EE 7715
RNS Hardware

Consider an RNS system defined by the moduli set $S = \{m_1, m_2, \cdots, m_L\}$; $(m_i, m_j) = 1$ for $i \neq j$. Such a system will consist of:

A. Weighted (binary) - to - RNS conversion hardware.

B. RNS processing hardware.

C. RNS - to - weighted (binary) conversion hardware.

The hardware organization of the entire RNS system is shown by the figure on next page.

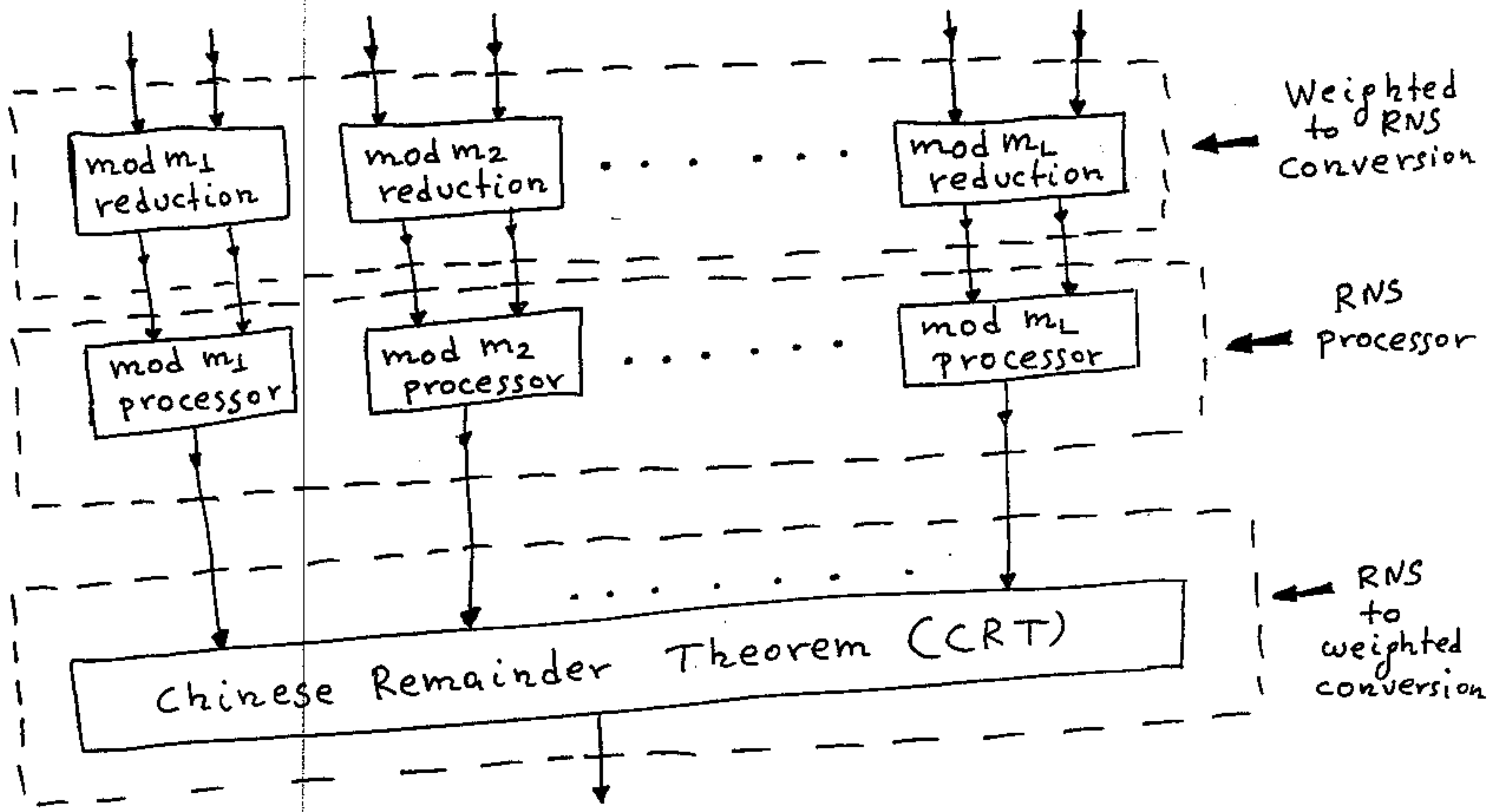


Figure 1: Hardware organization of the entire RNS system using set $S=\{m_1, m_2, \ldots, m_L\}$.

A brief discussion on weighted-to-RNS conversion, RNS processing, and RNS-to-weighted conversion follows.

A. Weighted-to-RNS conversion

The problem here is to evaluate $\langle X \rangle_m$ where X is an n-bit integer while the modulus m is a positive integer.

The evaluation of $\langle X \rangle_m$ can be accomplished by performing the division of X by m and keeping the remainder. This is not a good way, however, since division is a slow operation.

A better technique follows:

Let the n-bit number X be

$$X = (x_{n-1} x_{n-2} \cdots x_1 x_0)_2$$

Then

$$X = x_{n-1} \cdot 2^{n-1} + x_{n-2} \cdot 2^{n-2} + \cdots + x_1 \cdot 2 + x_0 .$$

Thus

4 e

$$\boxed{\langle X\rangle_m = \langle x_{n-1}\cdot\langle 2^{n-1}\rangle_m + x_{n-2}\cdot\langle 2^{n-2}\rangle_m + \cdots + \cdots + x_1\langle 2\rangle_m + x_0\rangle_m \qquad (1).}$$

The terms $\langle 2^i\rangle_m$ can be precomputed and stored; (these terms are preknown constants). A multioperand addition mod m can then complete the computation $\langle X\rangle_m$ of equation (1).

<u>Example:</u> Compute $\langle X\rangle_m$ where $m = 19$ and $X = (1111111100)_2$.

Here $X = 2^9 + 2^8 + 2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2$.

Thus,

$$\langle X\rangle_m = \langle\langle 2^9\rangle_{19} + \langle 2^8\rangle_{19} + \langle 2^7\rangle_{19} + \langle 2^6\rangle_{19} + \langle 2^5\rangle_{19} + \langle 2^4\rangle_{19} + \langle 2^3\rangle_{19} + \langle 2^2\rangle_{19}\rangle_{19}$$

$$= \langle -1 + 9 - 5 + 7 + 13 - 3 + 8 + 4\rangle_{19} = \langle 32\rangle_{19} = 13.$$

Double check to see that $\langle X\rangle_m =$

$= \langle 1020\rangle_{19} = 13$.

B. RNS Processing (Arithmetic mod m)

B-1. Addition modulo m

Let A, B be integers such that $A \in Z_m$, $B \in Z_m$ where $Z_m = \{0, 1, 2, \ldots, m-1\}$. The problem here is to compute $\langle A+B \rangle_m$.
Let A and B be n-bit numbers or

$$A = (a_{n-1}\, a_{n-2} \ldots a_1\, a_0)_2$$
$$B = (b_{n-1}\, b_{n-2} \ldots b_1\, b_0)_2$$

Let S be the summation of A and B or

$$S = A + B = (s_n s_{n-1} \cdots s_1 s_0)_2$$

Obviously, $0 \leq S \leq m+m-2$ (since $0 \leq A \leq m-1$, $0 \leq B \leq m-1$).

If $0 \leq S \leq m-1$ (which means that no overflow occured due to addition), then

$$\langle A+B \rangle_m = S$$

If $m \leq S \leq m+m-2$ (which means that overflow occured), then

$\langle A+B \rangle_m = S - m = S + 2\text{'s complement of } m$
(ignore carry out of this addition).

B-2. Additive inverse mod m (negation mod m)

Let A be an integer such that $A \in Z_m$.
Obviously

$$\langle -A \rangle_m = \langle m - A \rangle_m = m + 2\text{'s compl. of } A$$

(ignore carry out).

