Binary Multiplication based on Single Electron Tunneling

Casper Lageweg, Sorin Cotofana and Stamatis Vassiliadis
Microelectronics and Computer Engineering (ME&CE) department,
Delft University of Technology,
Delft, The Netherlands
E-mail: {casper,sorin,stamatis}@ce.et.tudelft.nl

Abstract

This paper investigates single electron tunneling based implementations of 16 and 32-bit tree multipliers operating according to the single electron encoded logic paradigm. First, we propose implementations for a set of basic components (3/2 counter, 7/3 counter) and verify them by means of simulation. Second, we propose 16 and 32-bit tree multipliers based on these components, and analyze these multipliers in terms of area, delay and power consumption. Third, we investigate alternative designs for the 32-bit multiplier and conclude that the 7/3 counter based implementations are less effective than expected. We consequently propose improved 7/3 counters and evaluate the implications of these new designs on the area, delay and power consumption of the 16 and 32-bit multipliers.

1: Introduction

Single Electron Tunneling (SET) [1, 2] is a novel technology candidate and offers greater scaling potential than MOS as well as the potential for ultra-low power consumption. Additionally, recent advances in silicon based fabrication technology (see for example [3]) show potential for room temperature operation. However, similar to other future technology candidates, SET devices display a switching behavior that differs from traditional MOS devices. This provides new possibilities and challenges for implementing digital circuits.

In this line of reasoning we investigate in this paper SET based implementations of 16 and 32-bit tree multipliers. The main contributions can be summarized as follows:

- We propose single-electron-encoded threshold logic gate based implementations of the following components: 3/2 counter and 7/3 counter. We verify these implementations by means of simulation.
- We propose 16 and 32-bit tree multipliers based on these components. The 16 / 32-bit multipliers can be implemented at the cost of 10155 / 32148 circuit elements, have a delay of 56.8 / 74.7 ns and a power consumption of 462 / 1666 meV.
- We investigate a 32-bit multiplier with partial production matrix reduction that is based on 3/2 counters only. A 3/2 counter based 32-bit multiplier can be implemented at the cost of 45181 circuit elements, has a delay of 64.8 ns and a power consumption of 2016 meV. Based on this we conclude that 7/3 counter based implementations are less effective than expected.
We propose improved 7/3 counters and evaluate the implications of these new designs on the area, delay and power consumption of the 16 and 32-bit multipliers. Improved 7/3 counter based 16 / 32-bit multipliers can be implemented at the cost of 10659 / 36450 circuit elements, have a delay of 49.9 / 64.4 ns and a power consumption of 555 / 2406 meV.

The remainder of this paper is organized as follows. Section 2 briefly presents the SET theory. Section 3 introduces the SET threshold logic gate. Section 4 proposes threshold gate based implementations of tree multiplier components. Section 5 investigates 16 and 32-bit tree multipliers based on these components. Finally, Section 6 concludes the paper.

2: Background

A tunnel junction can be thought of as a leaky capacitor. The transport of charge through a tunnel junction is referred to as tunneling, where the transport of a single electron is referred to as a tunnel event. Electrons are considered to tunnel strictly one after another. We assume that all conditions are met such that charge quantization is observable (Coulomb energy $E_C >> E_Q$, the quantum energy) and that tunnel events due to thermal energy can be ignored ($E_C >> K_b T$, where $K_b$ is Boltzman’s constant and $T$ the operating temperature). Under these conditions, the critical voltage $V_c$ across a tunnel junction is the voltage threshold that is needed across the tunnel junction in order to make a tunnel event through this tunnel junction possible.

For calculating the critical voltage of a junction, we assume a tunnel junction with a capacitance of $C_j$. The remainder of the circuit, as viewed from the tunnel junction’s perspective, has an equivalent capacitance of $C_e$. Given the approach presented in [4], we calculate the critical voltage $V_c$ for the junction as:

$$V_c = \frac{e}{2(C_e + C_j)} \quad (1)$$

Generally speaking, if we define the voltage across a junction as $V_j$, and assuming the conditions stated above, a tunnel event will occur through this tunnel junction if and only if:

$$|V_j| \geq V_c \quad (2)$$

If tunnel events cannot occur in any of the circuit’s tunnel junctions, i.e., $|V_j| < V_c$ for all junctions in the circuit, the circuit is in a stable state. For our research we only consider circuits where a limited number of tunnel events may occur, resulting in a stable state. Each stable state determines a new output value resulting from the distribution of charge throughout the circuit.

