A 51x51 Multiplier Design Based on Signed 18x18 and Unsigned 17x17 Multipliers

Jonathan Roth and Hani Mehrpouyan
Department of Electrical and Computer Engineering
Queen’s University, Kingston, Ontario, K7L3N6, Canada

Abstract

This report provides a brief overview of the two popular schemes of performing large operand multiplication, sign extension and Baugh-Wolley’s algorithm, and provides the implementation and performance analysis for an unsigned 17x17 and a signed 18x18 multiplier. Furthermore, an efficient scheme for combining the unsigned 17x17 or signed 18x18 multipliers is used to implement two different versions of a signed 51x51 multiplier and comparative analysis between the two methods is included, where the area, speed, and power consumption of the two schemes are compared. VHDL and synopsis are used to implement the logic blocks and layout of the 51x51 signed multiplier, respectively.

I. INTRODUCTION

In recent years, modern FPGAs contain large number of embedded blocks, such as embedded multipliers and DSP functions ([1] and [2]).

The availability of many efficient and optimized embedded logic blocks within today’s Field Programmable Gate Arrays (FPGAs) has greatly facilitated the design process for more complicated logic functions [1], [2]. Large multipliers are an example of such complicated functions with many applications in signal processing, fast fourier transform (FFT), and image processing ([3] and [4]). Therefore, there have been many schemes proposed to design larger size multipliers from smaller ones. The divide-and-conquer strategy presented in [5] has been used to realize matrix multiplication based on word-width decomposition. The broadcast algorithm was proposed in [6] which is sequential and requires less resources at the cost of a slower speed. The sliding window scheme, [7], has been reported to efficiently realize large size multipliers. The algorithm generates less partial products by skipping zeros in the operand. However, it is not efficient for implementation in FPGAs.
In [8] a new methodology is presented to realize multipliers in FPGAs. This method can be recursively applied to implement multipliers with large operands. However, the proposed algorithm is based on the sign extension algorithm which has been demonstrated (in [9]) to be inefficient compared to Baugh-Wooley’s algorithm for FPGA design. In [10] a new scheme based on the carry-save multiplier scheme first outlined in [11] is proposed. The algorithm promises to reduce the total number of cells required in and FPGA and reduce the number of interconnects. Thus, reducing delay. However, it is important to point out the algorithm does not shed any new light on how to reduce the overall complexity associated with the multiplication when smaller blocks are used and it is mainly focused around reducing the number of interconnects within the different cells of the FPGA.

Baugh-Wooley’s algorithm, [12], is a popular algorithm that can handle block level manipulation. As, it is known, Baugh-Wooley’s algorithm performs a transformation of the two’s complement multiplication into two components, unsigned and signed. In this report we use the more methodical and structured approach proposed in [9] to come up with the structure of a 51x51 signed multiplier based on 17x17 unsigned and 18x18 signed multipliers. The proposed scheme in [9], first converts the operands from a two’s complement notation to a sign-magnitude notation. Secondly, the large unsigned multiplication is carried out on the resulting magnitudes. Subsequently the result of the multiplication is converted into a two’s complement notation.

The two versions of the signed 51x51 multipliers are implemented using VHDL. The resulting structure is then laid out using Synopsis and based on the CMOSP18 Artisan (\(18\mu m\)) technology. Finally, the speed, layout area, and power consumption of the two different designs for the 51x51 multiplier are comparatively analyzed.

II. SIGN EXTENSION METHOD

The sign extension algorithm is commonly used in the design of large two’s complement multipliers based on smaller blocks for FPGA implementations. The scheme extends the partial product in each row based on the sign of the multiplicands. Therefore, a disadvantages of the sign extension method is that smaller blocks of unsigned multipliers can not be used to determine larger ones.

