A HIGH BIT RATE SERIAL–SERIAL MULTIPLIER

By

JISHA K R

Reg. No. 1020106006

of

KUMARAGURU COLLEGE OF TECHNOLOGY

(An Autonomous Institution affiliated to Anna University, Coimbatore)

COIMBATORE - 641049

A PROJECT REPORT

Submitted to the

FACULTY OF ELECTRONICS AND COMMUNICATION ENGINEERING

In partial fulfillment of the requirements
for the award of the degree
of

MASTER OF ENGINEERING
IN
APPLIED ELECTRONICS
MAY 2012

BONAFIDE CERTIFICATE

Certified that this project report entitled "**A HIGH BIT RATE SERIAL – SERIAL MULTIPLIER** " is the bonafide work of **JISHA K R** [Reg. no. 1020106006] who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other project or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

Project Guide	Head of the Department
Dr. Rajeswari Mariappan	Dr. Rajeswari Mariappan

The candidate with university Register no. 1020106006 is examined by us in the project viva-voce examination held on ……

Internal Examiner External Examiner

ACKNOWLEDGEMENT

I express my profound gratitude to our chairman **Padmabhusan Arutselvar Dr.N.Mahalingam B.Sc.,F.I.E.** for giving this opportunity to pursue this course.

I wish to acknowledge my sincere gratitude and heartfelt thanks to our beloved principal **Dr.S.Ramachandran Ph.D.,** for having given me the adequate support and opportunity for completing this project work successfully.

I express my sincere thanks to **Dr.Ms.Rajeswari Mariappan Ph.D.,** Head of the Department of Electronics and Communication Engineering and my project guide for her ideas and suggestion, which have been very helpful for the completion of this project work.

In particular, I wish to thank and everlasting gratitude to the project coordinator **Ms.R.Hemalatha M.E.,** Assistant Professor, Department of Electronics and Communication Engineering for her expert counseling and guidance to make this project to a great deal of success. With her careful supervision and ensured me in the attaining perfection of work.

Last, but not the least, I would like to express my gratitude to my family members, friends and to all my staff members of Electronics and Communication Engineering department for their encouragement and support throughout the course of this project.

ABSTRACT

Multipliers are essential building blocks in VLSI circuits. Hardware implementation of a multiplication operation consists of three stages, specifically the generation of partial products (PPs), the reduction of partial products (PPs), and the final carry propagation addition. The partial products can be generated either in parallel or serially, depending on the target application and the availability of input data. The Project highlights a technique for serial multiplication, which completes the partial product formation in n cycles. This is accomplished by revamping the partial product formation architecture. The proposed technique effectively forms the entire partial product matrix in just n sampling cycles for an n x n multiplication instead of at least 2n cycles in the conventional serial-serial multipliers.

In this project, an algorithm named Serial to Serial algorithm is proposed to arrive at the architecture. The algorithm helps in reducing the complexity of partial product formation and reduction . The architecture consists of a series of asynchronous 1's counters so that the critical path is limited to only an AND gate and a D flip-flop (DFF). The use of 1's counter to column compress the partial products preliminarily reduces the height of the partial product matrix from n to log (n + 1), resulting in a significant complexity reduction of the resultant adder tree. Using this algorithm the multiplier is designed to carry out both signed and unsigned multiplication. The latency of operation is compared with the existing Carry Save Add and Shift (CSAS) multiplier.

TABLE OF CONTENT

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

Multipliers are fundamental and essential building blocks of VLSI systems. The design and implementation on approaches of multipliers contribute substantially to the area, speed and power consumption of computational intensive VLSI systems. Often, the delay of multipliers dominates the critical path of these systems and due to issues concerning reliability and portability, power consumption is a critical criterion for applications that demand low-power as its primary metric . While low power and high speed multiplier circuits are in high demand, it is not always possible to achieve both criteria simultaneously.

Typically, hardware implementation of a multiplication operation consists of three stages, specifically the generation of partial products (PPs), the reduction of PPs and the final carry-propagation addition[9]. The partial products can be generated either in parallel or serially, depending on the target application and the availability of input data. The partial products are generally reduced by carry-save adders (CSAs) using an array or a tree structure[3]. Carry propagation addition is inevitable when the number of partial products is reduced to two rows. This final adder can be a simple ripple carry adder (RCA) for low power or a carry look-ahead adder (CLA) for high speed [9]. As the height of PP tree increases linearly with the wordlength of the multiplier, it increases the area, delay and power dissipation of the two subsequent stages. Therefore, a good multiplier design requires some tradeoff between speed and power consumption.

Parallel multipliers are more popular as the size is less critical due to technology scaling. However, with the emerging development of the on-chip serial-link bus architectures [16],[17], serial-serial multipliers could find their potential roles in the new generation of System on Chips and Field Programmable Gate Arrays to the design of serial multiplier that is capable of processing input data at Gb/s without input buffering and with reduced total number of computational cycles is proposed.

1.1 PROJECT GOAL

The goal of this project is to design a Serial-Serial multiplier for the purpose of fast performance with less area overhead and power consumption. Unlike existing serial multipliers in which one of the operand is fed in parallel, here both the operands are fed serially with reduced delay. The complete partial product formation and accumulation gets completed within n cycles for an nxn multiplier when compared to the existing serial multiplier which takes about 2n cycles for partial product formation. The Accumulator is Designed using Asynchronous 1's counter so that the critical delay path is reduced to and gate and a D flip-flop.

1.2 OVERVIEW

The basic steps for designing a Serial-Serial multiplier is based on an algorithm named Serial –Serial algorithm. Using the algorithm an architecture is being proposed for Unsigned multiplication and this is extended to carry out Signed multiplication. The partial products are generated, accumulated and the column height of the partial product is reduced using the algorithm. Once the products are so formed and accumulated , it can be used for addition for generating the final product.

1.3 SOFTWARES USED

- ModelSim XE III 6.2g
- Xilinx ISE 9.1i

1.5 ORGANIZATION OF THE REPORT

- **Chapter 2** discusses an Introduction to Serial Multipliers
- **Chapter 3** Explains The Serial-Serial Algorithm
- **Chapter 4** Proposed Serial Accumulator
- **Chapter 5** Explains Proposed Serial-Serial Multiplier
- **Chapter 6** discusses Extension of logic For Signed Multi[placation
- **Chapter 7** discusses the Existing Serial multipliers
- **Chapter 8** discusses simulation results
- **Chapter 9** shows the Conclusion and future scope of the project.

CHAPTER 2
INTRODUCTION TO SERIAL MULTIPLIERS

Serial multipliers are popular for their low area and power , and are more suitable for bit serial signal processing applications with I/O constraints and on-chip serial-link bus architectures. They are broadly classified into two categories, namely serial-serial and serial-parallel multiplier. In a serial-serial. multiplier both the operands are loaded in a bit serial fashion, reducing the data input pads to two . On the other hand, a serial-parallel multiplier loads one operand in a bit-serial fashion and the other is always available for parallel operation. In two's complement pipeline multipliers consists of a bit-serial input output multiplier which features high throughput at the expense of truncated output. A full precision bit serial multiplier for unsigned numbers consists of a 5:3 counter and some DFFs. Later, the first bit-serial multiplier was developed that directly handles the negative weight of the most significant bit (MSB) in 2's complement representation. This method needs only n cells for an nxn bit multiplication but it introduces an XOR gate in the critical path, which ends up with a more complicated overall design.

In all these designs the it requires about 2n computational cycles for nxn multiplication. To reduce the number of computational cycles from 2n to n cycles in an nxn serial multiplier, several serial-parallel multipliers have been developed over the years . Most of them are based on a carry save add shift (CSAS) structure. In CSAS multiplier it can be observed that the critical path consists of an FA, a DFF, and an AND gate for the unsigned multiplier and an extra XOR gate for the 2's complement multiplier. Although the sampling frequency has been improved in the serial-parallel multipliers and the total number of computational cycles is halved, one of the operands has to be loaded in parallel. Recently, there has not been any new development in serial-serial multiplier design due to the maturity of conventional architectures. Parallel multipliers are more popular as the size is less critical due to technology scaling. However, with the emerging development of the on-chip serial-link bus architectures ,

serial-serial multipliers could find their potential roles in the new generation of SoCs and FPGAs. In this project, an approach to the design of serial multiplier that is capable of processing input data at Gb/s without input buffering and with reduced total number of computational cycles is proposed. By exploiting the relationship among the bits of a partial product matrix, it is possible to generate all the rows serially in just n cycles for an nxn multiplication. Employing counters to count the number of 1's in each column allows the partial product bits to be generated on-the-fly and partially accumulated in place with a critical path delay of only an AND gate and a DFF. The counter-based accumulation reduces the PP height logarithmically and makes it possible to achieve an effective reduction rate of log n using an FA-based CSA tree The proposed counter-based multiplier outperforms many serial-serial and serial-parallel multipliers in speed but its hybrid architecture does carry an area overhead. Overall, the ADP is comparable to other serial-serial multipliers

CHAPTER 3
THE SERIAL-SERIAL ALGORITHM

The Serial multiplication in the project is based on an algorithm named serial algorithm that reduces the computation time of the partial products such that they can be formed in just n cycles for an n x n multiplier. According to this algorithm the partial product row and column structure is revamped. The figures 3.1 and 3.2 shows the partial product formation of the conventional and the proposed multiplier.

