



# Recursive Architectures for 2DLNS Multiplication

Mahzad Azarmehr

Supervisor: Dr. M. Ahmadi

Summer 2009



# Outline

- Multidimensional Logarithmic Number System (MDLNS)
  - Introduction
  - Definition
- Multiplication
  - Shift/Add Multiplication Algorithm
  - 1-digit 2DLNS Multiplication
  - 2-digit 2DLNS Multiplication
- Recursive Multiplication
  - One-level of Recursion
  - Two-level of Recursion
- Synthesis Results
- Conclusion

# MDLNS (Introduction)

- Desired characteristics of a number system used in DSP
  - Smaller size of corresponding representations
  - More error-free mapping approximations
  - Less complexity of arithmetic operations
  - More accurate representation of smaller values ( like less than one coefficient values in a filter )

## MDLNS (Definition)

- A representation of the real number  $X$ , in the form:

$$X = \sum_{i=1}^n s_i \prod_{j=1}^b p_j^{e_j^{(i)}}$$

- where  $s_i$  is sign  $(-1,0,1)$ ,  $p_j$  is a real, and  $e_j^{(i)}$  are integers, is called an  $n$  digit multi-dimensional logarithmic representation of  $X$

-  $b$  is the number of bases used (at least two) and the first one,  $p_1$ , is always be assumed to be 2

# MDLNS (Optimal Base)

- The proper second base (optimal base) should be selected in accordance to the specific design considerations
- Different bases as  $D$ , are used to find the minimal error or score between the set of values and their MDLNS approximations
- With optimal bases, error-free input data mapping can be significantly improved
- The value of optimal base doesn't affect the hardware structure
- Every MDLNS value is represented in hardware with its sign and corresponding exponents



# MDLNS (Properties)

- Larger dynamic range
- More degrees of freedom by virtue of having two or more orthogonal bases and the ability to use multiple digits
- A significant reduction in hardware complexity
- Simplified mathematical computation

# MDLNS (Arithmetic)

- Multiplication and Division

Given a single-digit representation of (one-bit sign):

$$x = \{s_x, a_x, b_x\} \text{ and } y = \{s_y, a_y, b_y\}$$

$$x \cdot y = \{s_x \text{ xor } s_y, a_x + a_y, b_x + b_y\}$$

$$x \div y = \{s_x \text{ xor } s_y, a_x - a_y, b_x - b_y\}$$

## MDLNS (Arithmetic)

- Addition and Subtraction

$$2^{a_x} \cdot D^{b_x} + 2^{a_y} \cdot D^{b_y} = (2^{a_x} \cdot D^{b_x}) \cdot (1 + 2^{a_y - a_x} \cdot D^{b_y - b_x}) \\ \approx (2^{a_x} \cdot D^{b_x}) \cdot \Phi(a_y - a_x, b_y - b_x)$$

$$2^{a_x} \cdot D^{b_x} - 2^{a_y} \cdot D^{b_y} = (2^{a_x} \cdot D^{b_x}) \cdot (1 - 2^{a_y - a_x} \cdot D^{b_y - b_x}) \\ \approx (2^{a_x} \cdot D^{b_x}) \cdot \Psi(a_y - a_x, b_y - b_x)$$

