NEW APPROACH OF SIGNED BINARY NUMBERS MULTIPLICATION AND ITS IMPLEMENTATION ON FPGA

[1]Sarifuddin Madenda, [2]Suryadi Harmanto
[1,2]Fakultas Teknologi Industri, Universitas Gunadarma
Jl. Margonda Raya No. 100, Depok 16424, Jawa Barat
[1]sarif@staff.gunadarma.ac.id, [2]misdie@staff.gunadarma.ac.id

Abstract

This paper proposes a new model of signed binary multiplication. This model is formulated mathematically and can handle four types of binary multipliers: signed positive numbers multiplied by signed positive numbers (SPN-by-SPN); signed positive numbers multiplied by signed negative numbers (SPN-by-SNN); signed negative numbers multiplied by signed positive numbers (SNN-by-SPN); and signed negative numbers multiplied by signed negative numbers (SNN-by-SNN). The proposed model has a low complexity algorithm, is easy to implement in software coding and integrated in a hardware FPGA (Field-Programmable Gate Array), and is more powerful compared to the modified Baugh-Wooley's model.

Keywords*: Signed binary numbers, signed multiplication, algorithm, multiplier circuit, FPGA*

INTRODUCTION

Currently, automation technology related to multimedia data analysis and processing based on artificial intelligence continues to be developed. The use of Convolutional Neural Network (CNN), one of the artificial intelligence methods, in the field of image processing is constantly expanding: biometrics recognition for personal identification [1], image recognition [2] [3], autonomous vehicles [4], medical diagnostics [5], and so on. In [6] proposes the implementation of CNN architecture in FPGA (Field-Programmable Gate Array) based on mapping and pipeline implementation methods on all its layers. A convolutional method using Sobel kernels in the convolutional layer of CNN and its hardware implementation on FPGA was proposed in [7]. Other implementation methods of acceleration in the deep learning network on FPGA are discussed in [8].

The multiplication and division operations are a major part of data processing algorithms in AI: Machine learning, CNN, and Deep learning. In [9] proposes the Stochastic Computing Multiplier method that is applied to the implementation of Deep Convolutional Neural Networks, where the embedment of each perceptron in the NN layer performs the same number of mathematical operations (additions, products, and threshold functions) [10]. Xilinx [11] is also developing an implementation model for the FPGA Acceleration of Matrix Multiplication for artificial neural networks. The implementation of all AI algorithms into the SoC (FPGA and ASIC) is often constrained by the implementation of multiplication and division operations. The

constraint in question not only concerns the amount of space occupation on the integrated circuits (IC) that is related to production costs, but also the complexity of its implementation method. The multiplication model depends on the type of variable values used for the multiplicand and multiplier. The values of both variables can be unsigned and unsigned numbers or signed and unsigned numbers or unsigned and signed numbers or signed and signed numbers, respectively. Signed binary numbers mean that both positive and negative numbers may be represented. The most significant bit (MSB) indicates the sign, where bit sign "0" for signed positive number (*SPN*) and "1" for signed negative number (*SNN*). Unsigned binary numbers (*UNS)* refer to the numbers that only have a positive value without a sign bit.

Signed binary multiplication is one of the multiplications that is still a part of research topics. Particularly how to develop the implementation methods with low complexity, low-cost hardware implementation, low-power consumption, and faster. Signed binary multiplication was introduced by Baugh-Wooley [12] and then modified into two's complement multiplication, also known as modified Baugh-Wooley multiplication [13]. Mathematically, the two multiplication models are given in equations (1) and (2). Figures 1 and 2 show their shift-and-add or matrix structure for n = 4. Noted that the two's complement multiplication (equation 2) has a limitation, it only applies to *SNN-by-SNN* multiplication. Equation (1) can process four types of multiplication: *SPN-by-SPN*, *SPN-by-SNN*, *SNN-by-SPN*, and *SNN-by-SNN*, but it needs 3 additional full adders (FA: gray color), so there is an increase in cost and time delay. This is especially impactful when used in algorithms that require tens or hundreds of multipliers such as in CNN. This paper is focused on modifying equation (1), so it has a low complexity algorithm and low-cost hardware implementation.

$$Y = -2^{2n-1} + (a_{n-1}b_{n-1} + \bar{a}_{n-1} + \bar{b}_{n-1})2^{2n-2} + (a_{n-1} + b_{n-1})2^{n-1} + \sum_{i=0}^{n-2} \bar{a}_i b_{n-1} 2^{i+n-1} + \sum_{j=0}^{n-2} a_{n-1}\bar{b}_j 2^{j+N-1} + \sum_{j=0}^{n-2} \sum_{i=0}^{n-2} a_i b_j 2^{i+j} \quad (1)$$

$$Y = 2^{2n-1} + a_{n-1}b_{n-1}2^{2n-2} + 2^n + \sum_{i=0}^{n-2} \overline{b_{N-1}a_i} 2^{i+n-1} + \sum_{j=0}^{n-2} \overline{b_j a_{n-1}} 2^{j+n-1} + \sum_{j=0}^{n-2} \sum_{i=0}^{n-2} a_i b_j 2^{i+j} \quad (2)$$

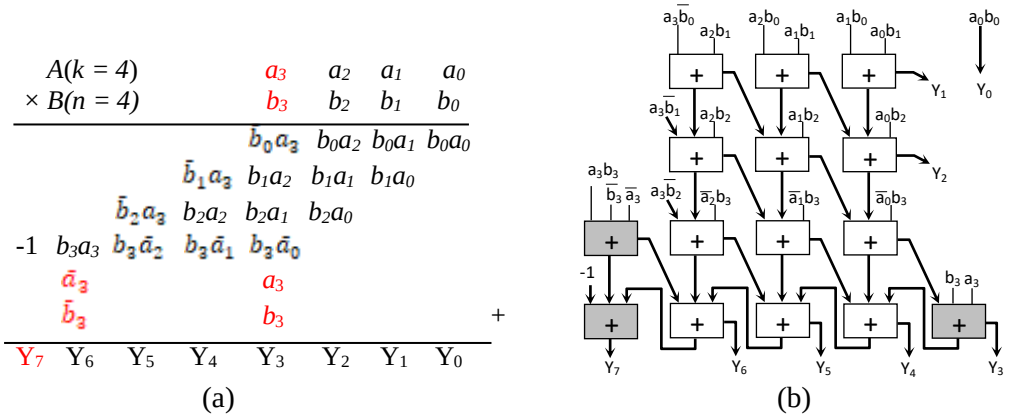


Figure 1. Shift-and-add structure of Baugh-Wooley's signed binary multiplication.