B-3. Subtraction mod m

Let A, B be integers such that $A \in Z_m$, $B \in Z_m$. Then

$$\langle A - B \rangle_m = \langle A + \langle -B \rangle_m \rangle_m .$$

B-4. Multiplication mod m

Let A, B be n-bit integers such that $A \in Z_m$, $B \in Z_m$. Let P be the full precision product of A and B or

7 e

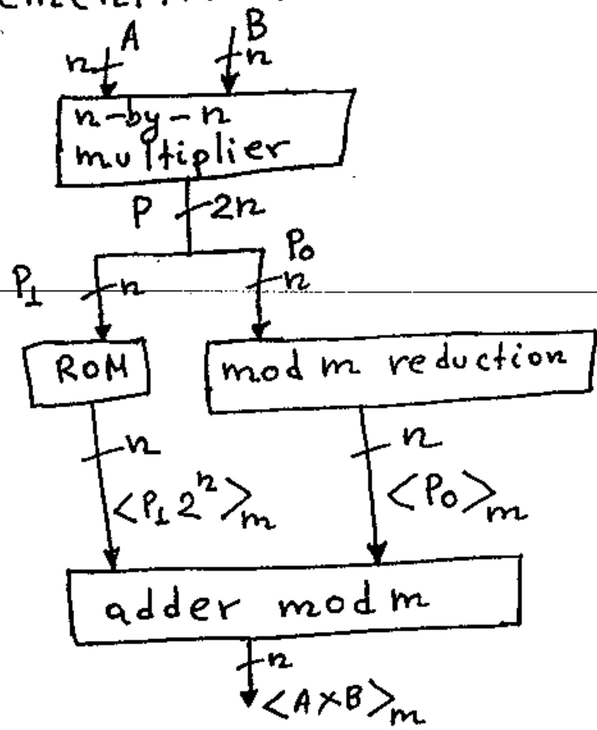
$P = A \times B$. Since A, B are n-bit numbers, the product P is a $2n$-bit number. Let P be decomposed into two n-bit blocks as $P = P_1 P_0$. Obviously

$$P = P_1 \cdot 2^n + P_0 .$$

Thus

$$\boxed{\begin{aligned} \langle A \times B \rangle_m &= \langle P \rangle_m = \langle P_1 \cdot 2^n + P_0 \rangle_m = \\ &= \langle \langle P_1 \cdot 2^n \rangle_m + \langle P_0 \rangle_m \rangle_m \qquad (2) \end{aligned}}$$

The above eq. (2) suggests the following possible implementation.



Another technique used for mod m multiplication could be the index calculus technique (generators etc), in case where m is prime.

C. RNS-to-weighted conversion using CRT

Let the moduli set be

$S = \{m_1, m_2, \ldots, m_L\}$; $(m_i, m_j) = 1$ for $i \neq j$

and let the RNS representation of Z be

$Z \xrightarrow{RNS} (Z_1, Z_2, \ldots, Z_L)$.

The CRT then dictates

$$\boxed{\begin{array}{l} Z = \left\langle \langle Z_1 N_1 \rangle_{m_1} \cdot M_1 + \langle Z_2 N_2 \rangle_{m_2} \cdot M_2 + \cdots + \langle Z_L N_L \rangle_{m_L} \cdot M_L \right\rangle_M \\ \text{where } M = \prod_{i=1}^{L} m_i \text{ ; } \quad M_i = \dfrac{M}{m_i} \text{ ; } \quad N_i = \langle M_i^{-1} \rangle_{m_i} \end{array}}$$

In the above, $M_1, M_2, \ldots, M_L, N_1, N_2, \ldots, N_L, M$ are preknown constants.

The operations $\langle Z_i \cdot N_i \rangle_{m_i}$ can be performed using the existing mod m_i multipliers which belong to the RNS processing hardware. Alternatively speaking, $\langle Z_i \cdot N_i \rangle_{m_i}$ can be performed using special purpose ~~hardware~~ ~~processing~~

multipliers performing $\langle Z_i \times \text{constant} \rangle_{m_i}$.

Post multiplying $\langle Z_i \cdot N_i \rangle_{m_i}$ by M_i can also be implemented by special purpose multipliers performing multiplications by constents.

Finally, an L-operand mod M adder is needed to perform the final CRT addition.

The 3-moduli RNS based on set $S = \{2^n - 1,\ 2^n,\ 2^n + 1\}$

A residue number system that has been given considerable attention by several researchers is the 3-moduli RNS based on set

$$S = \{2^n - 1,\ 2^n,\ 2^n + 1\}; \quad n = \text{integer}.$$

Such RNS systems imply simple weighted-to-RNS and RNS-to-weighted conversions, simple RNS arithmetic and very balanced decomposition of the dynamic range.

<u>Lemma:</u> Let a be an odd integer. Then the numbers a, $a+1$, $a+2$ are pairwise relatively prime.

<u>Proof:</u>

Let d be a common divisor of $a+2$ and $a+1$. Then d must divide their difference, or $d \mid 1$ which means that $d = 1$. Thus, the only common divisor of $a+2$ and $a+1$ is 1 and $(a+2,\ a+1) = 1$. The fact that $a+1$

and a are relatively prime can be proven in a similar way.

Regarding the numbers $a+2$ and a, let d' be a common divisor of them. Then d' must divide their difference $a+2-a=2$, or $d'|2$ which means that $d'=1$ or 2. The number $d'=2$, however, does not divide $a+2$ nor a since both $a+2$ and a are odd numbers. Thus $d'=1$ and $(a+2, a)=1$. The proof is completed.

The above general lemma [redacted] dictates that the set $S=\{2^n-1, 2^n, 2^n+1\}$ consists of pairwise relatively prime integers.

A. Weighted-to-RNS conversion based on set $S=\{m_1, m_2, m_3\}=\{2^n-1, 2^n, 2^n+1\}$

An RNS based on set $S=\{m_1, m_2, m_3\}= =\{2^n-1, 2^n, 2^n+1\}$ achieves a dynamic range of approximately $3n$ bits (see that $M=m_1\cdot m_2\cdot m_3 = 2^{3n}-2^n$).

Let X be an integer such that $X\in Z_M$ ([redacted] $Z_M=\{0, 1, 2, \ldots, M-1\}$).

③ f

The residues that need to be computed here are $X_1 = \langle X \rangle_{2^n-1}$, $X_2 = \langle X \rangle_{2^n}$, $X_3 = \langle X \rangle_{2^n+1}$.

Let the 3n-bit integer X be decomposed into three n-bit blocks as $X = X_2 X_1 X_0$.

Obviously, $X = X_2 \cdot 2^{2n} + X_1 \cdot 2^n + X_0$.

Thus,

$$X_1 = \langle X \rangle_{2^n-1} = \langle X_2 2^{2n} + X_1 2^n + X_0 \rangle_{2^n-1} =$$

$$= \langle X_2 (2^n)^2 + X_1 2^n + X_0 \rangle_{2^n-1} = \langle X_2 + X_1 + X_0 \rangle_{2^n-1}$$

or

$$\boxed{X_1 = \langle X \rangle_{2^n-1} = \langle X_2 + X_1 + X_0 \rangle_{2^n-1}} \quad (1)$$

$$X_2 = \langle X \rangle_{2^n} = \langle X_2 (2^n)^2 + X_1 2^n + X_0 \rangle_{2^n} = \langle X_0 \rangle_{2^n} = X_0$$

or

$$\boxed{X_2 = \langle X \rangle_{2^n} = X_0} \quad (2)$$

$$X_3 = \langle X \rangle_{2^n+1} = \langle X_2 (2^n)^2 + X_1 2^n + X_0 \rangle_{2^n+1} =$$

$$= \langle X_2 - X_1 + X_0 \rangle_{2^n+1}$$

or

$$\boxed{X_3 = \langle X \rangle_{2^n+1} = \langle X_2 - X_1 + X_0 \rangle_{2^n+1}} \quad (3)$$

The facts $\langle 2^n \rangle_{2^n-1} = 1$, $\langle 2^n \rangle_{2^n} = 0$ and $\langle 2^n \rangle_{2^n+1} = \langle -1 \rangle_{2^n+1}$ have been taken into account in deriving equations (1), (2) and (3).

Equations (1), (2), (3) demonstrate that the weighted-to-RNS conversion is simple when using an RNS based on set $S = \{2^n-1, 2^n, 2^n+1\}$.