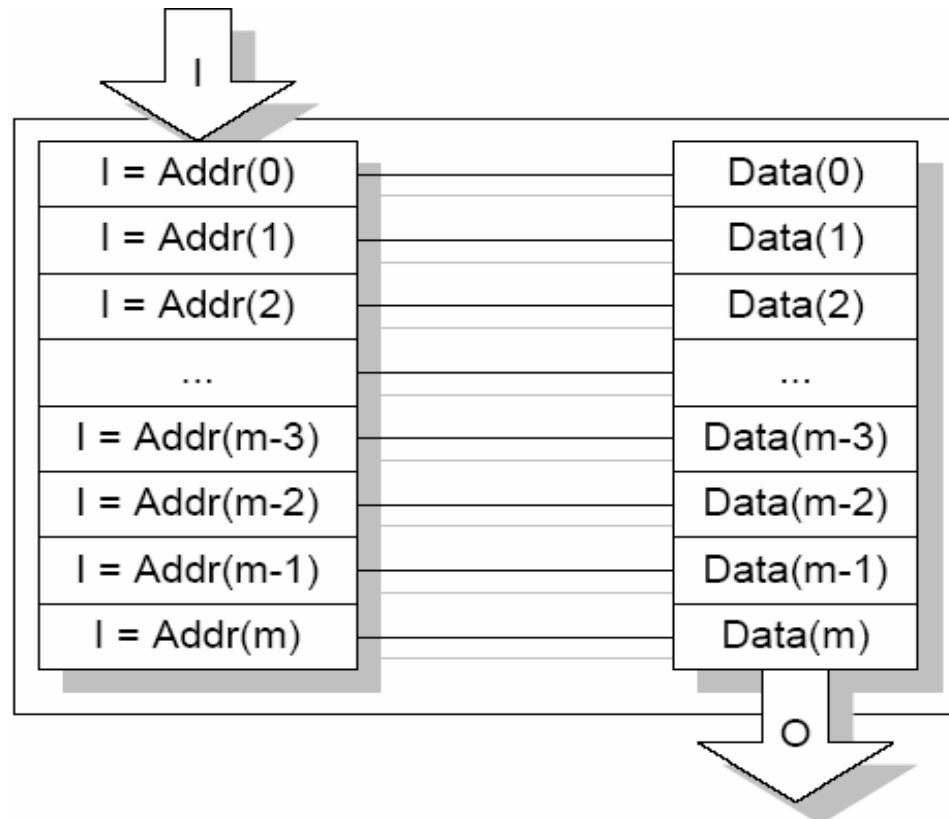
The operators  $\Phi$  and  $\Psi$  are lookup tables that store the pre-computed 2DLNS values

# MDLNS (Conversion)

- There is no functional relationship between Binary and MDLNS representations
- Conversions between Binary and MDLNS representations are efficiently implemented with Range Addressable Look-up Tables ( RALUT )
- The virtue of using multiple digits makes appropriate size of RALUTs reasonably small

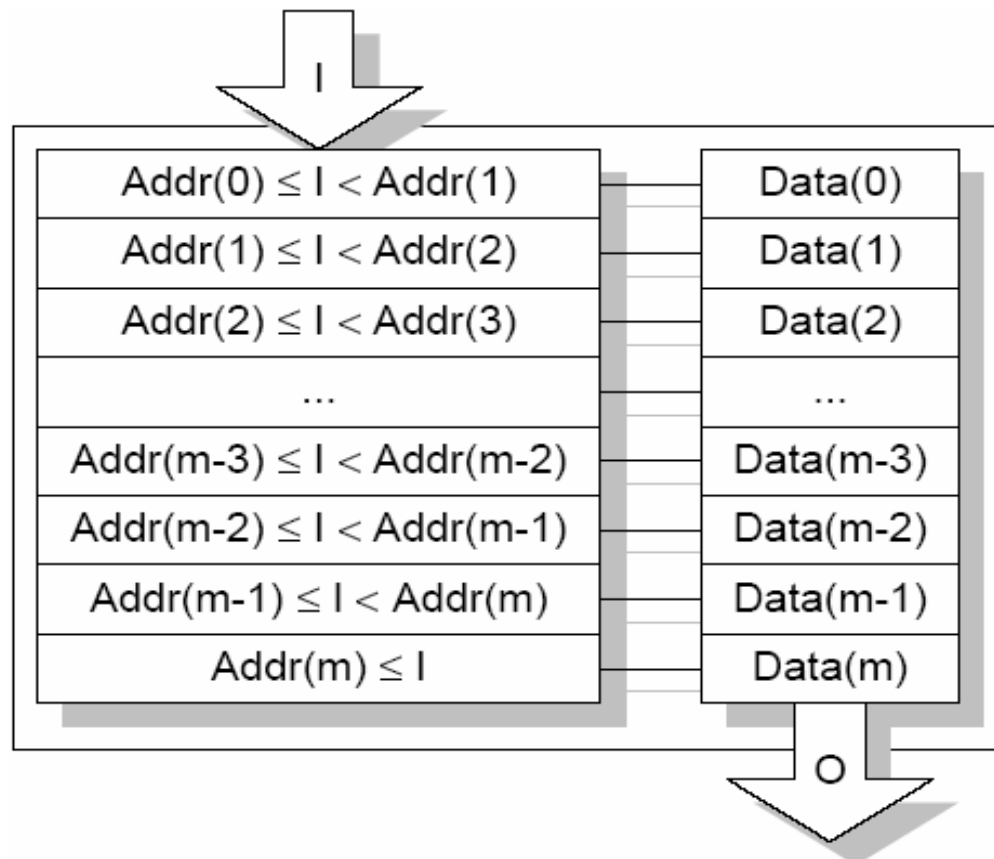
# Standard Address Decode

- LUT does not scale well and is redundant, and therefore can be modified by changing the address decode system from exact matching to range matching



