(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
22 November 2001 (22.11.2001)

PCT

(10) International Publication Number
WO 01/88694 A1

(51) International Patent Classification[7]: G06F 7/72, H03M 7/18

(21) International Application Number: PCT/US00/27221

(22) International Filing Date: 3 October 2000 (03.10.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/569,944 12 May 2000 (12.05.2000) US

(71) Applicant: THE ATHENA GROUP, INC. [US/US]; 3424 N.W. 31st Street, Gainesville, FL 32605 (US).

(72) Inventor: MELLOTT, Jonathon, D.; 1717 N.W. 39th Terrace, Gainesville, FL 32605 (US).

(74) Agents: PARKER, James, S. et al.; Saliwanchik, Lloyd & Saliwanchik, A Professional Association, Suite A-1, 2421 N.W. 41st Street, Gainesville, FL 32606-6669 (US).

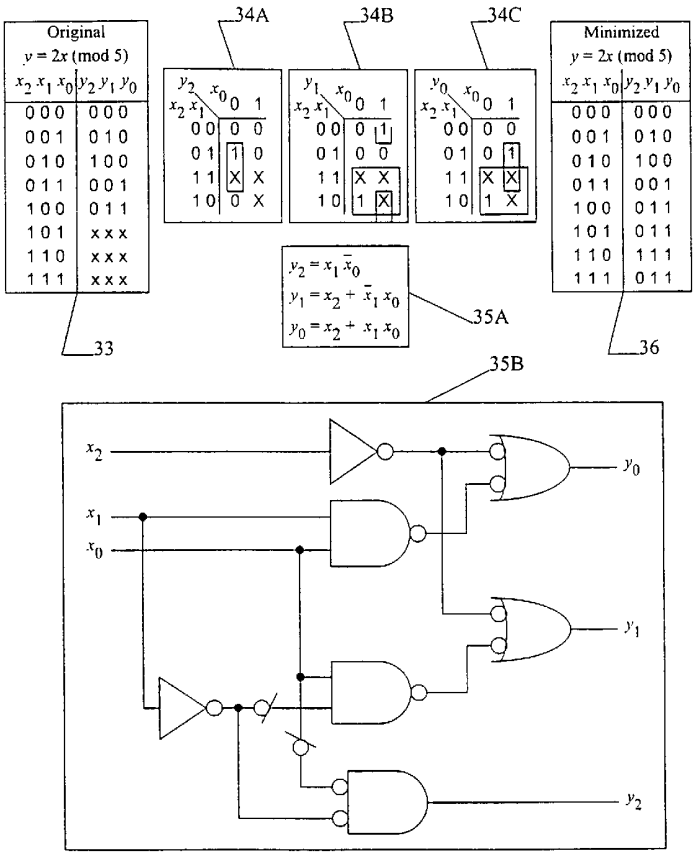
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*

[Continued on next page]

(54) Title: METHOD AND APPARATUS FOR PERFORMING COMPUTATIONS USING RESIDUE ARITHMETIC



(57) Abstract: The subject invention pertains to a method and apparatus for performing computations using residue arithmetic. The subject method and apparatus can utilize logic gates for performing calculations such as multiplication by a constant, computing a number theoretic logarithm of a residue for a given base α_i and modulus p, and computing the product of two residues, modulo p. The use of logic gates can offer advantages when compared with the use of ROMs for table look-up functions in integrated RNS digital signal processor implementations.

WO 01/88694 A1

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

DESCRIPTION

METHOD AND APPARATUS FOR PERFORMING COMPUTATIONS USING RESIDUE ARITHMETIC

The subject invention was made with government support under a research project supported by the National Institutes Standards and Technology Cooperative Agreement No. F0NANB7H3021. The government may have certain rights in this invention.

Background of the Invention

The subject invention relates to a method and apparatus for performing computations using residue arithmetic. The subject method and apparatus can utilize the Residue Number System (RNS) to implement automatic computing machinery. The use of the RNS has been proposed in Garner, H.L., "The Residue Number System," *IRE Transactions on Electronic Computers*, vol. EL-8, No. 6, June 1959, pp. 140-147, and Taylor, F.J., "Residue Arithmetic: A Tutorial with Examples," *IEEE Computer*, vol. 17, No. 5, May 1984, pp. 50-61. The RNS is generally used to implement automatic computing machinery for digital signal processing. Digital signal processing (DSP) is dominated by the repetitive computation of sums of products. The RNS is well-suited to performing computations of this type, as demonstrated in Mellott, J.D., Lewis, M.P., Taylor, F.J., "A 2D DFT VLSI Processor and Architecture," *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, Atlanta, 1996, and Mellott, J.D., Smith, J.C., Taylor, F.J., "The Gauss Machine – A Galois-Enhanced Quadratic Residue Number System Systolic Array," *Proceedings of IEEE 11th Symposium on Computer Arithmetic*, Windsor Ontario, 1993, pp. 156-162.

In the past, it has often been impractical to implement large-scale digital signal processors using a single semiconductor device due to the limitations of the amount of logic that can be placed on such a device. Instead, large-scale digital signal processors were typically implemented using discrete logic. The RNS is well-suited to this implementation methodology since its need for small adders and table lookup functions corresponds with the common availability of discretely packaged small adders and small programmable read-only memories (PROMs). An example of this implementation methodology is the Gauss Machine, discussed in the aforementioned reference by Mellott, *et al.* As it became possible to integrate large-scale digital signal processors onto a single semiconductor device, the methodology of using small adders and memories was carried forward. An example of such a digital signal processor is given

by Smith, J.C., Taylor, F.J., "The Design of a Fault Tolerant GEQRNS Processing Element for Linear Systolic Array DSP Applications," *Proceedings of IEEE Great Lakes Symposium on VLSI*, Notre Dame, Indiana, 1994. Other examples of RNS digital signal processors can be found in U.S. Patent No. 5,117,383 (Fujita *et al.*), issued May 26, 1992; U.S. Patent No. 5,008,668 (Takayama, *et al.*), issued April 16, 1991, U.S. Patent No. 4,949,294 (Wambergue), issued August 14, 1990; and U.S. Patent No. 4,281,391 (Huang), issued July 28, 1981.

The aforementioned examples disclose the use of ROMs for implementation of table lookup functions. For the small table lookup functions typically found in RNS digital signal processor implementations, ROMs are attractive because they are easy to program and have known speed, area, and power characteristics. In contrast, the manual design of a collection of logic gates to realize a table lookup function can be a daunting task, and the speed, area, and power characteristics are generally not fully known until the time that the circuit is designed. Another feature associated with prior use of ROMs in integrated, as opposed to discrete, RNS digital signal processor implementations is that the ROMs offer favorable die area compared to other possible means of implementing small table lookups.

Prior techniques for performing computations using RNS suffer from one or more disadvantages related to the use of memories, usually ROMs, to implement table lookup functions. Some of these disadvantages include: memories with the required properties for use in RNS computations are not available in sufficient quantity in all ASIC implementation technologies; memories often contain analog circuitry that uses significant power even if there is no switching activity in the circuit; the analog circuitry found in most memory devices does not scale well into deep sub-micron semiconductor fabrication technologies; memories, since they are dependent upon analog circuits (e.g., differential amplifiers), can be more difficult to test than digital logic circuits, can require separate tests and test mechanisms than digital logic circuits, and are not generally compatible with leakage current (I_{DDQ}) test methodologies; there is little or no flexibility to optimize a memory with respect to one or more of speed, power, and area; memories can be difficult to pipeline, and in many implementation technologies there is no realistic option to pipeline memory; the size of the memory is typically fixed by the number of inputs and outputs, and is essentially independent of the contents of the memory; for reliability reasons, wires unrelated to a memory are not usually allowed to pass over a memory on a semiconductor device, such that the presence of many small memories on a semiconductor device, such as would be used in an apparatus to perform computations using the RNS, can impair the ability to connect various functions, both memory and non-memory, on the device.

Brief Summary of the Invention

The subject invention pertains to a method and apparatus for performing computations using the Residue Number System (RNS). In a specific embodiment, a plurality of logic gates can be utilized to implement computations using the RNS. In light of recent semiconductor device scaling and design methodology changes, the subject invention can offer advantages over the use of ROMs for small table lookup functions in integrated RNS digital signal processor implementations. Some of these advantages include: logic gates can scale down in size or power better than the analog portions of the ROM circuitry, for example the differential sense amplifier; for integrated RNS implementations, small table lookup functions implemented with gates require less die area than the same functions implemented with ROMs; in general, logic gates are compatible with quiescent current test methodologies, while memory devices are not compatible with quiescent, or leakage, current test methodologies (also known as I_{DDQ} testing); logic gates are generally scan testable whereas memory devices can require special test structures and are typically not directly compatible with scan test methodologies; and signal wires may be routed over logic gates, whereas most design methodologies do not allow signal wires to be routed over on-chip memories such that the presence of many small memories in a design may congest wire routing, potentially leading to higher design costs, slower circuit operation, greater power consumption, greater silicon die area consumption, and, thus, greater manufacturing cost.

The present invention can provide one or more of the following advantages: provide a means of implementing residue arithmetic computational circuitry with a reduced use of, or entirely without the use of, memories for table lookup operations so that the circuitry can be easily implemented using a variety of technologies, including, but not limited to, custom digital logic, standard cell logic, cell-based arrays of logic, gate arrays, field programmable gate arrays, and programmable logic devices; provide a means of implementing residue arithmetic computational circuitry that does not consume significant power in the absence of switching activity in the circuit; to provide a means of implementing residue arithmetic computational circuitry that scales directly into deep sub-micron semiconductor fabrication technologies; to provide a means of implementing residue arithmetic computational circuitry that is compatible with standard logic test methodologies (e.g., scan, I_{DDQ}); provide a means of optimizing the mathematical functions in the residue arithmetic computational circuitry for one or more of speed, power, and area; provide a means of implementing the mathematical functions in residue arithmetic computational circuitry that allows pipelining and is fully compatible with Electronic Design Automation (EDA) methodologies for automatic pipelining; provide a means of implementing the mathematical functions in residue arithmetic computational circuitry that takes

advantage of the structure of the values resulting from a mathematical function to produce an implementation that is smaller and faster than is possible with any memory-based implementation; and provide a means of implementing mathematical functions in the residue arithmetic computational circuitry that does not unduly interfere with the routing of wires on the semiconductor device.

Brief Description of the Drawings

Figure 1 shows a block diagram of an apparatus, for digital signal processing, that uses residue arithmetic to operate on real operands and produces real results.