\(^1\)The VHDL code for both designs was first compiled using Quartus II software provided by Altera [13].
The first step partitions the large size operands of the multiplier into smaller size segments. In general using $n$-bit wide embedded blocks, the operands $A$ and $B$ are decomposed into $m$ segments.

\[
A = [A_{m-1}, A_{m-2} \ldots A_1, A_0]2^n \\
B = [B_{m-1}, B_{m-2} \ldots B_1, B_0]2^n,
\]

where only the most significant segments, $A_{m-1}$ and $B_{m-1}$ take the sign bits of $A$ and $B$, respectively. Therefore, all other segments can be treated as positive numbers and $A$ and $B$ can be represented as

\[
A = 2^{(m-1)n}A_{m-1} + 2^{(m-2)n}A_{m-2} + \ldots + 2^n A_1 + A_0 \\
B = 2^{(m-1)n}B_{m-1} + 2^{(m-2)n}B_{m-2} + \ldots + 2^n B_1 + B_0.
\]

To correctly exploit the signed multiplier for the positive number multiplication, the sign bits of the embedded multipliers have to be forced to 0. To illustrate the process involved better the design of a 35x35 bit multiplier based on smaller blocks of signed 18x18 multipliers is discussed here. In this scenario, the number of segments $m$ is equal to 2 and the bit size of the positive number multiplication is 1717. The input operands are

\[
A = [A_1 A_0]2^n \\
B = [B_1 B_0]2^n,
\]

where $A_1$ and $B_1$ are the most significant bit positions and $A_0$ and $B_0$ are the least significant bit positions of $A$ and $B$, respectively.

\[
A \times B = [A_1 2^n + A_0] \times [B_1 2^n + B_0] \\
= A_1 B_1 2^{2n} + A_0 B_0 + A_1 B_0 2^n + A_0 B_1 2^n
\]

The first partial product in Equation (4) is $A_1 B_1 2^{2n}$, which requires no sign extension because the sign is already embedded in both $A_1$ and $B_1$. The second partial product, $A0 \times B0$, is also an unsigned multiplication and again does not require sign extension. The last two partial products in Equation (4) are $A_1 B_0 2^n$ and $A_0 B_1 2^n$. Since $A1$ and $B1$ keep the sign of the operands $A$ and $B$ and $A_0$ and $B_0$ are positive numbers, the partial products $A_1 B_0$ or $B_1 A_0$ have exactly the same sign as that of $A1$ and $B1$, which is also the sign of $A$ and $B$, respectively. Therefore the
sign bit of $A_1B_0$ and $A_0B_1$ have to be extended to the most significant bit of the final product before adding the partial products all together. This idea is illustrated in Figure 1.

It is important to point out that the sign-extension strategy increases the burden of additions for segmented partial products. Since sign extension process results in more bit that need to be added, therefore, increasing the amount of delay and complexity.

III. 17X17 MULTIPLIER

Consider two numbers $A$ and $B$

\[
(A)_{10} = \sum_{i=0}^{m-1} a_i 2^i \\
(B)_{10} = \sum_{i=0}^{n-1} b_i 2^i.
\]

The product $A \times B$ is

\[
(P)_{10} = A \times B \\
= \sum_{i=0}^{m-1} a_i 2^i \sum_{j=0}^{n-1} b_j 2^j \\
= \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} a_i b_j 2^{i+j} \\
= \sum_{k=0}^{m+n-1} P_k 2^k
\]
Figure 2 represents the summand representation of Equation (6). Furthermore, Figure 3 represents the structural design for a 5x5 unsigned multiplier.

Fig. 2. The bit by bit representation for an unsigned multiplier.

\[
\begin{array}{cccccccccc}
  a_{m-1} & a_{m-2} & \ldots & a_1 & a_0 \\
  b_{m-1} & b_{m-2} & \ldots & b_1 & b_0 \\
  a_{m-1} & b_1 & a_{m-2} & b_1 & \ldots & a_1b_1 & a_0b_1 & a_0b_0 & a_0b_0 \\
  a_{m-1} & b_1 & a_{m-2} & b_1 & \ldots & a_1b_1 & a_0b_1 & a_0b_1 & a_0b_1 \\
\end{array}
\]

Clearly as pointed out in Figure 2 the process of multiplication can be performed using many full adders, where the input bits are first aned together and then added.