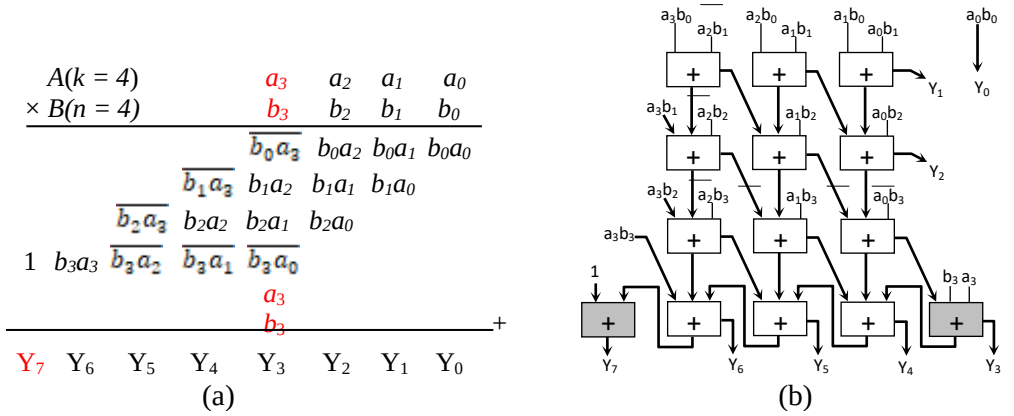


Figure 2. Modified Baugh-Wooley or two's complement multiplication.

PROPOSED SIGNED MULTIPLICATION MODEL

In this section, the new model of signed binary multiplication will be outlined. The proposed model is expressed mathematically and can be easily implemented into software algorithms and hardware on FPGA. This multiplication model will be proven by using several examples. Consider two unsigned binary numbers $A = \{a_{k-1}, a_{k-2}, \ldots, a_1, a_0\}$ and $B = \{b_{n-1}, b_{n-2}, \ldots, b_1, b_0\}$, where k and n respectively are the number of bits. Their decimal values can be expressed as shown in Equation (1), $a_i, b_j \in \{0,1\}$, $i = \{0, 1, \ldots, k-1\}$ and $j = \{0, 1, \ldots, n-1\}$, a_i is the i^{th} magnitude bit of A and b_j is the j^{th} magnitude bit of B. Furthermore, if A and B are signed numbers, both can be represented as shown in Equation (2). $A = \{\hat{a}_{k-1}, a_{k-2}, \ldots, a_1, a_0\}$ and $B = \{\hat{b}_{n-1}, b_{n-2}, \ldots, b_1, b_0\}$, where bits $\hat{a}_{k-1}$ and $\hat{b}_{n-1}$ are sign bits, or we call as borrow bits which mean $\hat{a}_{k-1} = -1a_{k-1}$ and $\hat{b}_{n-1} = -1b_{n-1}$. If a_{k-1} = "0", A has a positive value and a_i represents the magnitude bit. Conversely, if a_{k-1} = "1" then A is negative and a_i indicates its two's complement bit. The same thing applies to B.

$$A=\sum_{i=0}^{k-1} a_i 2^i \quad \text{and} \quad B=\sum_{j=0}^{n-1} b_j 2^j \tag{1}$$

$$A=\hat{a}_{k-1}2^{k-1}+\sum_{i=0}^{k-2} a_i 2^i \quad \text{and} \quad B=\hat{b}_{n-1}2^{n-1}+\sum_{j=0}^{n-2} b_j 2^j \tag{2}$$

$$-A=-1.2^k+\left(\sum_{i=0}^{k-1}\bar{a}_i 2^i\right)+2^0 \quad \text{or} \quad -A=\hat{1}.2^k+\left(\sum_{i=0}^{k-1}\bar{a}_i 2^i\right)+2^0 \tag{3}$$

$$A=\hat{a}_{k-1}2^{k-1}+\sum_{i=0}^{k-2} a_i 2^i \quad \text{and} \quad -A=\hat{\bar{a}}_{k-1}2^{k-1}+\sum_{i=0}^{k-2}\bar{a}_i 2^i+2^0 \tag{4}$$

In the multiplication process, the conversion of unsigned binary numbers to signed numbers and from positive to a negative value, vice-versa, is necessary. For example, if *A* is an unsigned binary number that is always positive, the conversion of *A* to a signed negative number is done through the two's complement process as shown in equation (3). Because *A* does not have a sign bit, then one bit indicating a negative sign "-1" must be added to the bit position of 2^n and then the one's complement process is done by inverting the a_i bits into $\bar{a}_i$ and adding "1" to the LSB position of a_0. The sign bit "-1" can be replaced by a bit symbol "$\hat{1}$", which means a borrowed bit that has a negative value. Take an example, for unsigned $A = 1010_2$ or its decimal numbers is $A = 2^3+2^1 = 10_{10}$. Referring to Equation (3), the two's complement of *A* is $-A = \hat{1}0101_2 + 1_2 = \hat{1}0110_2$ or in decimal is $-2^4+2^2+2^1 = -10_{10}$.

Next, the conversion of positive to negative binary numbers and vice versa is given by equation (4). Example, for signed binary numbers $A = \hat{1}111_2$ or in decimal $A = -2^3+2^2+2^1+2^0 = -1_{10}$, then its two's complement is $-A = \hat{0}000_2 +1_2 = \hat{0}001_2$ or $-A = 2^0 = 1_{10}$. Another example, if $A = \hat{0}111_2$ or $A = 2^2+2^1+2^0 = 7_{10}$, then its two's complement is $-A = \hat{1}000_2 +1_2 = \hat{1}001_2$ or $-A = -2^3+2^0 = -7_{10}$. The mathematical representations of unsigned and signed binary numbers and their conversion will be used to explain our proposed multiplication model.