# RALUT Structure

- A RALUT differs from the classic LUT by changing the address decoder system to match on a range of values rather than exact values



# RALUTs in MDLNS

- Most of the designs which are used in MDLNS circuits can be efficiently implemented using RALUTs
- The RALUT is optimal for 2DLNS conversion architectures as its size mainly depends on the value of  $R$  (The number of bits to represent second base index); this RALUT has  $2^R + 1$  rows
- MDLNS addition/subtraction is also can be implemented using RALUTs

# Multiplication

- Multiplication is a heavily used arithmetic operation that figures prominently in signal processing and scientific applications
- Multiplication is hardware intensive, and the main criteria of interest are higher speed, lower power, and less VLSI area
- The main concern in classic multiplication, often realized by  $K$  cycles of shifting and adding, is to speed up the underlying multi-operand addition of partial products
- Using MDLNS, multiplication can be replaced by parallel small adders

# Multiplication

- Based on some previous work, considering  $B = 6$  and  $R = 5$  provides most error free mapping of 16-bit binary data, while for 32-bit binary data,  $B = 11$  and  $R = 9$  have been considered
- In order to achieve the desired precision, 2-digit 2DLNS representations are considered
- The optimal base for each 2DLNS representation has been computed. The optimal base for 24-bit architecture obtained as  $D = 0.92024380912663017$ , and for 42-bit representation calculated as  $D = 0.870789850126489$

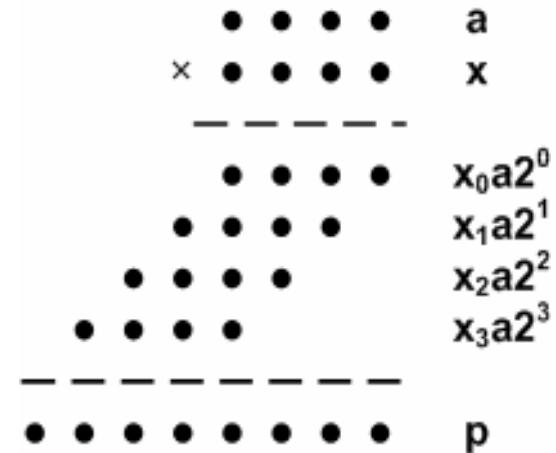
# Shift/Add Multiplication Algorithm

- With the following notation:

a Multiplicand  $a_{k-1}a_{k-2}\dots a_1a_0$

x Multiplier  $x_{k-1}x_{k-2}\dots x_1x_0$

p Product  $p_{2k-1}p_{2k-2}\dots p_1p_0$

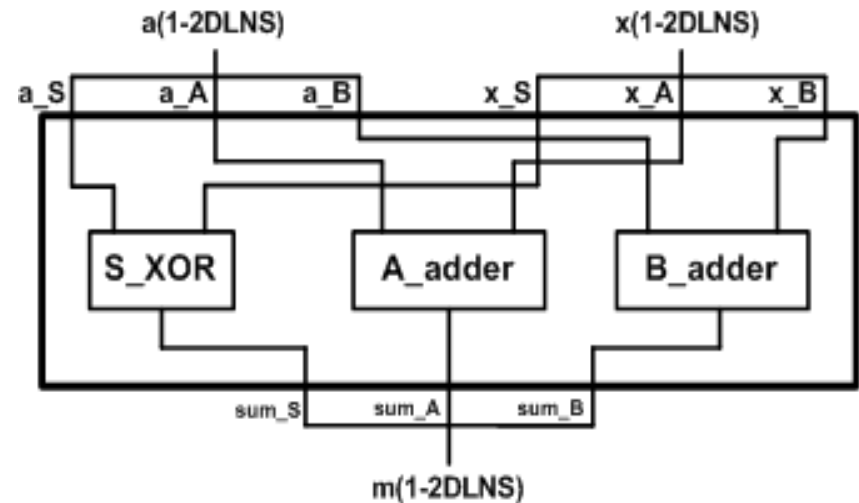


Each row corresponds to the product of the multiplicand and a single bit of multiplier. Each term is either 0 or a