Figure 2 shows a block diagram of an apparatus, for digital signal processing, that uses residue arithmetic to operate on complex operands and produces complex results.

Figure 3 shows a block diagram of a modular product table lookup for a constant multiplier.

Figure 4 shows a table for the product of two and a modulo 5 variable, modulo 5, the minimization of the equations for the table using Karnaugh maps, a plurality of logic gates implementing the reduced equations, and the resulting table.

Figure 5 shows a number theoretic logarithm lookup table.

Figure 6 shows a block diagram of a multiplier that computes products using the number theoretic logarithms of the operands.

Figure 7 shows a structure to compute the residue of an N bit unsigned or two's complement number.

Figure 8 shows a block diagram of a multi-operand modular adder tree.

Figure 9 shows a structure to convert a value from RNS representation to binary representation using the Chinese Remainder Theorem.

Figure 10 shows a structure to convert a value from RNS representation to binary representation using the L-CRT algorithm.

Figure 11 shows a structure to convert a complex RNS value to a QRNS value.

Figure 12 shows a structure to convert a QRNS value to a complex RNS value.

Detailed Description of Invention

Enabling Mathematical Theory

The following subsections present the mathematics which are relevant to the operation of the invention. While the mathematics are well-known, the theory is presented here so as to provide a consistent framework of notation and symbols.

The Chinese Remainder Theorem

Let $S=\{p_0, p_1, p_2, \ldots, p_{L-1}\}$, where $\gcd(p_i, p_j)=1$ for all $i,j \in \{0, 1, 2, \ldots, L-1\}$ and $i \neq j$, wherein gcd stands for greatest common denominator. Let $M = \prod_{i=0}^{L-1} p_i$, and let $X \in \mathbf{Z}/M\mathbf{Z}$, where $\mathbf{Z}$ denotes the ring of integers. By the Chinese Remainder Theorem, there exists an isomorphism

$$\varphi: \mathbf{Z}/M\mathbf{Z} \leftrightarrow \mathbf{Z}/p_0\mathbf{Z} \times \mathbf{Z}/p_1\mathbf{Z} \times \mathbf{Z}/p_2\mathbf{Z} \times \ldots \times \mathbf{Z}/p_{L-1}\mathbf{Z}.$$

The mapping φ is given by

$$\varphi(X) \rightarrow (x_0, x_1, x_2, \ldots, x_{L-1})$$

where $(x_0, x_1, x_2, \ldots, x_{L-1}) \in \mathbf{Z}/p_0\mathbf{Z} \times \mathbf{Z}/p_1\mathbf{Z} \times \mathbf{Z}/p_2\mathbf{Z} \times \ldots \times \mathbf{Z}/p_{L-1}\mathbf{Z}$, and $x_i \equiv X \pmod{p_i}$ for all $i \in \{0,1,2,\ldots L-1\}$. The inverse mapping is given by

$$\varphi^{-1}[(x_0, x_1, x_2, \ldots, x_{L-1})] \rightarrow X$$

where

$$X \equiv \left(\sum_{i=0}^{L-1} m_i \left\langle m_i^{-1} x_i \right\rangle_{p_i} \right) \pmod{M},$$

$m_i=M/p_i$, $m_i m_i^{-1} \equiv 1 \pmod{p_i}$, and $\langle x \rangle_p$ denotes the value in the set $\{0,1,2,\ldots,p-1\}$ that is congruent to x modulo p.

Number Theoretic Logarithms

If p_i is prime then there exists a generator $\alpha_i \in \mathbf{Z}/p_i\mathbf{Z}$ such that

$$\{\alpha_i^k \mid k=0, 1, 2, \ldots, p_i-2\} = \{1, 2, 3, \ldots, p_i-1\}$$

in the ring $\mathbf{Z}/p_i\mathbf{Z}$. If $x_i \in (\mathbf{Z}/p_i\mathbf{Z})\backslash\{0\}$, then there exists a unique $l_{x_i} \in \mathbf{Z}/(p_i-1)\mathbf{Z}$ such that

$$x_i \equiv \alpha_i^{l_{x_i}} \pmod{p_i}$$

The value l_{x_i} is said to be the number theoretic logarithm of x_i to the base α_i modulo p_i.

The number theoretic logarithm may be exploited to compute products in the ring $\mathbf{Z}/p_i\mathbf{Z}$. If $x_i, y_i \in (\mathbf{Z}/p_i\mathbf{Z})\backslash\{0\}$, then there exist unique l_{x_i}, $l_{y_i} \in \mathbf{Z}/(p_i-1)\mathbf{Z}$ such that

$$x_i y_i \equiv \left(\alpha_i^{l_{x_i}}\right)\left(\alpha_i^{l_{y_i}}\right) \pmod{p_i}$$

$$x_i y_i \equiv \alpha_i^{\left\langle l_{x_i} + l_{y_i} \right\rangle_{p_i-1}} \pmod{p_i}$$

$$x_i y_i \equiv f_{\alpha_i}\left(\left\langle l_{x_i} + l_{y_i} \right\rangle_{p_i-1}\right) \pmod{p_i}$$

If either or both of x_i, y_i is zero, then the product $x_i y_i$ is zero.

Complex Arithmetic

Let $\mathbf{Z}[j]/(j^2+1)$ denote the ring of Gaussian integers under the usual operations of addition and multiplication, numbers of the form $a+jb$ where $a,b \in \mathbf{Z}$, and $j^2=-1$. Then $(\mathbf{Z}[j]/(j^2+1))/p_i\mathbf{Z}$ denotes the ring of Gaussian integers modulo p_i, and if $a+jb \in \mathbf{Z}[j]/(j^2+1)$ then the mapping $\varphi: \mathbf{Z}[j]/(j^2+1) \rightarrow (\mathbf{Z}[j]/(j^2+1))/p_i\mathbf{Z}$ is given by

$$\varphi((a+jb)) \rightarrow a_i+jb_i,$$

where $a_i \equiv a \pmod{p_i}$ and $b_i \equiv b \pmod{p_i}$. The set $(\mathbf{Z}[j]/(j^2+1))/p_i\mathbf{Z}$ is a ring under the usual complex arithmetic operations of multiplication and addition. That is, if $(a_i+jb_i),(c_i+jd_i) \in (\mathbf{Z}[j]/(j^2+1))/p_i\mathbf{Z}$, then

$$(a_i+jb_i) + (c_i+jd_i) = ((a_i+c_i) + j(b_i+d_i))$$
$$(a_i+jb_i) \times (c_i+jd_i) = ((a_ic_i-b_id_i) + j(a_id_i+b_ic_i)).$$

Suppose p_i is a prime and $p_i=4k_i+1$, where $k_i \in \mathbf{Z}$. Then there exists an isomorphism between the Gaussian integers modulo p_i under the usual complex arithmetic operations as shown above, and the Gaussian integers modulo p_i under component-wise addition and multiplication, $\Psi:(\mathbf{Z}[j]/(j^2+1))/p_i\mathbf{Z} \leftrightarrow (\mathbf{Z}[j]/(j^2+1))/p_i\mathbf{Z}$, with the mapping

$$\Psi((a_i+jb_i)) \rightarrow (z_i, z_i^*)$$

where $z_i = a_i + \hat{j}b_i$, $z_i^* = a_i - \hat{j}b_i$ and $\hat{j}^2 \equiv -1 \pmod{p_i}$.

The inverse mapping is given by

$$\Psi^{-1}((z_i, z_i^*)) \rightarrow (a_i+jb_i)$$

where $a_i = 2^{-1}\left(z_i + z_i^*\right), b_i = \hat{j}2^{-1}(z_i - z_i^*)$, and $2 \cdot 2^{-1} \equiv 1 \pmod{p_i}$.

The Chinese Remainder Theorem (CRT) may be exploited to perform addition, subtraction, and multiplication of values in the ring of integers modulo M, $\mathbf{Z}/M\mathbf{Z}$, by breaking the computation into L independent computations in $\mathbf{Z}/p_i\mathbf{Z}$, for $i \in \{0,1,2,\ldots,L-1\}$. If each $p_i \in S$ is prime then number theoretic logarithms may be exploited to reduce the complexity of multiplication. Furthermore, if each $p_i \in S$ is prime and $p_i=4k_i+1$ where $k_i \in \mathbf{Z}$, then it is possible to exploit the

isomorphism Ψ to reduce the number of arithmetic operations required to implement complex multiplication from four real multiplies and two real additions to two real multiplies.

Figure 1 shows a specific embodiment of the subject invention which can be used to perform sums of products on real binary unsigned or two's complement, one's complement, sign-magnitude, or other fixed-radix or floating-radix operands using residue arithmetic. The system shown in Figure 1 can have a circuit **1** to convert data from a conventional representation such as, but not limited to, one's complement, sign-magnitude, unsigned binary or two's complement to a set of L residues. If multiplication is needed, the residues of the input operands can be multiplied by one or more coefficients by a circuit **3**. Circuit **3** can be removed if only addition is to be achieved. These coefficients can be fixed and/or programmed coefficients. The modular products produced by circuit **3** can then be added by a circuit **4** to produce modular sums of products. The modular sums of products can then be converted to a conventional representation by a circuit **6**. The specific arrangement of the modular products and sums are dependent upon the algorithm design and can be optimized as desired.

Referring to Figure 1, an embodiment which can process real operands is shown. Data operands, for example, in a conventional format such as two's complement, can be input to circuit **1** (the details of which are summarized in the discussion of Figure 7) to convert the operands into RNS form. If the algorithm requires multiplication, the products can be computed next by a circuit **3**, which can comprise one or more elements from Figure 3, and/or Figure 5 and Figure 6. Any sums, if required, can be computed next by a circuit **4**, which comprises two operand modular adders and, optionally, one or more modular adder trees from Figure 8. The specific arrangement of the arithmetic elements and intermediate storage elements, including, but not limited to, registers, latches, and random access memory (RAM)s, can be varied depending on the situation. For example, the arithmetic elements and intermediate storage elements may be arranged to implement functions including, but not limited to, convolution, correlation, finite impulse response filters, fast Fourier transforms, discrete cosine transforms, wavelet transforms, filter banks, cascaded integrator comb filters, digital receivers, and digital transmitters. The results of the computation can then be converted to a conventional format such as two's complement by a circuit **6**, which can comprise, for example, a CRT conversion as shown in from Figure 9 or an L-CRT conversion as shown in Figure 10.