Based on equation (4), mathematically, the binary representation of signed multiplication of $Y = B \times A$ is shown by equation (5) and then equation (6), where $\hat{a}_{k-1}$ and $\hat{b}_{n-1}$ are respectively the sign bits of *A* and *B*. Furthermore, referred to the equation (3), the second and third parts of this equation can be written in the form of two's complement as

presented in equations (7) and (8). By inserting both into equation (6), then equation (9) is obtained. The first part of this equation is ($\hat{a}_{k-1}\hat{b}_{n-1}+\hat{a}_{k-1}+\hat{b}_{n-1}$) as the sign bit at position 2^{n+k-2} of *Y*. It should be noted that the maximum value of the multiplication result is when the values of $A = \hat{1}.2^{k-1} = -2^{k-1}$ and *B* $= \hat{1}.2^{n-1} = -2^{k-1}$, then the MSB of *Y* $= \hat{1}.2^{n-1} \times \hat{1}.2^{k-1} = 2^{k+n-2}$ and *Y* has a positive value. This shows that the sign bit is not located at bit position of 2^{n+k-2}, but will be relocated at bit position of 2^{n+k-1}. Thus by using the logical operation "OR" (symbolized by ||), $(\hat{a}_{k-1}\hat{b}_{n-1}+\hat{a}_{k-1}+\hat{b}_{n-1})2^{n+k-1}$ can be replaced by $(\hat{a}_{k-1}||\hat{b}_{n-1})2^{n+k-1}$ + $(a_{k-1}||b_{n-1})2^{n+k-2}$ as given in Equation (10). This proposed signed binary multiplication (*SNN/SPN-by- SNN/SPN*) simplifies Baugh-Wooley's model.

$$Y = \left(\hat{a}_{k-1}2^{k-1} + \sum_{i=0}^{k-2} a_i 2^i\right)\left(\hat{b}_{n-1}2^{n-1} + \sum_{j=0}^{n-2} b_j 2^j\right) \tag{5}$$

$$Y = \left(\hat{b}_{n-1}2^{n-1}\right)\left(\hat{a}_{k-1}2^{k-1}\right) + \sum_{j=0}^{n-2} b_j 2^j \left(\hat{a}_{k-1}2^{k-1}\right) + \sum_{i=0}^{k-2}\left(\hat{b}_{n-1}2^{n-1}\right)a_i 2^i + \sum_{j=0}^{n-2}\sum_{i=0}^{n-2} b_j a_i 2^{i+j} \tag{6}$$

where $$\sum_{j=0}^{n-2} b_j 2^j \left(\hat{a}_{k-1}2^{k-1}\right) = \hat{a}_{k-1}2^{k+n-2} + a_{k-1}2^{k-1} + \sum_{j=0}^{n-2}\bar{b}_j a_{k-1}2^{j+k-1} \tag{7}$$

and $$\sum_{i=0}^{k-2}\left(\hat{b}_{n-1}2^{n-1}\right)a_i 2^i = \hat{b}_{n-1}2^{k+n-2} + b_{n-1}2^{n-1} + \sum_{i=0}^{k-2} b_{n-1}\bar{a}_i 2^{i+n-1} \tag{8}$$

then

$$Y = \left(\hat{a}_{k-1}\hat{b}_{n-1} + \hat{a}_{k-1} + \hat{b}_{n-1}\right)2^{k+n-2} + a_{k-1}2^{k-1} + b_{n-1}2^{n-1} + \sum_{j=0}^{n-2}\bar{b}_j a_{k-1}2^{j+k-1} + \sum_{i=0}^{k-2} b_{n-1}\bar{a}_i 2^{i+n-1} + \sum_{j=0}^{n-2}\sum_{i=0}^{k-2} b_j a_i 2^{i+j} \tag{9}$$

$$Y = \underbrace{\left(\hat{a}_{k-1} \,\|\, \hat{b}_{n-1}\right)2^{k+n-1}}_{\text{Sign bit}} + \underbrace{\left(a_{k-1} \,\|\, b_{n-1}\right)2^{k+n-2}}_{\text{MSB}} + \underbrace{\left(\sum_{j=0}^{n-2} a_{k-1}\bar{b}_j 2^{j+k-1}\right) + a_{k-1}2^{k-1}}_{\text{Multiplication: } a_{k-1} \text{ and two's complement of } B} + \underbrace{\left(\sum_{i=0}^{k-2}\bar{a}_i b_{n-1}2^{i+n-1}\right) + b_{n-1}2^{n-1}}_{\text{Multiplication: } b_{n-1} \text{ and two's complement of } A} + \underbrace{\sum_{j=0}^{n-2}\sum_{i=0}^{k-2} a_i b_j 2^{i+j}}_{\text{Multiplication: } b_j a_i} \tag{10}$$

The multiplication process of equation (10) can be implemented in the software mode using algorithm *Algo-1*. Both variables *A (k bits)* and *B (n bits)* are signed binary numbers, each having one sign bit and *k*-1, and *n*-1 magnitude bits. The *shift-left* and *AND* logic operations are respectively symbolized by "«" and "&". Conforming to Equation (10), this algorithm consists of five parts. First, steps 1 – 3 are accumulator initialization and determining MSB value: $S_y = (a_{k-1} \| b_{n-1})$, and then save it to the accumulator at position *Y*(*n*-1). The second and third parts in steps 4–6 and steps 7–9 are the multiplication processes of $\hat{a}_{k-1}2^{k-1} \times$ (two's complement of *B)* and $\hat{b}_{n-1}2^{n-1} \times$ (two's complement of *A)*, respectively. Each of these processes is carried out when the conditions are met, and their results are added to the accumulator *Y*. Fourth, steps 10–15 are the multiplication process of $b_j 2^j \times A(k\text{-}2 : 0)$, and the last part (step 16) sets up the sign bit value of the multiplication result $(\hat{a}_{K-1} \| \hat{b}_{N-1})2^{N+K-1} + C_{out}$. In this step, the logic process S_y & $\overline{Y(p-1)}$ is used. It means if carry-out (C_{out}) at *Y*(*p*-1) = "1" and S_y = "1" (borrow), then the sign bit at *Y*(*p*-1) is set to be "0", otherwise if carry-out at *Y*(*p*-1) = "0" then the sign bit at *Y*(*p*-1) = S_y. Finally, the multiplication result consists of *Y*(*p*-1) as the sign bit and *Y*(*p*-2 : 0) as the magnitude bits.