The transport of an electron through a tunnel junction is a stochastic process. This means that we cannot analyze delay in the traditional sense. Instead, assuming a non-zero probability for charge transport ($|V_j| > V_c$), the switching delay $t_d$ of a single electron transport can be calculated based on an error probability $P_{error}$ that the desired transport did not occur as

$$t_d = -\frac{\ln(P_{error}) q_e R_t}{|V_j| - V_c} \quad (3)$$

where $R_t = 10^5 \Omega$ is the tunnel resistance (though depending on the physical implementation this value is typically assumed). The error probability $P_{error}$ will determine the reliability of
the circuit. Given that the switching behavior is stochastic in nature, the error probability cannot be reduced to 0. It is therefore assumed that an error correction mechanism, as for example suggested in [5, 6], will be present in the form of hardware or data redundancy in order to achieve the desired accuracy.

When charge transport occurs through a tunnel junction, the difference in the total amount of energy present in the circuit before and after the tunnel event can be calculated by

$$\Delta E = E_{final} - E_{initial} = -q_e(|V_j| - V_c)$$

(4)

Therefore, the energy consumed by a single tunnel event occurring in a single tunnel junction can be calculated by taking the absolute value of $\Delta E$. In order to calculate the power consumption of a gate, the energy consumption of each tunnel event is multiplied by the frequency of switching. The switching frequency in turn depends on the frequency at which the gate’s inputs change and is input data dependent, as a new combination of inputs may or may not result in charge transport.

In addition to the switching error probability as described in Equation (3) there are two fundamental phenomena that may cause errors: thermally induced tunneling and co-tunneling [7]. We assume in here that the operating temperature is sufficiently low, such that the error probability due to thermally induced tunneling is comparable to the switching error probability or less. Also, we assume that sufficient measures [8] have been taken to equally reduce the co-tunneling error probability.

The biggest technological challenge currently comes from the fact that thus far all experimental circuits have displayed random offset charge (random charge present on circuit nodes), which is assumed to be the result of trapped charge particles. This random charge results in a random additional voltage across tunnel junctions, which can cause errors in their switching behavior. Therefore, SET tunnel junction based circuits remain as-of-yet mostly of theoretical interest. However, SET tunnel junctions can be fabricated in many different ways, and have for example been demonstrated in conventional lithographic technologies such as silicon (see for example [9]), but also in carbon nanotube based technologies (see for example [10]). Additionally, there are indications [1] that the offset charge problem may reduce or even disappear entirely for the nanometer-scale feature size circuits required for room temperature operations. Consequently, SET circuit may become of practical interest in the near future. Given the discussion above, and the fact that in our investigation we focus on the utilization of the SET behavioral properties, we ignore the aspects related to offset charge.

The next section introduces a SET generic threshold logic gate (TLG), which operates according to the Single Electron Encoded Logic (SEEL) paradigm. SEEL gates encode the Boolean logic values 0 and 1 as a net charge of $0e$ and $1e$ on the gate’s output node. A SEEL gate switches output value by transporting just 1 electron, resulting in minimal power consumption. Also, due to the sequential nature of the charge transport through tunnel junctions, one can in general assume that the less charge transport implies less switching delay. The SEEL TLG is utilized as the basic building block of all circuits proposed in the remainder of the paper.
3: Single Electron Threshold Logic Gates

Threshold Logic Gates (TLG) are devices which are able to compute any linearly separable Boolean function given by:

\[
F(X) = \text{sgn}\{F(X)\} = \begin{cases} 
0 & \text{if } F(X) < 0 \\
1 & \text{if } F(X) \geq 0
\end{cases}
\]

(5)

\[
F(X) = \sum_{i=1}^{n} \omega_i x_i - \psi
\]

(6)

where \(x_i\) are the \(n\) Boolean inputs and \(w_i\) are the corresponding \(n\) integer weights. The TLG performs a comparison between the weighted sum of the inputs \(\Sigma_{i=1}^{n} \omega_i x_i\) and the threshold value \(\psi\). If the weighted sum of inputs is greater than or equal to the threshold, the gate produces a logic 1. Otherwise the output is a logic 0.