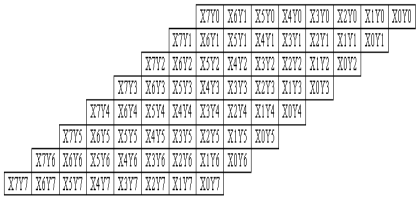


Figure 3.1 Conventional PP Generation Scheme

Fig. 2.1 illustrates the PP generation of a conventional 8x8 multiplier for two unsigned numbers X and Y and Fig. 3.2 shows the corresponding partial product generation in the proposed serial multiplier. The PPs in Fig. 3.2 are generated in such an unconventional way.

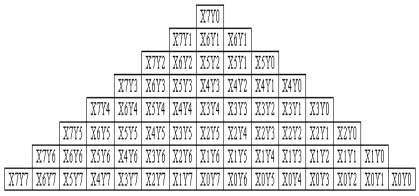


Figure 3.2 Proposed PP Generation Scheme

The partial products so formed are counted column wise for the number of ones. The counter output is also arranged column wise. The bit representation of the counter is based on the maximum number of ones n each column. The counter output format is as shown in the fig 3.3

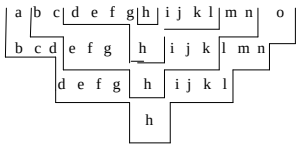


Figure 3.3 Counter output Scheme

The counter output is grouped as shown in fig 3.3. After grouping the groups are selected to form the rows. The outermost group is taken as the first row. The second outermost

group is taken as the second row bit then the row is shifted once to left. The third outermost group is taken as the third row but it is shifted twice to the left. The method is continued till the last group and the final format of partial product is as shown in the fig3.4

a b c d e f g h i j k l m n o
b c d e f g h i j k l m n
d e f g h i j k l
h

Figure 3.4 Reduced PP Scheme

By comparing fig. 3.2 and fig. 3.3 we can see that the pp column height is reduced from 8 to 4. this reduction in the column height helps in reducing the computational complexity. The above partial products ca then be further reduced by using any adders. demodulator function is concerned, it is best performed digitally in a DSP processor outside the digital receiver chip.

The following example illustrates the multiplication of two operands using the Serial Serial Algorithm.

Let the operand length be 8 and the operands be

A = 10000000001
B = 10000000000

```
                1 0 0 0 0 0 0 0
          X
                1 0 0 0 0 0 0 1
-------------------------------------------
                1 0 0 0 0 0 0 0
              0 0 0 0 0 0 0 0
            0 0 0 0 0 0 0 0
          0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
      0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0
  1 0 0 0 0 0 0 0
-------------------------------------------
  1 | 0 0 | 0 0 0 0 | 0 | 0 0 0 0 | 0 0 | 0
      | 0 0 | 0 0 0 0 | 0 | 0 0 0 0 | 0 0
          | 0 0 0 0 | 0 | 0 0 0 0
                1
```

Figure 3.5 Example Of Serial Algorithm

After arrangement of groups the partial products are as shown below. The partial products so arranged can be added to get the final result.

```
  1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0
          0
-------------------------------------------
  1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
```

So the final output is binary 100000010000000.The operands are 100000000001 and 100000000000 whose binary equivalent is 128 and 129. Product of 128 and 129 are 16512 whose binary equivalent is 100000010000000 and thus the verification.

CHAPTER 4
THE PROPOSED SERIAL ACCUMULATOR

Accumulation is an integral part of serial multiplier design. A typical accumulator is simply an adder that successively adds the current input with the value stored in its internal register. Generally, the adder can be a simple RCA but the speed of accumulation is limited by the carry propagation chain. The accumulation can be speed up by using a CSA with two registers to store the intermediate sum and carry vectors, but a more complex fast vector merged adder is needed to add the final outputs of these registers. In either case, the basic functional unit is an FA cell. A new approach to serial accumulation of data by using asynchronous counters is suggested which essentially count the number of 1's in respective input sequences (columns).

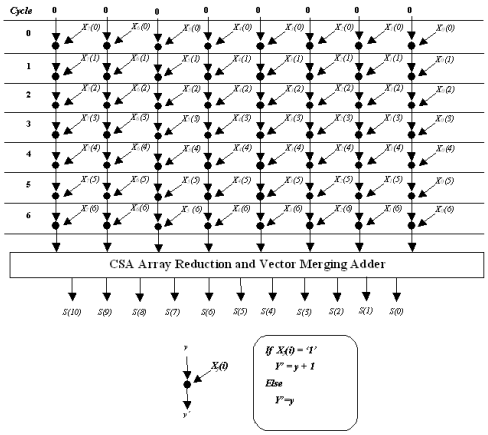


Figure 4.1 Dependency Graph Of Proposed Accumulator

An m bit accumulator can be realized with dedicated counters to concurrently accumulate the 1's in each bit position. The dependency graph (DG) of such a scheme with m=8 and n=7 is shown in Fig. 4.1.

An example illustrating the serial accumulation of the inputs, "10110001," "00000001," "11001101," "11100011," "01110001," "11001101," and "10001101" in the DG is shown in Fig 4.2.

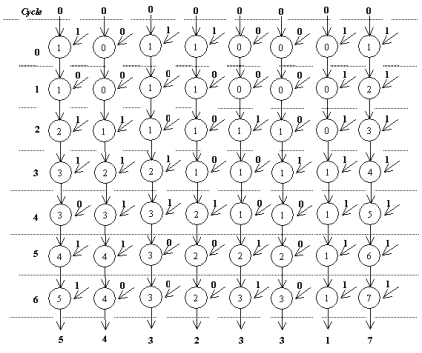


Figure 4.2 Example for fig 4.1

The accumulated output after each cycle is indicated by the integer in each node and the final accumulated output in each column can be represented by a 3-bit binary number. At the end of the th iteration, each counter produces an n bit binary weighted output. By arranging all counter output bits of the same positional weight in the same column with the least significant bit (LSB) of each counter output aligned at the jth column, the result of the accumulation can be obtained by column compressing the

counter outputs with a CSA tree structure of height h. Note that the height of the CSA tree would be a binary exponent of should the summation be carried out without the 1'scounters. The architecture of the accumulator corresponding to Fig. 4.2 is shown in Fig 4.3 . The bits of the input operands are serially fed into their corresponding counters from column 0 (right-most in Fig. 5) to column 7. These counters execute independently and concurrently. In each cycle of accumulation, a new operand is loaded and the counters corresponding to the columns that have a 1 input are incremented. The counters can be clocked at high frequency and all the operands will be accumulated at the end of the nth clock. The final outputs of the counters need to be further reduced to only two rows of partial products by a CSA tree. A carry propagate adder is then used to obtain the final sum.

In Fig. 4.3, the counters C are used to count the number of 1's in a column. Each of them is a simple DFF-based ripple counter. The clock is provided to the first DFF and all the other DFFs are triggered by the preceding DFF outputs. The CS block stands for carry save adders.