B. RNS Processing (Arithmetic mod (2^n-1), mod 2^n, mod (2^n+1))

B-1. Arithmetic mod (2^n-1)

B.1.1 Addition mod (2^n-1)

Let A, B be integers such that $A \in Z_{2^n-1}$, $B \in Z_{2^n-1}$, where $Z_{2^n-1} = \{0, 1, 2, \cdots, 2^n-2\}$.

Then $\langle A+B \rangle_{2^n-1}$ = result of 1's complement addition of A and B (add A and B and end around carry). The reason for the end around carry is that the weight factor of the carry out is 2^n while $\langle 2^n \rangle_{2^n-1} = 1 = 2^0$.

B.1.2 Additive inverse mod(2^n-1); (negation mod(2^n-1))

Let A be such that $A \in Z_{2^n-1}$. A is then an n-bit number or $A = (a_{n-1} a_{n-2} \cdots a_1 a_0)_2$. The additive inverse of A is

$$\langle -A \rangle_{2^n-1} = \langle 2^n - 1 - A \rangle_{2^n-1} = \langle (\overline{a_{n-1}}\, \overline{a_{n-2}} \cdots \overline{a_1}\, \overline{a_0}) \rangle_{2^n-1}$$

$$= (\overline{a_{n-1}}\, \overline{a_{n-2}} \cdots \overline{a_1}\, \overline{a_0}) = 1\text{'s complement of } A.$$

Thus

$$\langle -A \rangle_{2^n-1} = 1\text{'s complement of } A.$$

B.1.3 Subtraction mod(2^n-1)

Obviously,

$$\langle A - B \rangle_{2^n-1} = A + 1\text{'s complement of } B \text{ (end around carry).}$$

B.1.4 Scaling by power of two (scaling by 2^k)

Let A be such that $A \in Z_{2^n-1}$. The problem here is to compute $\langle 2^k \cdot A \rangle_{2^n-1}$.

Let A be decomposed into two blocks A_1 and A_0 where A_1 is a k-bit block while A_0 is a $(n-k)$-bit block.

Then,

$$A = \underbrace{A_1}_{\substack{k-\\ \text{bit}}}\ \underbrace{A_0}_{\substack{n-k\\ \text{bit}}} = A_1 \cdot 2^{n-k} + A_0 .$$

Thus, $\langle 2^k \cdot A \rangle_{2^n-1} = \langle 2^k \cdot (A_1 \cdot 2^{n-k} + A_0 \rangle_{2^n-1}$

$$= \langle A_1 \cdot 2^n + A_0 \cdot 2^k \rangle_{2^n-1} = \langle A_0 \cdot 2^k + A_1 \rangle_{2^n-1} =$$

$$= \langle (\underset{\substack{\uparrow\\ n-k\\ \text{bit}}}{A_0}\ \underbrace{000\cdots000}_{k \text{ zeros}}) + \underbrace{A_1}_{\substack{k\\ \text{bit}}} \rangle_{2^n-1} = \langle A_0 A_1 \rangle_{2^n-1} = A_0 A_1 =$$

= result of rotating A left by k bits.

So

$$\boxed{\langle 2^k \cdot A \rangle_{2^n-1} = \text{result of left-rotating } A \text{ by } k \text{ bits}}$$

Example: Consider $n = 8$ and $A = (a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0)_2$.
Then $\langle 2^3 \cdot A \rangle_{2^8-1} = (a_4 a_3 a_2 a_1 a_0 a_7 a_6 a_5)_2$.

B.1.5 Multiplication mod $(2^n - 1)$

Let A, B be n-bit integers such that $A \in Z_{2^n-1}$, $B \in Z_{2^n-1}$. Here, $\langle A \times B \rangle_{2^n-1}$ needs to be computed. Let P be the full precision product of A and B or $P = A \times B$. Then P is a $2n$-bit number which can be decomposed into two n-bit blocks as $P = P_1 P_0$. Obviously $P = P_1 \cdot 2^n + P_0$ and thus

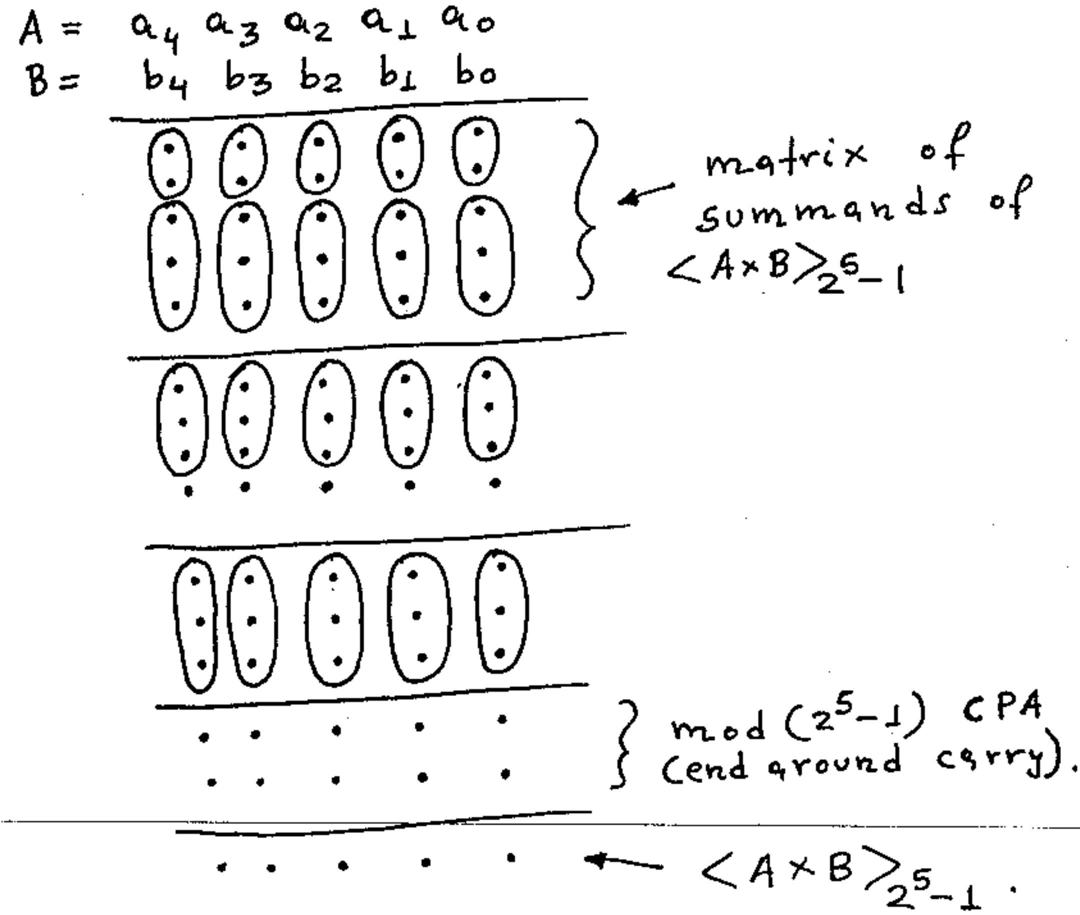
$$\boxed{\begin{aligned} \langle A \times B \rangle_{2^n-1} &= \langle P \rangle_{2^n-1} = \langle P_1 \cdot 2^n + P_0 \rangle_{2^n-1} \\ &= \langle P_1 + P_0 \rangle_{2^n-1}. \end{aligned}}$$

The above equation suggests that $\langle A \times B \rangle_{2^n-1}$ can be computed by an n-by-n full precision multiplier computing $A \times B = P = P_1 P_0$, followed by a mod (2^n-1) adder performing $\langle P_1 + P_0 \rangle_{2^n-1}$.

The above technique is not the most hardware efficient technique. A much better approach is a direct design of a mod (2^n-1) multiplier based on counters. This approach is shown by an example on next page.

8f

Let A, B be 5-bit integers such that $A \in Z_{2^5-1}$, $B \in Z_{2^5-1}$ and suppose that $\langle A \times B \rangle_{2^5-1}$ needs to be computed. Below you can see a counter-based implementation of $\langle A \times B \rangle_{2^5-1}$. The counters used are (3,2) and (2,2) counters.



B.2 Arithmetic mod 2^n

B.2.1 Addition mod 2^n

Let A, B be such that $A \in Z_{2^n}$, $B \in Z_{2^n}$.