Fig. 3. The logic structure of a 5x5 unsigned multiplier.

The structure represented in Figure 3 can be easily extended to operands with 17 bits or more.
VHDL and Quartus II were used to implement the 17x17 multiplier and simulate the result of the implementation, respectively. Figure 4 represents the functional verification performed in Quartus II. As expected, the implemented 17x17 multiplier functions correctly. Furthermore, the provided inputs ensure that the functional verification covers 98% of all the possible scenarios.

The VHDL design was further analyzed in synopsis based on the CMOSP18 Artisan structure. The result of the analysis is represented in Table I.

IV. 18x18 MULTIPLIER

If we consider two 2’s complement integers, $A$ and $B$ with $m$ and $n$ bits respectively, then $A$ and $B$ can be represented by

$$
(A)_{10} = -a_{m-1}2^{m-1} + \sum_{i=0}^{m-2} a_i2^i
$$

$$
(B)_{10} = -b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i2^i.
$$

The product $P = A.B = (P_{m+n-1}P_{m+n-2}...P_1P_0)_{2}$ can be further represented as

$$
(P)_{10} = a_{m-1}b_{n-1}2^{m+n-2}
$$

$$
+ \sum_{i=0}^{m-2} \sum_{j=0}^{n-2} a_ib_j2^{i+j}
$$

$$
- \sum_{i=0}^{m-2} a_i b_{n-1}2^{n-1+i}
$$

$$
- \sum_{i=0}^{n-2} a_{m-1} b_i 2^{m-1+i}.
$$

Figure 5 provides a representation of Equation 8 and clearly demonstrates how the signed multiplication is different from the unsigned version. It is important to note that the subtraction in
Equation 8 can be converted to addition by finding the two’s complement of $\sum_{i=0}^{m-2} a_i b_{n-1} 2^{n-1+i}$ and $\sum_{i=0}^{n-2} a_{m-1} b_i 2^{m-1+i}$ terms,

$$- \sum_{i=0}^{m-2} a_i b_{n-2} 2^{n-1+i} = 2^{n-1} (-1.2^m + 1.2^{m-1} + 1 + \sum_{i=0}^{m-2} a_i b_{n-1} 2^i).$$

Figure 5. The bit by bit representation of a Baugh-Wooley signed multiplier.

Figure 6 illustrates the structural design for a 6x4 signed multiplier. The full adders dealing with the sign section of the multiplication are represented in red.

The VHDL implementation of the 18x18 signed multiplier was first performed in Quartus II and the result of the functional verification analysis is presented in Figure 7 (the inputs ensure a 94% coverage by the functional verification).

The VHDL design was further analyzed in synopsis based on the CMOSP18 Artisan structure. Please refer to Table I for the result of the analysis corresponding to maximum operating frequency (slack time=0). As expected the 18x18 signed multiplication requires more power, cell area, and operates at a slower speed. Table I also compares the two designs when the slack time is not zero. Thus, corresponding to the frequency of operation in a practical system. The 18x18 signed multiplier is expected to be slower and use more power since it requires more full adders (refer to Figures 6 and 3) and the operands need to be inverted after being anded in more cases, resulting in more delay and complexity.
Fig. 6. The logic structure of a 6x4 signed Baugh-Wooley signed multiplier, the res representing the signed section of the multiplication.

Fig. 7. Functional verification for the 18x18 signed multiplier

V. 51x51 Multiplier

This section discusses the design and implementation of the two 51×51-bit two’s complement multipliers. One multiplier uses the 17×17-bit unsigned multiplier as the basic building block, while the other uses the 18×18-bit two’s complement multiplier as the basic building block. The two multipliers designed and implemented use the segmentation method discussed in [9], which is based on the Baugh-Wooley algorithm [12].