***Algo-1**. (SNN/SPN)-by-(SNN/SPN) Multiplication Algorithm: (+B)×(+A); (+B)×(-A); (-B)×(+A) and (-B)×(-A)*

```
Input: signed A(k bits), signed B(n bits)          % unsigned number A and signed number B
Output: signed Y(p bits)                           % signed number Y:  p = (k + n) bits;
Process :
1   Y(p-1 : 0) ← 0;                                % Accumulator Y (Acc. Y ) initialized to 0
2   Sy ← A(k-1) || B(n-1);                         % Set Sy = ā(k-1) || b̄(n-1) = a(k-1) || b(n-1)
3   Y(n-1) ← Sy;                                   % Set Sy as MSB to Acc. at position Y(n-1)
4   if A(k-1) = 1                                  % If A has a negative value then
5      Y ← Y +{Comp(B(n-2 :0))+A(k-1)};            % Add ā(k-1)2^(k-1) ×two's complement of B without
6   endif                                            sing bit to Acc. Y
7   if B(n-1) = 1                                  % If B has a negative value then
8      Y ← Y +{Comp(A(k-2 :0))+B(n-1)};            % Add b̄(n-1)2^(n-1) ×two's complement of A without
9   endif                                            sing bit to Acc.Y
10  for j = (n-2) downto 0                         % repeat process of bj2^j×A(k-2: 0), until j = 0.
11     Y ← Y « 1;                                  % Shift-left one-bit the value of Acc. Y
12     if B(j) = 1                                 % if bj=1, then
13        Y ← Y + A(k-2 : 0);                      % add A without sing bit, to Acc. Y
14     endif
15  endfor j                                       % end repeat
16  Y(p-1) ← (Sy & Y(p-1) );                       % Set sign bit at Y(p-1) =(ā(k-1) || b̄(n-1)) + Cout
endprocess
```

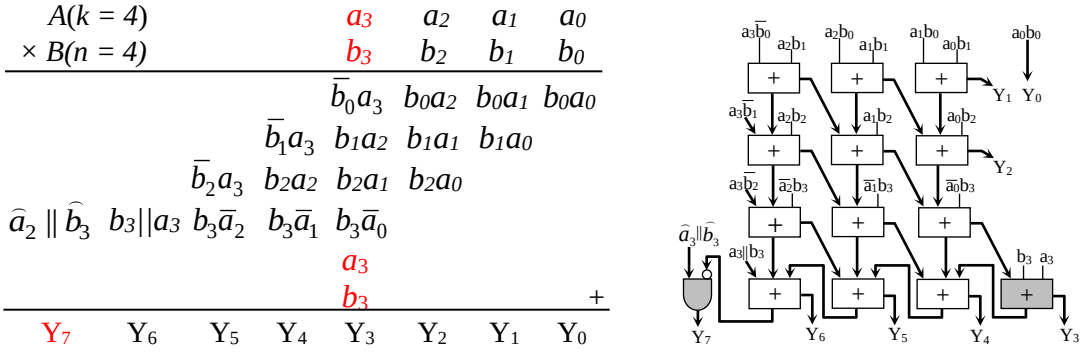


Figure 3. Shift-and-add and matrix structures of proposed signed binary multiplication.

The shift-and-add structure of our proposed model can be presented as given by an example in Figure 3, where *A* and *B* respectively have a sign bit a_3 and b_3, seven integer bits $a_2a_1a_0$ (k = 4) and $b_2b_1b_0$ (n = 4). Figure 3-a, the bits $\bar{b}_j a_3$ on the left diagonal-column and LSB a_3 are part of the two's complement process of *B*. On the last line, the bits $b_3\bar{a}_i$ and LSB b_3 are part of the two's complement process of *A*. Sign bit $\hat{b}_3||\hat{a}_3$ and MSB $b_3||a_3$ of the multiplication are successively located in the bit positions of 2^7 (k+n-1=7) and 2^6 (k+n-2=6). It should also be noted that if the sign bit $\hat{b}_3||\hat{a}_3$ has a value of "$\hat{1}$", then this bit is negative or is a borrowed bit. Furthermore, if the sum of all multiplication bits has *carry-out* equals "1" at the bit position of 2^7, then the sign bit Y_7 = *carry-out* + *borrow* = "1" + "$\hat{1}$" = "1" + "-1" = "0". In Figure 3-b, the proposed multiplication model only requires an addition of one adder and one AND gate. This is more efficient than Baugh-Wooley's model.

IMPLEMENTATION AND RESULTS

The proposed signed binary multiplication algorithm has been implemented using Scilab software and in FPGA hardware using Xilinx. The processes and results obtained from programming coding are illustrated in figures and tables. Based on the *shift-and-add* structure model (Figure 3), the proposed *(SNN/SPN)-by-(*part of two's complement process of *B* and *A* successively. All bits of $\bar{b}_j a_2$="**0**" because a_2="**0**" and all bits of $b_3\bar{a}_i$="**0**" because b_3="**0**". So, the multiplication result is Y=**0**10010.01001$_2$, which has a positive value because its sign bit Y_5="**0**" and its decimal value is Y = 18.28125$_{10}$.