- Binary multiplication reduces to adding a set of numbers, each of which is 0, or shifted version of the multiplicand a

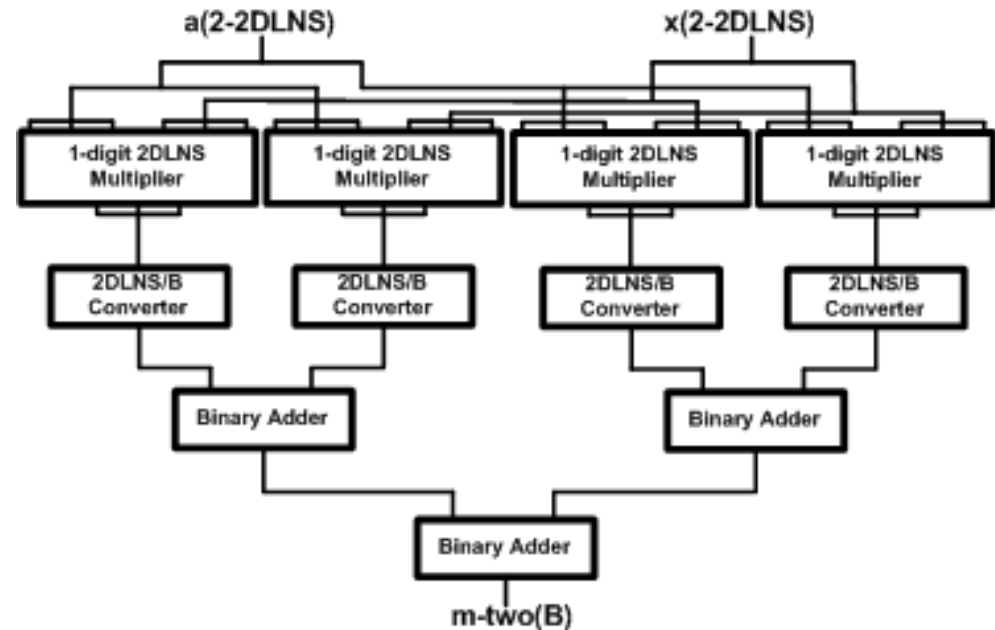
# 1-digit 2DLNS Multiplication

- Two parallel adders determine the corresponding exponents of the product
- Since 1-bit sign representations are assumed, 0 for positive and 1 for negative sign, an XOR gate specifies the sign of product



## 2-digit 2DLNS Multiplication

- Each number is represented with a summation of 2 digits, in HW is shown with a set of sign, first and second base exponents
- There are four partial products that should be summed up to form the final result



# Recursive Multiplier (One-level of Recursion)

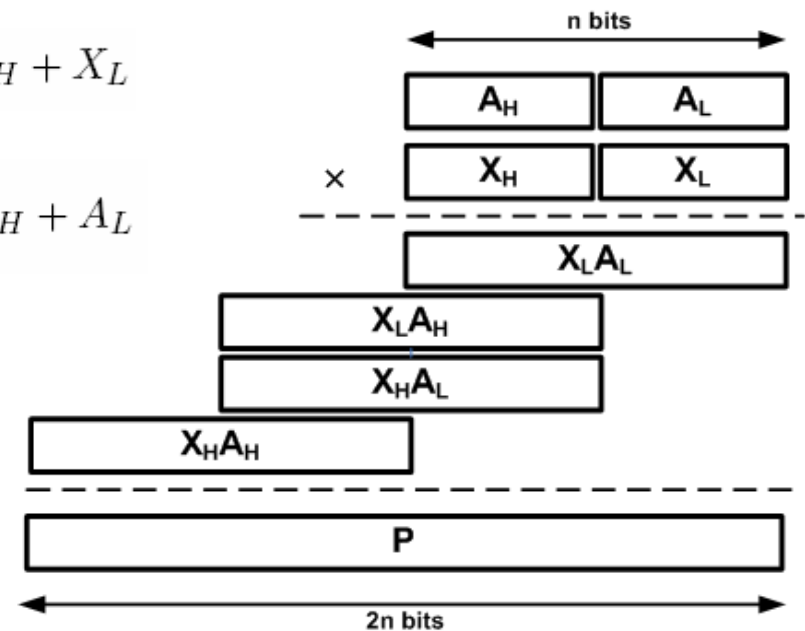
$$X = \sum_{k=0}^{n-1} x_k \cdot 2^k = \sum_{k=0}^{n/2-1} x_k \cdot 2^k + \sum_{k=n/2}^{n-1} x_k \cdot 2^k = X_H + X_L$$