There are two specific sections outlined here. Since the process of multiplication in [9] is segmented to two specific sections, specifically the signed and unsigned.
Given a $KK$-bit multiplier with two inputs in two’s complement number representation, the values of the inputs, denoted as $X_v$ and $Y_v$, are

$$X_v = -x_{k-1}2^{k-1} + \sum_{i=0}^{k-2} x_i2^i$$

$$Y_v = -y_{k-1}2^{k-1} + \sum_{i=0}^{k-2} y_i2^i. \quad (10)$$

According to Baugh-Wooley’s algorithm [12], the value of the output of the multiplication, denoted as $P_v$, where

$$P_v = Y_v \times X_v = -P_{2k-1}2^{2k-1} + \sum_{i=0}^{2k-2} P_i2^i$$

$$= \left(-y_{k-1}2^{k-1} + \sum_{i=0}^{k-2} y_i2^i\right) \times \left(-x_{k-1}2^{k-1} + \sum_{i=0}^{k-2} x_i2^i\right)$$

$$= \left(x_{k-1}y_{k-1}2^{2k-2} + \sum_{i=0}^{k-2} \sum_{j=0}^{k-2} x_i y_{j}2^{i+j}\right) - \sum_{i=0}^{k-2} x_{k-1}y_i2^{k-1+i} - \sum_{i=0}^{k-2} y_{k-1}x_i2^{k-1+i}. \quad (11)$$

Equation (11) can be simplified to

$$P_v = -2^{2k} + \sum_{i=0}^{k-2} x_{i}y_{j}2^{i+j} + 2^{2k-1} + 2^{2k-2}(x_{k-1}y_{k-1} + x_{k-1} + y_{k-1})$$

$$+ 2^{k-1}(x_{k-1} + y_{k-1} + \sum_{i=0}^{k-2} x_{k-1}y_{i} + \sum_{i=0}^{k-2} y_{k-1}x_{i}. \quad (12)$$

In equation (12), the partial products are partitioned into two sections as shown in Figure 8 (the red represents the signed and the black represents the unsigned section). First, the unsigned section and secondly the signed section. The unsigned section represents partial products for multiplication of the operands of $(k-1)$ bits. The rest of the products represent the signed section.

The approach in [9] divides each section presented in Figure 8, into two section, specifically
Fig. 8. The signed and unsigned sections for kxk signed multiplier

the signed and unsigned sections. Thus, both sections need to be optimized to ensure they can be efficiently designed based on the available, smaller embedded blocks. Furthermore, the result of the multiplication from the smaller blocks forming the larger ones need to be added together (Figure 9. Therefore, and optimized addition operation is also proposed in [9] that will be discussed further here.

A. Unsigned Section

The design of the large size unsigned multiplier used in the unsigned portion of the Baugh-Wooley algorithm is done using small size multipliers as the basic building blocks. The large
size operands are decomposed into smaller segments, as shown in equations 13 and 14, and then the segmented inputs are then multiplied using the small size multipliers.

\[
X = [X_{m-1}X_{m-2}\ldots X_1X_0]
\]  
\[
Y = [Y_{m-1}Y_{m-2}\ldots Y_1Y_0]
\]

Using large size operands of \(k\)-bits each, the unsigned multiplier uses the first \((k - 1)\) bits to perform the multiplication. If \(n \times n\)-bit multipliers are used as the basic block, and the large, unsigned operands are \((k - 1)\)-bits each, the large size operands have to be decomposed into \(m\) segments, where \(n(m - 1) < k - 1 \leq nm\). For our particular design, \(n = 17\) and \((k - 1) = 50\), and so \(m = 3\), meaning the operands have to be divided into three segments. These segmented inputs can be expressed as:

\[
X = 2^{34}X_2 + 2^{17}X_1 + X_0
\]

\[
Y = 2^{34}Y_2 + 2^{17}Y_1 + Y_0
\]

For the design using the \(18 \times 18\)-bit two’s complement multiplier, \(n = 17\), as in the case of using the \(17 \times 17\)-bit unsigned multiplier, since all numbers dealt with in this section are unsigned. The sign bit (bit 18) of the \(18 \times 18\)-bit multiplier is simply set to 0.