Then $\langle A+B \rangle_{2^n}$ = result of 2's complement addition of A and B; (add A and B and ignore carry out).

The reason for ignoring the carry out is that its weight factor is 2^n while $\langle 2^n \rangle_{2^n} = 0$.

B.2.2 Additive inverse mod 2^n

Let $A \in Z_{2^n}$. Then

$\langle -A \rangle_{2^n} = \langle 2^n - A \rangle_{2^n}$ = 2's complement of A.

B.2.3 Subtraction mod 2^n

$\langle A - B \rangle_{2^n}$ = A + 2's complement of B; (ignore carry out).

B.2.4 Scaling by power of two (scaling by 2^k)

Let A be such that $A \in Z_{2^n}$. Decompose A into two blocks A_1 and A_0 where A_1 is a k-bit block while A_0 is a $(n-k)$-bit block. Then

$$A = A_1 A_0 = A_1 \cdot 2^{n-k} + A_0 .$$

Thus, $\langle 2^k \cdot A \rangle_{2^n} = \langle 2^k \cdot (A_1 \cdot 2^{n-k} + A_0) \rangle_{2^n} =$

$= \langle A_1 \cdot 2^n + A_0 \cdot 2^k \rangle_{2^n} = \langle A_0 \cdot 2^k \rangle_{2^n} =$

$$= \langle (\underbrace{A_0}_{\substack{n-k \\ \text{bit}}} \underbrace{000 \cdots 000}_{k \text{ zeros}}) \rangle_{2^n} = A_0 000 \cdots 000$$

So

$$\boxed{\langle 2^k \cdot A \rangle_{2^n} = \text{result of left-shifting } A \text{ by } k \text{ bits zero-filling at the right}}$$

Example: Consider $n=8$ and $A = (a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0)_2$.
Then $\langle 2^3 \cdot A \rangle_{2^8} = (a_4 a_3 a_2 a_1 a_0 000)_2$.

B.2.5 Multiplication mod 2^n

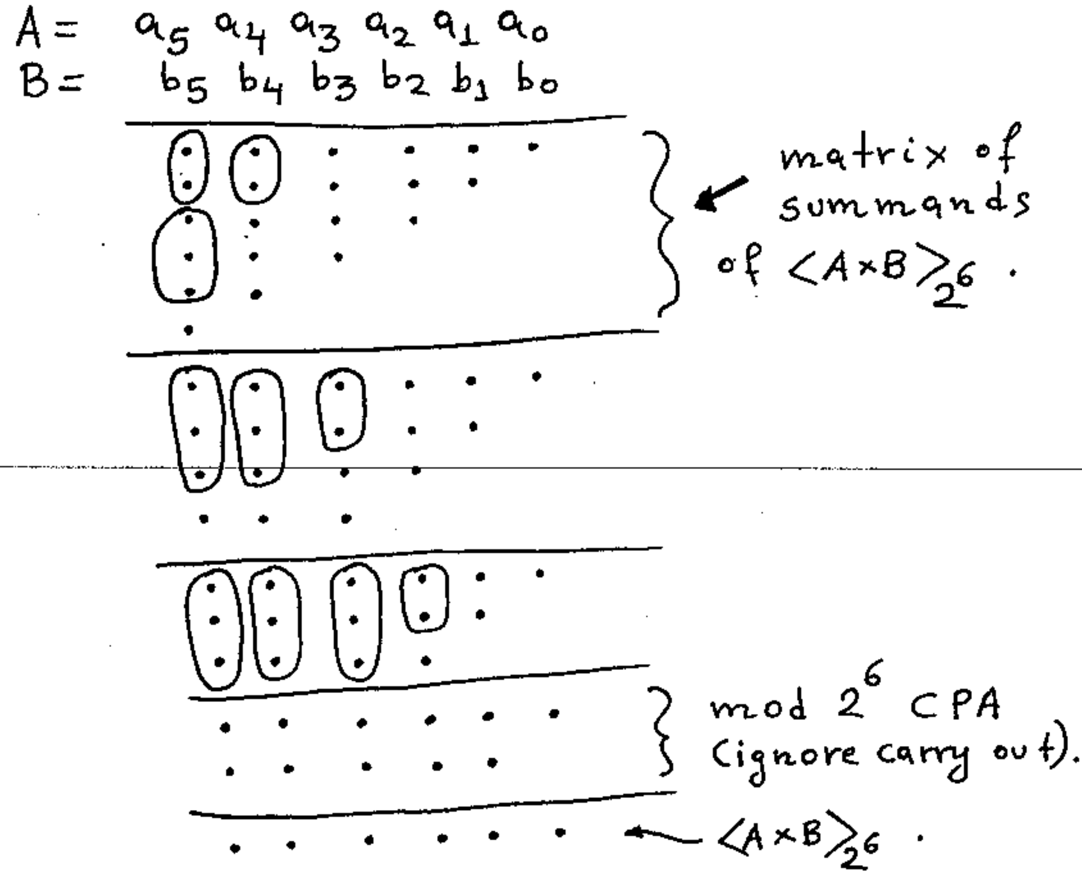
Let $A \in Z_{2^n}$, $B \in Z_{2^n}$ and suppose that $\langle A \times B \rangle_{2^n}$ needs to be computed. Let P be $P = A \times B$; (P is the full precision product of A and B). Then $P = \underbrace{P_1}_{\substack{n \\ \text{bit}}} \underbrace{P_0}_{\substack{n \\ \text{bit}}} = P_1 \cdot 2^n + P_0$.

Thus, $\boxed{\langle A \times B \rangle_{2^n} = \langle P \rangle_{2^n} = \langle P_1 \cdot 2^n + P_0 \rangle_{2^n} = P_0}$

The above approach which is based on computing the full precision product of A and B

and keeping its right most n-bit part P_0 is not the most hardware efficient approach. A much better direct design of a mod 2^n multiplier based on counters can be used. This is demonstrated by the following example.

Let A, B be 6-bit integers such that $A \in Z_{2^6}$, $B \in Z_{2^6}$ and suppose that $\langle A \times B \rangle_{2^6}$ needs to be computed. A counter-based implementation of $\langle A \times B \rangle_{2^6}$ is shown below. The counters used are (3,2) and (2,2) counters.



B.3 Arithmetic mod (2^n+1).

B.3.1 Addition mod (2^n+1)

Let A, B be integers such that $A \in Z_{2^n+1}$, $B \in Z_{2^n+1}$ where $Z_{2^n+1} = \{0, 1, 2, \dots, 2^n\}$. Then A and B need $n+1$ bits for their representation or $A = (a_n a_{n-1} \cdots a_1 a_0)_2$; $B = (b_n b_{n-1} \cdots b_1 b_0)_2$.

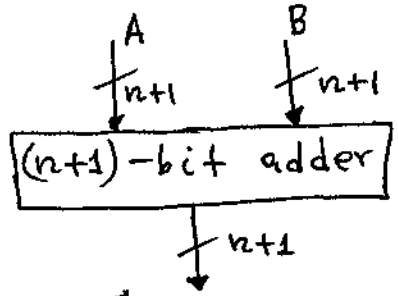
<u>Case (i)</u>: <u>$a_n = b_n = 1$</u>: In this case, $A = B = (1000\cdots00)_2 = 2^n$. Thus $\langle A+B \rangle_{2^n+1} = \langle 2^n + 2^n \rangle_{2^n+1}$ $= \langle (-1) + (-1) \rangle_{2^n+1} = \langle -2 \rangle_{2^n+1} = 2^n + 1 - 2 = 2^n - 1 =$ $= (0\underbrace{111\cdots111}_{n \text{ ones}})_2$. This case can be detected and treated separately.

<u>Case (ii)</u>: <u>At least one of a_n, b_n is zero:</u>

In this case, at least one of the two numbers A or B is smaller than 2^n.

~~Let S be the full precision summation of~~ A and B or $S = A + B$. Here, S is an $(n+1)$-bit number since $S_{max} = (A+B)_{max} = 2^n + (2^n - 1) =$ $= 2^{n+1} - 1$; (recall that not both numbers A, B are equal to 2^n).