Figure 2 shows another specific embodiment of the subject invention which can be used to perform sums of products on complex binary unsigned or two's complement operands using residue arithmetic. The system shown in Figure 2 can have a circuit **1** to convert data from a conventional representation such as, but not limited to, one's complement, sign-magnitude,

unsigned binary, or two's complement to a set of L residues for each of the real and imaginary components of each operand. The complex residues can then be converted to quadratic residue representation by a circuit **2**. The quadratic residues of the input operands can be multiplied by one or more coefficients by a circuit **3**. These coefficients can be fixed and/or programmed coefficients. The modular products produced by circuit **3** can then be added by a circuit **4** to produce modular sums of products. The quadratic modular sums of products can then be converted to complex residues by a circuit **5**. The complex sums of products can then be converted to a conventional representation, such as complex unsigned binary or two's complement, by a circuit **6**. The specific arrangement of the modular products and sums are dependent upon the algorithm design and can be optimized as desired. In some instances, an algorithm can be designed to accept real inputs as operands and produce complex results, or to accept complex inputs and produce real results. In such case, the circuit **2** and/or the circuit **5** may be removed, as desired.

Referring to the embodiment shown in Figure 2, the subject invention can process complex operands. Data operands, for example, in a conventional form such as two's complement, can be input to circuit **1** (the details of which are summarized in the discussion of Figure 7) to convert the operands into CRNS form. The CRNS operands can be passed to a circuit **2** to convert the operands to QRNS format. An example of such a circuit **2** is shown in Figure 11. If the algorithm requires multiplication, the products can be computed next by a circuit **3**, which can comprise one or more elements from Figure 3, and/or Figure 5 and Figure 6. Any sums, if required, can be computed next by a circuit **4**, which can comprise two operand modular adders and, optionally, one or more modular adder trees as shown in Figure 8. The specific arrangement of the arithmetic elements and intermediate storage elements, including, but not limited to, registers, latches, and RAMs, can be varied depending on the situation. For example, the arithmetic elements and intermediate storage elements may be arranged to implement functions including, but not limited to, convolution, correlation, finite impulse response filters, fast Fourier transforms, discrete cosine transforms, wavelet transforms, filter banks, cascaded integrator comb filters, digital receivers, and digital transmitters. The QRNS results of the computation can then be converted back to CRNS representation by a circuit **5**, for example, as shown in Figure 12. The CRNS results can then be converted to a conventional format such as two's complement by a circuit **6**, which can comprise, for example, a CRT conversion as shown in Figure 9 or an L-CRT conversion as shown in Figure 10.

An embodiment for computation of modular products of a constant and a modular data operand is shown in Figure 3. The product can be generated by a circuit **7** that accepts an N_i bit

operand and produces the product of the operand and a constant c_i modulo p_i , producing an N_i bit result. Figure 3 shows a block diagram of an embodiment of a circuit **7** to produce the modular product of an operand and a constant where such constant is fixed by the design of the circuit. Circuit **7** can utilize a plurality of logic gates selected by first computing the value of a multiply by a constant function for each possible modular data operand, then extracting the logical equations representing the computed values of the multiply by a constant function. The logical equations can then be mapped to a plurality of logic gates. If desired, prior to mapping to a plurality of logic gates, the logical equations can be minimized by, for example, using well-known logic minimization techniques which take advantage of the fact that for any invalid input the value of the output is allowed to be any value. After the logic equations is reduced to a minimized logical function, it can be mapped to an implementation utilizing a plurality of logic gates. Mapping to a plurality of logic gates can be performed, for example, manually or using software such as DESIGN COMPILER, available from Synopsys, Inc. of Mountain View, California.

Figure 4 shows an example of a product lookup table for the constant multiplier 2 and a modulo 5 value x (bits x_2, x_1, and x_0, ordered from most significant to least significant). A truth table **33** shows all possible inputs to the table as well as the output of the table y (bits y_2, y_1, and y_0, ordered from most significant to least significant). The "x" entries in the table indicate that the value of the output can be anything. The table is reduced to a minimized set of logical equations **35A** using Karnaugh maps **34A**, **34B**, and **34C**. One example of a plurality of logic gate **35B** which can be used to implement the logical equations **35A** are shown in Figure 4. For larger moduli, and thus larger tables, minimization of the logical equations for the table by manual means can be impractical, so a computer program can be employed to minimize the logical equations. The results of the minimized logical equations, given all possible inputs are shown in a truth table **36**.

An embodiment for computation of number theoretic logarithms for a given base α_i and modulus p_i is shown in Figure 5. To multiply two operands in the RNS, the logarithms of the operands can be computed by a circuit **8** as shown in Figure 5. The logarithm can be generated by a circuit **8** that accepts an N_i bit operand and produces the N_i bit logarithm of the operand. If the input operand is zero then the output of the circuit **8** is a symbol that is not a valid number theoretic logarithm.

Figure 5 shows a block diagram of an embodiment of a circuit **8** to produce the number theoretic logarithm of a residue, or a special zero symbol if the input operand is zero. For a given base α_i and modulus p_i, the number theoretic logarithm of a value in the set $\{1,2,3,\ldots, p_i\text{-}1\}$ will

lie in the set $\{0,1,2,\ldots, p_i\text{-}2\}$. In the preferred embodiment of circuit **8**, the special symbol that results when the input is zero is the binary word that is all ones. The table lookup function **8** can be reduced to a circuit using the procedure discussed in the description of Figure 3.

Figure 6 shows a block diagram of an embodiment of a circuit to compute the product of two residues, modulo p_i, using the sum of the number theoretic logarithms of the operands. The circuit of Figure 6 can accept two operands, the number theoretic logarithms of the residues to be multiplied or the symbol for zero that is produced by a circuit **8** when presented with an input of zero. The operands can be presented to a modular adder circuit **9**, which produces the sum of the operands modulo p_i-1, the output of which is valid only if neither of the operands is the zero symbol. The operands can also be presented to a circuit **10** to detect the symbol for zero. The sum of the logarithms produced by the circuit **9** can then be an input to a number theoretic exponentiation table lookup circuit **11**. The table lookup function **11** can be reduced to a circuit using the procedure discussed in the description of Figure 3. The output of the zero detection circuits **10** can then be logically ORed by, for example, an OR gate **12**. If the output of the OR gate **12** indicates that either of the input operands were the zero symbol, then the output of a multiplexer **13** can be set to zero, otherwise the output of the exponentiation circuit **11** can then be passed to the output of the multiplexer. In most implementations of the systems shown in Figure 1 and Figure 2, the number theoretic exponentiation table lookup circuit **11** will be the most common table lookup in the system. In general, for a specific $(\mathbf{Z}/p_i\mathbf{Z})\backslash 0$, there are many possible generators. For any modulus p_i, there may be as much as a twenty percent variation in the size of the exponentiation circuit over the entire set of possible generators. Accordingly, generators can be selected based on one or more factors. In a preferred embodiment of the subject invention, for each modulus p_i, an optimum generator α_i can be selected based on one or more criterion such as size, speed, power, or some other cost function. This optimum generation can then be used to create the number theoretic exponentiation circuit **11** and/or the number theoretic logarithm circuit **8**.

In the embodiments shown in Figure 6, the logarithms of the operands are checked by a zero detection circuit **10**; if either of the logarithm inputs are the special symbol for zero, as determined by a logical OR gate **12**, then the product output is set to zero by a multiplexer **13.** Otherwise, the logarithms can be added modulo p_i-1 by a modular adder circuit **9**, the output of which can be input to an exponentiation circuit **11.** The output of the exponentiation circuit **11**, can then be passed to the multiplexer **13**, and if neither of the operands were the special zero symbol, as determined by the output of the OR gate **12**, then the output of the multiplexer **13** can be set to the output of the exponentiation circuit **11.**

Figure 7 shows a block diagram of an embodiment for reduction of an N bit binary operand to its residue modulo p_i. This binary operand can be, for example, unsigned or two's complement. A zero extension **14** can take the least significant N_i-1 bits of the input operand and produce its N_i bit residue modulo p_i. The N bit conventional operand can be partitioned into q_i+1 groups of bits. The N_i -1 least significant bits are already reduced modulo p_i but are zero extended to N_i bits by a zero extension **14.** The remaining N-N_i+1 bits of the input operand can be partitioned into q_i groups of bits which are inputs to q_i table lookups **15A**, **15B**, and **15C**. Each partition of bits Q_{ij} for $j \in \{0,1,2,..., q_i -1\}$ can be input to a table lookup circuit **15A**, **15B**, and **15C**. Table lookups **15A**, **15B**, and **15C** can then produce the residues of the weighted inputs. The mathematical functions performed by table lookups **15A**, **15B**, and **15C**, can be reduced to circuits using the procedure discussed in the description of Figure 3. The q_i+1 residues can be added by a q_i+1 operand modular adder **16** to produce the residue of the original input operand modulo p_i. For example, the output of the splitter **14** and the table lookup circuits **15A**, **15B**, **15C** can be added by a $q_i + 1$ operand modular adder circuit **16**, the sum of which is the original N bit operand reduced modulo p_i.

Figure 8 shows a block diagram of an embodiment of a circuit to compute the sum of L>2 operands (L residues) modulo p_i. The L operands can be added by a binary adder tree **17** to produce the full sum of the L operands. For example, binary adder **17** can produce the $N_i + \lceil \log_2 L \rceil$ bit unsigned sum of the input operands. The N_i-1 least significant bits can be split from the full sum by a splitter **20** and zero extended to N_i bits by a zero extension **21.** As shown, the output of the binary adder **17** can be split by a bus splitter **20**, and the most significant $\lceil \log_2 L \rceil + 1$ bits passed to a modulo p_i table lookup circuit **18**, while the least significant N_i-1 bits are passed to a zero extension **21**. The table lookup function **18** can be reduced to a circuit using the procedure discussed with respect to the embodiment of Figure 3. The outputs of the modulo p_i table lookup circuit **18** and the zero extension **21** are combined by a modulo p_i adder **19**, producing the sum of the L operands modulo p_i.

An embodiment of the subject invention can be utilized for conversion of an L operand RNS value to a conventional value using the Chinese remainder theorem. Figure 9 shows a block diagram of an embodiment of a circuit to convert the L residue representation of a value to its unsigned binary representation by the Chinese remainder theorem. The L residues, $\{x_0, x_1, x_2, \ldots, x_{L-1}\}$ can be input to L separate CRT function table lookup circuits **22A**, **22B**, **22C**, and **22D**, producing L results. The table lookup functions **22A**, **22B**, **22C**, and **22D**, can be reduced to circuits using the procedure discussed in the description of Figure 3. These results modular adder circuit **23** to produce, for example, the unsigned binary representation of the input value.

