DESIGN OF DADDYA MULTIPLIER WITH OPTIMIZED POWER USING ANT ARCHITECTURE

M. Sukanya¹, Dr. B. Rama Rao ², Y. Srinivasa Rao ³

¹PG Student [VLSI], Dept. of ECE, AITAM, Tekkali, A.P., India.
²Professor, Dept. of ECE, AITAM, Tekkali, A.P., India.
³Assistant professor Dept. of ECE, AITAM, Tekkali, A.P., India.

ABSTRACT:
One of the most important hardware blocks for the DSP systems is multiplier block. In digital filtering, communication and analysis of the digital signals i.e in DSP applications the key role is played by the multiplier. In present day digital applications are focused for being portable and can be used as portable devices which means the devices are upcoming battery powered. Thus power dissipation becomes the important constraint in designing a system. Typically the multipliers are the complex systems and requires by clock rates, for reducing delay of the design for satisfying overall design performance.

In this paper two different multipliers are designed based on ANT Architecture. The simulation and synthesis results are obtain by using XILINX ISE 12.3i. The modified Dadda multiplier and array multiplier are designed with combination of truncated multiplier. The multiplier circuit area in fixed width reduced precision replica can be lower by 39.99% and the power can be reduced.

Keywords: Truncated multiplier, Array multiplier, Dadda multiplier, Multiplexer.

[1] INTRODUCTION
The rapid growth of portable and wireless computing systems in modern years drives the need for ultralow power systems.
To lower the power dissipation, supply voltage scaling is widely used as an effective low-power technique since the power consumption in CMOS circuits is proportional to the square of supply voltage [1]. However, in deep-sub micrometer process technologies, noise interference problems have raised difficulty to design the reliable and efficient microelectronics systems; hence, the design techniques to enhance noise tolerance have been widely developed [2]. A novel algorithmic noise tolerant (ANT) technique [2] combined VOS main block with reduced-precision replica (RPR), which combats soft errors efficiently while achieving significant energy saving. Some ANT deformation designs are presented in [3]–[6] and the ANT design concept is further extended to system level in [7]. However, the RPR designs in the ANT designs of [3]–[4] are designed in a customized manner, which are not easily adopted and repeated. The RPR designs in the ANT designs of [5] and [6] can operate in a very fast manner, but their hardware complexity is too complex. As a result, the RPR design in the ANT design of [2] is still the a good number popular design because of its simplicity. However, adopting with RPR in [2] should still pay extra area overhead and power consumption. In this paper, we further proposed an easy way using the fixed-width RPR to replace the full-width RPR block in [2].

The problem with multipliers is, they are very costly and poor in overall performance. The performance speed of multiplier influences a computational problem [8]. Assume that we consider two unsigned binary numbers as X and Y with respective bit lengths of M and N. It is very useful to represent X and Y in binary notation for performing multiplication operation for them.

\[
x = \sum X_i 2^i \quad i = 0 \text{ to } M \\
Y = \sum Y_j 2^j \quad j = 0 \text{ to } N \\
Z = X \ast Y = \sum Z_k 2^k \quad k = 0 \text{ to } M + N - 1 \\
= ( \sum X_i 2^i \quad i = 0 \text{ to } M ) ( \sum Y_j 2^j \quad j = 0 \text{ to } N ) \\
= \sum (\sum X_i Y_j 2^{i+j}) \quad i = 0 \text{ to } M - 1, j = 0 \text{ to } N - 1
\]

The multiplication process will have two steps as generating partial products and addition for obtaining product of the two values. One of such easiest ways is to use a two input adder. For the two digits having a width of M bits N width, multiplication procedure requires M cycles and uses N-bit adders. The partial products will be added together by using shift add algorithm. The partial products are generated my multiplying every element (bit) in multiplicand with every element (bit) in multiplier. By adding all partial products by using shift and add method the final sum relates to the product of the values with M and N bits wide. On observing the actual scenario, multiplication for any radix is actually performed in binary and then converted to the respective radix. Partial products are produced by ending the multiplier and multiplicand, which is a copy of multiplier or a ‘zero’. It is because the binary value contains two digits as 0 and 1.

Speed: Multiplier should perform operation at high speed.
Area: Multiplier should occupy less number of slices and LUTs.

Power: Multiplier should consume less power [10].

By implementing a new method, comparable to the manual computing method, multiplication may be a bit faster using the technique. Partial products generation took little time, because all the partial products are produced in the same time and are arranged in an array. For completing and computing the multiplication and addition is done. The multiplication is shown in the below [Figure 1] [10].

\[
\begin{array}{cccc}
1101 & 4\text{-bits} & \text{Multiplicant} \\
1101 & 4\text{-bits} & \text{Multiplier} \\
1101 & \text{partial} \\
1101 & \text{product} \\
+ & 1101 & \text{Result} \\
\end{array}
\]

Figure:1. Example of manual multiplication.