---

**Figure 1.** Threshold logic gates and inverting buffer.

As stated in Section 2, a SET tunnel junction requires a minimum voltage \(|V_j| \geq V_c\) in order for a tunnel event to occur. This critical voltage \(V_c\) acts as a naturally occurring threshold \(\psi\) with which the junction voltage \(V_j\) is compared. If we add capacitively coupled inputs to the circuit nodes on either side of the tunnel junction, the inputs will make a positively or negatively weighted contribution to the voltage across this junction (depending on the sign definition of \(V_j\)). Similarly, we can add a capacitively coupled biasing voltage in order to adjust the threshold to the desired value. This approach results in a generic SEEL TLG implementation [11] as displayed in Figure 1(a).

In this figure, the input signals \(V^p = \{V_1^p, V_2^p, \ldots, V_r^p\}\) are weighted by their corresponding capacitors \(C^p = \{C_1^p, C_2^p, \ldots, C_r^p\}\) and added to the voltage across the tunnel junction. The input signals \(V^n = \{V_1^n, V_2^n, \ldots, V_r^n\}\) are weighted by their corresponding capacitors.
$C^n = \{ C_1^n, C_2^n, \ldots, C_r^n \}$ and subtracted from the voltage across the tunnel junction. The biasing voltage $V_b$, weighted by the capacitor $C_b$, is used to adjust the gate threshold to the desired value $\psi$. If $\text{sgn}\{|V_j| - V_c\} = 1$, a single electron is transported from node $y$ to node $x$, which results in a high output. The resulting threshold function calculated by the circuit is:

$$
\mathcal{F}(X) = C_\Sigma^n \sum_{k=1}^r C_k^n V_k^p - C_\Sigma^n \sum_{l=1}^s C_l^n V_l^m - \psi
$$

(7)

$$
\psi = \frac{1}{2} (C_\Sigma^n + C_o^n) e - C_\Sigma^n C_b V_b,
$$

(8)

where $C_\Sigma^n = C_b + \sum_{k=1}^r C_k^n$ and $C_o^n = C_o + \sum_{l=1}^s C_l^n$.

Note that the SET TLG allows for both positive and negative weights and thus can potentially be used to calculate any threshold function in a single gate. If a TLG implementation only allows for positive weight, the potential of the threshold gate for efficient implementations of algorithms is limited [12].

Due to the passive nature of the threshold gate, buffers are required in order for the gate to operate correctly in networks [13]. A buffer requires active components, for which SET transistors [14] can be utilized. If two SET transistors share a single load capacitor, such that one transistor can remove a single electron from the load capacitor (resulting in high output) while the other can replace it, we arrive at the non-inverting static buffer [15] depicted in Figure 1(b).

In the remainder of this paper, unless otherwise specified, the following parameters are utilized: for input variables and supply voltages we use logic '0' = 0 Volt and logic '1' = $V_b = V_s = 16mV$. For the threshold gate and the inverting buffer we use: $C_i = C_g = C_1 = C_2 = C_3 = C_4 = 0.1aF$, $C_{c1} = C_{c2} = 4.85aF$, $C_i = 9.8aF$, and $\Sigma C^n + C_o = 9.8aF$. For the delay calculations we assume an error probability $P_{error} = 10^{-8}$.

$$
\begin{array}{c}
\text{(a) TLG gate symbol.} \\
\text{(b) TLG based of 2-input AND gate.}
\end{array}
$$

**Figure 2. Threshold Logic Gate (TLG).**

The SET threshold gate combined with the inverting output buffer, graphically depicted in Figure 2(a), serves as a basic building block for the proposed implementations of the tree multiplier components that are discussed in the next section.

### 4: Multiplication Building Blocks

The 2-input AND gate, the 3/2 counter (full adder) and the 7/3 counter are the basic components for tree multipliers. This section presents the threshold gate based implementations of these three components.
4.1: 2-input AND Gate

A threshold gate based version of a 2-input AND gate can be implemented in a single gate, and it is defined in correspondence with Equation (5) as follows:

\[
AND(a_i, b_i) = sgn\{a_i + b_i - 2\}
\]  \hspace{1cm} (9)