An embodiment of the subject invention can be utilized for conversion of an L operand RNS value to a conventional value using L-CRT. Figure 10 shows a block diagram of an embodiment of a circuit to convert the L residue representation of a value to a scaled unsigned binary or two's complement representation using the L-CRT conversion. The L residues, $\{x_0, x_1, x_2, \ldots, x_{L-1}\}$ can be input to L separate L-CRT function table lookup circuits **24A**, **24B**, **24C**, and **24D**, producing L scaled results. The table lookup functions **24A**, **24B**, **24C**, and **24D**, can be reduced to circuits using the procedure discussed in the description of Figure 3. These results produced by the table lookup circuits **24A**, **24B**, **24C**, and **24D** can then be added by a binary adder circuit **25** to produce, for example, the scaled unsigned binary or two's complement representation of the input value.

An embodiment of the subject invention can be utilized for conversion of CRNS operands to QRNS form. Figure 11 shows a block diagram of an embodiment of a circuit to convert a complex residue number system (CRNS) value to a quadratic residue number system (QRNS) value. The imaginary component of the CRNS input b_i, can be input to a constant multiplication by $\hat{j}$ circuit **26.** For example, the imaginary residue operand, b_i, can be input to a circuit **26** that looks up the product of the operand with $\hat{j}$. The table lookup function **26** can be reduced to a circuit using the procedure discussed with respect to the embodiment of Figure 3. The output of the table lookup circuit **26** and the real portion of the CRNS input, a_i, can be added modulo p_i, by a modular adder circuit **27** to produce the QRNS component z_i. The output of the table lookup circuit **26** can then be subtracted, modulo p_i, from the real portion of the CRNS input by a modular subtractor circuit **28** to produce the QRNS component z_i^*.

An embodiment of the subject invention can be utilized for conversion of QRNS operands to CRNS form. Figure 12 shows a block diagram of an embodiment of a circuit to convert a quadratic residue number system value to a complex residue number system value. The QRNS components z_i and z_i^* can be added modulo p_i by a modular adder circuit **29.** The QRNS component z_i^* can be subtracted, modulo $p_{,i}$ from the component z by a modular subtractor circuit **30.** The output of the modular adder circuit **29** can be input to a constant multiplication by 2^{-1} table lookup circuit **31**, the output of which is the real component of the CRNS representation of the data. The output of the modular adder **29** can be the input to a circuit **31** that looks up the product of the sum with 2^{-1}. The output of the modular subtractor circuit **30** can be input to a constant multiplication by $\hat{j}^{-1}2^{-1}$ table lookup circuit **32**, the output of which is the imaginary component of the CRNS representation of the data. The output of the modular subtractor **30** can be the input to circuit **32** that looks up the product of the sum with $\hat{j}^{-1}2^{-1}$.

The product table lookup functions **31** and **32** can be reduced to circuits using the procedure discussed with respect to the embodiment of Figure 3.

The use of logic gates to implement various table lookup operations in accordance with this invention can provide manifold advantages over the previous method of using memory devices. The use of logic gates can allow RNS computational circuitry to be efficiently implemented in a variety of technologies, some of which would not have been previously amenable to the use of RNS techniques. Additionally, the use of logic gates rather than memories for RNS computational circuitry can provide one or more of the following benefits: logic gates implemented in complimentary metal oxide semiconductor (CMOS) static logic can consume very low power in the absence of switching activity in the circuit; logic gates can scale directly into deep sub-micron semiconductor fabrication technologies; logic gates can be compatible with standard logic test methodologies; groups of logic gates can be optimized for speed, power, and area; groups of logic gates can be easily pipelined through manual or automatic means; and logic gates can reduce interference with the routing of wires on a semiconductor device as compared with memories.

Unlike memories, which have a fixed area and speed for any given table lookup function of a given input and output size, groups of logic gates can be minimized for the specific table lookup function to be implemented. In many cases, the logic function to be minimized can have some underlying structure that is not obvious from inspection of the table. This structure can lead to significant area and speed advantages for groups of logic gates over memories. For example, a table lookup for the product of an eight bit input modulo 241, and 2^{-1}, modulo 241, produced in a read only memory (ROM) in a 0.2 micron standard cell application specific integrated circuit (ASIC) process requires the equivalent area of 2,250 gates, and at 100 MHZ and has a power dissipation of 3.6 mW, while the same table produced as gates requires only the area of 36 gates, and at the same speed has a power dissipation of 0.23 mW. Another table of the same size, an exponentiation table modulo 241, requires only an area of 675 gates, and at the same speed has a power dissipation of 1.3 mW.

These results were obtained using the process previously described with respect to the embodiment of Figure 3. The aforementioned ROM has a minimum clock period of 3.0 ns, while the aforementioned product lookup implemented as gates has a maximum delay from input to output of 1.0 ns, and the exponentiation lookup implemented as gates has a maximum delay of 3.0 ns. In the case of the exponentiation lookup, a delay of 1.2 ns can be achieved, although the area of the function is increased to 957 gates. This example is a compelling demonstration of the subject invention's ability to allow the optimization of the balance between speed, area, and

power, by implementing RNS table lookups using logic gates rather than memories such as ROMs. For a given implementation technology, ROMs have the highest storage density of all the types of memory. For example, a static RAM implemented in the same technology as the aforementioned ROM, and with the same size and speed characteristics, requires the equivalent area of 3,660 gates. This example also demonstrates that by using logic gates to implement table lookup functions, area and speed may be traded to best suit the needs of a particular design.

It should be understood that the examples and embodiments described herein are for illustrative purposes only and that various modifications or changes in light thereof will be suggested to persons skilled in the art and are to be included within the spirit and purview of this application and the scope of the appended claims.

Claims

1. A method of performing mathematical computations using residue arithmetic, comprising one or more of the following steps:

converting data in binary code to residues;

converting residues from CRNS to QRNS;

computing modular products of residues;

computing modular sums of residues;

converting residues from QRNS to CRNS; and

converting residues to data in binary code,

wherein at least one of said steps is implemented utilizing a corresponding at least one plurality of logic gates.

2. The method according to claim 1, wherein said method comprises the steps of:

receiving input data in binary code; and

converting said input data in binary code to residues.

3. The method according to claim 2, wherein said method comprises the step of:

converting said residues from CRNS to QRNS.

4. The method according to claim 3, wherein said method comprises the step of:

computing modular products of the QRNS residues.

5. The method according to claim 4, wherein said method comprises the step of:

computing modular sums of the products of residues.

6. The method according to claim 5, wherein said method comprises the step of:

converting the modular sums from QRNS to CRNS.

7. The method according to claim 6, wherein said method comprises the step of:
converting the CRNS residues to data in binary code.

8. The method according to claim 5, wherein said method comprises the step of:
converting the QRNS residues to data in binary code.

9. The method according to claim 4, wherein said method comprises the step of:
converting the modular products from QRNS to CRNS.

10. The method according to claim 9, wherein said method comprises the step of:
converting the CRNS residues to data in binary code.

11. The method according to claim 4, wherein said method comprises the step of:
converting the modular products to data in binary code.

12. The method according to claim 3, wherein said method comprises the step of:
computing modular sums of the QRNS residues.

13. The method according to claim 12, wherein said method comprises the step of:
converting the modular sums from QRNS to CRNS.

14. The method according to claim 13, wherein said method comprises the step of:
converting the CRNS residues to data in binary code.

15. The method according to claim 12, wherein said method comprises the step of:
converting the modular sums to data in binary code.

16. The method according to claim 3, wherein said method comprises the step of: converting the QRNS residues from QRNS to CRNS.

17. The method according to claim 16, wherein said method comprises the step of: converting the CRNS residues to data in binary code.

18. The method according to claim 3, wherein said method comprises the step of: converting the QRNS residues to data in binary code.

19. The method according to claim 2, wherein said method comprises the step of: computing modular products of the residues.

20. The method according to claim 19, wherein said method comprises the step of: computing modular sums of the modular products of residues.

21. The method according to claim 20, wherein said method comprises the step of: converting the modular sums from QRNS to CRNS.

22. The method according to claim 21, wherein said method comprises the step of: converting the CRNS residues to data in binary code.

23. The method according to claim 20, wherein said method comprises the step of: converting the modular sums to data in binary code.

24. The method according to claim 19, wherein said method comprises the step of: converting the modular products from QRNS to CRNS.

25. The method according to claim 24, wherein said method comprises the step of:

converting the CRNS residues to data in binary code.

26. The method according to claim 19, wherein said method comprises the step of:

converting the modular products to data in binary code.

27. The method according to claim 2, wherein said method comprises the step of:

computing modular sums of the residues.

28. The method according to claim 27, wherein said method comprises the step of:

converting the modular sums from QRNS to CRNS.

29. The method according to claim 28, wherein said method comprises the step of:

converting the CRNS residues to data in binary code.

30. The method according to claim 27, wherein said method comprises the step of:

converting the modular sums to data in binary code.

31. The method according to claim 2, wherein said method comprises the step of:

converting the residues from QRNS to CRNS.

32. The method according to claim 31, wherein said method comprises the step of:

converting the CRNS residues to data in binary code.

33. The method according to claim 2, wherein said method comprises the step of:

converting the residues to data in binary code.

34. The method according to claim 1, wherein said method comprises the steps of:
receiving input data in residue format; and
converting the residues from CRNS to QRNS.

35. The method according to claim 34, wherein said method comprises the step of:
computing modular products of the QRNS residues.

36. The method according to claim 35, wherein said method comprises the step of:
computing modular sums of the products of residues.

37. The method according to claim 36, wherein said method comprises the step of:
converting the modular sums from QRNS to CRNS.

38. The method according to claim 37, wherein said method comprises the step of:
converting the CRNS residues to data in binary code.

39. The method according to claim 36, wherein said method comprises the step of:
converting the QRNS residues to data in binary code.

40. The method according to claim 35, wherein said method comprises the step of:
converting the modular products residues from QRNS to CRNS.

41. The method according to claim 40, wherein said method comprises the step of:
converting the CRNS residues to data in binary code.

42. The method according to claim 35, wherein said method comprises the step of:
converting the modular products to data in binary code.

43. The method according to claim 34, wherein said method comprises the step of: computing modular sums of the QRNS residues.

44. The method according to claim 43, wherein said method comprises the step of: converting the modular sums from QRNS to CRNS.