$$A = \sum_{k=0}^{n-1} a_k \cdot 2^k = \sum_{k=0}^{n/2-1} a_k \cdot 2^k + \sum_{k=n/2}^{n-1} a_k \cdot 2^k = A_H + A_L$$

$$P = A \cdot X$$

$$= (A_H + A_L) \cdot (X_H + X_L)$$

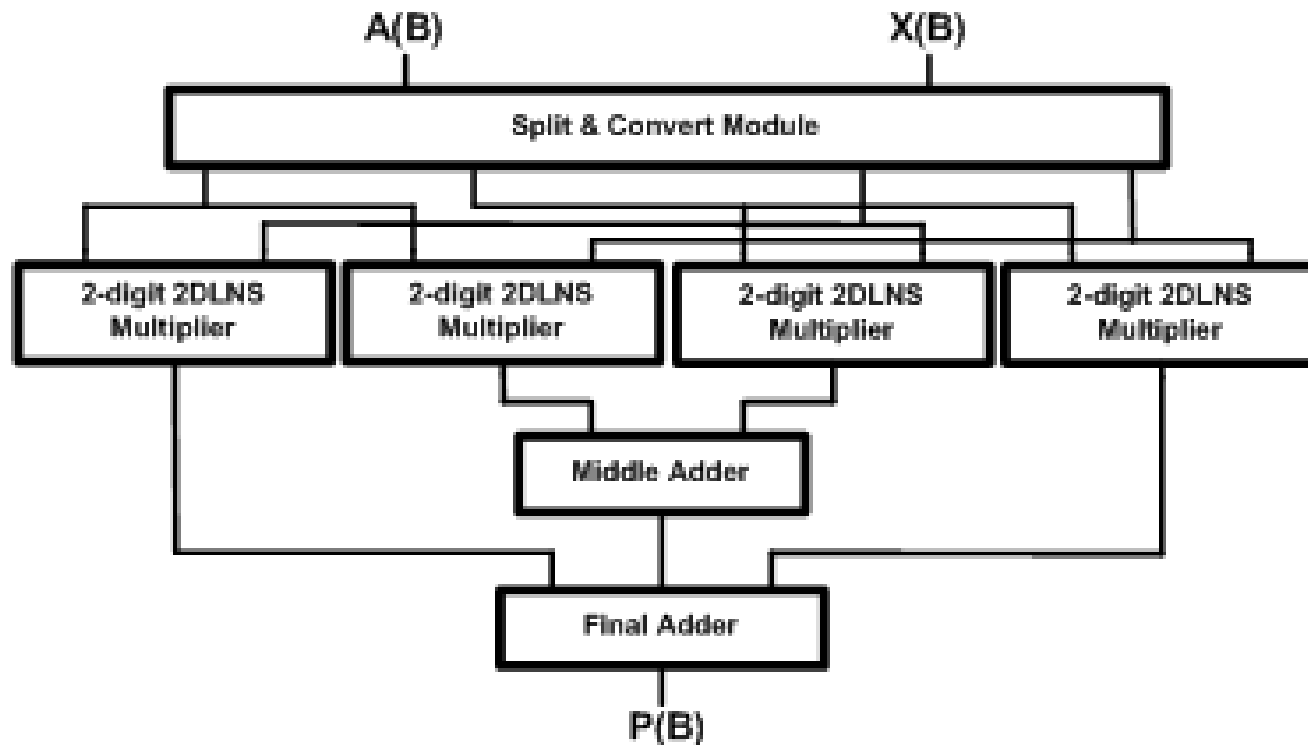
$$= X_H \cdot A_H + X_H \cdot A_L + X_L \cdot A_H + X_L \cdot A_L$$



- For n×n multiplication one can use n/2-bit adders exclusively to accumulate the partial products

# Recursive Multiplier (One-level of Recursion)

- Applying one level of recursion to a  $64 \times 64$  bit multiplier lead to having four number of base modules which are 2DLNS equivalents to  $32 \times 32$  bit multipliers