Given that the threshold gate discussed in Section 2 requires an inverting buffer, the above threshold equation is implemented as a threshold gate calculating its inverse (calculating NAND\((a_i, b_i)\) instead of AND\((a_i, b_i)\)). Thus, when combined with an inverting buffer, the gates produce the correct output. In general, inverted threshold equations can be derived in a straightforward manner by inverting the sign of each weight, subtracting 1 from the threshold value and inverting the sign of the result. Consequently, the 2-input AND gate implementation based on buffered threshold gates adheres to the structure displayed in Figure 2(b).

\[ \text{Figure 3. Threshold gate based 2-input AND gate - simulation results.} \]

In order to evaluate the 2-input AND gate implementation the following circuit parameters are utilized (in addition to the general parameters described in Section 2): \(C_n^1(\omega = -1) = C_n^2(\omega = -1) = 0.5 aF, C_b = 12.1 aF\). The AND gate implementation has been verified by means of simulation using the single-electron device and circuit simulator SIMON (SIMulation Of Nanostructures) [16]. The simulation results are depicted in Figure 3. As can be observed, the AND logic function is correctly implemented.

4.2: 3/2 Counter

A 3/2 counter, more commonly referred to as full adder, counts the number of 1s among its 3 inputs \((x_0, x_1, \text{and } x_2)\), and represents the result as a 2-bit number \((c_1, s)\) as summarized by Table 1. A threshold gate based 3/2 counter can be implemented in two gates and in two logic levels [17], and it is defined in correspondence with Equation (5) as follows:

\[
c_1 = sgn\{x_0 + x_1 + x_2 - 2\}
\]

\[
s = sgn\{x_0 + x_1 + x_2 - 2c_1 - 1\}
\]  \hspace{1cm} (10)  \hspace{1cm} (11)

Each of the above threshold equations is implemented as a threshold gate calculating the inverse (derived by means of the method described in Section 4.1) and an inverting
buffer. Consequently, the FA implementation based on buffered threshold gates adheres to the structure displayed in Figure 4(a).

Table 1. Input and matching output values of 3/2 counter.

<table>
<thead>
<tr>
<th>$\sum_{i=0}^{2} x_i$</th>
<th>$c_1$</th>
<th>$s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 4. Threshold gate based 3/2 counter.

In order to evaluate the 3/2 counter implementation, the following circuit parameters are utilized (in addition to the general parameters described in Section 2). For tlg1 we use $C_1^0(\omega = -1) = C_2^0(\omega = -1) = C_3^0 = (\omega = -1) = 0.5aF$, $C_b = 12.1aF$. For tlg2 we use $C_1^0(\omega = -1) = C_2^0(\omega = -1) = C_3^1 = (\omega = -1) = 0.2aF$, $C_b^1(\omega = 2) = 0.6aF$, $C_b = 12.1aF$. The resulting implementation has been verified by means of simulation and the simulation results are depicted in Figure 4(b). The top 3 bars represents the three inputs $x_0$, $x_1$ and $x_2$, the bottom 2 represents the outputs $c_1$ and $s$. As can be observed, the 3/2 counter’s logic function is correctly implemented.

4.3: 7/3 Counter

A 7/3 counter counts the number of 1s among its 7 inputs $(x_0, x_1, \ldots, x_6)$, and represents the result as a 3-bit number $(c_2, c_1, s)$ as summarized by Table 2. A threshold gate based 7/3 counter can be implemented in three gates and in three logic levels [17], and it is defined in correspondence with Equation (5) as follows:

\[
\begin{align*}
c_2 &= \text{sgn}\{\sum_{i=0}^{6} x_i - 4\} \\
c_1 &= \text{sgn}\{\sum_{i=0}^{6} x_i - 4c_2 - 2\} \\
s &= \text{sgn}\{\sum_{i=0}^{6} x_i - 4c_2 - 2c_1 - 1\}
\end{align*}
\]
Above each of the above threshold equations is implemented as a threshold gate calculating the inverse (derived by means of the method described in Section 4.1) and an inverting buffer. Consequently, the FA implementation based on buffered threshold gates adheres to the structure displayed in Figure 5(a). Note that each of the 7 inputs \( x_i \) serves as an input with weight \( \omega_i = -1 \) to each of the 3 TLGs.

\[
\sum_{i=0}^{6} x_i c_2 c_1 s
\]

\[
\begin{array}{cccc}
0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 \\
2 & 0 & 1 & 0 \\
3 & 0 & 1 & 1 \\
4 & 0 & 0 & 0 \\
5 & 0 & 0 & 1 \\
6 & 0 & 1 & 0 \\
7 & 0 & 1 & 1 \\
\end{array}
\]

Table 2. Input and matching output values of 7/3 counter.