The next example is the *SNN-by-SNN* multiplication, where A = -3.25$_{10}$ or in binary two's complements A = 100.11$_2$, which has sign bits a_2="**1**" and B = -5.625$_{10}$ or in binary two's complement B = 1010.011$_2$, with sign bit b_3="**1**". The binary multiplication process of Y = 1010.011$_2$×100.11$_2$ is given in figure 4-b, where its two's complement parts have sign bit $\hat{b}_3||\hat{a}_2$="1", MSB $b_3||a_2$ ="1", LSB a_2="**1**" and b_3="**1**", and then the multiplication bits of

$\bar{b}_j a_2$="**0**" when *bj*="1" and $\bar{b}_j a_2$="**1**" when *bj*="0", as well as for the multiplication bits of $b_3 \bar{a}_i$="**0**" when a_i="1" and $b_3 \bar{a}_i$="**1**" when a_i="0". For this example, the sum of all bits has a carry-out "1" at the sign bit position of 2^5, then the new sign bit equals carry-out + sign bit = "1" + ("-1") = "0", so the multiplication result Y = 010010.01001$_2$ has the sign positive and its decimal value is 18.28125$_{10}$.

The third example is the *SNN-by-SPN* multiplication shown in figure 15-c, which is Y = 1010.011$_2$×011.01$_2$ or in decimal Y = (-5.625$_{10}$)×(+3.25$_{10}$). The two's complement part of this multiplication has a sign bit $\hat{b}_3 || \hat{a}_2$="1", MSB $b_3||a_2$ ="1", LSB a_2="**0**" and b_3="**1**", all bits of $\bar{b}_j a_2$="**0**" because a_2="**0**" and the multiplication bits of $b_3 \bar{a}_i$="**0**" when a_i="1" and $b_3 \bar{a}_i$="**1**" when a_i="0". The multiplication result Y = **1**01101.10111$_2$ has the sign negative and its decimal value is -18.28125$_{10}$. The fourth example shown in figure 4-d is the *SPN-by-SNN* multiplication, which is Y = **0**101.101$_2$×100.11$_2$ or in decimal, Y = (+5.625$_{10}$)×(-3.25$_{10}$). The two's complement part of this multiplication has a sign bit $\hat{b}_3 || \hat{a}_2$="1", MSB $b_3||a_2$ ="1", LSB a_2="**1**" and b_3="**0**", all bits of $b_3 \bar{a}_i$="**0**" because b_3="**0**" and the multiplication bits of $\bar{b}_j a_2$="**0**" when b_j="1" and $\bar{b}_j a_2$="**1**" when b_j="0". The multiplication result Y = **1**01101.10111$_2$ has the sign negative and its decimal value is -18.28125$_{10}$. All the results of examples in figures 4-a to 4-p are summarized in Table 1.

The proposed multiplication model is implemented in FPGA using Xilinx software, ISE Design Suite 14.7. Two implementation approaches are carried out by employing LUTs (LUT6 and LUT5), fast carry logics: Carry4, MUXCY, and XORCY resources. The first approach uses a sequential shift-and-add process or serial-parallel multiplier based on Algo-1 and the second one is a parallel multiplier or array multiplier referring to matrix structure in Figure 4-b. Both are designed for 8 bits (n = k = 8) signed binary integer numbers.

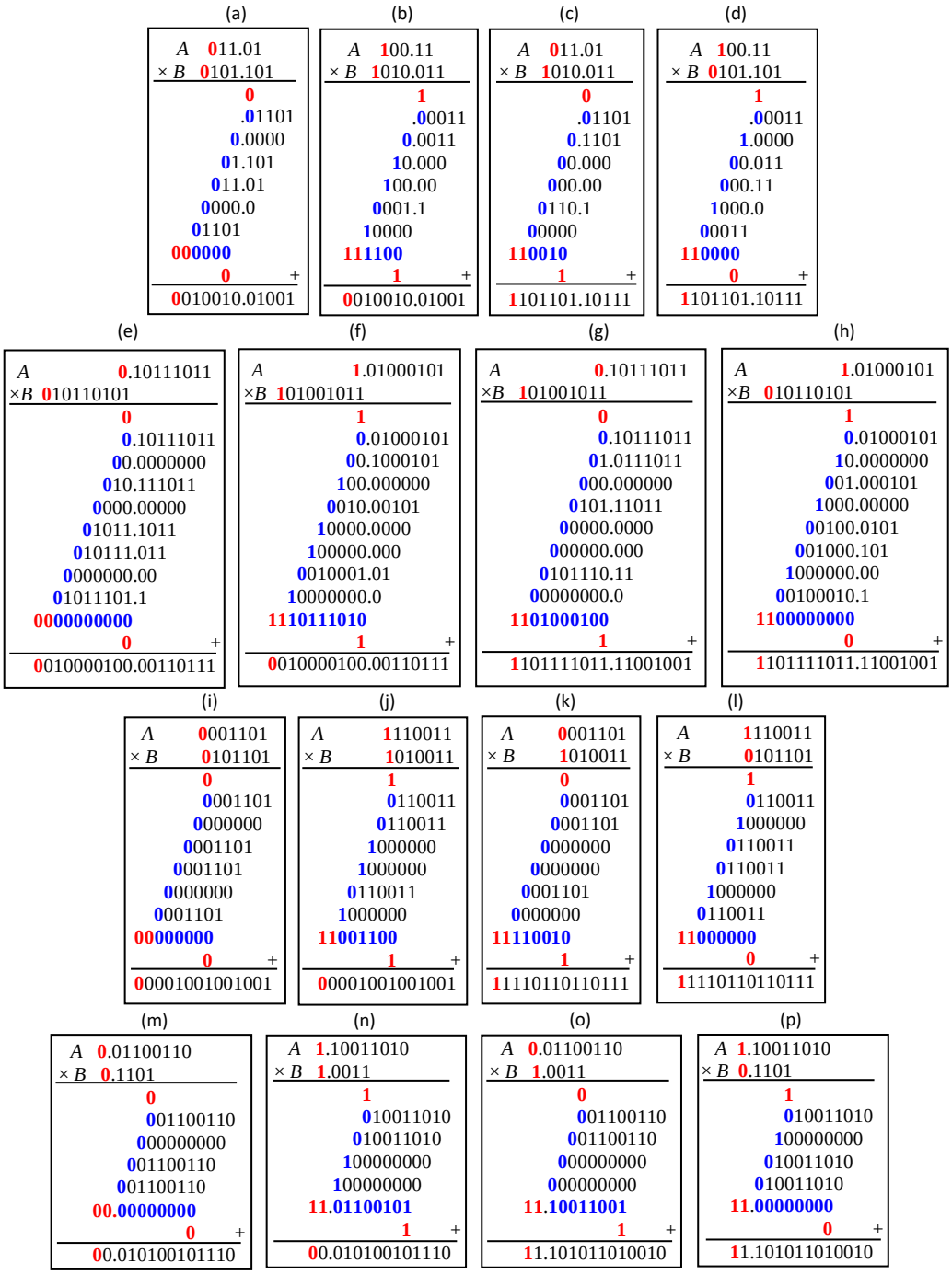


Figure 4. Examples of the proposed (*SNN/SPN*)-*by*-(*SNN/SPN*) multiplication.