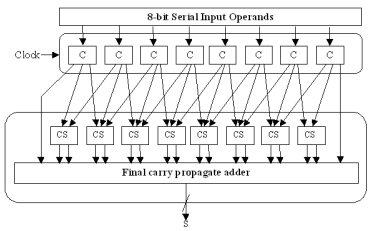


Figure 4.3 Architecture of Accumulator with Asynchronous 1's counters and Carry save adders

4.1 Asynchronous 1's Counters

The counter is designed using Asynchronous counters using D flip flops. The clock input is synchronized with the input data rate and thus the operands can be accumulated with a high frequency defined by the setup time and propagation delay of a DFF. Moreover, the counters change states only when the input is "1," which leads to low switching power .This simple and efficient bit accumulation technique is used to design the proposed serial-serial multiplier. A typical 3 bit 1's counter is shown in Fig. 4.4

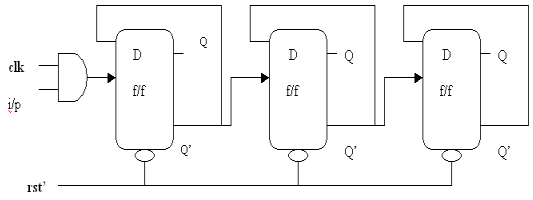


Figure 4.4 Architecture of Asynchronous 1's counters

CHAPTER 5
THE PROPOSED SERIAL-SERIAL MULTIPLIER

This Chapter proposes a new technique of generating the individual row of partial products by considering two serial inputs, one starting from the LSB and the other from MSB. Using this feeding sequence and the proposed counter-based accumulation technique presented inchapter4, it takes only n cycles to complete the entire partial product generation and accumulation process for an nxn multiplication. The theoretical underpinning of this design is elaborated as follows.

The product of two bit unsigned binary numbers and can be expressed as

$$P = X.Y = \sum_{i=0}^{n-1} \sum_{j}^{n-1} X_i Y_j 2^{i+j} \qquad (1)$$

where X_i and Y_j are the ith and jth bits of X and Y, respectively, with bit 0 being the LSB.

By reversing the sequence of index of (1) and rearranging we get

$$P = \Sigma_r PP_r \qquad (2)$$

Therefore $PPr = PP_r^L + PP_r^C + PP_r^R$

Where

$$PP_r^L = \{ 0 \quad r = 0$$
$$\Sigma_k x_{n-k-1} . y_r . 2^{n+r-k-1}, \quad r = 1,2,\ldots,n-1\}$$

$$PP_r^C = \{ 0 \quad r = 0$$
$$\Sigma x_{n-r-1} . y_r . 2^{n-1}, \quad r = 1,2,...n-1\} \text{ and}$$

$$PP_r^R = \{ 0 \quad r = 0$$
$$\Sigma_k x_{n-r-1} . y_{r-k-1} . 2^{n-k-2}, \quad r = 1,2,....n-1 \}$$

If X is fed MSB (bit 7) first and Y is fed LSB(bit0) first, then in cycle r,PP_r^C is a partial product bit generated by the current input bits $X_{n-r-1}$and Y_r,and PP_r^L are partial product bits of the current input bit,Y_r and each of the preceding input bits of X and $PP_r^R$are the partial product bits of the input bit, X_{n-r-1} and each of the preceding input bits of Y.By appropriately sequencing the input bits of X and Y into a shift register, one PPr in each cycle can be generated. Consequently, P can be obtained in n cycles.

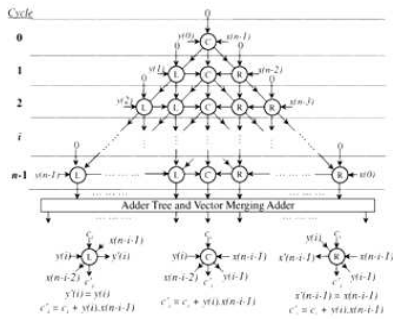


Figure 5.1 DG of nxn serial-serial partial product generation and accumulation.

The DG shown in Fig. 5.1 illustrates the complete operation of the PP generation and accumulation for a general nxn multiplication. Each node (circle) in the DG represents a binary counter and an ancillary AND gate to generate a partial product bit. All nodes are identical in functionality and have three inputs and three outputs, except that the data are propagated in different directions as depicted in Fig. 5.1

It can be seen from Fig. 3.2 that the nodes (L) on the left of the middle column have the identical properties of shifting X_{n-i-1} in cycle r computing the PP bit and the counter output. Similarly the nodes(R) on the right of the middle column exhibit identical property but shift Yi instead. The nodes (C) on the middle column receive a new pair of input bits Xn-i-1 and Y0 in each clock cycle, process the pair to form the PP bit and propagate and to the left and right nodes respectively. At cycle n-1 the registered output of the nodes contains the sum of all 1's in respective columns.

The complete architecture of the proposed counter-based serial-serial multiplier (8x8) is shown in Fig. 5.2.The functionality of the nodes in Fig.5.1 is mapped to the structure of Fig. 5.2 with two shift registers to perform the left and right shift, 2n-1 counters of different bit widths sufficient to accumulate the number of 1's in their respective columns and an array of 2n-1 AND gates to generate the PP bits serially. . It is evident that for an 8x8 multiplication, the middle column has a maximum height of 8 and a 4-bit counter is sufficient to account for the maximum column sum. The column height decreases gradually to either side and the counter width also decreases correspondingly.

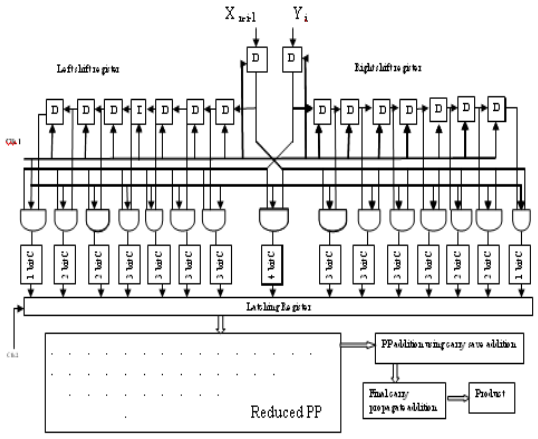


Figure 5.2 Proposed Architecture for 8X8 unsigned serial multiplier

The operation of the multiplier can be explained with the aid of the structure in fig.5.2. A PP bit corresponding to the middle column of the PP is produced by the center AND gate when a new pair of input bits (Xn-i-1 and) is latched by the two DFFs (top middle) in each clock cycle. In the next cycle ,X n-i-1 and Yi are shifted to the left and right, respectively, to produce the partial product bits with another pair of input bits Yi+1 and X n-i-2 by the array of AND gates. The AND gates are gated by clock to ensure that the outputs driving the counters are free of glitches. Each counter changes state at the rising edge of the clock line only if a "1" is produced by its driving AND gate. After cycles, the counters hold the sums of all the 1's in the respective columns and their outputs are latched to the second stage for summation. The latched outputs are wired to the correct FAs and HAs (half adders) according to the positional weights of the output bits produced by the counters. From Fig.3.2, it is observed that the column height has been reduced from 8 to 4 and the final product P can be obtained with two stages of CSA tree and a final RCA. . This drastic reduction in column height leads to a much simpler CSA tree, and hence reducing the overall hardware complexity and power consumption. The latching register between the counter and the adder stages not only makes it possible to pipeline the serial data accumulation and the CSA tree reduction, but also prevents the spurious transitions from propagating into the adder tree. Two clocks, namely Clock 1 and Clock 2, are employed to synchronize the data flow between the two stages. The counter stage is driven by to process the inputs at high speed as the critical path is defined only by the delay of the And gate and the D Flip flop. The counters and shift registers are reset to "0" in every cycles to allow a new set of operands to be loaded. Clock 2 is derived from Clock 1to drive the latching register.

5.1 Partial Product reduction Using Carry Save addition

For the partial product reduction Carry save addition is used. This helps in reducing the delay. The straightforward way of adding together m numbers (all n bits wide) is to add the first two, then add that sum to the next, and so on. This requires a total of m − 1 additions, for a total gate delay of O(mlg n) (assuming look ahead carry adders).

Instead, a tree of adders can be formed, taking only O(lgm · lg n) gate delays. Using carry save addition, the delay can be reduced further still. The idea is to take 3 numbers that we want to add together, x + y + z, and convert it into 2 numbers c + s such that x + y + z = c + s, and do this in O(1) time. The reason why addition can not be performed in O(1) time is because the carry information must be propagated. In carry save addition, we refrain from directly passing on the carry information until the very last step.

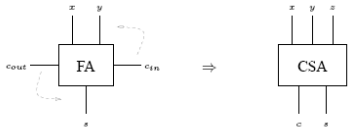


Figure 5.2 Full Adder and Carry Save Adder

Figure 5.2 shows a full adder and a carry save adder. A carry save adder simply is a full adder with the c_{in} input renamed to z, the z output (the original "answer" output) renamed to s, and the c_{out} output renamed to c.

In this project array type carry save addition is used.. The partial product formed using the serial algorithm is reduced to 3 stages using the carry save adder and the final stage is reduced using a carry propagation addition. The combination of carry save adder and ripple carry adder gives the fastest result. So that combination is preferred.