In order to evaluate the 7/3 counter implementation, the following circuit parameters are utilized (in addition to the general parameters described in Section 2). For tlgl we use \( C_1^T(\omega = -1) \ldots C_7^T(\omega = -1) = 0.1aF, C_b = 11.0aF \). For tlg2 we use \( C_1^T(\omega = -1) \ldots C_7^T(\omega = -1) = 0.1aF, C_2^T(\omega = 4) = 0.5aF, C_b = 11.0aF \). For tlg3 we use \( C_1^T(\omega = -1) \ldots C_7^T(\omega = -1) = 0.1aF, C_2^T(\omega = 4) = 0.5aF, C_2^T(\omega = 2) = 0.25aF, C_b = 11.1aF \). The resulting implementation has been verified by means of simulation and the simulation results are depicted in Figure 5(b). The top 7 bars represents the inputs \( x_0, x_1, \ldots, x_6 \), the bottom 3 represents the outputs \( C_2, c_1 \) and \( s \). As can be observed, the 7/3 counter’s logic function is correctly implemented.

**Figure 5. Threshold gate based 7/3 counter.**
B

adder

formation

partial product
matrix reduction

P

(a) General structure.  
(b) Partial product matrix reduction for 16-bit multiplier.

Figure 6. Tree multiplier.

5: Tree Multiplier Implementations

The tree based multiplication of two $n$-bit binary numbers $A = a_{n-1}a_{n-2}\ldots a_0$ and $B = b_{n-1}b_{n-2}\ldots b_0$, as depicted in Figure 6(a) [18], consists of three steps. During the first step, a partial product matrix is formed by multiplying each bit of the multiplier $A$ with the multiplicand $B$. Given that both $A$ and $B$ are binary numbers, each of these multiplication steps consists of $n$ logic AND operations (resulting in a total of $n^2$ parallel AND operations). The second step involves the reduction of the partial product matrix to two rows, and it is generally implemented with a tree of carry save adders or counters. The third and final step involves the carry-propagate addition of the two intermediate sums, forming the final product $P = p_{2n-1}p_{2n-2}\ldots p_0$.

The second step reduces the number of partial products in the matrix by counting the

<table>
<thead>
<tr>
<th>Component</th>
<th>Area</th>
<th>Delay</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-input AND gate</td>
<td>14</td>
<td>1.7 ns</td>
<td>0.8 meV</td>
</tr>
<tr>
<td>3/2 counter</td>
<td>31</td>
<td>4.5 ns</td>
<td>1.1 meV</td>
</tr>
<tr>
<td>7/3 counter</td>
<td>60</td>
<td>13.8 ns</td>
<td>2.6 meV</td>
</tr>
<tr>
<td>32-bit carry-lookahead adder (CLA)</td>
<td>1423</td>
<td>23.3 ns</td>
<td>60.8 meV</td>
</tr>
<tr>
<td>64-bit carry-lookahead adder (CLA)</td>
<td>2900</td>
<td>27.1 ns</td>
<td>102.1 meV</td>
</tr>
</tbody>
</table>

Table 3. Area, delay and power of tree multiplier components.
number of logic '1's in each of the matrix' columns, and replacing them by their binary representation [19]. Such reduction can be achieved by 7/3 and 3/2 counter based implementations, which results in a reduced partial product matrix with approximately the same number of columns but with less rows. This process is repeated until only two rows remain. Possible 7/3 and 3/2 counter based partial product matrix reduction schemes for 16 and 32-bit multiplication are depicted in Figures 6(b) and 7(a), respectively. The output generated by each counter is depicted as a set of connected dots. One can observe the following. The delay of the partial product matrix reduction corresponding to the 16-bit multiplication is 1 7/3 counter and 4 3/2 counter delays. For 32-bit multiplication this delay is 3 7/3 counter and 1 3/2 counter delays.

The third step of tree multiplication is implemented by a 2n-bit adder. Given that tree multipliers are optimized for speed, a fast adder is typically utilized. In this paper we assume TLG based carry-lookahead (CLA) adders as described in [20].

