the incremented segmented sum portion $\mathbf{X'_m}$ as the final segment sum portion $\mathbf{S_m}$.

[0023] For the segment where **M=1**, the carry factor for segment 0, $\mathbf{Cf_0}$, equals the segment carry out, $\mathbf{CX_0}$ because the XOR between segment carry out and the incremented segment carry out, if computed, is always 1. No carry logic is necessary for this segment.

[0024] A final segment carry logic 26_M is used to select between the final segment carry out $\mathbf{CX_M}$ and the final incremented segment carry out $\mathbf{CX'_M}$ in multiplexer 28_{M+1}. Selected final segment carry out is assembled with final segment sum portions $\mathbf{S_0}$,..., $\mathbf{S_{M-1}}$, and $\mathbf{S_M}$ to give the sum **S** as shown.

[0025] FIG. 4 shows an alternate embodiment. This embodiment is generally suboptimal because it introduces an additional computation cycle. It may be applied to efficiently use FPGA resources. In this embodiment, an incrementer 34_m is joined to each adder 34_1 ... 34_M. Incrementer 34m adds 1 to segmented sum portion $\mathbf{X_m}$ to compute incremented segmented sum portion $\mathbf{X'_m}$ and

the incremented segment carry out $\mathbf{CX'_m}$. As in the embodiment shown in FIG. 2, multiplexer 26_m is joined to receive segmented sum portion and incremented segmented sum portion to provide a final segment sum portion $\mathbf{S_m}$ based on a carry factor $\mathbf{Cf_m}$. Carry factor $\mathbf{Cf_m}$ is computed from the carry outs in the previous segment and the previous carry factor as shown in FIG. 3. All other components operate as in the embodiment shown in FIG. 2.

[0026] It will be understood that many additional changes in the details, materials, steps and arrangement of parts, which have been herein described and illustrated in order to explain the nature of the invention, may be made by those skilled in the art within the principle and scope of the invention as expressed in the appended claims. For example, the addends can be segmented into different bit lengths in order to best fit the programmable logic configuration. Similarly, as shown in FIG. 4, a two stage adder incrementer configuration can be used. Other configurations and optimizations can also be applied.

[0027] The foregoing description of the preferred embodiments of the invention has been presented for purposes of illustration and description only. It is not intended to be exhaustive, nor to limit the invention to the precise form disclosed; and obviously, many modification and variations are possible in light of the above teaching. Such modifications and variations that may be apparent to a person skilled in the art are intended

to be included within the scope of this invention as defined by the accompanying claims.

PARALLEL HYBRID ADDER

ABSTRACT OF THE DISCLOSURE

A combined adder for **N** logical bits to produce a sum from a first addend having **N** first addend bits and a second addend having **N** second addend bits. A least significant adder produces a segment sum of the least significant bits and a carry out. Segment adder pairs are used for each higher order of significant sums. One segment adder produces a segment sum portion, and the other produces an incremented segment sum portion. Carry logic associated with each segment is utilized with a multiplexer to select the incremented segment sum portion or the segment sum portion. The selected segment sum portions are assembled with a most significant carry out to produce the sum.