Figure 5.3 shows the 3 stages of PP reduction.

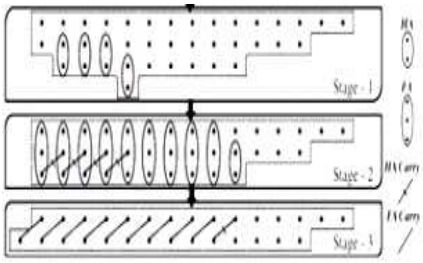


Figure 5.3 Stages of reduction of PP

The grouping of 2 shows half adder and the grouping of 3 shows full adder. For stage 3 reduction a simple ripple carry addition is carried out.

A ripple-carry adder works in the same way as pencil-and-paper methods of addition. Starting at the rightmost (least significant) digit position, the two corresponding digits are added and a result obtained. It is also possible that there may be a carry out of this digit position (for example, in pencil-and-paper methods, "9+5=4, carry 1"). Accordingly all digit positions other than the rightmost need to take into account the possibility of having to add an extra 1, from a carry that has come in from the next position to the right. This means that no digit position can have an absolutely final value until it has been established whether or not a carry is coming in from the right. Moreover, if the sum without a carry is 9 (in pencil-and-paper methods) or 1 (in binary arithmetic), it is not even possible to tell whether or not a given digit position is going to pass on a carry to the position on its left. At worst, when a whole sequence of sums comes to ...99999999... (in decimal) or ...11111111... (in binary), nothing can be deduced at all

until the value of the carry coming in from the right is known, and that carry is then propagated to the left, one step at a time, as each digit position evaluated "9+1=0, carry 1" or "1+1=0, carry 1". It is the "rippling" of the carry from right to left that gives a ripple-carry adder its name, and its slowness. When adding 32-bit integers, for instance, allowance has to be made for the possibility that a carry could have to ripple through every one of the 32 one-bit adders.

CHAPTER 6
EXTENSION OF LOGIC FOR SIGNED MULTIPLICATION

Most digital systems operate on signed numbers commonly represented in 2's complement. In this section, the unsigned serial-serial multiplication architecture of Fig. 5.2 is extended to deal with signed multiplication. The concept is that with the same unsigned architecture if signed multiplication is carried out, the difference in the final answer compared to the original signed multiplication is found out and that difference is added to the architecture such that the difference is nullified. Here the signed multiplication is compared with the 2's complement multiplication using Baugh Wooley Algorithm.

6.1 Baugh Wooley Algorithm

Two's complement is the most popular method of representing signed integers in computer science. It is also an operation of negation (converting positive to negative numbers or vice versa) in computers which represent negative numbers using two's complement. Its use is ubiquitous today because it does not require the addition and subtraction circuitry to examine the signs of the operands to determine whether to add or subtract, making it both simpler to implement and capable of easily handling higher precision arithmetic.

Two's complement and one's complement representations are commonly used since arithmetic units are simpler to design. In an n-bit binary number, the most significant bit is usually the 2n−1 s place. But in the two's complement representation, its place value is negated; it becomes the −2n−1 s place and is called the sign bit. If the sign bit is 0, the value is positive; if it is 1, the value is negative. To negate a two's complement number, invert all the bits then add 1 to the result. If all bits are 1, the value is −1. If the sign bit is 1 but the rest of the bits are 0, the value is the most negative number, −2n−1 for an n-bit number. The absolute value of the most negative number

cannot be represented with the same number of bits because it is greater than the most positive number that two's complement number by exactly 1.

Figure 6.1, shows two's complement and one's complement representations.

+N	Positive integers	-N	Negative integers		
			Sign Magnitude	2's Complement	1's Complement
+0	0000	-0	1000	……	1111
+1	0001	-1	1001	1111	1110
+2	0010	-2	1010	1110	1101
+3	0011	-3	1011	1101	1100
+4	0100	-4	1100	1100	1011
+5	0101	-5	1101	1011	1010
+6	0110	-6	1110	1010	1001
+7	0111	-7	1111	1001	1000
+8	……	-8	……	1000	…….

Figure 6.1 2's complement and 1's complement representations

A two's complement 8-bits binary numeral can represent every integer in the range −128 to +127. If the sign bit is 0, then the largest value that can be stored in the remaining seven bits is 27 − 1, or 127.Using two's complement to represent negative numbers allows only one representation of zero, and to have effective addition and subtraction while still having the most significant bit as the sign bit.

Baugh-Wooley technique was developed to design direct multipliers for two's complement numbers. When multiplying two's complement numbers directly, each of the partial products to be added is a signed number. Thus, each partial product has to be sign-extended to the width of the final product in order to form the correct sum by the Carry Save Adder tree. According to the Baugh- Wooley approach, an efficient

method of adding extra entries to the bit matrix is suggested to avoid having to deal with the negatively weighted bits in the partial product matrix.

Partial product array's of two's complement multiplication of 5-bits x 5- bits are shown in Figure 6.1.2

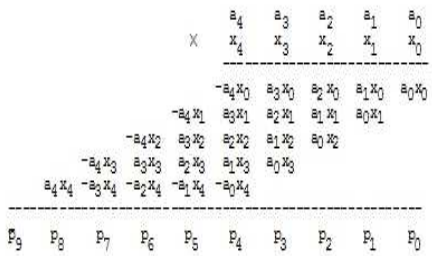


Figure 6.2 2's complement multiplication

Here is how the algorithm works. Knowing that the sign bit in two's complement numbers has a negative weight, the entry the term can be written in terms of − a4 x0 .

$$-a_4 x_0 = a_4(1 - x_0) - a_4 = \overline{a_4} x_0 - a_4$$

Hence the term $-a_4 x_0$ is replaced by $\overline{a_4} x_0$ and $-a_4$. If a_4 is used instead of $-a_4$, the column sum increases by $2a_4$. So $-a_4$ is inserted in the next highest column to compensate the effect of $2a_4$. The same is done for $a_4 \overline{x_1}$, $a_4 \overline{x_2}$ and $a_4 \overline{x_3}$. In each column a_4 and $-a_4$ cancel out each other. The column P8 gets an –a4 entry which can be

replaced by a_4 -1.This can be repeated for all entries. Now there are two -1's in the eighth column which is equivalent to a -1 entry in P9 and can be replaced by with a 1 and a borrow in to the non existing tenth column.

$a_4\ a_3\ a_2\ a_1\ a_0$

x

$x_4\ x_3\ x_2\ x_1\ x_0$

$1\quad \overline{a_4 x_0}\ a_3 x_0\ a_2 x_0\ a_1 x_0\ a_0 x_0$

$a_4x_1\ a_3x_1\ a_2x_1\ a_1x_1\ a_0x_1$

$\overline{a_4x_2}\ a_3x_2\ a_2x_2\ a_1x_2\ a_0x_1$

$\overline{a_4x_3}\ a_3x_3\ a_2x_3\ a_1x_3\ a_0x_3$

$a_4x_4\ \overline{a_3x_4}\ \overline{a_2x_4}\ \overline{a_1x_4}\ \overline{a_0x_4}$

P9 P8 P7 P6 P5 P4 P3 P2 P1 P0

Figure 6.3 Baugh Wooley 2's complement signed multiplication

Baugh-Wooley method increases the height of the longest column by two, which may lead to a greater delay through the Carry Save Adder tree

6.2 Proposed Serial Signed Multiplier

By combining the Baugh Wooley and the serial algorithm, the architecture could be modified to perform signed multiplication. Using Baugh Wooley algorithm the equation for the multiplication of two signed numbers in two's

complement form can be written as

$$P' = X'Y' = \sum_{i}^{n-2}\sum_{j}^{n-2} X_i Y_j 2^{i+j} + 2^{n-1}\Big(\sum_{i}^{n-2}\overline{X_i Y_{n-1}}2^i + \sum_{i}^{n-2}\overline{X_{n-1}Y_j}2^j\Big) + X_{n-1}Y_{n-1}2^{2n-2} + 2^n - 2^{2n-1} \quad (1)$$

Using the proposed architecture the multiplication of the signed numbers can be written as

$$P = X.Y = \sum_{i}^{n-2}\sum_{j}^{n-2} X_i Y_j 2^{i+j} + 2n-1\Big(\sum_{i}^{n-2}\sum_{j}^{n-2}\overline{XiYn-12}i + \sum_{i}^{n-2}\overline{xn-1Yi2}j\Big) + X_{n-1}Y_{n-1}2^{2n-2} \quad (2)$$