45. The method according to claim 44, wherein said method comprises the step of: converting the CRNS residues to data in binary code.

46. The method according to claim 43, wherein said method comprises the step of: converting the modular sums to data in binary code.

47. The method according to claim 34, wherein said method comprises the step of: converting the QRNS residues from QRNS to CRNS.

48. The method according to claim 47, wherein said method comprises the step of: converting the CRNS residues to data in binary code.

49. The method according to claim 34, wherein said method comprises the step of: converting the CRNS residues to data in binary code.

50. The method according to claim 1, wherein said method comprises the steps of: receiving input data in residue format; and
computing modular products of the residues.

51. The method according to claim 50, wherein said method comprises the step of: computing modular sums of the products of residues.

52. The method according to claim 51, wherein said method comprises the step of:
converting the modular sums from QRNS to CRNS.

53. The method according to claim 52, wherein said method comprises the step of:
converting the CRNS residues to data in binary code.

54. The method according to claim 51, wherein said method comprises the step of:
converting the modular sums to data in binary code.

55. The method according to claim 50, wherein said method comprises the step of:
converting the modular products from QRNS to CRNS.

56. The method according to claim 55, wherein said method comprises the step of:
converting the CRNS residues to data in binary code.

57. The method according to claim 50, wherein said method comprises the step of:
converting the modular products to data in binary code.

58. The method according to claim 1, wherein said method comprises the steps of:
receiving input data in residue format; and
computing modular sums of the residues.

59. The method according to claim 58, wherein said method comprises the step of:
converting the residues from QRNS to CRNS.

60. The method according to claim 59, wherein said method comprises the step of:
converting the CRNS residues to data in binary code.

61. The method according to claim 58, wherein said method comprises the step of: converting the residues to data in binary code.

62. The method according to claim 1, wherein said method comprises the steps of: receiving input data in residue format; and converting the residues from QRNS to CRNS.

63. The method according to claim 62, wherein said method comprises the step of: converting the CRNS residues to data in binary code.

64. The method according to claim 1, wherein said method comprises the steps of: receiving input data in residue format; and converting the residues to data in binary code.

65. The method according to claim 1, wherein said plurality of logic gates are selected from the group consisting of: custom digital logic, standard cell logic, cell-based arrays of logic, gate arrays, field programmable gate arrays, and programmable logic devices.

66. The method according to claim 2, wherein said binary code is selected from the group consisting of: one's complement, sign-magnitude, unsigned binary, two's complement, fixed-radix, and floating-radix.

67. A method for implementing a computation of a product of a constant and a modular data operand, comprising the following steps:

computing a value of a multiply by a constant function for each possible modular data operand;

extracting logical equations representing computed values of the multiply by a constant function; and

mapping the logical equations to a plurality of logic gates,

wherein the plurality of logic gates receives a modular data operand and outputs a product of a constant and the modular data operand.

68. The method according to claim 67, further comprising the step of minimizing the logical equations prior to the step of mapping the logical equations to the plurality of logic gates.

69. A method for implementing computation of a number theoretic logarithm of a residue for a given base α_i and modulus p_i, comprising the steps of:

computing a value of a number theoretic logarithm function for each possible residue input;

extracting logical equations representing the computed values of the number theoretic logarithm function; and

mapping the logical equations to a plurality of logic gates,

wherein the plurality of logic gates receives a residue input and outputs a number theoretic logarithm of the residue input for a given base α_i and modulus p_i.

70. The method according to claim 69, further comprising the step of minimizing the logical equations prior to the step of mapping the logical equations to the plurality of logic gates.

71. The method according to claim 69, further comprising the step of outputting a zero symbol if the input operand is zero.

72. The method according to claim 71, wherein the zero symbol is a binary word having all ones.

73. The method according to claim 69, further comprising the step of:

implementing the number theoretic logarithm function for each of a plurality of possible generators; and

selecting an optimum generator,

wherein the selection of the optimum generator is based on at least one criterion selected from the group consisting of: speed, area, and power.

74. A method for implementing computation of a number theoretic exponentiation of a residue for a given base α_i and modulus p_i, comprising the steps of:

computing a value of a number theoretic exponentiation function for each possible residue input;

extracting logical equations representing the computed values of the number theoretic exponentiation function; and

mapping the logical equations to a plurality of logic gates,

wherein the plurality of logic gates receives a residue input and outputs a number theoretic exponentiation of the residue input for a given base α_i and modulus p_i.

75. The method according to claim 74, further comprising the step of minimizing the logical equations prior to the step of mapping the logical equations to the plurality of logic gates.

76. The method according to claim 74, further comprising the step of outputting a zero if the input operand is a zero symbol.

77. The method according to claim 74, further comprising the step of:

implementing the number theoretic exponentiation function for each of a plurality of possible generators; and

selecting an optimum generator,

wherein the selection of the optimum generator is based on at least one criterion selected from the group consisting of: speed, area, and power.

78. A method for computing the product of two residues, modulo p_i, comprising the steps of:

receiving a first number theoretic logarithm of a first residue corresponding to a first input operand, wherein the first number theoretic is a zero symbol if the first residue is zero;

receiving a second number theoretic logarithm of a second residue corresponding to a second input operand, wherein the second number theoretic is the zero symbol if the second residue is zero;

adding the first number theoretic logarithm and the second number theoretic logarithm, modulo p_i-1, to produce a sum of the first input operand and the second input operand, modulo p_i-1;

generating an exponentiation of the sum of the first input operand and the second input operand, modulo p_i-1, wherein the step of generating an exponentiation is accomplished by inputting the sum, modulo p_i-1, to a plurality of logic gates, wherein the plurality of logic gates is selected by the steps of:

computing a value of a number theoretic exponentiation function for each possible input;

extracting logical equations representing the computed values of the number theoretic exponentiation function;

mapping the logical equation to a plurality of logic gates;

wherein the output of the plurality of logic gates is the product of the first residue and the second residue, modulo p_i.

79. The method according to claim 78, wherein the step of adding the first and second number theoretic logarithms is accomplished utilizing a modular adder circuit.

80. The method according to claim 78,

further comprising the step of:

setting the output of the plurality of logic gates to zero when one or both of the first number theoretic logarithm and the second number theoretic logarithm is a zero symbol.

81. A method for reduction of an N bit binary operand to a residue, modulo p_i, comprising the steps of:

receiving an N-bit operand which is partitioned into $q_i + 1$ groups of bits;

zero extending the N_i-1 least significant bits;

inputting each of the remaining q_i partitions of bits Q_{ij} for $j \in \{0,1,2,..., q_i -1\}$ to a corresponding q_i pluralities of logic gates, wherein each of the q_i pluralities of logic gates produce a residue of the input partition;

adding the zero-extended N_i-1 least significant bits and the q_i residues of the q_i partitions, modulo p_i, to produce a residue, modulo p_i, of the N bit operand.

82. The method according to claim 81, wherein the q_i pluralities of logic gates are selected by the steps of:

computing a value of a modular reduction function for each possible input for each of the remaining q_i partitions;

extracting logical equations representing the computed values of the modular reduction function; and

mapping the logical equations to q_i pluralities of logic gates.

83. The method according to claim 82, further comprising the step of minimizing the logical equations prior to the step of mapping the logical equations to the plurality of logic gates.

84. A method of performing the computation of the sum of L operands modulo p_i, comprising the steps of:

adding L operands to produce a $N_i + \lceil \log_2 L \rceil$ bit unsigned sum;

inputting the $\lceil \log_2 L \rceil$+1 most significant bits to a plurality of logic gates, wherein said plurality of logic gates produces a residue of the $\lceil \log_2 L \rceil$+1 most significant bits;

zero extending the N_i-1 least significant bits; and

adding, modulo p_i, the zero-extended N_i-1 least significant bits and the $\lceil \log_2 L \rceil$+1 most significant bits to produce the sum of the L operand modulo p_i,

wherein the plurality of logic gates is selected by the steps of:

computing a value of a modular reduction function for each possible input;

extracting logical equations representing the computed values of the modular reduction function;

mapping the logical equation to a plurality of logic gates;

wherein the output of the plurality of logic gates is a residue of the $\lceil \log_2 L \rceil + 1$ most significant bits.

85. The method according to claim 84, further comprising the step of minimizing the logical equations prior to the step of mapping the logical equations to the plurality of logic gates.

86. A method for converting an L operand RNS value to a binary representation using the Chinese Remainder Theorem (CRT), comprising the steps of:

inputting L residues to a corresponding L pluralities of logic gates to produce L results;

adding the L results modulo M to produce a binary representation of an L operand RNS value,

wherein the plurality of logic gates is selected by the steps of:

computing a value of a CRT function for each possible input;

extracting logical equations representing the computed values of the CRT function; and

mapping the logical equation to a plurality of logic gates.

87. The method according to claim 86, further comprising the step of minimizing the logical equations prior to the step of mapping the logical equations to the plurality of logic gates.

88. A method for converting an L operand RNS value to a binary representation, comprising the steps of:

inputting L residues associated with an L operand RNS value to L pluralities of logic gates to produce L scaled results;

adding the L scaled results, modulo 2^k, to produce a binary representation of an L operand RNS value.

wherein the plurality of logic gates is selected by the steps of:

computing a value of a LCRT function for each possible input;

extracting logical equations representing the computed values of the *L*CRT function; and

mapping the logical equation to a plurality of logic gates.

89. The method according to claim 88, further comprising the step of minimizing the logical equations prior to the step of mapping the logical equations to the plurality of logic gates.

90. A method for converting a complex residue number system (CRNS) value to a quadratic residue number system (QRNS) value, comprising the steps of:

inputting an imaginary component of a CRNS value, b_i, to a plurality of logic gates to produce a product $\hat{j}\,b_i$;

adding, modulo p_i, a real operand, a_i, and the product $\hat{j}\ b_i$ to produce a first QRNS component, z_i;

subtracting the product $\hat{j}\,b_i$, modulo p_i, from the real operand, a_i, to produce a second QRNS component, z_i*,

wherein the plurality of logic gates is selected by the steps of:

computing a value of a multiply by $\hat{j}$function for each possible input;

extracting logical equations representing the computed values of the multiply by $\hat{j}$ function;

mapping the logical equation to a plurality of logic gates;

wherein the output of the plurality of logic gates is the product of the imaginary component of a CRNS value and $\hat{j}$.

91. The method according to claim 90, further comprising the step of minimizing the logical equations prior to the step of mapping the logical equations to the plurality of logic gates.