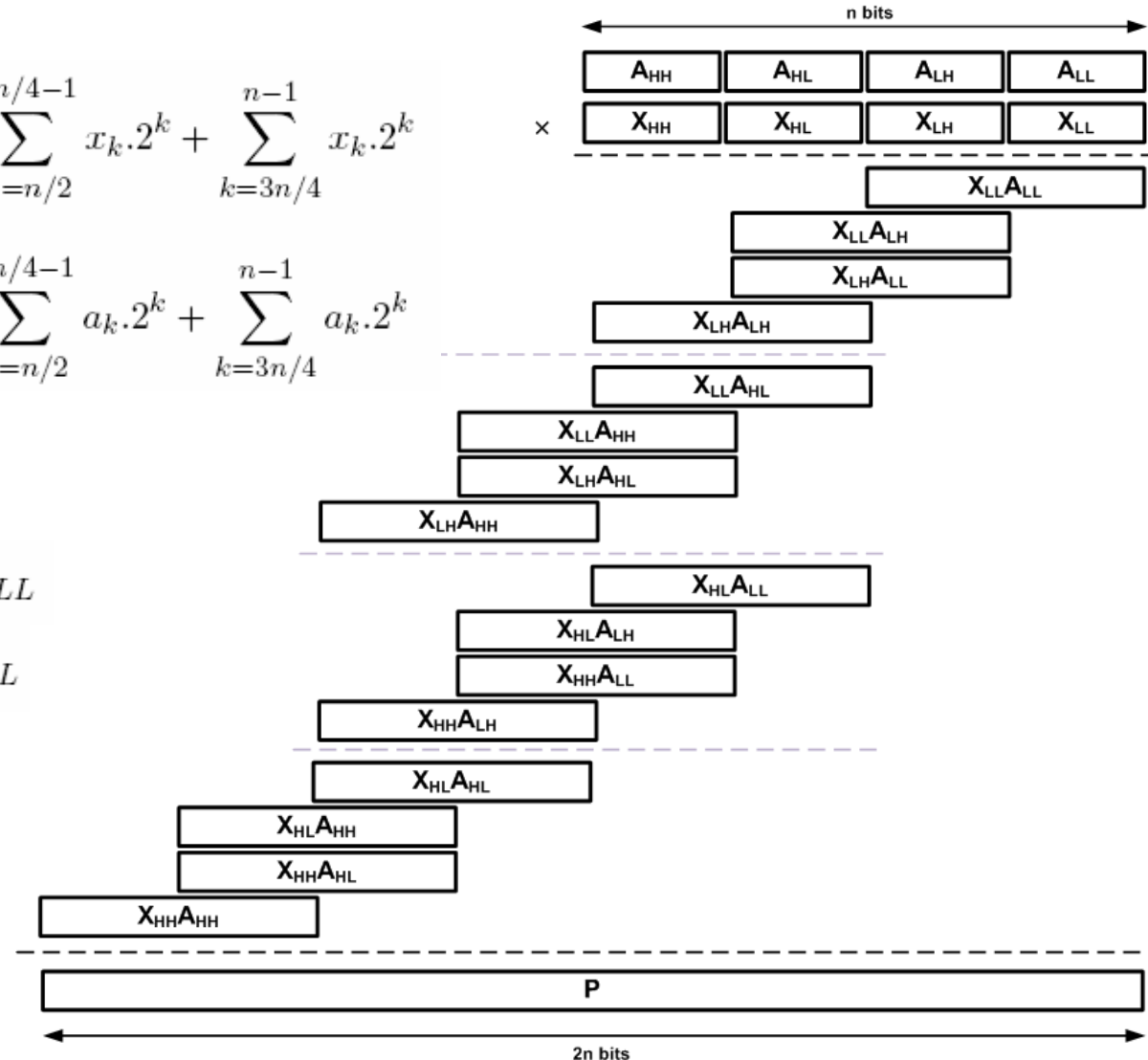
# Recursive Multiplier (Two-level of Recursion)

$$X = \sum_{k=0}^{n/4-1} x_k \cdot 2^k + \sum_{k=n/4}^{n/2-1} x_k \cdot 2^k + \sum_{k=n/2}^{3n/4-1} x_k \cdot 2^k + \sum_{k=3n/4}^{n-1} x_k \cdot 2^k$$

$$A = \sum_{k=0}^{n/4-1} a_k \cdot 2^k + \sum_{k=n/4}^{n/2-1} a_k \cdot 2^k + \sum_{k=n/2}^{3n/4-1} a_k \cdot 2^k + \sum_{k=3n/4}^{n-1} a_k \cdot 2^k$$