The difference between 1 and 2 can be written as

$$\Psi = P' - P$$

$$= 2^{n-1}\Big(\sum_{i}^{n-2} X_i Y_{n-1}2^i + \sum_{i}^{n-2} X_{n-1}Y_j 2^j\Big) - 2^{n-1}\Big(\sum_{I}^{N-2} X_i Y_{n-1}2^i + \sum_{i}^{n-2} X_{n-1}Y_i 2^j\Big) + 2^n\text{-}2^{2n-1} \quad (3)$$

The above expression could be simplified as

$$\Psi = 2^{n-1}\Big[\Big(\sum_{i}^{n-2}(1-2X_iY_{n-1})2^i\Big) + \Big(\sum_{j}^{n-2}(1-2X_{n-1}Y_j)2^j\Big)\Big] + 2^n - 2^{2n-1} \quad (4)$$

To extend the unsigned multiplier architecture for signed multiplication without introducing a high over-head, the difference expressed in 6must be simplified. Since $\Sigma_i^{n-2} 2^i = \Sigma_j^{n-2} 2^j$, the following summation terms embedded in (4) can be simplified by the closed form expression of a geometric progression

$$2^0 + 2^1 + 2^2 + ... + 2^{2n-2} = 2^{2n-1} - 1$$

Therefore

$$2^{n-1}\Big(\sum_{i}^{n-2}2^i + \sum_{j}^{n-2}2^j\Big) = 2^{2n-1} - 2^n \quad (5)$$

hence Ψ can be written as

$$\Psi = -2^n\Big(\sum_{i}^{n-2} X_i Y_{n-1}2^i\Big) + \Big(\sum_{j}^{n-2} X_{n-1}Y_j 2^j\Big) \quad (6)$$

The difference Ψ is added to the proposed architecture such that the architecture can be used for signed multiplication. Thus we can write

$$P' = P + \Psi \quad (7)$$

In the proposed PP generation method,Yn-1 arrives only in cycle n-1. The generation of $\Sigma_i^{n-2} X_i Y_{n-1} 2^i$ has to be delayed until cycle n-1. The remaining terms $\Sigma_j^{n-2} X_{n-1} Y_j 2^j$ can be computed during the initial n-1cycles. Hence, the difference can be corrected in the CSA tree. It is trivial that a n-1-bit shift register, a NAND gate and several FAs are required for adding Ψ.

The architecture of the proposed 2's complement serial-serial multiplier is depicted in Fig.6.4

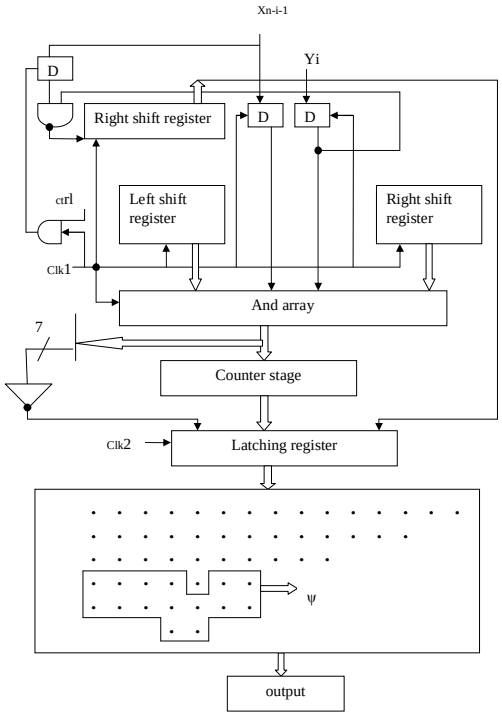


Figure 6.4 Proposed Serial Signed Multiplier

It is noted that the input can be fed at the same speed as the unsigned multiplier. A control input Ctrl is required to latch Xn-1 in the first clock cycle to generate $\Sigma jn^{-2}X_{n-1}\,Y_j\,2^j$ serially in n-1cycles. The bits of to be added in the CSA tree are marked in Fig. 13. The addition of the term raises the height of CSA tree by only two bits regardless of the word length of the operands.

CHAPTER 7
EXISTING SERIAL MULTIPLIER

Serial multipliers are popular for their low area and power and are more suitable for bit-serial signal processing applications with I/O constraints and on-chip serial-link bus architectures. They are broadly classified into two categories, namely serial-serial and serial-parallel multiplier. In a serial serial multiplier both the operands are loaded in a bit-serial fashion, reducing the data input pads to two. On the other hand, a serial-parallel multiplier loads one operand in a bit-serial fashion and the other is always available for parallel operation . Several serial-parallel multipliers have been developed over the years. It is observed that all the reported multipliers have a common computational unit known as the 5:3 counter in the critical path. In addition, there are also DFFs and AND gates in the critical path, which lower the operation speed and limit the input bit rate. Many attempts have been made to reduce either the hardware cost or latency but there is no improvement on the critical path. Modifications are done in many multipliers but then the computational latency is 2n cycles. Most of them are based on a carry save add shift (CSAS) structure. Here for comparison the existing serial CSAS multiplier is used . It can be observed that the critical path consists of an FA, a DFF, and an AND gate for the unsigned multiplier and an extra XOR gate for the 2's complement multiplier .Here a CSAS multiplier is implemented using the right shift algorithm.

7.1 Right Shift Algorithm

The basic algorithm to be used is the same as the one we use to multiply decimal numbers. We multiply each digit of the multiplier times the multiplicand to form a partial product and we add all of the partial products together to form the final product. This is illustrated on the left side of Fig. 8.1. Ordinarily, we produce all of the partial products and then add all of them together; however, this is not necessary and, instead, as each partial product is produced, it can be added to the sum of previous ones. Since adders in computers usually can only add two numbers

together, the addition of partial products must be done in the latter way. Binary multiplication of positive numbers can be done using the same algorithm, but it is simpler since each partial product is either zero or equal to the multiplicand properly aligned with respect to the multiplicand. This is illustrated on the right side of Fig. 7.1.1 for a 4-bit multiplicand and 4-bit multiplier.

```
   0110 ->Multiplicand
 X 1101 ->Multiplier
--------
0000110 ->Partial product1
0000000 ->Partial product2
0011000 ->Partial product3
0110000 ->Partial product4
1001110 ->Final product
```

Figure 7.1 Basic Multiplication Algorithm

As presented in Fig. 7.1 the addition of the partial products requires an adder that is twice as long as the length of the multiplicand and multiplier. Since a portion (almost half) of each partial product is 0, this is inefficient. (Note that the nonzero portion of each addition is just for the same number of bits as contained in the multiplicand, except for a possible carry into the next more significant position.) The double length adder can be avoided by shifting the accumulated partial products to the right with respect to the multiplicand rather than shifting the multiplicand to the left with respect to the accumulated partial products.

An example of this algorithm is shown in figure 7.1.2. The initial partial sum is zero.


```
Bit 0 of Adder
   ↓
0000  XXXX-- -> Initial partial sum (zero)
+0110  ---- --------> PP0
0110  XXXX ---> PP0 + partial sum (zero)
0011  0XXX ----> Shift partial sum right
+0000  ---- --------> PP1
0011  0XXX ----> PP1 + partial sum
0001  10XX -----> Shift partial sum right
+0110  --------
0111  10XX -----> PP2 + partial sum
0011  110X ------> Shift partial sum right
+0110  --------
1001  110X -------> PP3 + partial sum
0100  1110 --------> Shift partial sum right (final
                                        answer)
        ↑
Bit 0 of Partial Sum
```

Figure 7.2 Add and Shift Multiply Algorithm

Since bit 0 of the multiplier is 1, the first partial product (PP0) equals the multiplicand. This is added to the left half of the initial partial sum, and the partial sum is shifted right.

Since bit 1 of the multiplier is 0, PP1 is zero and after addition to the partial sum, the partial sum is shifted right. This is repeated for bits 2 and 3 of the multiplier. At each step the new PP is either zero or the multiplicand and it is added to the upper half of the partial sum. Then the partial sum is shifted 1 bit to the right. This Add and Shift algorithm is done once for each bit of the multiplier.

7.2 CSAS Signed Multiplier

CSAS stands for Carry Save Add and Shift Multiplier. The CSAS technique accepts multiplier bits serially (lsb first) and produces outputs serially (lsb first). Multiplication of two n bit unsigned numbers requires 2n clock cycles to complete the process out of which n clocks are used for n-row carry-save additions, and the other n clocks are utilized only to propagate the remaining carries.