92. The method according to claim 2, wherein the step of converting the residues from CRNS to QRNS comprises the steps of:

inputting an imaginary component of a CRNS value, b_i, to a plurality of logic gates to produce a product $\hat{j}\,b_i$;

adding, modulo p_i, a real operand, a_i, and the product $\hat{j}\ b$ to produce a first QRNS component, z_i;

subtracting the product $\hat{j}\,b_i$, modulo p_i, from the real operand, a_i, to produce a second QRNS component, z_i*,

wherein the plurality of logic gates is selected by the steps of:

computing a value of a multiply by $\hat{j}$function for each possible input;

extracting logical equations representing the computed values of the multiply by $\hat{j}$ function;

mapping the logical equation to a plurality of logic gates;

wherein the output of the plurality of logic gates is the product of an imaginary component of a CRNS value and $\hat{j}$.

93. The method according to claim 92, further comprising the step of minimizing the logical equations prior to the step of mapping the logical equations to the plurality of logic gates.

94. A method for converting a quadratic residue number system (QRNS) value to a complex residue number system (CRNS) representation, comprising the steps of:

adding a first QRNS component, z_i, modulo p_i, to a second QRNS component, z_i*, to produce a sum;

subtracting the second QRNS component, z_i*, modulo p_i, from the first QRNS component, z_i, to produce a difference;

inputting the sum to a first plurality of logic gates to produce a real component, a_i, of a CRNS representation; and

inputting the difference to a second plurality of logic gates to produce an imaginary component, b_i, of the CRNS representation,

wherein the first plurality of logic gates is selected by the steps of:

computing a value of a multiply by 2^{-1} function for each possible input;

extracting logical equations representing the computed values of the multiply by 2^{-1} function;

mapping the logical equation to a plurality of logic gates;

wherein the output of the first plurality of logic gates is the product of the sum and 2^{-1},

wherein the second plurality of logic gates is selected by the steps of:

computing a value of a multiply by 2^{-1} function for each possible input;

extracting logical equations representing the computed values of the multiply by 2^{-1} function;

mapping the logical equation to a plurality of logic gates;

wherein the output of the second plurality of logic gates is the product of the difference and $2^{-1}\hat{j}^{-1}$.

95. The method according to claim 94, further comprising the step of minimizing the logical equations prior to the step of mapping the logical equations to the first plurality of logic gates.

96. The method according to claim 94, further comprising the step of minimizing the logical equations prior to the step of mapping the logical equations to the second plurality of logic gates.

97. An apparatus for performing mathematical computations using residue arithmetic, comprising one or more of the following:

a means for converting data in binary code to residues;

a means for converting residues from CRNS to QRNS;

a means for computing modular products of residues;

a means for computing modular sums of residues;

a means for converting residues from QRNS to CRNS; and

a means for converting residues to data in binary code,

wherein at least one of said means comprises a corresponding at least one plurality of logic gates.

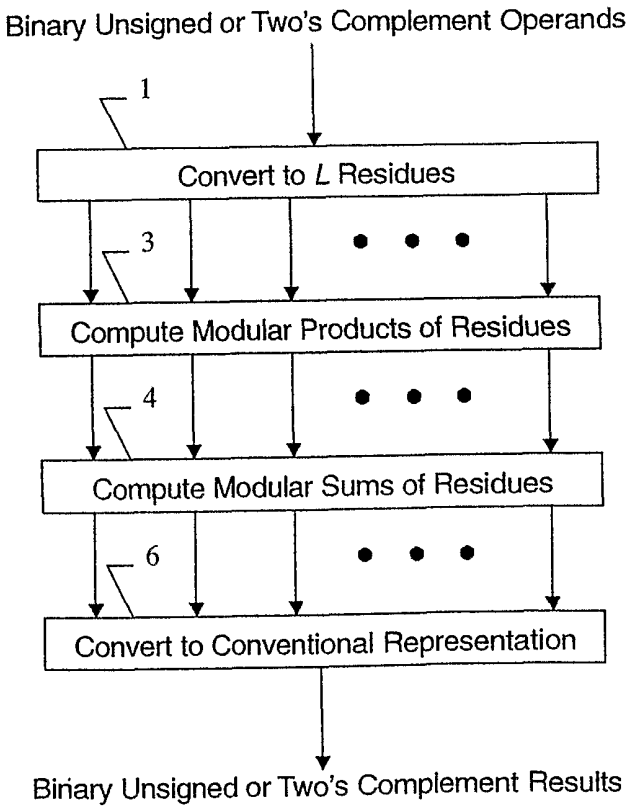

Binary Unsigned or Two's Complement Operands
1
Convert to L Residues
3
Compute Modular Products of Residues
4
Compute Modular Sums of Residues
6
Convert to Conventional Representation
Binary Unsigned or Two's Complement Results

FIG. 1

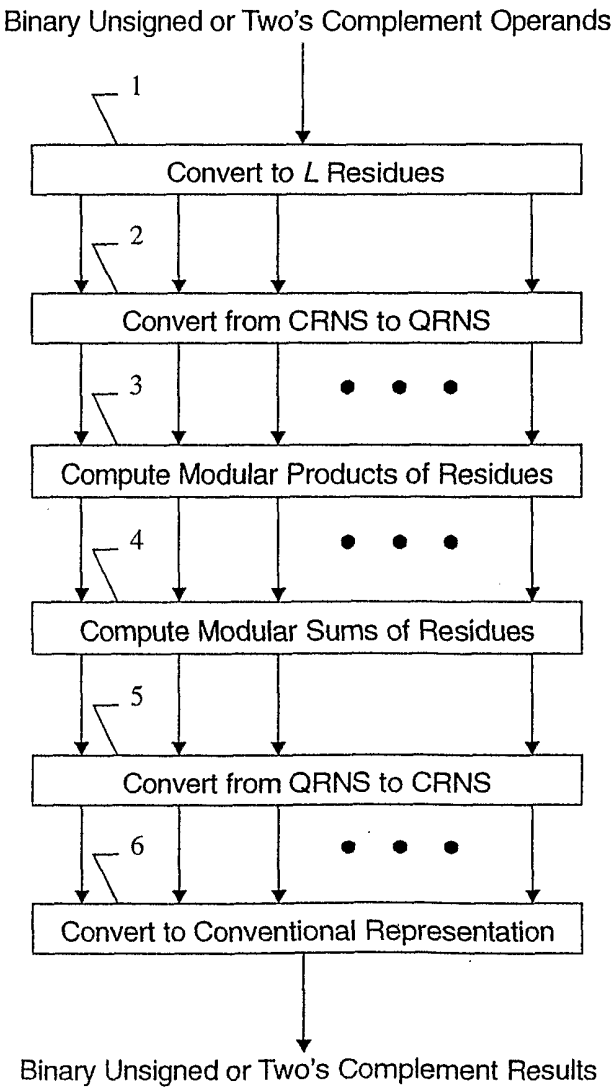

Binary Unsigned or Two's Complement Operands
1
Convert to L Residues
2
Convert from CRNS to QRNS
3
Compute Modular Products of Residues
4
Compute Modular Sums of Residues
5
Convert from QRNS to CRNS
6
Convert to Conventional Representation
Binary Unsigned or Two's Complement Results