[2] ANT ARCHITECTURE

The ANT technique [9] [8] includes both main digital signal processor (MDSP) and error correction (EC) block, as shown in [Figure 2]. To meet ultralow power demand, VOS is used in MDSP. However, under the VOS, once the critical path delay \( T_{CP} \) of the system becomes greater than the sampling period \( T_{samp} \), the soft errors will occur. It leads severe degradation in signal precision.

![ANT Architecture Diagram](image-url)
In the ANT technique, a replica of the MDSP but with reduced precision operands and shorter computation delay is used as EC block. Under VOS, there are a number of input dependent soft errors in its output \( y_a[n] \). However, RPR output \( yr[n] \) is still correct since the critical path delay of the replica is smaller than \( Tsamp \) [4]. Therefore, \( yr[n] \) is applied to detect errors in the MDSP output \( ya[n] \). Error detection is accomplished by comparing the difference \( |ya[n] - yr[n]| \) against a threshold \( Th \). Once the difference between \( ya[n] \) and \( yr[n] \) is larger than \( Th \), the output \( y'[n] \) is \( yr[n] \) instead of \( ya[n] \). Where \( yo[n] \) is error free output signal. In this way, the power consumption can be greatly lowered while the SNR can still be maintained without severe degradation [9].

The basic premise is that a replica of the DSP logic is designed with significantly reduced precision such that its critical path delay is less than that of the Main DSP (MDSP). Hence, it will operate at lower voltage than the main DSP block. The output of the RPR block \( yr[n] \) can thus be used to detect timing-errors in the output of the main DSP block \( yo[n] \), whereupon the replica precision is chosen as the output for the system. Designing a RPR replica DSP function with significantly reduced critical path length is possible for ripple-carry adders and multipliers, because the critical path length is linearly dependent on the operand precision. When a timing-error is detected, the main DSP block output sample is replaced with the RPR output sample leading to a decrease in the system SNR, due to a decrease in the RPR block.

[3] MULTIPLIER ARCHITECTURES

One of the multiplication procedures, the figure shows the array multiplier design. The total numbers of partial products generated are proportional to the bit widths of the two values. Typically the total number of partial products are proportional to the product of bit widths i.e. \( M*N \) and it requires \( M*N \) two input and gates. Adding the generated partial products is the second step, this is done by using \( N-1 \) \( M \) –bit adders. No logic required for the perfectly aligning the partial products and for adding them. The efficient layout of the partial products adding structure is same as a rectangle.

![Array Multiplier Architecture](image-url)
The truncated multiplier technique [12] is shown in the [Figure 3], by reducing the usage of the lower truncated triangle part, area requirements are considerably reduced. Truncation method eliminates least significant columns from partial product matrix. The degree of truncation is indicated by $T$. The most significant bits are always zeros. Truncation method also consists of the steps as generation of partial product and addition. Additional steps involved in truncated multiplier are deleting, truncating and finally rounding.

The primary step in truncated multiplier is deleting, the process start with deleting. The degree of truncation is generally 50% of the size of the product. i.e. we delete more than half of the partial products, the remaining partial products are used for the truncated process. In truncation least significant bits are replaced by zeros. The least significant partial product matrix is eliminated and deleted [Figure 4]. The remaining most significant partial product matrices are added. For both fixed width and non-fixed width truncation is same apart from the truncation degree.

In any multiplication if we have n-bit multiplicand and n-bit multiplier will generally have 2n-bit product. By eliminating the LSP from the array multiplier, it forms the truncated multiplication scheme. Thus obtained truncated multiplier satisfies the low power requirements. The truncated multiplier is most preferable for the low power applications and where the exact value is not necessary. Truncated multiplier is the area power efficient than the normal multiplier. Area is the primary restraint that is concentrated. With the use of the truncated multiplier cost factor will be reduced in the FIR filters.

$$Y2X3 = Y2 \times X3$$:

<table>
<thead>
<tr>
<th>multiplicand</th>
<th>X3</th>
<th>X2</th>
<th>X1</th>
<th>X0</th>
</tr>
</thead>
<tbody>
<tr>
<td>multiplier</td>
<td>Y3</td>
<td>Y2</td>
<td>Y1</td>
<td>Y0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Y0X3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y1X3 Y1X2</td>
</tr>
<tr>
<td>Y2X3 Y2X2 Y2X1</td>
</tr>
<tr>
<td>Y3X3 Y3X2 Y3X1 Y3X0</td>
</tr>
</tbody>
</table>

| Product      | P7 | P6 | P5 | P4 | P3 | 0  | 0  | 0  |

Figure: 4. 4x4 bit Binary Multiplication with truncation

[4] DADDA MULTIPLIER

Luigi Dadda, the computer scientist has invented the Dadda hardware multiplier during 1965. Dadda multiplier is extracted form of parallel multiplier [12]. It is slightly faster and requires fewer gates. Different types of schemes are used in parallel multiplier. The Dadda scheme is one of the parallel multiplier schemes that essentially minimize the number of adder stages required to perform the summation of partial products. This is achieved by using full and half adders to reduce the number of rows in the matrix number of bits at each
summation stage. Even though the Dadda multiplication has regular and less complex structure, the process is slower in manner due to serial multiplication process. Further, Dadda multiplier is less expensive compared to that of Dadda multiplier. Hence, in this paper, Dadda multiplier is designed and analyzed by considering different methods using full adders involving different logic styles.