Given the methodology presented in Section 2 and the parameters listed in Sections 2, 4.1, 4.2, 4.3 and in [20], we calculated the area, delay and power consumption of all components required for the considered tree multiplier implementations. The results are

Figure 7. Schemes for 32-bit partial product matrix reduction.
summarized in Table 3. Based on these numbers and the reduction schemes presented in Figures 6(b) and 7(a), we have evaluated the costs of the 16 and 32-bit multipliers in terms of the required components, area, delay and power consumption. The results are summarized in Table 4.

As suggested earlier, depending on the available components, partial product matrix reduction can be implemented by a variety of different schemes. Focusing for example on 32-bit multiplication, we next investigate an alternative scheme, in which partial product matrix reduction is solely implemented with 3/2 counters. This results in the 32-bit reduction scheme depicted in Figure 7. Given the 3/2 counter implementation presented in Section 4.2, we calculated the cost of implementing this scheme in terms of required components, area, delay and power consumption. The results are summarized in Table 5. When comparing this implementation with the mixed 7/3 and 3/2 implementation presented earlier, the following can be observed. The 3/2 based scheme is about 13% faster than the mixed scheme, but requires approximately 40% extra area and increases power consumption by about 21%.

Based on the number of TLG levels in the critical path (16 gates for the 3/2 scheme versus 11 gates for the mixed scheme), we anticipated that the mixed implementation would be faster. Additionally, one would expect that 7/3 counters result in faster compression and hence in less delay. We therefore conclude that the TLGs in the 7/3 counter implementations are much slower than those utilized for the 3/2 counter. Examining the TLG gates in detail, we observe that the input capacitors of the TLGs utilized for the 7/3 counter are much smaller than those of the 3/2 counter TLGs. Consequently, each of the 7/3 counter’s gates is significantly slower. In the remainder of this section we investigate alternative TLG based 7/3 counter implementations that improve the performance.

The first optimization step is to reduce the delay of the buffer embedded in the TLGs. This can be done by assuming the following buffer circuit parameters: $C_g = 0.4aF$, $C_1 = 0.1aF$, $C_2 = 0.2aF$, $C_3 = 0.25aF$, $C_{c1} = 4.6aF$, $C_{c2} = 4.55aF$, $C_l = 9.5aF$. These parameters will be used in the remainder of the paper. We next focus on the threshold gate themselves.

In general, the fanout of a TLG (measured in total capacitive load) should be limited in order to reduce feedback effects in a network to workable levels [15]. Given that the TLG input weights are realized by input capacitors, they contribute to the capacitive load of the gate that is driving them. Consequently, if a TLG has high fanout, the input capacitors of the driven gates must be scaled down so that the total load remains bound. However, the delay of a TLG is inversely proportional to the size of the smallest input capacitor.
Thus, a TLG with small input capacitors is slower than a TLG with large input capacitors. As perceived from the CMOS design point of view, this is counter-intuitive. Normally smaller input capacitors should result in faster networks. In the case of the SET TLGs, however, delay is inversely proportional to the part of the input signal that contributes to the junction voltage $V_j$. Given that this part increases when the input capacitors are large, larger input capacitors result in faster gates.

Examining the original 7/3 counter implementation depicted in Figure 5(a) we observe that $tlg_1$ drives 2 inputs that each have a weight of 4. Given that these weights must be realized with limited size capacitors, the capacitor used to implement a weight of 1 must be relatively small. This results in large delays for $tlg_2$ and $tlg_3$. Thus, to speed up the design, one should investigate alternative implementations that perform the same calculations but with weights that can be implemented with larger capacitors.

The first design optimization is based on the following observation. The 7/3 counter implementation depicted in Figure 5(a) utilizes both positive and negative weights. Given our choice of circuit parameters and TLG behavior specified in Equation (7), the following holds true. When implementing equal sized positive and negative weights $|\omega^p| = |\omega^n|$, the positive weight $\omega^p$ requires a larger input capacitor than the negative weight $\omega^n$. We can thus reduce the size of input capacitors by transforming positive weights into negative weights. This can be done by inverting the corresponding input signal. The inverters themselves are implemented as inverting buffers. We can apply this approach to the output of both $tlg_1$ and $tlg_2$. This results in an optimized 7/3 counter implementation (version v1), which is depicted in Figure 8(a).