Once the inputs have been segmented, using equations 15 and 16, the multiplication of \(X\) and \(Y\) can be expressed as:

\[
Z = X \times Y
\]

\[
= (2^{34}X_2 + 2^{17}X_1 + X_0) \times (2^{34}Y_2 + 2^{17}Y_1 + Y_0)
\]

\[
= X_0 \times Y_0 + X_1 \times Y_1 \times 2^{34} + X_2 \times Y_2 \times 2^{68}
\]

\[
+ X_1 \times Y_0 \times 2^{17} + X_2 \times Y_1 \times 2^{51} + X_2 \times Y_0 \times 2^{34}
\]

\[
+ X_0 \times Y_1 \times 2^{17} + X_1 \times Y_2 \times 2^{51} + X_0 \times Y_2 \times 2^{34}
\]

The partial products generated by equation 17 can be organized as shown in Figure 10. Each partial product is labeled \(P_{ij}\), which can be interpreted as the \(j\)th partial product of level \(i\). Once the partial products have been generated using the small size multipliers, additions must be performed to generate the final product of the unsigned portion of the algorithm. To characterize the additions required, three parameters are defined below:

- \(\tau_i\): The arrival time of the operand \(i\) created from a previous level.
• $\delta_{ij}$: The execution delay created by adding the operands $i$ and $j$ at the current level.
• $\Delta_{ij}$: The accumulated delay that is the total value of the arrival time and the execution delay at the current level, which is also the arrival time for the next level.

The execution delay is assumed to only depend on the operand size and relative offset of the operands, and can be calculated as:

$$\delta_{ij} = \max(size(i), size(j)) - \text{abs}(\text{offset}(i) - \text{offset}(j))$$

(18)

Also, the accumulated delay is calculated as:

$$\Delta_{ij} = \max(\tau_i, \tau_j) + \delta_{ij}$$

(19)

Fig. 10. The multiplication segmentation for the 51x51 multiplier

The first level of additions involve adding the partial products that are of the same length. Referring to Figure 10, the first level includes the addition of $P_{11}$ and $P_{12}$, and the addition of $P_{13}$ and $P_{14}$. The second and third level of additions are organized taking into account the arrival times of the partial products from the previous level. The second level partial products to be added are $P_{20}$, which is the same as $P_{10}$, $P_{21}$, which is the addition of $P_{11}$ and $P_{12}$, and $P_{22}$, which is the addition of $P_{13}$ and $P_{14}$. Table II shows the arrival times, operand sizes, and offsets of the second level partial products.

A delay table, table III, for the addition of the second level operands is also created, which shows the accumulated delay of adding operands $i$ and $j$, $\Delta_{ij}$. From the algorithm, the row and
column corresponding to $P_{21}$ are removed, since $P_{21}$ has the largest accumulated delay. This leaves us with one entry left, which tells us to add $P_{20}$ and $P_{22}$.

Once the second level addition is performed, one third level addition is need. This is the addition of $P_{30}$, which is the addition of $P_{20}$ and $P_{22}$ from the second level, and $P_{31}$, which is the addition of $P_{11}$ and $P_{12}$ from the first level. The delay of this addition can be calculated to be 83. Figure 11 and 12 show the additions performed at each level.