Table 1. Results of sixteen examples in figure 4.

***Y* = (+*B*) × (+*A*)**	***Y* = (-*B*) × (-*A*)**	***Y* = (-*B*) × (+*A*)**	***Y* = (+*B*) × (-*A*)**
$0101.101_2 \times 011.01_2$ $= 010010.01001_2$ $5.625_{10} \times 3.25_{10}$ $= 18.28125_{10}$	$1010.011_2 \times 100.11_2$ $= 010010.01001_2$ $-5.625_{10} \times -3.25_{10}$ $= 18.28125_{10}$	$1010.011_2 \times 011.01_2$ $= 101101.10111_2$ $-5.625 \times 3.25_{10}$ $= -18.28125_{10}$	$0101.101_2 \times 100.11_2$ $= 101101.10111_2$ $5.625 \times -3.25_{10}$ $= -18.28125_{10}$
$010110101_2 \times 0.10111011_2$ $= 010000100.00110111_2$ $181_{10} \times 0.73046875_{10}$	$101001011_2 \times 1.01000101_2$ $= 010000100.00110111_2$ $-181_{10} \times -0.73046875_{10}$	$101001011_2 \times 0.10111011_2$ $= 101111011.11001001_2$ $-181_{10} \times 0.73046875_{10}$	$010110101_2 \times 1.01000101_2$ $= 101111011.11001001_2$ $181_{10} \times -0.73046875_{10}$

= 132.21484375_{10}	= 132.21484375_{10}	= -132.21484375_{10}	= -132.21484375_{10}
$\mathbf{0}101101_2 \times \mathbf{0}001101_2$ = $\mathbf{0}001001001001_2$ $45_{10} \times 13_{10} = 585_{10}$	$\mathbf{1}010011_2 \times \mathbf{1}110011_2$ = $\mathbf{0}001001001001_2$ $-45_{10} \times -13_{10} = 585_{10}$	$\mathbf{1}010011_2 \times \mathbf{0}001101_2$ = $\mathbf{1}110110110111_2$ $-45_{10} \times 13_{10} = -585_{10}$	$\mathbf{0}101101_2 \times \mathbf{1}110011_2$ = $\mathbf{1}110110110111_2$ $45_{10} \times -13_{10} = -585_{10}$
$\mathbf{0}.1101_2 \times \mathbf{0}.01100110_2$ = $\mathbf{0}.010100101110_2$ $0.8125_{10} \times 0.3984375_{10}$ = 0.32373046875_{10}	$\mathbf{1}.0011_2 \times \mathbf{1}.10011010_2$ = $\mathbf{0}.010100101110_2$ $-0.8125_{10} \times -0.3984375_{10}$ = 0.32373046875_{10}	$\mathbf{1}.0011_2 \times \mathbf{0}.01100110_2$ = $\mathbf{1}.101011010010_2$ $-0.8125_{10} \times 0.3984375_{10}$ = -0.32373046875_{10}	$\mathbf{0}.1101_2 \times \mathbf{1}.10011010_2$ = $\mathbf{1}.101011010010_2$ $0.8125_{10} \times -0.3984375_{10}$ = -0.32373046875_{10}