The computation $S = A+B$ can be performed by a regular $(n+1)$-bit binary adder



$$S = (s_n s_{n-1} \cdots s_1 s_0)_2$$

As explained $0 \leq S \leq 2^{n+1} - 1$.

• If $0 \leq S \leq 2^n$ (which means that overflow did not occur or $S \in Z_{2^n+1}$), then

$$\langle A+B \rangle_{2^n+1} = S$$

•• If $S > 2^n$ then an overflow occured ($S \notin Z_{2^n+1}$) and in this case

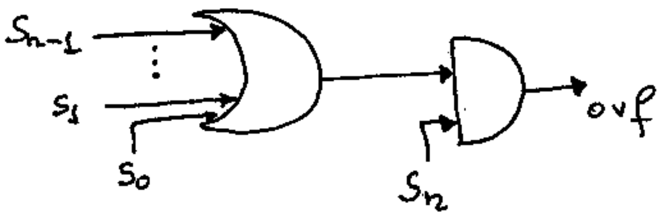
$$\langle A+B \rangle_{2^n+1} = \langle S \rangle_{2^n+1}.$$

The overflow is easily detected. Overflow occurs if

$$S = (s_n s_{n-1} \cdots s_1 s_0)_2 > 2^n.$$

This means:

$s_n = 1$ and not all $s_{n-1}, s_{n-2}, \cdots, s_1, s_0$ are zeros. As shown below the hardware for overflow detection consists of one OR and one AND gate



In this case of an overflow

$$2^n+1 \le S \le 2^{n+1}-1$$

or

$$2^n+1 \le S \le 2^n+2^n-1$$

Then

$$\langle A+B \rangle_{2^n+1} = \langle S \rangle_{2^n+1} = S-(2^n+1) =$$

$$= (s_n s_{n-1} \cdots s_1 s_0)_2 - (2^n+1) = 2^n + (s_{n-1} \cdots s_1 s_0)_2 - 2^n - 1$$

$$= (s_{n-1} \cdots s_1 s_0)_2 - 1 = (s_{n-1} \cdots s_1 s_0)_2 + 2\text{'s compl. of } 1 =$$

$$= (s_{n-1} s_{n-2} \cdots s_1 s_0)_2 + (111\cdots111)_2$$ (ignore carry out of this addition).

Thus, in this case of $S > 2^n$, the

desired $\langle A+B \rangle_{2^n+1}$ can be obtained by adding $(s_{n-1} s_{n-2} \cdots s_1 s_0)_2$ with a vector consisting of n ones and keeping the right most n bits of the result.

B.3.2 Additive inverse mod (2^n+1)

Let A be an integer such that $A \in Z_{2^n+1}$; $(Z_{2^n+1} = \{0, 1, 2, \ldots, 2^n\})$.

Then $A = (a_n a_{n-1} \cdots a_1 a_0)_2$.

Obviously

$$\langle -A \rangle_{2^n+1} = \langle 2^n + 1 - A \rangle_{2^n+1} .$$

Case (i): $A = 0$

In this case $\langle -A \rangle_{2^n+1} = 0$. This case can be detected and treated separately.

Case (ii): $A = 2^n$

Here, $\langle -A \rangle_{2^n+1} = \langle 2^n + 1 - 2^n \rangle_{2^n+1} = 1$. This case can be detected and treated separately.

Case (iii): $0 < A < 2^n$

In this case A can be represented by n bits or $A = (a_{n-1} a_{n-2} \cdots a_1 a_0)_2$.

Then $\langle -A \rangle_{2^n+1} = \langle 2^n + 1 - A \rangle_{2^n+1} =$

$$= \langle 2^n + 1 - (a_{n-1} \cdots a_1 a_0)_2 \rangle_{2^n+1} =$$

$$= \langle 2^n - 1 - (a_{n-1} a_{n-2} \cdots a_1 a_0)_2 + 2 \rangle_{2^n+1} =$$

$$= \langle (\overline{a_{n-1}}\, \overline{a_{n-2}} \cdots \overline{a_1}\, \overline{a_0})_2 + 2 \rangle_{2^n+1}.$$

The fact that $A \neq 0$ implies that

$(\overline{a_{n-1}}\, \overline{a_{n-2}} \cdots \overline{a_1}\, \overline{a_0})_2 \leq 2^n - 2$ or

$(\overline{a_{n-1}}\, \overline{a_{n-2}} \cdots \overline{a_1}\, \overline{a_0})_2 + 2 \leq 2^n$. Thus,

$$\langle (\overline{a_{n-1}}\, \overline{a_{n-2}} \cdots \overline{a_1}\, \overline{a_0})_2 + 2 \rangle_{2^n+1} =$$

$$= (\overline{a_{n-1}}\, \overline{a_{n-2}} \cdots \overline{a_1}\, \overline{a_0})_2 + 2.$$

In conclusion, in this case where $0 < A < 2^n$,

$$\boxed{\langle -A \rangle_{2^n+1} = (\overline{a_{n-1}}\, \overline{a_{n-2}} \cdots \overline{a_1}\, \overline{a_0})_2 + 2}$$

which means that $\langle -A \rangle_{2^n+1}$ is the 1's compl. of A incremented by 2.

B.3.3 Subtraction mod (2^n+1)

Let A, B be such that $A \in Z_{2^n+1}$, $B \in Z_{2^n+1}$.
Obviously

$$\langle A - B \rangle_{2^n+1} = \langle A + \langle -B \rangle_{2^n+1} \rangle_{2^n+1}.$$

B.3.4 Scaling by power of two (scaling by 2^k)

Let A be such that $A \in Z_{2^n+1}$. Then $A = (a_n a_{n-1} \cdots a_1 a_0)_2$. The problem here is to compute $\langle 2^k \cdot A \rangle_{2^n+1}$.

Let A be decomposed into two blocks A_1 and A_0, where A_1 is a $(k+1)$-bit block while A_0 is a $(n-k)$-bit block.

Then

$$A = \underbrace{A_1}_{\substack{k+1\\ \text{bit}}}\underbrace{A_0}_{\substack{n-k\\ \text{bit}}} = A_1 \cdot 2^{n-k} + A_0$$

Thus, $\langle 2^k \cdot A \rangle_{2^n+1} = \langle 2^k \cdot (A_1 \cdot 2^{n-k} + A_0) \rangle_{2^n+1}$

$$= \langle A_1 \cdot 2^n + A_0 2^k \rangle_{2^n+1} = \langle -A_1 + A_0 \cdot 2^k \rangle_{2^n+1}.$$

B.3.5 Multiplication mod (2^n+1)

Let A and B be integers such that $A, B \in \mathbb{Z}_{2^n+1}$. Here $\langle A \times B \rangle_{2^n+1}$ needs to be computed. Let the binary representations of A and B be

$$A = (a_n a_{n-1} \cdots a_1 a_0)_2 \; ; \; B = (b_n b_{n-1} \cdots b_1 b_0)_2 .$$

Case (i): $a_n = b_n = 1$:

In this case $A = B = (100\cdots0)_2 = 2^n = \langle -1 \rangle_{2^n+1}$.

Thus $\langle A \times B \rangle_{2^n+1} = \langle (-1) \times (-1) \rangle_{2^n+1} = 1$.

This case can be detected and treated separately.

Case (ii): $a_n = 1$, $b_n = 0$:

Here $A = 2^n = \langle -1 \rangle_{2^n+1}$ and $\langle A \times B \rangle_{2^n+1} = \langle -B \rangle_{2^n+1}$. In this case the necessary hardware is just a negator mod (2^n+1).

Case (iii): $a_n = 0$, $b_n = 1$:

Here $B = 2^n = \langle -1 \rangle_{2^n+1}$ and $\langle A \times B \rangle_{2^n+1} = \langle -A \rangle_{2^n+1}$.