FIG. 2

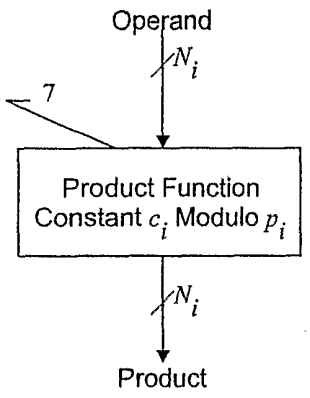


FIG. 3

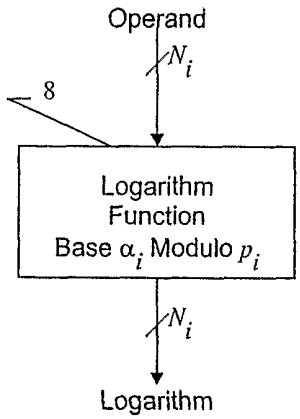


FIG. 5

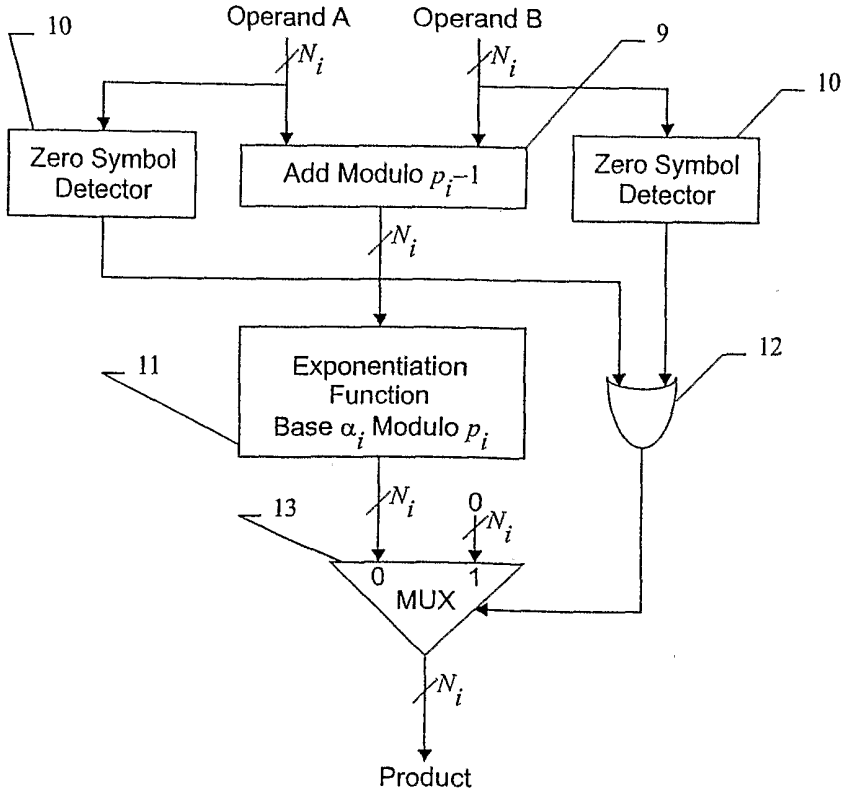


FIG. 6

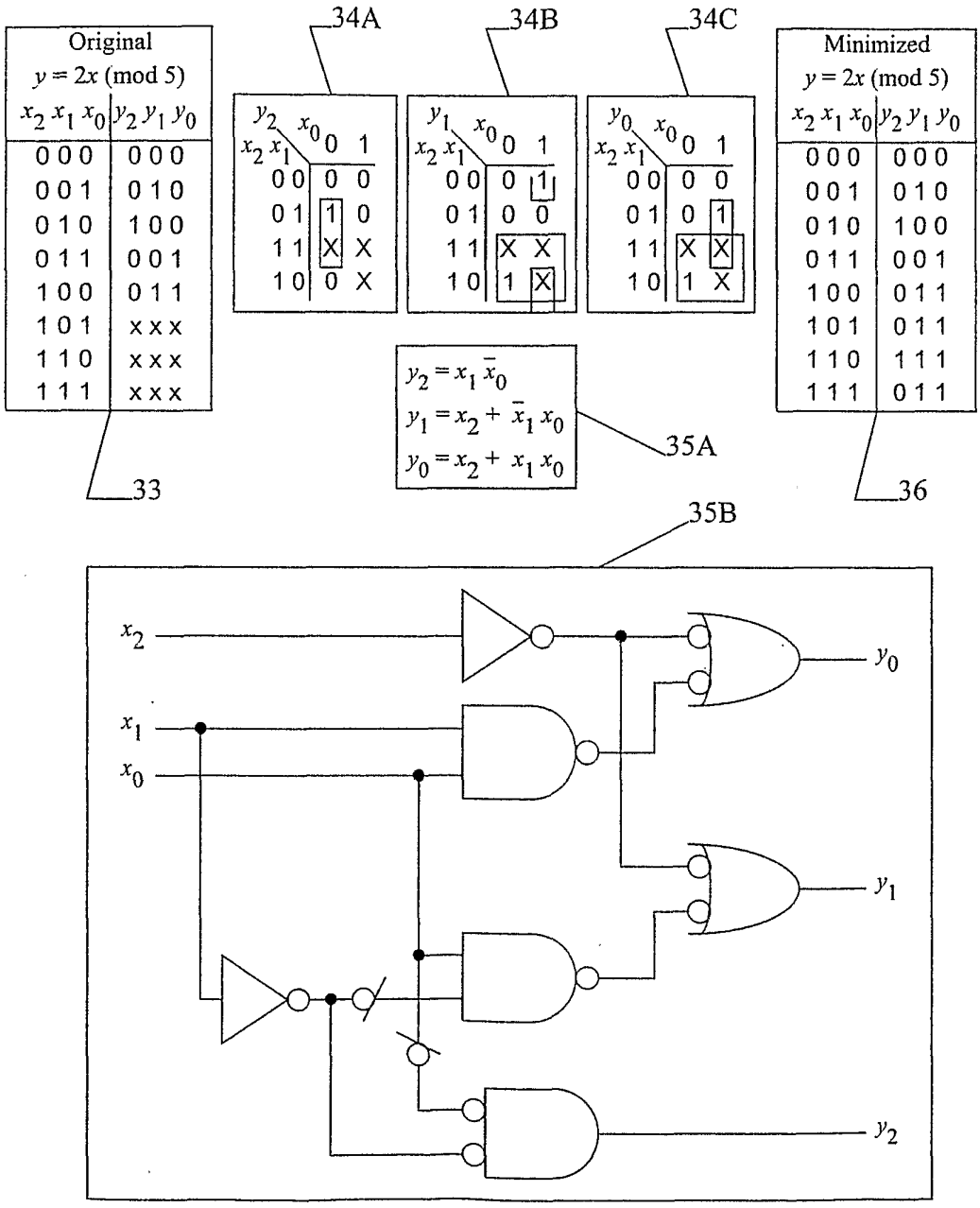

Original
$y = 2x \pmod 5$
$x_2\ x_1\ x_0$ | $y_2\ y_1\ y_0$
0 0 0 | 0 0 0
0 0 1 | 0 1 0
0 1 0 | 1 0 0
0 1 1 | 0 0 1
1 0 0 | 0 1 1
1 0 1 | x x x
1 1 0 | x x x
1 1 1 | x x x
33
34A
34B
34C
$y_2 = x_1 \bar{x}_0$
$y_1 = x_2 + \bar{x}_1 x_0$
$y_0 = x_2 + x_1 x_0$
35A
Minimized
$y = 2x \pmod 5$
$x_2\ x_1\ x_0$ | $y_2\ y_1\ y_0$
0 0 0 | 0 0 0
0 0 1 | 0 1 0
0 1 0 | 1 0 0
0 1 1 | 0 0 1
1 0 0 | 0 1 1
1 0 1 | 0 1 1
1 1 0 | 1 1 1
1 1 1 | 0 1 1
36
35B
x_2
x_1
x_0
y_0
y_1
y_2

FIG. 4

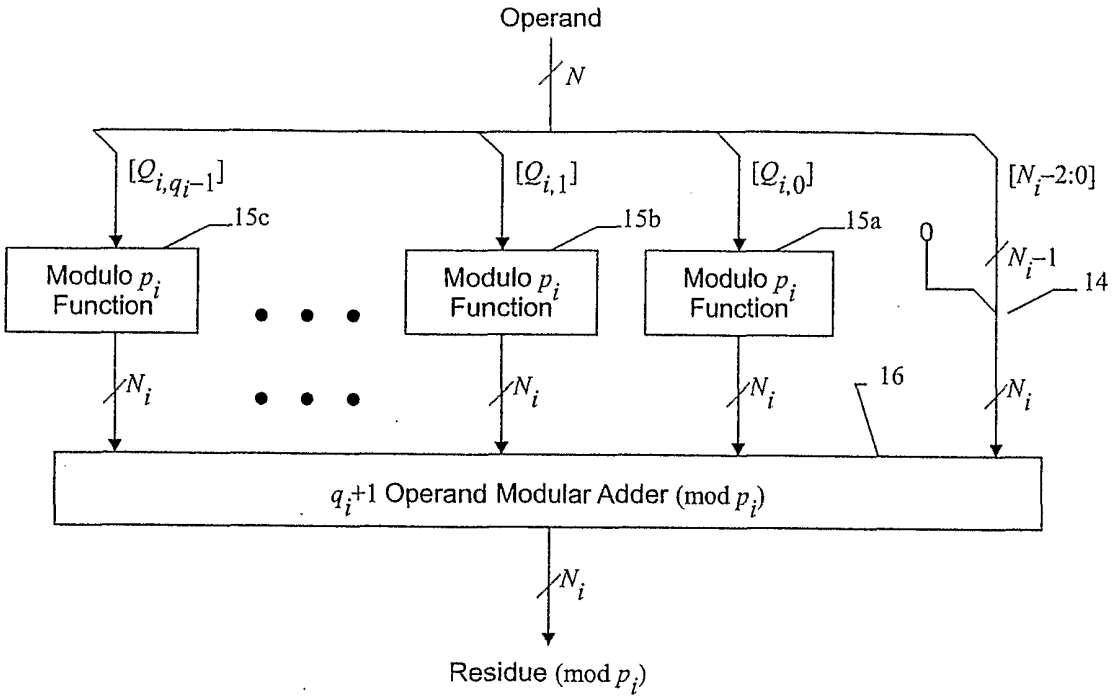

Operand
N
[Q_{i,q_i-1}]
[Q_{i,1}]
[Q_{i,0}]
[N_i-2:0]
15c
15b
15a
0
N_i-1
14
Modulo p_i Function
Modulo p_i Function
Modulo p_i Function
N_i
N_i
N_i
16
N_i
q_i+1 Operand Modular Adder (mod p_i)
N_i
Residue (mod p_i)

FIG. 7

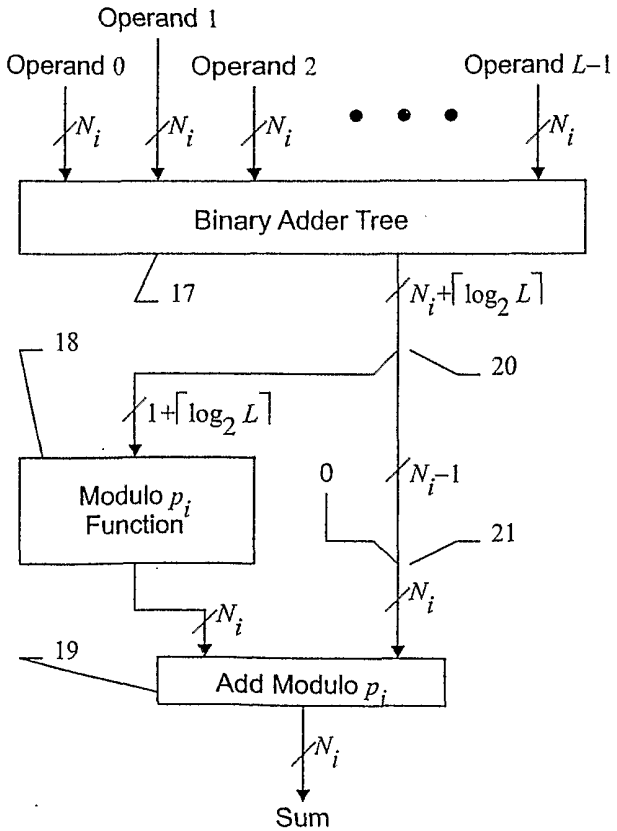

Operand 1
Operand 0
Operand 2
Operand L-1
N_i
N_i
N_i
N_i
Binary Adder Tree
17
N_i+⌈log_2 L⌉
18
20
1+⌈log_2 L⌉
0
N_i-1
21
Modulo p_i Function
N_i
N_i
19
Add Modulo p_i
N_i
Sum