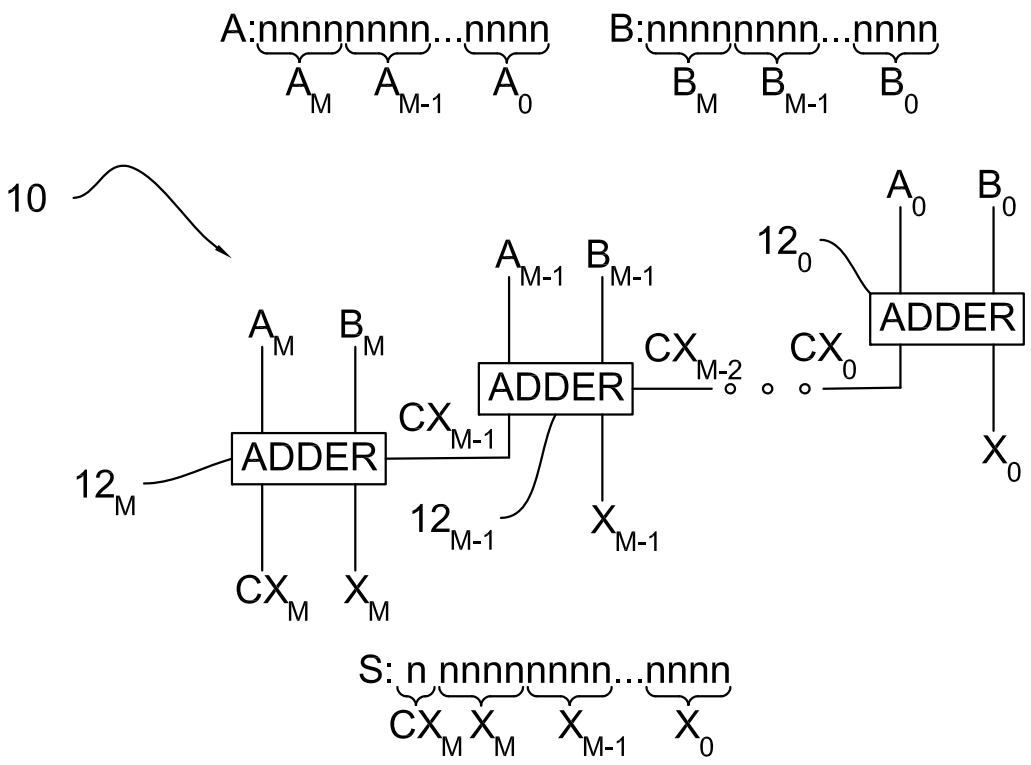

FIG. 1
(PRIOR ART)

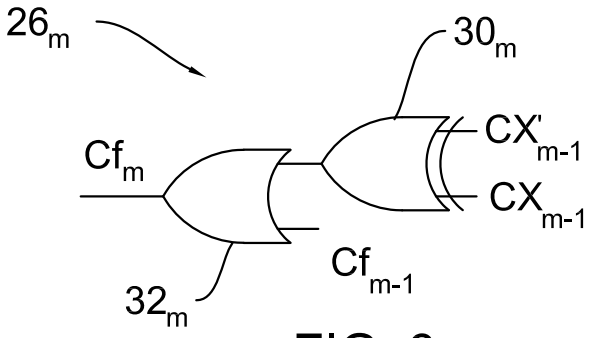

FIG. 3

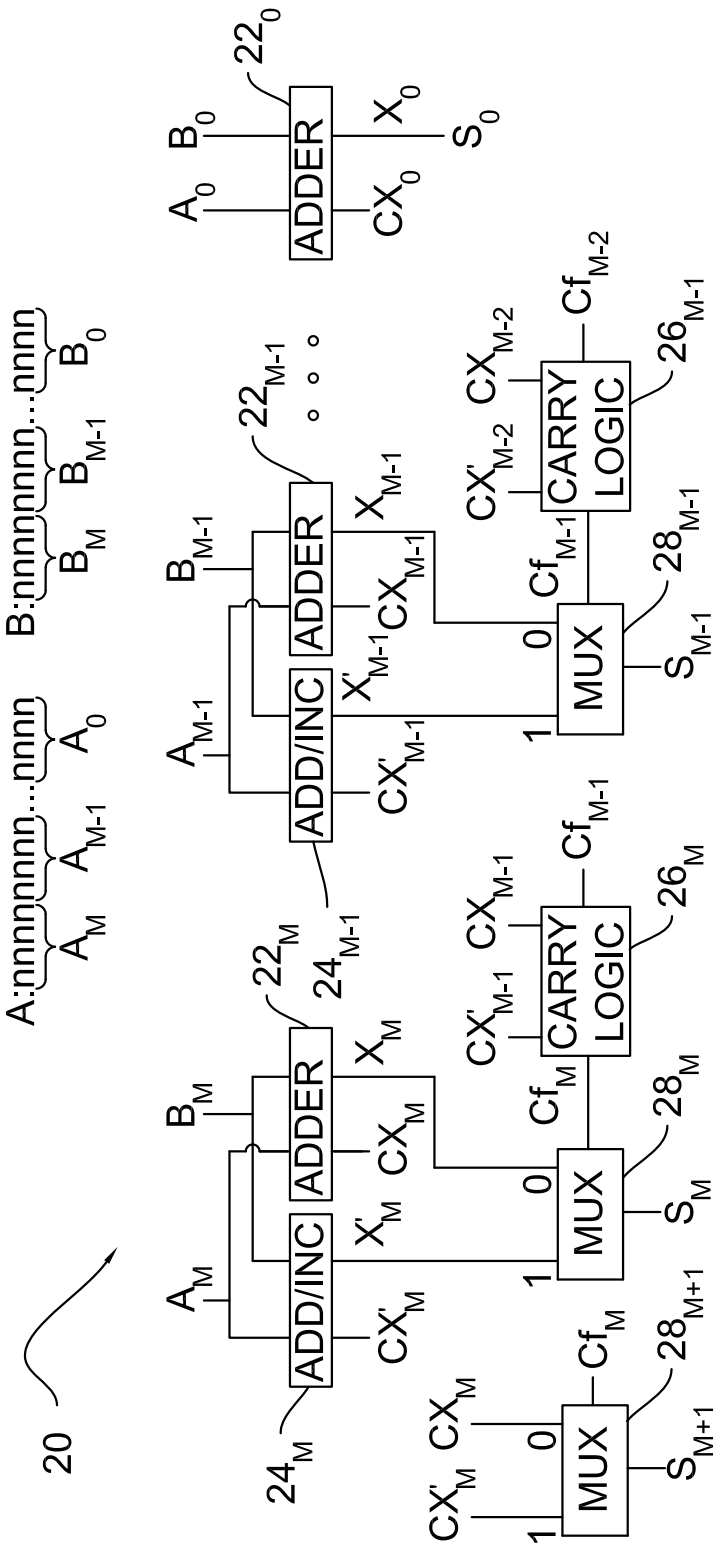
20
A:nnnnnnnn...nnnn
B:nnnnnnnn...nnnn
S: n nnnnnnnn...nnnn
ADD/INC
ADDER
MUX
CARRY LOGIC
22M
24M
26M
28M
22M-1
24M-1
26M-1
28M-1
22 0
24 0
26 0
28 0
28M+1