Almost all signal processing applications demand a considerable number of multiplications. Some of these applications require the multiplication to be performed at a faster rate, and others concentrate on less hardware and moderate speed. In order to meet the demand for high speed, various parallel array multiplication algorithms have been proposed by a number of authors . The array multipliers use a large amount of hardware, consequently consuming a large amount of power. The applications which aim at hardware simplicity traditionally use, in one form or another, the add-shift method for multiplication. Referring to Fig.7.1.1, the basic add-shift with the ripple-carry propagation technique adds, one row at a time, the multiplication matrix to the partial product using an n bit adder. In every addition, the carry is allowed to propagate full length. This carry propagation, along with n-row addition time, contributes largely to the slow speed. The speed can be improved by using the carry look ahead technique, which introduces more hardware in the implementation.

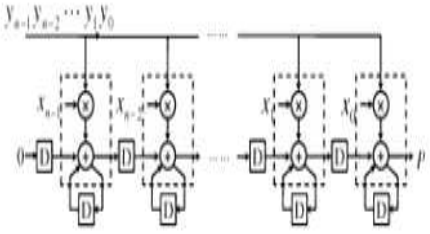
The figure below shows the basic PP formation of 4x4 multiplication

```
                a4  a3  a2  a1
            x
                b4  b3  b2  b1
-------------------------------------
                a4b1 a3b1 a2b1  a1b1
            a4b2 a3b2  a2b2 a1b2
        a4b3 a3b3 a2b3  a1b3
    a4b4 a3b4 a2b4 a1b4
-------------------------------------
P8  P7   P6   P5   P4   P3    P2   P1
```

Figure 7.3 Partial product matrix

The partial products are formed based on the normal multiplication and after each pp formation . the PP is shifted one position to the left and is continued till the last row. This is shown in the above figure.

The working of CSAS multiplier is based on the right shift algorithm. A simple CSAS multiplier implementation for positive numbers is shown in Fig. 7.4



.

Figure 7.4 CSAS Unsigned Multiplier

For n bit multiplication it takes 2n clock pulses to complete the process. At the end of the first n clocks the structure contains a sum word {Sn(n), Sn-1(n),…. S2(n)} and a carry word {C.n-1,(n), Cn-2(n), , C,(n)} the sum of which will yield n msb's of the product. This sum is accomplished by the next n clocks. Therefore, out of 2n clocks the last n clocks are expended only to propagate the carries residing in the feedback latches.

Figure 7.1.3 shows a CSAS signed multiplier

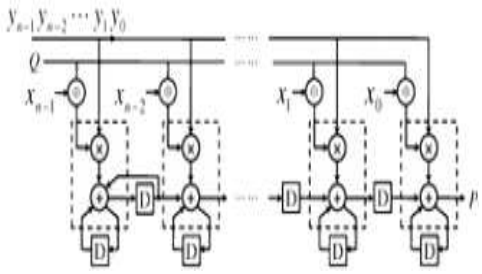


Figure 7.5 CSAS Signed Multiplier

Fig.7.4 shows the unsigned and 2's complement serial-parallel multiplier based on the CSAS structure. It can be observed that the critical path consists of an FA, a DFF, and an AND gate for the unsigned multiplier in Fig. 7.4 and an extra XOR gate for the 2's complement multiplier in Fig. 7.5. Although the sampling frequency has been improved in the serial-parallel multipliers and the total number of computational cycles is halved, one of the operands needs to be loaded in parallel. Recently, there has not been any new development in serial-serial multiplier design due to the maturity of conventional architectures.

CHAPTER 8
SIMULATION RESULTS

The simulation of this project has been done using ModelSim XE III 6.2g and Xilinx ISE 9.1i.

ModelSim is a simulation tool for programming {VLSI} {ASIC}s, {FPGA}s, {CPLD}s, and {SoC}s. Modelsim provides a comprehensive simulation and debug environment for complex ASIC and FPGA designs. Support is provided for multiple languages including Verilog, SystemVerilog, VHDL and SystemC. The following diagram shows the basic steps for simulating a design in ModelSim.

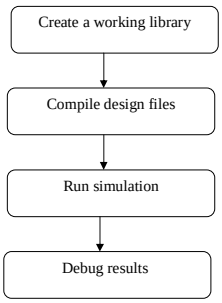


Figure 8.1 Basic Steps of Simulation in Model Sim

In ModelSim, all designs, be they VHDL, Verilog, or some combination thereof, are compiled into a library. We typically start a new simulation in ModelSim by creating a working library called "work". "Work" is the library name used by the compiler as the default destination for compiled design units. After creating the working library, we've to compile your design units into it. The ModelSim library format is compatible across all supported platforms. We can simulate our design on any platform without having to recompile design. With the design compiled, invoke the simulator on a top-level module (Verilog) or a configuration or entity/architecture pair (VHDL). Assuming the design loads successfully, the simulation time is set to zero, and enter a run command to begin simulation. If the results are not as expected , we can use ModelSim's robust debugging environment to track down the cause of the problem.

Xilinx was founded in 1984 by two semiconductor engineers, Ross Freeman and Bernard Vonderschmitt, who were both working for integrated circuit and solid-state device manufacturer Zilog Corp. The Virtex-II Pro, Virtex-4, Virtex-5, and Virtex-6 FPGA families are particularly focused on system-on-chip (SOC) designers because they include up to two embedded IBM PowerPC cores The ISE Design Suite is the central electronic design automation (EDA) product family sold by Xilinx. The ISE Design Suite features include design entry and synthesis supporting Verilog or VHDL, place-and-route (PAR), completed verification and debug using Chip Scope Pro tools, and creation of the bit files that are used to configure the chip .Xilinx is a synthesis tool which converts schematic/HDL design entry into functionally equivalent logic gates on Xilinx FPGA, with optimized speed & area. So, after specifying behavioral description for HDL, the designer merely has to select the library and specify optimization criteria; and Xilinx synthesis tool determines the net list to meet the specification; which is then converted into bit-file to be loaded onto FPGA PROM . Also, Xilinx tool generates post-process simulation model after every implementation step, which is used to functionally verify generated net list after processes, like map, place & route.

8.1 SIMULATION RESULTS

The following sections shows the simulation results of the existing and the proposed multipliers.

8.1.1 8x8 Unsigned CSAS Multiplier

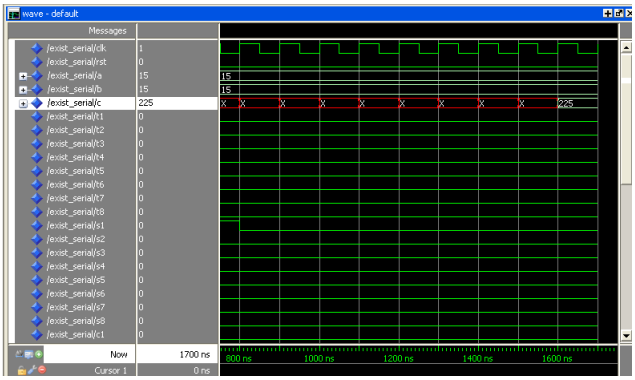


Figure 8.1 Simulation Result of 8x8 CSAS unsigned multiplier

In the Simulation result , clk denotes the clock and rst denotes reset, a and b are the serial input operands and c is the output .t1 to t8 etc are the and outputs , s1 to 18 are the full adder sum output and c1 to c8 are the carries.