FIG. 8

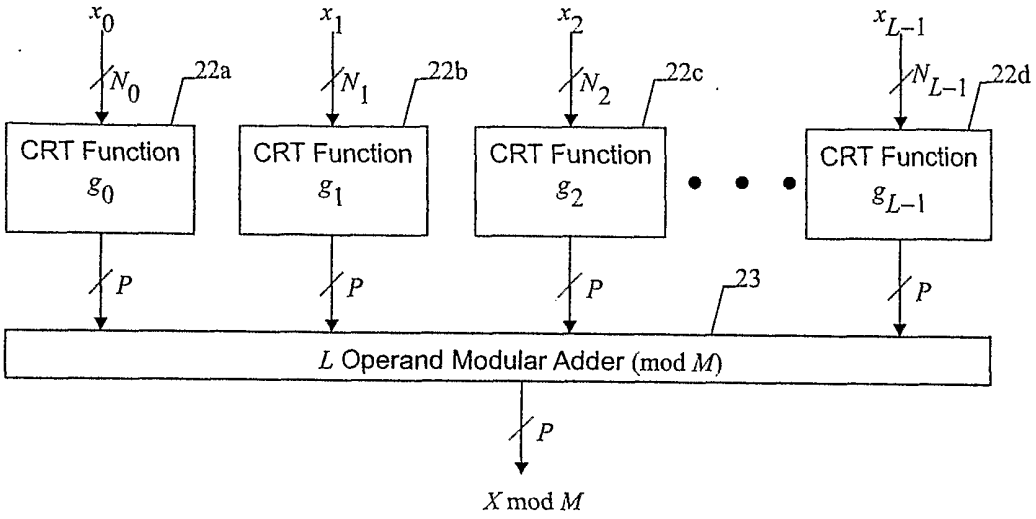

x_0
N_0
22a
CRT Function g_0
x_1
N_1
22b
CRT Function g_1
x_2
N_2
22c
CRT Function g_2
x_{L-1}
N_{L-1}
22d
CRT Function g_{L-1}
P
23
L Operand Modular Adder (mod M)
X mod M

FIG. 9

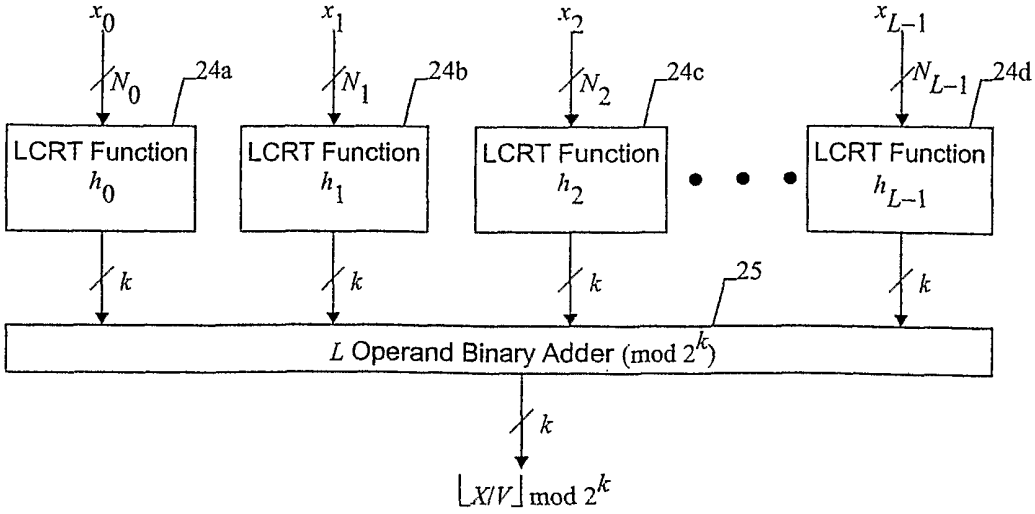

x_0
N_0
24a
LCRT Function h_0
x_1
N_1
24b
LCRT Function h_1
x_2
N_2
24c
LCRT Function h_2
x_{L-1}
N_{L-1}
24d
LCRT Function h_{L-1}
k
25
L Operand Binary Adder (mod 2^k)
$\lfloor X/V \rfloor$ mod 2^k

FIG. 10

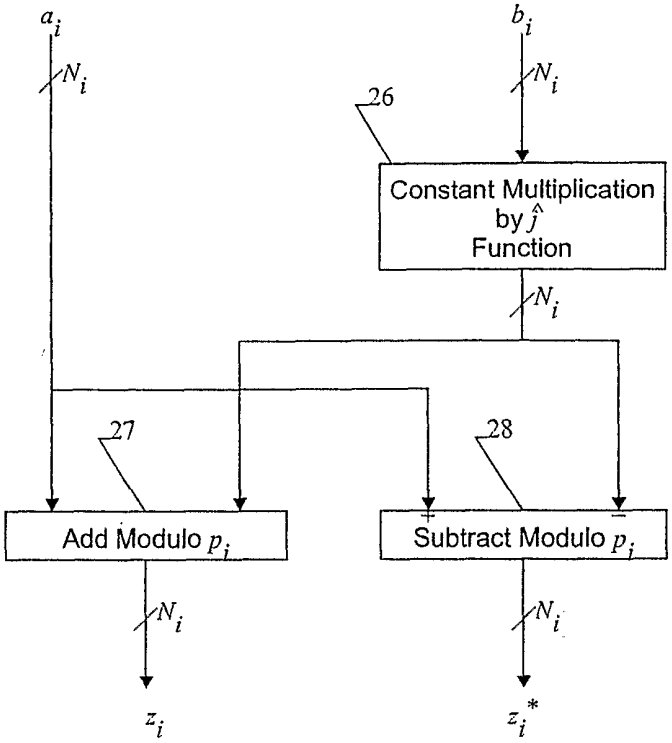

a_i
N_i
b_i
N_i
26
Constant Multiplication by $\hat{j}$ Function
N_i
27
28
Add Modulo p_i
+
Subtract Modulo p_i
−
N_i
N_i
z_i
z_i^*

FIG. 11

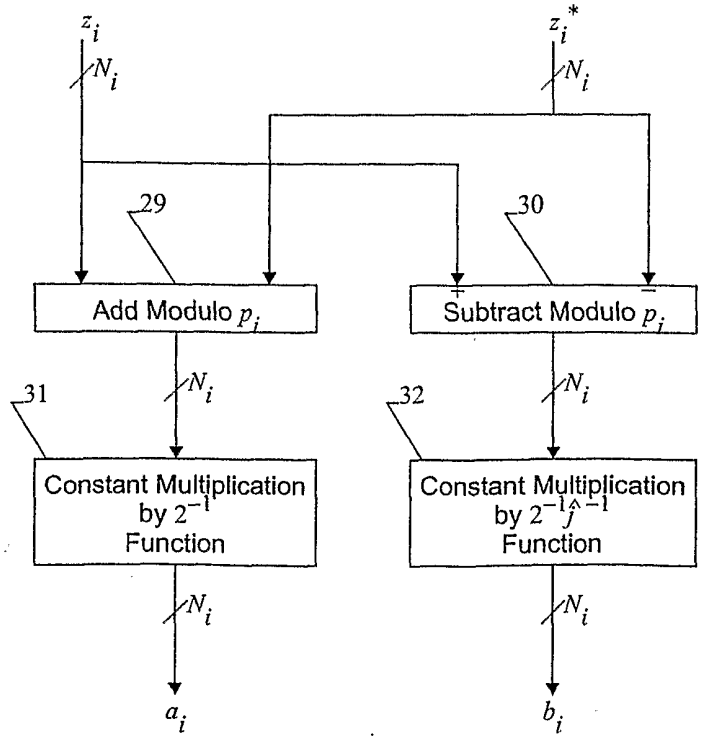

z_i
N_i
z_i^*
N_i
29
30
Add Modulo p_i
+
Subtract Modulo p_i
−
N_i
N_i
31
32
Constant Multiplication by 2^{-1} Function
Constant Multiplication by $2^{-1}\hat{j}^{-1}$ Function
N_i
N_i
a_i
b_i

FIG. 12

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 00/27221

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F7/72 H03M7/18

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)
EPO-Internal, INSPEC, PAJ

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	EP 0 795 819 A (CIRRUS LOGIC INC) 17 September 1997 (1997-09-17) the whole document	1-83,97

X	US 5 424 971 A (YANG LIN ET AL) 13 June 1995 (1995-06-13)	67,68
A	abstract	70,75, 83,87

	-/--	

[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
30 January 2001	29. 06. 2001

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	VERHOOF, P

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

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US 00/27221

Box I Observations where certain claims were found unsearchable (Continuation of item 1 of first sheet)

This International Search Report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:

2. ☐ Claims Nos.:
because they relate to parts of the International Application that do not comply with the prescribed requirements to such an extent that no meaningful International Search can be carried out, specifically:

3. ☐ Claims Nos.:
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box II Observations where unity of invention is lacking (Continuation of item 2 of first sheet)

This International Searching Authority found multiple inventions in this international application, as follows:

see additional sheet .

1. ☐ As all required additional search fees were timely paid by the applicant, this International Search Report covers all searchable claims.

2. ☐ As all searchable claims could be searched without effort justifying an additional fee, this Authority did not invite payment of any additional fee.

3. ☐ As only some of the required additional search fees were timely paid by the applicant, this International Search Report covers only those claims for which fees were paid, specifically claims Nos.:

4. ☒ No required additional search fees were timely paid by the applicant. Consequently, this International Search Report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:

1-83,86,87,97

Remark on Protest

☐ The additional search fees were accompanied by the applicant's protest.

☐ No protest accompanied the payment of additional search fees.

FURTHER INFORMATION CONTINUED FROM PCT/ISA/ 210

This International Searching Authority found multiple (groups of) inventions in this international application, as follows:

1. Claims: 1-83,86,87,97

   Implementation of any logic function by means of logic gates

   1.1. Claims: 67,68
   multiplication by a constant

   1.2. Claims: 69-80
   multiplication via logarithmic domain

   1.3. Claims: 81-83
   residue operation

   1.4. Claims: 86,87
   RNS to binary conversion using CRT functions

2. Claims: 84,85

   multiple input modulo addition

3. Claims: 88,89

   RNS to binary conversion using LCRT functions

4. Claims: 90-93

   CNRS to QNRS conversion

5. Claims: 94-96

   QRNS to CRNS conversion

Please note that all inventions mentioned under item 1, although not necessarily linked by a common inventive concept, could be searched without effort justifying an additional fee.

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 00/27221

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	SMITH J C ET AL: "A FAULT-TOLERANT GEQRNS PROCESSING ELEMENT FOR LINEAR SYSTOLIC ARRAY DSP APPLICATIONS" IEEE TRANSACTIONS ON COMPUTERS,US,IEEE INC. NEW YORK, vol. 44, no. 9, 1 September 1995 (1995-09-01), pages 1121-1130, XP000526283 ISSN: 0018-9340 cited in the application page 1123, right-hand column, paragraph 1 -page 1124, right-hand column, paragraph 1; figures 2,9 -----	69-80, 86,87

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No
PCT/US 00/27221

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP 0795819 A	17-09-1997	US 5987487 A	16-11-1999
		JP 10032493 A	03-02-1998
US 5424971 A	13-06-1995	NONE	