Case (iv): $a_n = b_n = 0$

In this case the integers A and B are n-bit numbers with binary representations $A = (a_{n-1} a_{n-2} \cdots a_1 a_0)_2$; $B = (b_{n-1} \cdots b_1 b_0)_2$. Let P be the full precision product of A and B or $P = A \times B$. The product P is a $2n$-bit number and can be decomposed into two n-bit blocks as $P = P_1 P_0$. Obviously $P = P_1 \cdot 2^n + P_0$ and thus

$$\boxed{\begin{aligned} \langle A \times B \rangle_{2^n+1} &= \langle P \rangle_{2^n+1} = \langle P_1 \cdot 2^n + P_0 \rangle_{2^n+1} \\ &= \langle -P_1 + P_0 \rangle_{2^n+1} . \end{aligned}}$$

The above equation suggests that a possible implementation of $\langle A \times B \rangle_{2^n+1}$ could rely on an n-by-n full-precision multiplier computing $A \times B = P = P_1 P_0$, followed by a mod (2^n+1) subtractor performing $\langle -P_1 + P_0 \rangle_{2^n+1}$.

Note:

A more efficient way for performing arithmetic mod (2^n+1) is by using diminished-1 representations of numbers. Few references on this subject follow:

References

[1] L.M. Leibowitz, "A simplified binary arithmetic for the fermat number transform", IEEE Trans. Acoustics, Speech, and Signal Processing, vol. ASSP-24, no. 5, pp. 356-359, oct. 1976.

[2] Z. Wang, G.A. Jullien and W.C. Miller, "An algorithm for multiplication modulo (2^n+1)", in Proceedings of 29th Asilomar Conference on Signals, Systems, and Computers, (Pacific Grove, CA, Oct. 1995), pp. 956-960.

EE 7715 ① g

A. The Chinese Remainder Theorem (CRT) for the attractive 3-moduli RNS.

Consider the RNS based on the set

$S = \{m_1, m_2, m_3\} = \{2^n - 1, 2^n, 2^n + 1\}$; n integ.

Let the integer X be represented in the RNS as $X \xrightarrow{RNS} (X_1, X_2, X_3)$

Then the CRT reconstructs X from its residues as

$$X = \left\langle \langle X_1 N_1 \rangle_{m_1} M_1 + \langle X_2 N_2 \rangle_{m_2} M_2 + \langle X_3 N_3 \rangle_{m_3} M_3 \right\rangle_M$$

In the above $M = m_1 \times m_2 \times m_3$;

$M_1 = \frac{M}{m_1} = m_2 m_3 = 2^n(2^n+1)$;

$$N_1 = \langle M_1^{-1} \rangle_{m_1} = \langle [2^n(2^n+1)]^{-1} \rangle_{2^n-1} = \langle (1\times 2)^{-1} \rangle_{2^n-1}$$

$$= \langle 2^{-1} \rangle_{2^n-1} = \frac{2^n-1+1}{2} = 2^{n-1}$$

- In the above we used the fact that if m is an odd integer then $\langle 2^{-1} \rangle_m = \frac{m+1}{2}$; (double check to see that $\langle 2 \times \frac{m+1}{2} \rangle_m = \langle m+1 \rangle_m = 1$).

② g

$$M_2 = \frac{M}{m_2} = m_1 m_3 = (2^n-1)(2^n+1)$$

$$N_2 = \langle M_2^{-1} \rangle_{m_2} = \langle [(2^n-1)(2^n+1)]^{-1} \rangle_{2^n} = \langle [(-1)(1)]^{-1} \rangle_{2^n}$$

$$= \langle (-1)^{-1} \rangle_{2^n} = \langle -1 \rangle_{2^n}$$

$$M_3 = \frac{M}{m_3} = m_1 \times m_2 = (2^n-1)2^n$$

$$N_3 = \langle M_3^{-1} \rangle_{m_3} = \langle [(2^n-1)2^n]^{-1} \rangle_{2^n+1} = \langle [(-2)(-1)]^{-1} \rangle_{2^n+1}$$

$$= \langle 2^{-1} \rangle_{2^n+1} = \frac{2^n+1+1}{2} = 2^{n-1}+1$$

Then the CRT gives

$$\boxed{X = \left\langle \langle X_1 2^{n-1} \rangle_{2^n-1} \times 2^n(2^n+1) + \langle -X_2 \rangle_{2^n} (2^{2n}-1) + \langle X_3 (2^{n-1}+1) \rangle_{2^n+1} (2^n-1)2^n \right\rangle_{2^{3n}-2^n}}$$

B. Computation of the mixed radix digits for the attractive 3-moduli RNS.

Consider again the RNS based on the set $S = \{m_1, m_2, m_3\} = \{2^n - 1,\ 2^n,\ 2^n + 1\}$. Let the RNS representation of the integer X be $X \xrightarrow{RNS} (X_1, X_2, X_3)$ where X_1, X_2, X_3 are the residues of X.

The Mixed Radix formula is

$$X = X_1' + m_1 X_2' + m_1 m_2 X_3'$$

where X_1', X_2', X_3' are the mixed radix digit.

We have already shown that

$$\boxed{X_1' = X_1} \quad (1)$$

$$\boxed{X_2' = \left\langle m_1^{-1} (X_2 - X_1) \right\rangle_{m_2}} \quad (2)$$

$$\boxed{X_3' = \left\langle (m_1 m_2)^{-1} (X_3 - X_1 - m_1 X_2') \right\rangle_{m_3}} \quad (3)$$

Considering now that $m_1 = 2^n - 1$, $m_2 = 2^n$, $m_3 = 2^n + 1$ one gets

$$X_2' = \langle m_1^{-1}(X_2 - X_1)\rangle_{m_2} = \langle (2^n-1)^{-1}(X_2 - X_1)\rangle_{2^n}$$

$$= \langle (-1)^{-1}(X_2 - X_1)\rangle_{2^n} = \langle (-1)(X_2 - X_1)\rangle_{2^n}$$

$$= \langle X_1 - X_2 \rangle_{2^n}$$

or

$$\boxed{X_2' = \langle X_1 - X_2 \rangle_{2^n}} \qquad (4)$$

The computation dictated by eq (4) is nothing more than $X_2' = X_1 + 2$'s complement of X_2.

Regarding X_3' one gets from eq. (3)

$$X_3' = \langle (m_1 m_2)^{-1}(X_3 - X_1 - m_1 X_2')\rangle_{m_3}$$

$$= \langle [(2^n-1)2^n]^{-1}(X_3 - X_1 - (2^n-1)X_2')\rangle_{2^n+1}$$

$$= \langle [(-2)(-1)]^{-1}(X_3 - X_1 - (-1-1)X_2')\rangle_{2^n+1}$$

$$= \langle 2^{-1}(X_3 - X_1 + 2X_2')\rangle_{2^n+1}$$

$$= \langle 2^{-1}(X_3 - X_1) + X_2' \rangle_{2^n+1}$$

(5)g

$$= \langle (2^{n-1}+1)(X_3 - X_1) + X_2' \rangle_{2^n+1}$$

(recall that $\langle 2^{-1} \rangle_{2^n+1} = \frac{2^n+1+1}{2} = 2^{n-1}+1$).

So finally

$$\boxed{X_3' = \langle (2^{n-1}+1)(X_3 - X_1) + X_2' \rangle_{2^n+1}} \quad (5)$$

We'll now show how $\langle (2^{n-1}+1)(X_3 - X_1) \rangle_{2^n+1}$ can be computed. Let S be

$S = \langle X_3 - X_1 \rangle_{2^n+1}$. Let the binary representation of S be $S = (s_n s_{n-1} \cdots s_1 s_0)_2$

Then

$$\langle (2^{n-1}+1)(X_3 - X_1) \rangle_{2^n+1} = \langle (2^{n-1}+1)(s_n s_{n-1} \cdots s_1 s_0)_2 \rangle_{2^n+1}$$

$$= \langle (2^{n-1}+1)\left[(s_n s_{n-1} \cdots s_1) \times 2 + s_0 \right] \rangle_{2^n+1}$$

$$= \langle (2^n+2)(s_n s_{n-1} \cdots s_1) + (2^{n-1}+1) s_0 \rangle_{2^n+1}$$

$$= \langle (s_n s_{n-1} \cdots s_1) + (s_0 2^{n-1} + s_0) \rangle_{2^n+1}$$

$$= \langle (s_n s_{n-1} \cdots s_1) + (s_0 \underbrace{000 \cdots\cdots 000}_{n-2 \text{ zeros}} s_0) \rangle_{2^n+1}$$

$$= (s_n s_{n-1} \cdots s_1) + (s_0 \underbrace{000 \cdots\cdots 000}_{n-2 \text{ zeros}} s_0)$$

⑥g

It will now be shown that

$$\langle (s_n s_{n-1} \cdots s_1) + (s_0 \underbrace{000 \cdots\cdots 000}_{n-2 \text{ zeros}} s_0) \rangle_{2^n+1}$$

$$= (s_n s_{n-1} \cdots s_1)_2 + (s_0 \underbrace{000 \cdots\cdots 000}_{n-2 \text{ zeros}} s_0)$$

All that needs to be proven is that

$$(s_n s_{n-1} \cdots s_1) + (s_0 000 \cdots 000 s_0) < 2^n + 1.$$

Since $S = (s_n s_{n-1} \cdots s_1 s_0)_2 \in \mathbb{Z}_{2^n+1}$ (recall that $S = \langle X_3 - X_1 \rangle_{2^n+1}$) then $S \in [0 \quad 2^n]$.