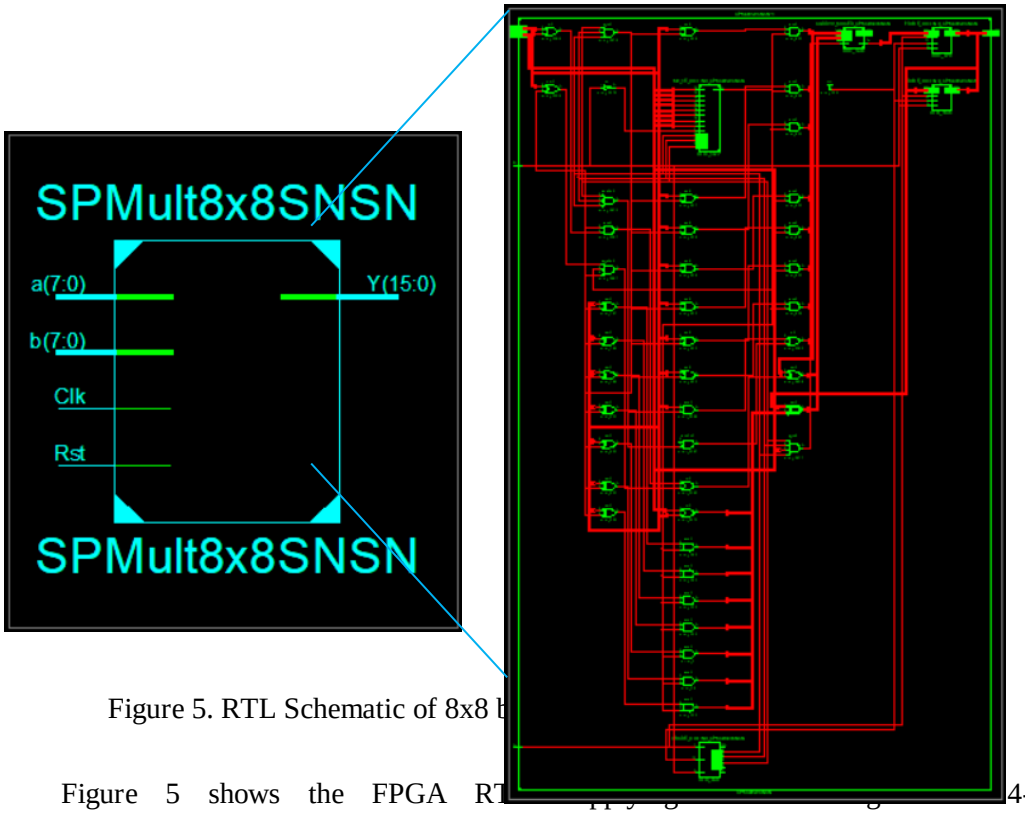
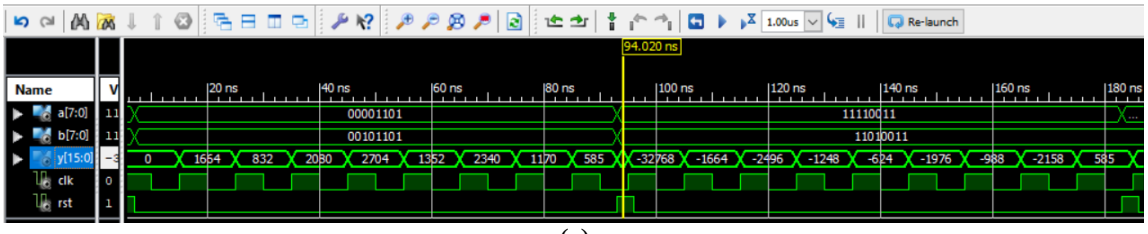
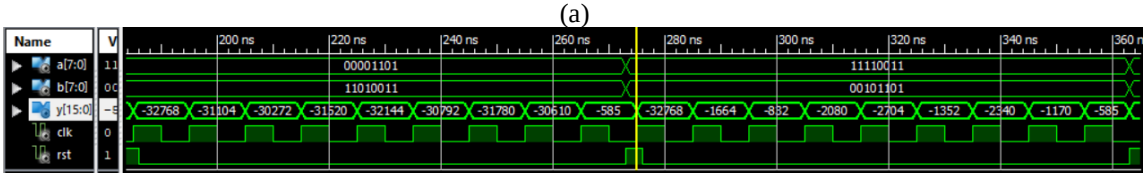


Figure 5. RTL Schematic of 8x8 b

Figure 5 shows the FPGA RT 4-l and in schematic of 8x8 bits signed binary serial-parallel multiplier. The FPGA resources used are presented in table 2: eleven LUT's as a logical multiplication function; two Carry4s and sixteen flip-flops as accumulators (adders and shift registers); eight flip-flops as a multiplier shift register; and one counter of 3 bits to control the multiplication process. This multiplier has a maximum combinational path delay of 1.485 ns, can be operated at a maximum frequency of 350.262 MHz, and needs 8 cycles to finish the multiplication process. Its simulation results, carried out by the third row of table 1, are shown in Figure 6. The multiplication process starts when the "Reset" signal changes from "1" to "0" and at the first rising edge clock. At each clock, the multiplication value continues to change, until the end process at the eighth clock and is followed by the "Reset" signal change from "0" to "1". The products of B×A are given in decimal and located at the 8^{th}, 16^{th}, 24^{th} and 32^{th} clock respectively for: $45_{10}\times13_{10}=585_{10}$, $(-45_{10})\times(-13_{10})=585_{10}$, $(-45_{10})\times13_{10}=-585_{10}$, and $45_{10}\times(-13_{10})=-585_{10}$.



(a)



(b)

Figure 6. Simulation results of 8x8 bits signed binary serial-parallel multiplier.

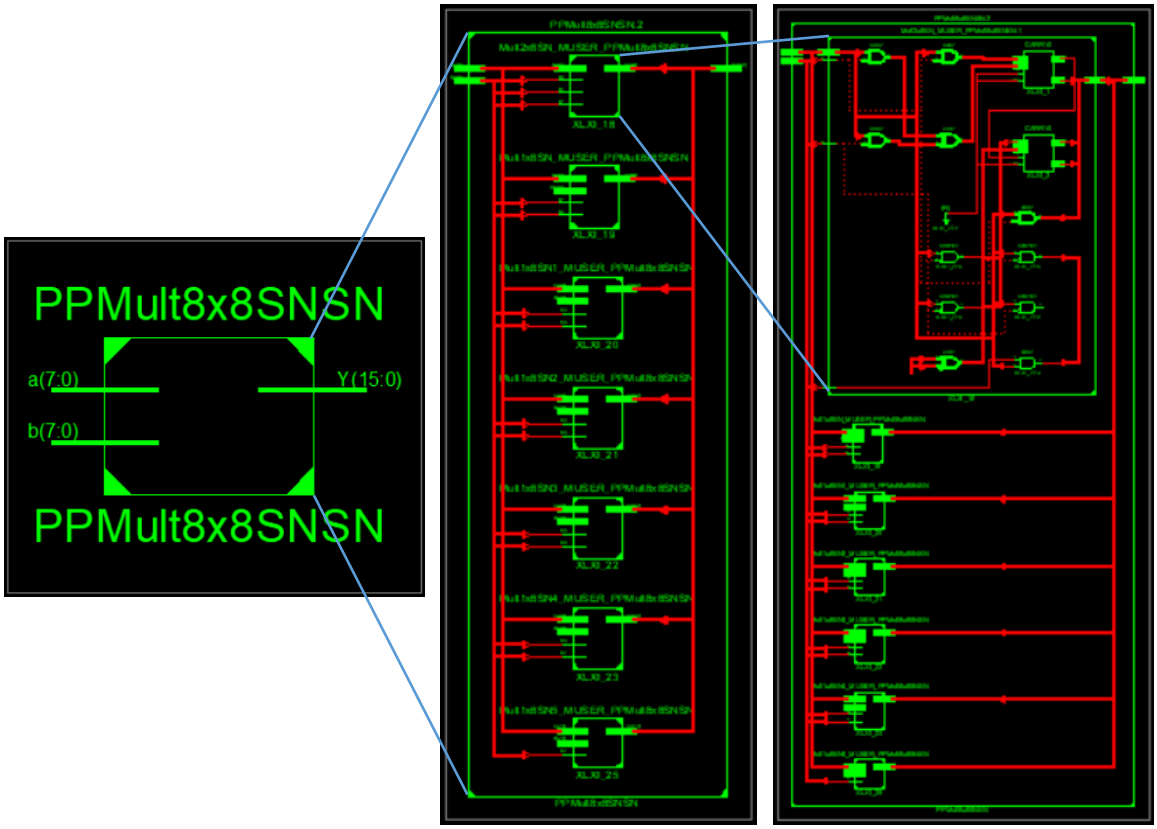


Figure 7. RTL Schematic of 8x8 bits signed binary parallel multiplier.