### B. Signed Section

The signed component of the Baught Wooley algorithm is computed as indicated in equation 12 and in Figure 8. The signed and unsigned components simply need to be added together to generate the final product. The final product is

$$P = P_{\Sigma s} + P_{s1} + P_{s2} + 2^{k-1}x_{k-1} + 2^{k-1}y_{k-1},$$

(20)

where

$$P_{\Sigma s} = 2^{2k-1} + 2^{2k-2}(x_{k-1}y_{k-1}) + P\Sigma,$$

(21)
where $P_2$ is the result of the unsigned multiplication generated in the previous section. $P_{s1}$ and $P_{s2}$ are

$$P_{s1} = 2^{2k-2}x_{k-1} + 2^{k-1} \left( \sum_{i=0}^{k-2} x_{k-1,j} \right)$$

(22) and

$$P_{s2} = 2^{2k-2}y_{k-1} + 2^{k-1} \left( \sum_{i=0}^{k-2} y_{k-1,i} \right).$$

(23)

To reduce the number of adders needed to generate the final product, some optimizations can be used. One adder can be used to compute the sum of $P_{s1}$ and $P_{s2}$, whose least significant bits are at position $k - 1$. Since the terms $2^{k-1}x_{k-1}$ and $2^{k-1}y_{k-1}$ are located at bit position $k - 1$ as well, they can be treated as incoming carries when summing $P_{s1}$ and $P_{s2}$. So, the first addition step involves adding $P_{s1}$ and $P_{s2}$ together, with carry $2^{k-1}x_{k-1}$, which requires a 51-bit adder. The resulting sum is then added to $P_2$, with $2^{k-1}y_{k-1}$ as a carry, which requires a 52-bit adder. The organization of these sums is shown in Figure 13.
C. Implementation

The implementation for the 51x51 multiplier involved two sections. The VHDL design and verification was performed in Quartus II software provided by Altera [13]. Furthermore, the functional analysis, test bench insertion, and the layout was done in synopsis based on the CMOS18 Artisan structure (the RMC design integrated circuit design flow manual [14] and the mentor design for test manual [15] were used to guide the process).
1) VHDL and Quartus II: The algorithm described above is used for the design of the 51x51 bit multiplier. Figure 14 represents the structure of the VHDL code and the input and output are registered for a synchronous design. The input and outputs of the 18x18 and 17x17 multipliers are both registered. Therefore, only the output of the 51x51 multiplier needs to be registered as outlined in Figure 14. The multiplication and summation are performed on each segment of the multiplication but represented as a black box in Figure 14. For further details the reader is encouraged to refer to the above sections on the signed and unsigned sections addition and multiplications specifically. It is also important to point out that the design illustrated in Figure 14 is based on the 18x18 multiplier but can be easily changed to include the 17x17 scenario.

The result of the functional verification for the 51x51 multiplier based on the unsigned 17x17 and signed 18x18 multipliers are presented in Figure 15 and Figure 16 respectively. The functional verification performed in Quartus II provides 94% coverage of all the possible scenarios in the case of the 51x51 multiplier designed based on the the unsigned 17x17 multiplier and 93% coverage for the 51x51 multiplier based on the signed 18x18 multiplier.

2) Functional analysis in Synopsis: Table IV represents the result of the functional analysis for both versions of the 51x51 multipliers. As expected the 51x51 based on the unsigned 17x17
multiplier tends to perform better compared to the one designed around the signed 18x18 multiplier. The maximum frequency of operation that we could achieve for this design was 28.9 and 28.8 for 51x51 based on 17x17 and 51x51 based on the 18x18, respectively.

Clearly the speed of operation for this design is slower than expected which is mainly due to the ripple carry adders within the addition section of the 51x51 multiplier. As outlined in Figure 14 there is no pipelining performed for the addition section of the 51x51 multiplier design. Therefore, justifying the slower speed of operation for the 51x51 multiplier. However, through the use of optimization and by introducing registers to the addition section the delay associated with the longest path could be considerably reduced, resulting in considerably higher frequency of operation.