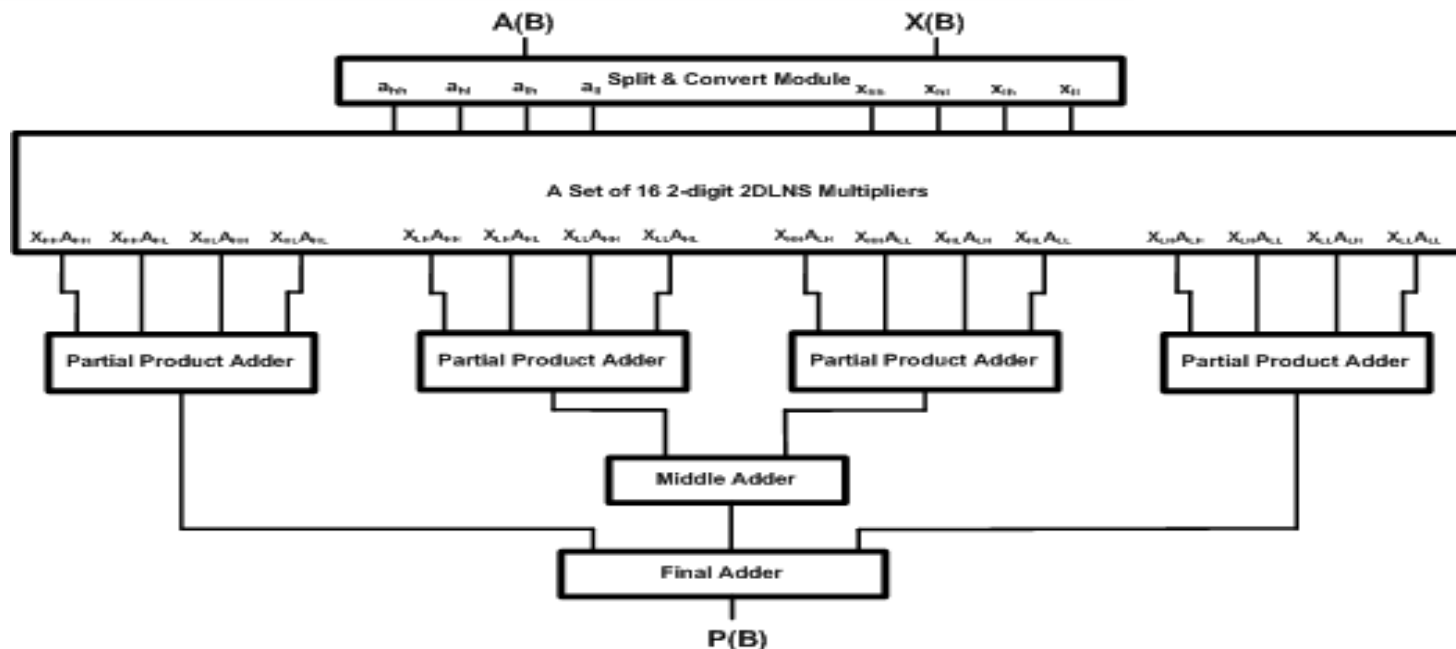
$$X = X_{HH} + X_{HL} + X_{LH} + X_{LL}$$

$$A = A_{HH} + A_{HL} + A_{LH} + A_{LL}$$



# Recursive Multiplier (Two-level of Recursion)

$$\begin{aligned}
 P &= A.X \\
 &= (A_{HH} + A_{HL} + A_{LH} + A_{LL}).(X_{HH} + X_{HL} + X_{LH} + X_{LL}) \\
 &= X_{HH}.A_{HH} + X_{HH}.A_{HL} + X_{HH}.A_{LH} + X_{HH}.A_{LL} \\
 &+ X_{HL}.A_{HH} + X_{HL}.A_{HL} + X_{HL}.A_{LH} + X_{HL}.A_{LL} \\
 &+ X_{LH}.A_{HH} + X_{LH}.A_{HL} + X_{LH}.A_{LH} + X_{LH}.A_{LL} \\
 &+ X_{LL}.A_{HH} + X_{LL}.A_{HL} + X_{LL}.A_{LH} + X_{LL}.A_{LL}
 \end{aligned}$$