a = 15
b = 15
c = 225

8.1.2 8x8 Signed CSAS Multiplier

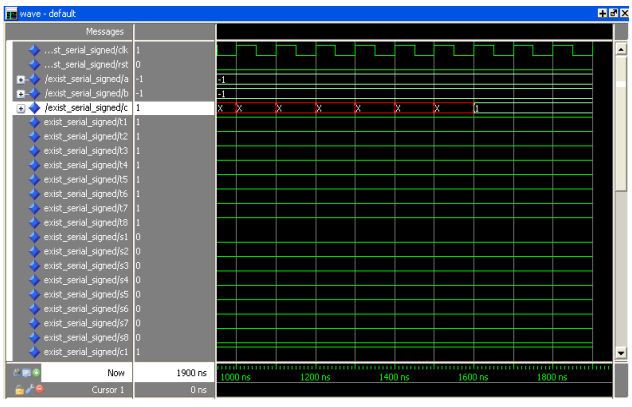


Figure 8.2 Simulation Result of 8x8 CSAS Signed multiplier

In the Simulation result , clk denotes the clock and rst denotes reset, a and b are the serial input operands and c is the output .t1 to t8 etc are the and outputs , s_1 to s_8 are the full adder sum output and c1 to c8 are the carries

a = -1
b = -1
c = +1

8.1.3 8x8 Unsigned Serial – Serial Multiplier

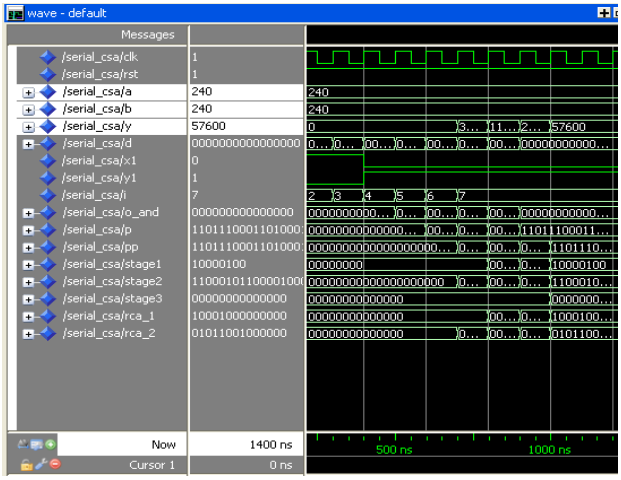


Figure 8.3 Simulation Result of 8x8 Serial unsigned multiplier

Here in the waveform a and b are the serial input operands, clk and rst are the clock and reset. 0-and denotes the output of and array, p and pp are the input and output of the latching register, Stage 1,2 and 3 are the PP reduction stages using CSA and rca_1 and rca_2 are the carry propagation adder inputs and y is the output (product).

a = 240
b = 240
y = 57600

8.1.4 8x8 Signed serial-serial Multiplier

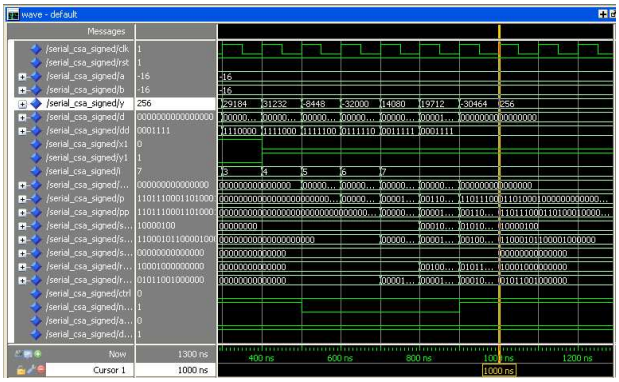


Figure 8.4 Simulation Result of 8x8 Serial signed multiplier

Here in the waveform a and b are the serial input operands, clk and rst are the clock and reset .d and dd denotes the outputs of the additional set of d flip flops added 0_and denotes the output of and array, p and pp are the input and output of the latching register, Stage 1,2 and 3 are the PP reduction stages using CSA and rca_1 and rca_2 are the carry propagation adder inputs and y is the output (product).

a = -16
b = -16
y = + 256

8.1.5 16x16 Unsigned serial-serial Multiplier

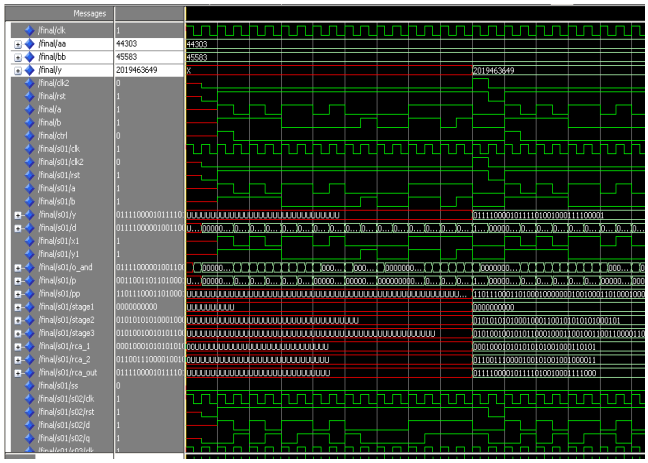


Figure 8.5 Simulation Result of 16x16 Serial signed multiplier

Here in the waveform a and b are the serial input operands, clk and rst are the clock and reset. 0-and denotes the output of and array, p and pp are the input and output of the latching register, Stage 1,2 and 3 are the PP reduction stages using CSA and rca_1 and rca_2 are the carry propagation adder inputs and y is the output (product).

a = 44303
b = 45583
y = 2019463649

8.2 SYNTHESIS REPORT

This section describes the area and power reports of the existing and the proposed serial multipliers.

8.2.1 Area Report of existing 8x8 CSAS unsigned multiplier

Design Summary

Number of errors: 0

Number of warnings: 0

Logic Utilization:

Total Number Slice Registers: 66 out of 2,400 2%

Number used as Flip Flops: 14

Number used as Latches: 52

Number of 4 input LUTs: 111 out of 2,400 4%

Logic Distribution:

Number of occupied Slices: 78 out of 1,200 6%

Number of Slices containing only related logic: 78 out of 78 100%

Number of Slices containing unrelated logic: 0 out of 78 0%

Total Number 4 input LUTs: 147 out of 2,400 6%

Number used as logic: 111

Number used as a route-thru: 36

Number of bonded IOBs: 33 out of 92 35%

Number of GCLKs: 1 out of 4 25%

Number of GCLKIOBs: 1 out of 4 25%

Total equivalent gate count for design: 1,311

Additional JTAG gate count for IOBs: 1,632

Peak Memory Usage: 119 MB

8.2.2 Area Report of existing 8x8 CSAS signed multiplier

Design Summary

Number of errors: 0

Number of warnings: 1

Logic Utilization:

Total Number Slice Registers: 67 out of 2,400 2%

Number used as Flip Flops: 15

Number used as Latches: 52

Number of 4 input LUTs: 134 out of 2,400 5%

Logic Distribution:

Number of occupied Slices: 92 out of 1,200 7%

Number of Slices containing only related logic: 92 out of 92 100%

Number of Slices containing unrelated logic: 0 out of 92 0%

Total Number 4 input LUTs: 170 out of 2,400 7%

Number used as logic: 134

Number used as a route-thru: 36

Number of bonded IOBs: 33 out of 92 35%

IOB Latches: 1

Number of GCLKs: 1 out of 4 25%

Number of GCLKIOBs: 1 out of 4 25%

Total equivalent gate count for design: 1,462

Additional JTAG gate count for IOBs: 1,632

Peak Memory Usage: 118 MB

8.2.3 Area Report of Proposed 8x8 Serial-Serial unsigned multiplier

Design Summary

Number of errors: 0

Number of warnings: 16

Logic Utilization:

Total Number Slice Registers: 136 out of 2,400 5%

Number used as Flip Flops: 100

Number used as Latches: 36

Number of 4 input LUTs: 81 out of 2,400 3%

Logic Distribution:

Number of occupied Slices: 158 out of 1,200 13%

Number of Slices containing only related logic: 158 out of 158 100%

Number of Slices containing unrelated logic: 0 out of 158 0%

Total Number of 4 input LUTs: 81 out of 2,400 3%

Number of bonded IOBs: 32 out of 92 34%

IOB Latches: 2

Number of GCLKs: 2 out of 4 50%

Number of GCLKIOBs: 1 out of 4 25%

Total equivalent gate count for design: 1,476

Additional JTAG gate count for IOBs: 1,584

Peak Memory Usage: 118 MB

8.2.4 Area Report of Proposed 8x8 Serial-Serial signed multiplier

Design Summary

Number of errors: 0

Number of warnings: 17

Logic Utilization:

Total Number Slice Registers: 160 out of 2,400 6%

Number used as Flip Flops: 110

Number used as Latches: 50

Number of 4 input LUTs: 123 out of 2,400 5%

Logic Distribution:

Number of occupied Slices: 189 out of 1,200 15%

Number of Slices containing only related logic: 189 out of 189 100%

Number of Slices containing unrelated logic: 0 out of 189 0%

Total Number of 4 input LUTs: 123 out of 2,400 5%

Number of bonded IOBs: 32 out of 92 34%

IOB Latches: 2

Number of GCLKs: 2 out of 4 50%

Number of GCLKIOBs: 1 out of 4 25%

Total equivalent gate count for design: 1,878

Additional JTAG gate count for IOBs: 1,584

Peak Memory Usage: 119 MB

The following table shows the comparison of logic utilization and logic distribution of the existing and proposed multipliers