[5] IMPLEMENTATION OF DADDA MULTIPLIER

The algorithm of Dadda multiplier is based on the below matrix [6] [7] [8] from shown in [Figure 5]. The partial product matrix is formed in the first stage by AND stages which is illustrated in [Figure 6].

\[
\begin{array}{cccc}
a_3 & a_2 & a_1 & a_0 \\
a_{3b_3} & a_{2b_3} & a_{1b_3} & a_{0b_3} \\
a_{3b_2} & a_{2b_2} & a_{1b_2} & a_{0b_2} \\
a_{3b_1} & a_{2b_1} & a_{1b_1} & a_{0b_1} \\
a_{3b_0} & a_{2b_0} & a_{1b_0} & a_{0b_0} \\
\end{array}
\]

Figure: 5. 4x4 Dadda Algorithm.

Steps involved in Dadda multipliers Algorithm:

- Multiply (that is - AND) each bit of one of the arguments, [13] [14] by each bit of the other, yielding N results. Depending on position of the multiplied bits, the wires carry different weights.

- Reduce the number of partial products to two layers of full adders.

- Group the wires in two numbers, and add them with a conventional adder [13] [14][15].
Figure: 6. Product terms generated by a collection of AND gates.

[6] RESULTS AND DISCUSSION

The simulation of the Dadda and truncated multiplier are carried out through XILINX ISE 12.3i. The Figure 7 shows the RTL schematic diagram of the Dadda and truncated multiplier. The truncated multiplier and Dadda multiplier outputs are given to the mux then we can find out the combine output.

Figure: 7. RTL schematic of truncated and Dadda multiplier.

The figure 8 shows how many look up tables (LUT) are used in the design of Dadda multiplier. In this paper the number of LUTs of truncated multiplier is 648 and the Dadda multiplier is 618. Finally the Dadda multiplier look up tables are reduced this is called the area.
DESIGN OF DADDA MULTIPLIER WITH OPTIMIZE D POWER USING ANT ARCHITECTURE

Simulation waveform of truncated and Dadda multiplier with combination of the array multiplier is shown in [Figure 9]. The Dadda multiplier is nothing but the Wallace tree multiplier. In this the inputs are a 11 x 0 bit, b 11 x0 bit and one selection line (0 or 1). The binary multiplication output is product 24 x 0 bit. The truncated output is product 1 and Dadda output is product 2.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>P0</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
<th>P6</th>
<th>P7</th>
<th>P8</th>
<th>P9</th>
<th>P10</th>
<th>P11</th>
<th>P12</th>
<th>P13</th>
<th>P14</th>
<th>P15</th>
<th>P16</th>
<th>P17</th>
<th>P18</th>
<th>P19</th>
<th>P20</th>
<th>P21</th>
<th>P22</th>
<th>P23</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product[0x0]</td>
<td>111011</td>
<td>101</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Product[0x1]</td>
<td>1001</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Product</td>
<td>11101</td>
<td>101</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Product[0x5]</td>
<td>10101</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Product[0x6]</td>
<td>10110</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Product[0x7]</td>
<td>11000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Product[0x8]</td>
<td>11011</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Product[0x9]</td>
<td>11100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure:9. Simulated waveform of truncated and Dadda multiplier.

TABLE 1: comparison between the Dadda and truncated multiplier.

The table 1 shows the comparison between the Dadda and truncated multiplier. It is observed from the table the number of LUT used in Dadda Multiplier is less as compared with Truncated multiplier. The memory size also reduced in Dadda Multiplier.

<table>
<thead>
<tr>
<th>Multiplier</th>
<th>No. of LUT's</th>
<th>Delay</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dadda</td>
<td>618</td>
<td>30.679ns</td>
<td>245 MB</td>
</tr>
<tr>
<td>Truncated</td>
<td>648</td>
<td>41.102ns</td>
<td>271 MB</td>
</tr>
</tbody>
</table>
[7] CONCLUSION

Here, in this paper two different multipliers were designed which are array multiplier and Dadda multiplier with the combination of truncated multiplier. In proposed design which is nothing but truncated with Dadda multiplier the area (in terms of LUT’s) is less which are 618 when compare to the existing truncated with array multiplier which are 648. In the same way the delay and memory requirements for the proposed design is better when compare with the existed design. This multipliers output are derived depending on multiplexer selection line, which depends on the user. In future, based upon the requirements there may be a chance to change the multipliers.

REFERENCES