If $s_n = 1$ then $s_{n-1} = s_{n-2} = \cdots = s_1 = s_0 = 0$ and in this case $(s_n s_{n-1} \cdots\cdots s_1) + (s_0 000 \cdots 000 s_0) =$

$$= (1 \underbrace{00 \cdots\cdots 000}_{n-1 \text{ zeros}}) + (00 \cdots 00) = 2^{n-1} < 2^n + 1.$$

If $s_n = 0$ then $(s_n s_{n-1} \cdots s_1) + (s_0 \underbrace{000 \cdots 000}_{n-2 \text{ zeros}} s_0) \leq$

$$(0 \underbrace{111 \cdots 111}_{n-1 \text{ ones}}) + (1 \underbrace{000 \cdots 000}_{n-2 \text{ zeros}} 1) = 2^{n-1} - 1 + 2^{n-1} + 1$$

$$= 2^n < 2^n + 1$$

Simplified arithmetic mod(2^n+1) using diminished-1 representations of numbers

In the handout entitled "The 3-moduli RNS based on set $S = \{2^n-1, 2^n, 2^n+1\}$", techniques for performing arithmetic mod(2^n+1) were offered; (see section B.3 starting on page 12).

A more efficient technique for performing arithmetic mod(2^n+1) is by using diminished-1 representations of numbers. This is the subject studied in this handout.

A. Code translation

Let A be an integer such that $A \in Z_{2^n+1}$ where $Z_{2^n+1} = \{0, 1, 2, \ldots, 2^n\}$; (the ring of integers $\bmod (2^n+1)$). Then A needs $n+1$ bits for its representation. The following shows the correspondence between normal and diminished-1 representations:

A		Diminished-1 of A or $A-1$
$0 = \overbrace{000\cdots000}^{n+1 \text{ bit}}$	$\longrightarrow$	$\overbrace{100\cdots000}^{n+1 \text{ bit}} = 2^n$
$1 = 000\cdots001$	$\longrightarrow$	$00\cdots000 = 0$
$2 = 000\cdots010$	$\longrightarrow$	$00\cdots001 = 1$
$3 = 000\cdots011$	$\longrightarrow$	$00\cdots010 = 2$
$4 = 000\cdots100$	$\longrightarrow$	$00\cdots011 = 3$
$\vdots$		
$2^n-1 = 011\cdots111$	$\longrightarrow$	$11\cdots110 = 2^n-2$
$2^n = 100\cdots000$	$\longrightarrow$	$11\cdots111 = 2^n-1$

As seen from the previous correspondence, the diminished-1 representation of the number zero is $10\ldots0 = 2^n$ and requires $n+1$ bits for its representation. The diminished-1 representation of the non-zero elements $1, 2, \cdots, 2^n-1, 2^n$ require only n bits for their representations.

$\boxed{B}$ Arithmetic mod (2^n+1)

When performing arithmetic mod (2^n+1) by using the diminished-1 system, all input-operands are in diminished-1 form and the results are returned in diminished-1 form as well.

B.1 Addition mod (2^n+1)

Let A, B be integers such that $A, B \in \mathbb{Z}_{2^n+1}$. If $A=B=0$ then $\langle A+B \rangle_{2^n+1} = 0$; if $A=0$ and $B \neq 0$ then $\langle A+B \rangle_{2^n+1} = B$; if $A \neq 0$ and $B=0$ then $\langle A+B \rangle_{2^n+1} = A$.

These cases (for which no addition is needed) can be detected and treated separately. The general case presented here is the case where $A \neq 0$, $B \neq 0$. The addition takes place as follows:

$$A \xrightarrow{\text{dim}-1} A-1$$
$$B \xrightarrow{\text{dim}-1} B-1$$

Add the n-bit numbers $A-1$ and $B-1$; complement (negate) the carry-out, end it around and add it back. The obtained result will be the diminished-1 form of $\langle A+B \rangle_{2^n+1}$.

Example 1: Perform $\langle A+B \rangle_{2^4+1}$ where $A = (10)_{10}$, $B = (13)_{10}$.

$A = 10 = (1010)_2 \xrightarrow{\text{dim}-1} (1001)_2 = A-1$

$B = 13 = (1101)_2 \xrightarrow{\text{dim}-1} (1100)_2 = B-1$

Adding $A-1$ and $B-1$ we get

$$\begin{array}{r} 1001 \\ 1100 \\ \hline 1\ \ 0101 \\ \llcorner\!\!\longrightarrow 0 \\ \hline 0101 \end{array} = 5 = \text{dim}-1 \text{ of } 6.$$

Double check to see that $\langle A+B \rangle_{2^4+1} = \langle 10+13 \rangle_{17} = 6$ and what we got is the diminished-1 form of 6.

Example 2: Perform $\langle A+B \rangle_{2^4+1}$ where $A = (8)_{10}$, $B = (7)_{10}$.

$A = 8 = (1000)_2 \xrightarrow{\text{dim}-1} 0111 = A-1$

$B = 7 = (0111)_2 \xrightarrow{\text{dim}-1} 0110 = B-1$

Adding A-1 and B-1 yields

$$\begin{array}{r} 0111 \\ 0110 \\ \hline 0\ \ 1101 \\ 1 \\ \hline 1110 \end{array} = 14 = \text{dim}-1 \text{ of } 15.$$

Double check to see that $\langle 8+7 \rangle_{17} = 15$ and what we got is the dim-1 form of 15.

Example 3: Perform $\langle A+B \rangle_{2^4+1}$ where $A = (8)_{10}$, $B = (9)_{10}$

Here $A = (8)_{10} = 1000 \xrightarrow{dim-1} 0111 = A-1$

$B = (9)_{10} = 1001 \xrightarrow{dim-1} 1000 = B-1$

$$
\begin{array}{rl}
A-1+B-1 = & 0111 \\
 & \underline{1000} \\
 & 0\,1111 \\
 & \;1 \\
 & \overline{1\,0000} = 2^4 = \text{dim-1 form of zero.}
\end{array}
$$

See that $\langle 8+9 \rangle_{17} = 0$

B.2 Additive inverse mod (2^n+1); (negation mod (2^n+1))

Let A belong to Z_{2^n+1}. If $A=0$ then $\langle -A \rangle_{2^n+1} = 0$; (no computation required; this case can be detected and treated separately). For the general case where $A \neq 0$ the computation $\langle -A \rangle_{2^n+1}$ occurs as follows:

$A \xrightarrow{dim-1} A-1$

Take the 1's complement of $A-1$. The obtained result will be the diminished-1 form of $\langle -A \rangle_{2^n+1}$.

Example 4: Compute $\langle -A \rangle_{2^4+1}$ where $A = (10)_{10}$.

Here $A = (10)_{10} = (1010)_2 \xrightarrow{\text{dim-1}} (1001)_2 = A-1$.

$\overline{A-1} = (0110)_2 = 6 =$ dim-1 form of 7.

Double check to see that $\langle -10 \rangle_{17} = 7$ and what we got is its dim-1 form.