Design summary		Existing Unsigned multiplier	Proposed Unsigned multiplier	Existing Signed multiplier	Proposed Signed multiplier
Logic Utilization	No: used as Flip flops	14	100	15	110
	No: used as Latches	52	36	52	50
	No: of 4 i/p LUTS	111 out of 2400	81 out of 2400	134 out of 2400	123 out of 2400
logic distribution	Total no: of 4 i/p LUTs	147 out of 2400	81 out of 2400	170 out of 2400	123 out of 2400
	total gate count	1311	1476	1462	1878

Figure 8.6 Comparison of the Existing and Proposed 8x8 Serial–Serial Multipliers

8.2.5 Power Report of existing 8x8 CSAS unsigned multiplier

Power summary:	I(mA)	P(mW)
Total estimated power consumption:		50
Vccint 2.50V:	18	44
Vcco33 3.30V:	2	7
Clocks:	14	34
Inputs:	2	5
Logic:	0	0
Outputs:		
Vcco33	0	0
Signals:	0	0
Quiescent Vccint 2.50V:	2	5
Quiescent Vcco33 3.30V:	2	7

8.2.6 Power Report of existing 8x8 CSAS signed multiplier

Power summary:	I(mA)	P(mW)
Total estimated power consumption:		53
Vccint 2.50V:	18	46
Vcco33 3.30V:	2	7
Clocks:	15	36
Inputs:	2	5
Logic:	0	0
Outputs:		
Vcco33	0	0
Signals:	0	0
Quiescent Vccint 2.50V:	2	5
Quiescent Vcco33 3.30V:	2	7

8.2.7 Power Area Report of Proposed 8x8 Serial-Serial unsigned multiplier

Power summary:	I(mA)	P(mW)
Total estimated power consumption:		53
Vccint 2.50V:	19	47
Vcco33 3.30V:	2	7
Clocks:	13	32
Inputs:	2	5
Logic:	1	3
Outputs:		
Vcco33	0	0
Signals:	1	2
Quiescent Vccint 2.50V:	2	5
Quiescent Vcco33 3.30V:	2	7

8.2.8 Power Area Report of Proposed 8x8 Serial-Serial signed multiplier

Power summary:	I(mA)	P(mW)
Total estimated power consumption:		57
Vccint 2.50V:	20	50
Vcco33 3.30V:	2	7
Clocks:	13	33
Inputs:	2	5
Logic:	2	5
Outputs:		
Vcco33	0	0
Signals:	1	2
Quiescent Vccint 2.50V:	2	5
Quiescent Vcco33 3.30V:	2	7

8.3 COMPARISON

This section shows the comparison of the existing and the proposed 8x8 serial-serial multipliers based on area , power and the no of clock cycles required for the output to occur.

Architecture	Area (Gate count)	Power (mW)	clk cycles required	% increase in Area
Existing CSAS Unsigned	1311	50	16	13%
Proposed Serial-Serial Unsigned	1476	53	8	
Existing CSAS Signed	1462	53	16	28%
Proposed Serial-Serial Signed	1878	57	8	

Figure 8.7 Comparison of the Existing and Proposed 8x8 Serial–Serial Multipliers

It can be seen that with an area overhead of 13 % the proposed unsigned multiplier generates the output with half the clocking speed when compared to the existing CSAS serial multiplier. Also it can be seen that the signed proposed multiplier produces the output with half the number of clock cycles required for the existing CSAS multiplier with an extra area overhead of 28%.

CHAPTER 7

CONCLUSION AND FUTURE SCOPE

In this project, a new method for computing serial-serial multiplication is introduced by using low complexity asynchronous counters. By exploiting the relationship among the bits of a partial product matrix, it is possible to generate all the rows serially in just n cycles for an nxn multiplication. Employing counters to count the number of 1's in each column allows the partial product bits to be generated on-the-fly and partially accumulated in place with a critical path delay of only an AND gate and a DFF. The counter-based accumulation reduces the PP height logarithmically and makes it possible to achieve an effective reduction rate of log n using an FA-based CSA tree The proposed counter-based multiplier outperforms many serial-serial and serial-parallel multipliers in speed but its hybrid architecture does carry an area overhead. Overall, the ADP is comparable to other serial-serial multipliers. Last but not least, the counter based approach has clear advantage of low I/O requirement and hence is most suitable for complex SoCs, advanced FPGAs and high-speed bit-serial applications.

FUTURE SCOPE

The future scope of this project is to implement the proposed Serial Serial Multiplier hardware architecture using Field-Programmable Gate Arrays (FPGAs).

REFERENCES

[1] Meher MR,Ching Chuen Jong;Chip Hong Cheng , "A High bit rate Serial-Serial multiplier with on the fly Accumulation by Asynchronous counters",IEEE Trans.VLSI,vol.19,pp.1733-1745,Sept 2010

[2] K. Z. Pekmestzi, P. Kalivas, and N. Moshopoulos, "Long unsigned number systolic serial multipliers and squarers," IEEE Trans. Circuits Syst. II, Brief Papers, vol. 48, no. 3, pp. 316-321, Mar. 2001.

[3] O. Nibouche, A. Bouridarie, and M. Nibouche, "New architectures for serial-serial multiplication," in Proc. IEEE Conf. Circuits Syst. (ISCAS), Sydney, Australia, 2001, vol. 2, pp. 705-708.

[4] M. R. Meher, C. C. Jong, and C. H. Chang, "High-speed and low power serial accumulator for serial/parallel multiplier," in Proc. IEEE Asia-Pacific Conf. Circuits Syst. (APCCAS), Macau,China, 2008, pp. 176-179.

[5] J R. Gnanasekaran, "A fast serial-parallel binary multiplier",*IEEE Trans. Comput.*, vol. C-34, no. 8, pp. 741-744, Aug. 1985.

[6] A. Aggoun, A.Ashur, and M. K. Ibrahim, "Area-time efficient serial-serial multipliers," in Proc.IEEE Conf. Circuits Syst. (ISCAS),Geneva, Switzerland, 2000, pp. 585-588.

[7] H. I. Saleh, A. H. Khalil, M. A. Ashour, and A. E. Salama, "Novel serial parallel multipliers," IEEE Proc-Circuits Devices Syst., vol. 148, no. 4, pp. 183-189, Aug. 2001.

[8] D. Bailey, E. Soenen, P. Gupta, P. Villarrubia, and D. Sang, "Challenges at 45 nm and beyond," in Proc. IEEE/ACM Int. Conf. Comput.- Aided Des. (ICCAD), San Jose, CA, 2008, pp. 11-18.

[9] B. Parhami, Computer Arithmetic: Algorithms and Hardware Designs. New York: Oxford Univ. Press, 2009

[10] A. D. Booth, "A signed binary multiplication technique," Quarterly J. Mechan. Appl. Math., vol. 4, no. 2, pp. 236-240, Aug. 1951.

[11] R. Menon and D. Radhakrishnan,"High performance 5:2 compressor architectures," IEE Proc-Circuits Devices Syst., vol. 153, no. 5, pp. 447-452, Oct. 2006.

[12] R. Dobkin, A. Morgenshtein, A. Kolodny, and R. Ginosar, "Parallel vs.serial on-chip communication," in Proc. ACM Int. Workshop Syst. Level Interconnect Prediction (SLIP), Newcastle, U.K., 2008, pp. 43-50.

[13] R. F. Lyon, "Two's complement pipeline multipliers," IEEE Commun. Lett., vol. 24, no. 4, pp. 418-425, Apr. 1976.

[14] P. Ienne and M. A Viredaz, "Bit-serial multipliers and squarers," IEEETrans. Comput., vol. 43, no. 12, pp. 1445-1450, Dec. 1994.

[15] G. Bi and E. V. Jones, "High-performance bit-serial adders and multipliers," IEEE Proc G-Circuits Devices Systs, vol. 139, no. 1, pp. 109-113, Feb. 1992.

[16] M. Ghoneima, Y. Ismail, M. Khellah, J. Tschanz and V. De, "Serial link bus: A low power on chip bus architecture" IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 56, no. 9, pp. 2020-2032, Sep. 2009

[17] R. Dobkin, A. Morgenshtein, A. Kolodny, and R. Ginosar, "Parallel vs.serial on chip communication," in Proc. ACM Int. Workshop Syst. Level Interconnect Prediction (SLIP), Newcastle, U.K., 2008, pp. 43-50