# Synthesis Results

Architecture 64 × 64 bit	2DLNS-based Proposed-1	2DLNS-based Proposed-1 32 × 32 bit	2DLNS-based Proposed-2	Binary-based [1]	Binary-based Synopsys
Level of Recursion	One	One	Two	One	None
Technology	90nm	90nm	90nm	0.18 $\mu$ m	90nm
Clock Frequency (MHz)	233	256	233	200	-
Overall Area ( $\mu$ m) <sup>2</sup>	24,041,537.19	936,852.64	3,788,860.25	360,417.44	64,308.38
Dynamic Power (mW)	1,412.51	5.20	6.01	582.31	62.72
Data Arrival Time (ns)	57.95	3.60	3.99	8.51	35.52

[1] Mokrian, P.; "A Reconfigurable Digital Multiplier Architecture", M.A.Sc. Thesis, University of Windsor, 2003

# Related Works

- The recursive multiplication schemes can be implemented in reconfigurable architectures due to their flexibility, performance, speed, and power consumption
- The 2DLNS/Binary conversions can be improved in terms of using more efficient RALUTs and hardware features
- 2DLNS addition/subtraction structures can be implemented in order to avoid data conversion
- Examining potential application in multiplication intensive algorithms in a variety of data widths
- The parallel and series architectures can be substituted to provide delay and area requirements

# Conclusion

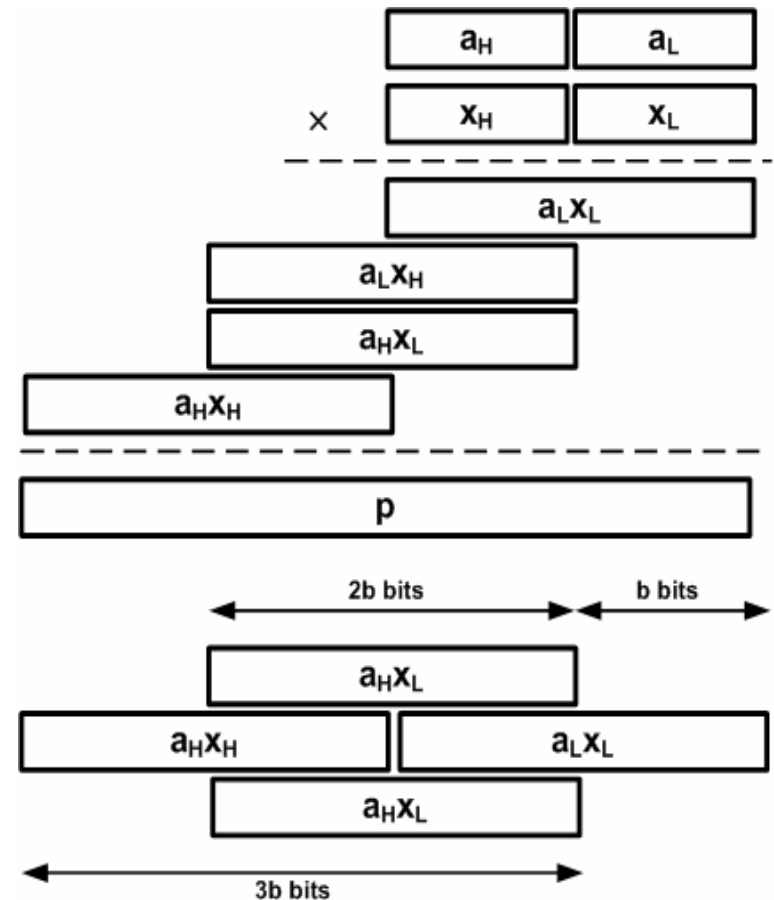
- These architectures benefit the 2DLNS properties as well as recursive multiplication that lead to low-power and high speed designs
- Multiple level of recursion can be applied to make the size of converters reasonably small
- These architectures provide optimistic context to implement high-performance reconfigurable digital signal processors



# Questions and Comments

# Divide and Conquer Design

- A  $2b \times 2b$  multiplier can be synthesized using  $b \times b$  multiplier
- Although there are four partial products, only three values need to be added
- $2b \times 2b$  multiplication has been reduced to 4  $b \times b$  multiplications and a three-operand addition





# Divide and Conquer Design

- For  $2b \times 2b$  multiplication one can use  $b$ -bit adders exclusively to accumulate the partial products