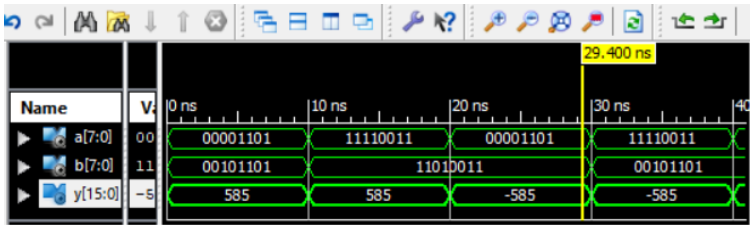


Figure 8. Simulation results of 8x8 bits signed binary parallel multiplier.

Table 2. FPGA resources occupied by 8x8 bits Serial-parallel and parallel multipliers.

Multiplier implementation approach	Carry logic. Carry4	Counter	Occupied FFs	Occupied LUTs	Occupied Slices	Delay (ns)	Clock cycle	Maximum Frequency (MHz)
Serial-parallel	2	1 (3 bits)	25	19	12	1.485	8	350.262
Parallel	14	-	-	57	15	11.233	1	87.719

Figure 7 shows the FPGA RTL schematic of 8x8 bits signed binary parallel multiplier. This multiplier occupies FPGA resources of 14 Carry4 and 57 LUTs in 15 slices and performs the multiplication process in one cycle with a time delay of 11.233 ns. Its simulation results are given in Figure 8 respectively for: $45_{10}\times13_{10}=585_{10}$, $(-45_{10})\times(-13_{10})=585_{10}$, $(-45_{10})\times13_{10}=-585_{10}$, and $45_{10}\times(-13_{10})=-585_{10}$.

CONCLUSION AND FUTURE WORK

New signed binary multiplication has been proposed and formulated mathematically. This formula is easily implemented in software coding with a low complexity algorithm. Its hardware implementation in the FPGA is also quite easy, either in the form of the serial-parallel multiplier and parallel multiplier. Both are implemented by optimizing the use of LUT5 and LUT6 for multiplication logic functions, fast carry logic Carry4, MUXCY, and XORCY so that fewer FPGA resources are occupied.

Further research is being carried out on hardware implementation using several existing methods: Cascade multiplier, Wallace tree multiplier, Vedic multiplier, and booth multiplier.

REFERENCES

[1] Y. Chen, S. Duffner, A. Stoian, J.-Y. Dufour, A. Baskurta, "Deep and low-level feature based attribute learning for person re-identification," *Image Vis. Comput.*, vol. 79. pp 25–34, 2018.

[2] X. Cheng, J. Lu, J. Feng, B. Yuan, J. Zhou, "Scene recognition with objectness," *Pattern Recognition*, vol. 74. pp 474–487, 2018.

[3] J. Zhang, K. Shao, X. Luo, "Small sample image recognition using improved Convolutional Neural Network," *J. Vis. Commun. Image Represent*, vol. 55, pp 640–647, 2018.

[4] S.S. Sarikan, A.M. Ozbayoglu, O. Zilcia, "Automated vehicle classification with image processing and computational intelligence," *Procedia Comput. Sci.*, vol. 114, pp 515–522, 2017.

[5] A. Qayyum, S.M. Anwar, M. Awais, M. Majid, "Medical image retrieval using deep convolutional neural network," *Neurocomputing*, vol. 266, pp 8–20, 2017.

[6] L. Gong, C. Wang, X. Li, H. Chen, X. Zhou, “MALOC: A fully pipelined FPGA accelerator for convolutional neural networks with all layers mapped on chip,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst*, vol. 37, no. 11, pp 2601–2612, 2018.

[7] N.I. Chervyakov, P.A. Lyakhov, M.V. Valueva, “Increasing of Convolutional Neural Network performance using residue number system,” *International Multi-Conference on Engineering, Computer and Information Sciences (SIBIRCON)*, pp. 135–140, 2017.

[8] A. Shawahna, S.M. Sait, A. El-Maleh, “FPGA-based accelerators of deep learning networks for learning and classification: A review,” *IEEE Access*, vol. 7, 7823–7859, 2019.

[9] H. Sim and J. Lee, "A New Stochastic Computing Multiplier with Application to Deep Convolutional Neural Networks", *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, Jun. 2017.

[10] Juan Renteria-Cedano 1 , Jorge Rivera 2,* , F. Sandoval-Ibarra 1 , Susana Ortega-Cisneros 1 and Raúl Loo-Yau 1, SoC Design Based on a FPGA for a Configurable Neural Network Trained by Means of an EKF, Electronics 2019, 8, 761; doi:10.3390/electronics8070761 www.mdpi.com/journal/electronics

[11] FPGA Acceleration of Matrix Multiplication for Neural Networks (xilinx.com) XAPP1332 (v1.0) February 27, 2020 www.xilinx.com Application Note.

[12] Baugh C.R., Wooley B.A., A Two’s Complement Parallel Array Multiplication Algorithm. IEEE Trans. Comput. C-22, pp 1045–1047, 1973.

[13] PramodiniMohanty, RashmiRanjan, “An Efficient Baugh Wooley Architecture for Both Signed & Unsigned Multiplication”, International Journal of Computer Science and Engineering Technology, vol. 3, no. 4, April 2012.