FIG. 2

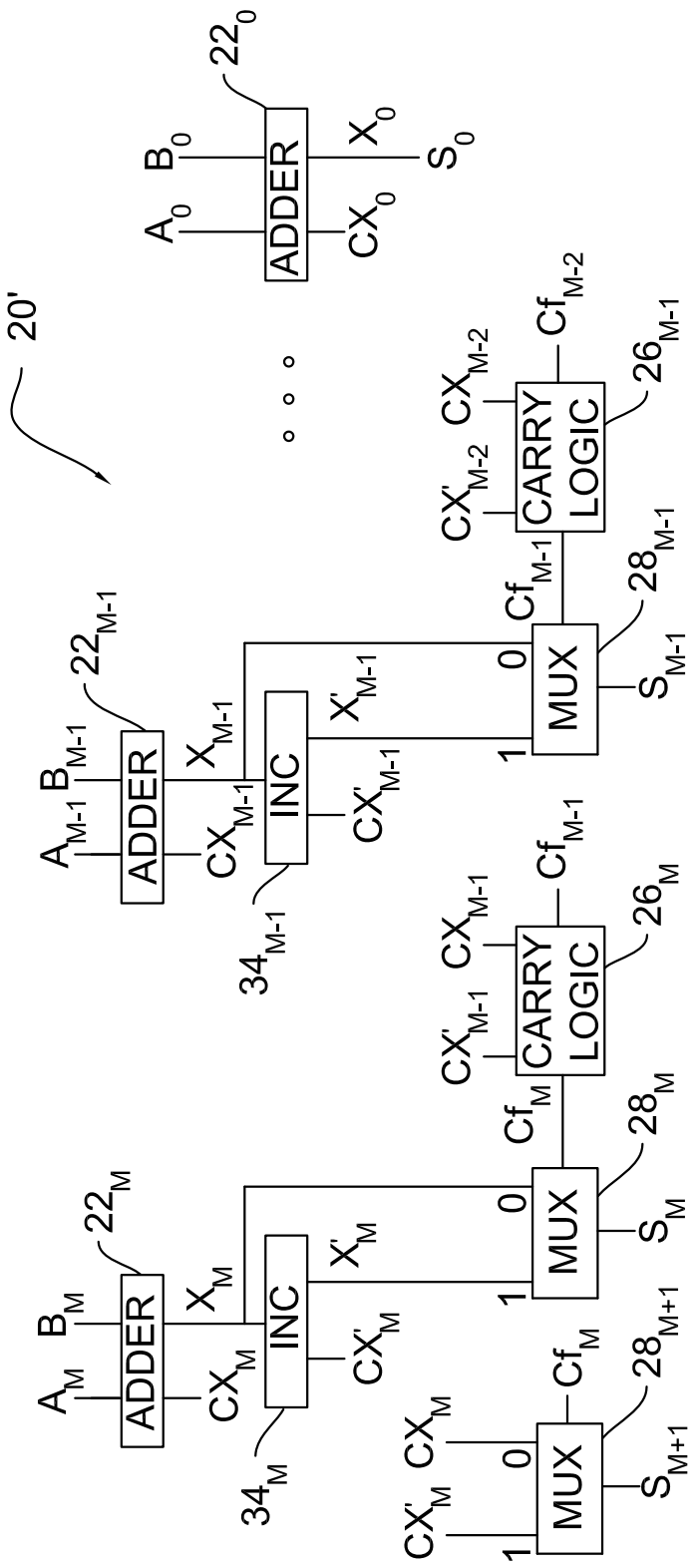
20'
ADDER
INC
MUX
CARRY LOGIC
A_0
B_0
CX_0
X_0
S_0
22_0
A_{M-1}
B_{M-1}
CX_{M-1}
X_{M-1}
X'_{M-1}
CX'_{M-1}
34_{M-1}
22_{M-1}
Cf_{M-1}
S_{M-1}
28_{M-1}
CX_{M-2}
CX'_{M-2}
Cf_{M-2}
26_{M-1}
A_M
B_M
CX_M
X_M
X'_M
CX'_M
34_M
22_M
Cf_M
S_M
28_M
CX_{M-1}
CX'_{M-1}
Cf_{M-1}
26_M
CX_M
CX'_M
Cf_M
S_{M+1}
28_{M+1}
0
1

FIG. 4