In order to evaluate the v1 improved 7/3 counter implementation, the following circuit...
parameters are utilized. For tlg1 we use $C_1^n(\omega = -1) \ldots C_7^n(\omega = -1) = 0.5aF$, $C_b = 15.3aF$. For tlg2 we use $C_1^p(\omega = -1) \ldots C_7^p(\omega = -1) = 0.15aF$, $C_8^n(\omega = -4) = 0.6aF$, $C_b = 12.2aF$. For tlg3 we use $C_1^n(\omega = -1) \ldots C_7^n(\omega = -1) = 0.15aF$, $C_8^n(\omega = 4) = 0.6aF$, $C_9^n(\omega = 2) = 0.3aF$, $C_b = 10.9aF$. The implementation has been verified by simulation and the results are depicted in Figure 8(b). We calculated the area, delay and power consumption of the v1 improved 7/3 counter and obtained the results summarized in Table 6. One can observe that even though we added 2 inverters in the critical path, the design is faster due to the fact that the delay added by the inverters is less than the delay reduction for the threshold gates. The v1 improved design reduces the 7/3 counter’s delay by about 40 % at the expense of increasing the area by 30 % and the power consumption by 120 %.

We next observe that the improved 7/3 counter has a total of 5 inverters in the critical delay path (3 as TLG output buffer and 2 stand-alone). We can further optimize the design by reducing the number of inverters in the critical path to the minimum of 3 while still limiting fanout. This results in a further optimized 7/3 counter implementation (version v2), which is depicted in Figure 8(a). Due to the removal of inverters in the critical path the circuit now contains a TLG (tlg3) with both positive and negative weights. However, the circuit is still faster due to the fact that the large load on tlg1 is now split between 2 inverters.

In order to evaluate the v2 improved 7/3 counter implementation, the following circuit parameters are utilized. For tlg1 we use $C_1^p(\omega = 1) \ldots C_7^p(\omega = 1) = 0.5aF$, $C_b = 10.0aF$. For tlg2 we use $C_1^p(\omega = 1) \ldots C_7^p(\omega = 1) = 0.25aF$, $C_8^p(\omega = 4) = 1aF$, $C_b = 10aF$. For tlg3 we use $C_1^n(\omega = -1) \ldots C_7^n(\omega = -1) = 0.15aF$, $C_8^n(\omega = -2) = 0.3aF$, $C_9^n(\omega = 4) = 0.75aF$, $C_b = 11.9aF$. The implementation has been verified by simulation and the results are depicted in Figure 8(b). We calculated the area, delay and power consumption of the v2 improved 7/3 counter and obtained the results summarized in Table 6. When compared with the original, the v2 improved design reduces the 7/3 counter’s delay by about 50 % at approximately the same expense in area and power consumption as the v1 improved 7/3 counter design.

Given the improved designs for the 7/3 counter, we re-calculated the area, delay and
<table>
<thead>
<tr>
<th>Multiplier</th>
<th>Area</th>
<th>Delay</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>16-bit</td>
<td>10659</td>
<td>49.9 ns</td>
<td>555 meV</td>
</tr>
<tr>
<td>32-bit</td>
<td>36450</td>
<td>64.4 ns</td>
<td>2406 meV</td>
</tr>
</tbody>
</table>

Table 7. Area, delay and power of improved multipliers.

power consumption of the mixed 7/3 and 3/2 counter based implementation of the 16 and 32-bit multipliers. The results are summarized in Table 7. When comparing the new 32-bit multiplier implementation with the 3/2 counter based implementation, the following can be observed. The delay of the 2 designs is approximately the same, while the 3/2 based scheme requires 24% more area and consumes 16% less power.

6: Conclusions

In this paper we investigated single electron tunneling based implementations of 16 and 32-bit tree multipliers operating according to the single-electron-encoded logic paradigm. First, we proposed implementations for a set of basic components (3/2 counter, 7/3 counter) and verified them by means of simulation. Second, we proposed 16 and 32-bit tree multipliers based on these components, and analyzed these multipliers in terms of area, delay and power consumption. Third, we investigated alternative designs for the 32-bit multiplier and concluded that the 7/3 counter based implementations are less effective than expected. We consequently proposed improved 7/3 counters and evaluated the implications of these new designs on the area, delay and power consumption of the 16 and 32-bit multipliers.

References