TABLE IV
17X17 AND 18X18 MULTIPLIER PERFORMANCE RESULTS @ MAXIMUM FREQUENCY AND AT NON ZERO SLACK

<table>
<thead>
<tr>
<th>Multiplier</th>
<th>Power Consumption (mW)</th>
<th>Cell Leakage Power (uW)</th>
<th>Cell Area</th>
<th>Frequency (MHz)</th>
<th>Slack Time (nSec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>51x51 based on 17x17</td>
<td>32.67</td>
<td>16.94</td>
<td>355795.06</td>
<td>28.9</td>
<td>0</td>
</tr>
<tr>
<td>51x51 based on 18x18</td>
<td>32.22</td>
<td>16.06</td>
<td>348516.91</td>
<td>28.8</td>
<td>0</td>
</tr>
<tr>
<td>51x51 based on 17x17</td>
<td>21.68</td>
<td>14.24</td>
<td>310785.56</td>
<td>20.0</td>
<td>0.01</td>
</tr>
<tr>
<td>51x51 based on 18x18</td>
<td>21.71</td>
<td>14.50</td>
<td>316390.53</td>
<td>20.0</td>
<td>0.04</td>
</tr>
</tbody>
</table>

Fig. 15. Functional verification for the 51x51 signed multiplier based on the 17x17 unsigned multiplier design.

Fig. 16. Functional verification for the 51x51 signed multiplier based on the 18x18 unsigned multiplier design.
3) **Scan chain insertion and verification in Synopsis:** In order to perform the scan chain insertion we needed to prepare the UNIX environment and provide many different scripts to generate the test patterns. In the next stage the test patterns were simulated to ensure that the test patterns generated are working properly with the design containing the test patterns. The tool used to perform this task is again NC-Verilog from Cadence. Figure 17 represents the timing diagram used testing the scan chain. The specific times used in the design were deduced from the result of timing analysis in the previous section.

![Timing diagram for the scan chain.](image)

**Fig. 17.** Timing diagram for the scan chain.

### D. Layout

For the lay out design the process described in [14]. Figure 18 represents the initial placement of the die for the 51x51 multiplier based on the 18x18 multiplier. The size of the die was measured to be $600\,\mu m \times 590\,\mu m$. Figure 18 also represents the power routing for the die represented as vertical lines crossing the die.

Figure 19 represents the routing for the different cells. Figure 20 represents the clock tree.

### E. LVS

The LVS for both designs of the 51x51 multiplier are represented in Figure 21 and Figure 22.

### VI. Conclusion

In this report a brief overview of Baugh-Wooley’s and sign-extension algorithms are provided. Specifically by segmenting a large size multiplier and using Baugh-Wooley’s algorithm we were
able to implement two different versions of a 51x51 multiplier based on signed 18x18 and unsigned 17x17 multipliers.

The result of functional analysis based on the .18\(\mu\)m CMOS18 Artisan structure demonstrates that using an 18x18 or a 17x17 multiplier for the design of the 51x51 multiplier does not make a significant difference in the operational speed, layout area, or power consumption of the final design. We were able to achieve a maximum operating frequency of 109 and 102 MHz for the 17x17 and 18x18 multipliers respectively. The operational speed for the 51x51 was considerably slower at almost 28 MHz. That’s mainly due to the large ripple carry adders in the design of the multiplier. The overall speed of the multiplier could be significantly improved through the introduction of registers at the different stages of the ripple carry adder, reducing the delay associated with the longest path.
VII. ACKNOWLEDGMENT

We would like to especially thank Jean-Luc Derome for all his help throughout the design, implementation, and layout of this project. We would also like to thank Dr. Al-Khalili for his excellent guidance and teaching efforts throughout the semester.

REFERENCES

Fig. 20. The clock tree.


[9] Shuli Gao, Dhamin Al-Khalili, and Noureddine Chabini, “Efficient realization of large size two’s complement multipliers using embedded blocks in FPGAs,” Accepted in the JN of circuits and systems and signal processing.


Fig. 21. LVS for 51x51 multiplier based on the signed 17x17.

Fig. 22. LVS for 51x51 multiplier based on the signed 18x18.