B.3 Subtraction mod (2^n+1)

Obviously subtraction can be accomodated by a negation and an addition

B.4 Scaling by power of two (scaling by 2^k)

Let A belong to Z_{2^n+1}. If $A=0$ then $\langle 2^k A \rangle_{2^n+1} = 0$; (no computation required; this case can be detected and treated separately). For the general case where $A \neq 0$ the computation $\langle 2^k A \rangle_{2^n+1}$ takes place as follows:

$A \xrightarrow{\text{dim}-1} A-1$

Rotate the number $A-1$ <u>k bits</u> to the left where the bits ~~[illegible]~~ shifted out of the left end and shifted into the right end must be complemented (negated)

The obtained result will be the diminished-1 form of $\langle 2^k A \rangle_{2^n+1}$

<u>Example 5:</u> Compute $\langle A \times 2^3 \rangle_{2^4+1}$ where $A = (10)_{10}$.

Here $A = 10 = (1010)_2 \xrightarrow{\text{dim}-1} (1001)_2 = A-1$.

$1001 \xrightarrow[\substack{\text{compl.}\\ \text{etc}}]{\substack{\text{rot 3-bit}\\ \text{left}}} 1\bar{1}\bar{0}\bar{0} = (1011)_2 = 11 =$

= diminished-1 form of 12.

Double check to see that $\langle A \times 2^3 \rangle_{2^4+1} =$

$= \langle 10 \times 8 \rangle_{17} = 12$ and what we got is its dim-1 form.

Example 6:

Compute $\langle 2^3(A-B) + 2^4(C-D)\rangle_{2^5+1}$

where $A = 11$, $B = 6$, $C = 13$, $D = 9$.

Solution:

$A = (11)_{10} = (01011)_2 \xrightarrow{\text{dim}-1} A-1 = (01010)_2$

$B = (6)_{10} = (00110)_2 \xrightarrow{\text{dim}-1} B-1 = (00101)_2$

$C = (13)_{10} = (01101)_2 \xrightarrow{\text{dim}-1} C-1 = (01100)_2$

$D = (9)_{10} = (01001)_2 \xrightarrow{\text{dim}-1} D-1 = (01000)_2$

The additive inverses of B and D in diminished-1 form are

$\overline{B-1} = (11010)_2$ = 1's compl. of $B-1$

$\overline{D-1} = (10111)_2$ = " " " $D-1$

dim-1 of $\langle A-B\rangle_{2^5+1}$ =

$$\begin{array}{r} 01010 \\ +)\ 11010 \\ \hline 1\ 00100 \\ 0 \\ \hline 00100 \end{array}$$

(10) h

dim-1 of $\langle 2^3(A-B)\rangle_{2^5+1}$ is

$$00100 \xrightarrow[\text{compl. etc}]{\text{rotate 3-bit left;}} 00\bar{0}\bar{0}\bar{1} = 00110$$

dim-1 of $\langle C-D\rangle_{2^5+1}$ =

$$\begin{array}{r} 01100 \\ +)\ 10111 \\ \hline 100011 \\ \rightarrow 0 \\ \hline 00011 \end{array}$$

dim-1 of $\langle 2^4(C-D)\rangle_{2^5+1}$ is

$$00011 \xrightarrow[\text{compl. etc}]{\text{rot. 4-bit left;}} 1\bar{0}\bar{0}\bar{0}\bar{1} = 11110$$

Finally

dim-1 form of $\langle 2^3(A-B)+2^4(C-D)\rangle_{2^5+1}$

$$= \begin{array}{r} 00110 \\ +)\ 11110 \\ \hline 100100 \\ \rightarrow 0 \\ \hline \end{array}$$

$(00100)_2 = 4$ = dim-1 form of 5.

Double check to see that

$$\langle 2^3(A-B)+2^4(C-D)\rangle_{2^5+1} =$$

$$= \langle 2^3(11-6)+2^4(13-9)\rangle_{33} = \langle 8\times 5+16\times 4\rangle_{33} =$$

$= \langle 40+64\rangle_{33} = \langle 104\rangle_{33} = 5$ and what we got is (00100) which is dim-1 of 5.

- This composite example makes it clear that in composite calculations, after entering the diminished-1 system we stay in the diminished-1 system all the time until the final result is computed. For this particular example, the inputs A, B, C, D were translated into their diminished-1 forms, and after that we stayed inside the diminished-1 system because all the intermediate results were in diminished-1 form.

B.5 Multioperand addition mod (2^n+1)

Let $A_1, A_2, A_3, \ldots, A_k$ be integers belonging to Z_{2^n+1}. We are interested in computing $\langle A_1 + A_2 + \cdots + A_k \rangle_{2^n+1}$ using the dim-1 approach. If one or more numbers A_i are zero, they do not contribute to the summation. For the general case where $A_i \neq 0$, $\forall i = 1, 2, \ldots, k$ the computation $\langle \sum_{i=1}^{k} A_i \rangle_{2^n+1}$ takes place as follows:

$$A_1 \xrightarrow{\text{dim}-1} A_1 - 1$$

$$A_2 \xrightarrow{\text{dim}-1} A_2 - 1$$

$$A_3 \xrightarrow{\text{dim}-1} A_3 - 1$$

$$\vdots \qquad \vdots \qquad \vdots$$

$$A_k \xrightarrow{\text{dim}-1} A_k - 1$$

Add the k diminished-1 forms $A_1 - 1$, $A_2 - 1$, $A_3 - 1$, $\ldots$, $A_k - 1$ using a CSA tree followed by a CPA (a 2-operand adder). The CSA tree will be a minimum delay (minimum # of levels) tree and will reduce the k numbers (rows) down to two. The CPA will add the two vectors to produce the final result. The CSA tree will consist of (2,2), (3,2) counters or in general counters of arbitrary size (say (p, q) counters). Both the CSA tree as well as the CPA will use the scheme complement the carry-out and end it around.

The obtained result will be the dim-1 form of $\langle A_1 + A_2 + \cdots + A_k \rangle_{2^n+1}$.

Example 7: Compute $\langle A+B+C+D \rangle_{2^4+1}$ where

$A = 12$, $B = 11$, $C = 10$, $D = 8$.

Here

$A = 12 = (1100)_2 \xrightarrow{\text{dim}-1} A-1 = (1011)_2$

$B = 11 = (1011)_2 \xrightarrow{\text{dim}-1} B-1 = (1010)_2$

$C = 10 = (1010)_2 \xrightarrow{\text{dim-1}} C-1 = (1001)_2$

$D = 8 = (1000)_2 \xrightarrow{\text{dim-1}} D-1 = (0111)_2$

The counters used here will be (3,2) counters. The sequence of #s is 3, 4, 6, 9, ... and obviously two levels of reduction will be needed to perform the 4-to-2 reduction. Then a CPA will give the final result.

$$
\begin{array}{lccccc}
A-1 = & & 1 & 0 & 1 & 1 \\
B-1 = & & 1 & 0 & 1 & 0 \\
C-1 = & & 1 & 0 & 0 & 1 \\
D-1 = & & 0 & 1 & 1 & 1 \\
\hline
 & & 0 & 1 & 1 & 1 \\
 & & 0 & 1 & 1 & 0 \\
 & & 1 & 0 & 0 & 1 \\
\hline
 & & 1 & 1 & 0 & 1 \\
 & & 1 & 0 & 0 & 0 \\
\hline
 & 1 & 0 & 1 & 1 & 0 \\
 & & & & & 0 \\
\hline
 & & 0 & 1 & 1 & 0
\end{array}
\quad \text{CPA}
$$

$0110 = 6 = \text{dim-1 of } 7$

Double check to see that

$\langle A+B+C+D \rangle_{2^4+1} = \langle 12+11+10+8 \rangle_{17} = 7$ and what we got is its dim-1 form.

B.6 General multiplication mod (2^n+1)

An efficient multiplier-design for performing $\langle A \times B \rangle_{2^n+1}$ using the diminished-1 approach is offered in reference [2]. The interested reader can find more details on arithmetic mod (2^n+1) using the dim-1 approach in references [1], [2]

References:

[1] L.M. Leibowitz, "A simplified binary arithmetic for the fermat number transform", IEEE Trans. Acoustics, Speech, and Signal Processing, vol. ASSP-24, no. 5, pp. 356-359, Oct. 1976.

[2] Z. Wang, G. A. Jullien, and W. C. Miller, "An algorithm for multiplication modulo (2^n+1)", in Proceedings of 29th Asilomar Conference on Signals, Systems, and Computers, (Pacific Grove, CA, Oct. 1995), pp. 956-960.