Binary Addition based on Single Electron Tunneling Devices

Casper Lageweg Sorin Cotofana Stamatis Vassiliadis
Electrical Engineering Department,
Delft University of Technology, Delft, The Netherlands
{Casper,Sorin,Stamatis}@Dutepp0.ET.TUDelft.NL

***Abstract*— The ability to control the transport of individual electrons within single electron tunneling based circuits creates the conditions for implementing single electron encoded threshold logic gates. This paper investigates the implementation of binary addition based on such gates. We first propose implementations of full adder and 4-bit lookahead carry generator blocks and verify the designs by means of simulation. We then evaluate the area, delay, and power consumption of 16-bit and 64-bit ripple carry and carry-lookahead adders based on these blocks.**

***Index Terms*— Single electron tunneling, threshold logic circuits, addition**

I. Introduction

Addition is a basic operation that is frequently used in arithmetic circuits. This paper investigates ripple carry (RC) and carry-lookahead (CLA) addition based on full adders and carry generators. We propose single electron encoded threshold gate based implementations of these blocks and calculate their area, delay and power consumption. Based on the obtained results we estimate the same for 16-bit and 64-bit RC and CLA adders.

The remainder of this paper is organized as follows. Section II briefly presents the SET theory and threshold logic gate. Section III proposes threshold gate based full adder and carry generator implementations. Section IV investigates 16 and 64 bit ripple carry and carry-lookahead adders based on these implementations. Finally, Section V concludes the paper.

II. Background and Generic Threshold gate

Single Electron Tunneling (SET) circuits [1], [2] are centered around tunnel junctions. Charge transport through a tunnel junction is referred to as *tunneling*, the transport of a single electron is referred to as a *tunnel event*. Tunneling is a stochastic process. The critical voltage V_c across a tunnel junction is the voltage threshold that is required to create a non-zero probability that a tunnel event will occur. If we define the voltage across a junction as V_j, it is assumed that a tunnel event through this tunnel junction will occur if and only if $|V_j| \geq V_c$.

Given the stochastic nature of electron tunneling, delay cannot be analyzed in the traditional sense. Instead, one can describe the switching delay as

$$t_d = \frac{-ln(P_{error})q_e R_t}{|V_j| - V_c} \tag{1}$$

where P_{error} is the chance that the desired tunnel event has not occurred after t_d seconds. For the tunnel resistance we assume $R_t = 10^5\Omega$ (though depending on the physical implementation this value is typically assumed).

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

$$\Delta E = E_{initial} - E_{final} = q_e(|V_j| - V_c) \tag{2}$$

Therefore the energy consumed by switching activity can be calculated by summarizing the energy ΔE consumed by each of the individual tunnel events.

The implementations discussed in here are independent of their physical implementation. Circuit area is evaluated in terms the total number of utilized circuit elements (capacitors and junctions) in order to provide a metric for comparison. Thermally induced tunneling and co-tunneling are beyond the scope of this investigation.

Threshold logic gates are devices that are able to compute any linearly separable Boolean function given by:

$$F(X) \quad = \quad sgn\{\mathcal{F}(X)\} = \begin{cases} 0 & \text{if } \mathcal{F}(X) < 0 \\ 1 & \text{if } \mathcal{F}(X) \geq 0 \end{cases} \tag{3}$$

where $\mathcal{F}(X) = \sum_{i=1}^{n} \omega_i x_i - \psi$, x_i are the n Boolean inputs and w_i are the corresponding n integer weights. The linear threshold gate performs a comparison between the weighted sum of the inputs $\Sigma_{i=1}^{n} \omega_i x_i$ and the threshold value ψ. 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.

A generic SEEL threshold gate structure has been proposed earlier in [3] and is depicted in Figure 1(a). The generic threshold gate can be used as a basis for implementing linear threshold gates with both positive and negative weights. However, due to the passive nature of the threshold gate, buffers are required in order for the gate to operate correctly in networks [4]. A buffer requires active components, for which SET transistors can be utilized (see for example [5]). 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 [6] depicted in Figure 1(b).

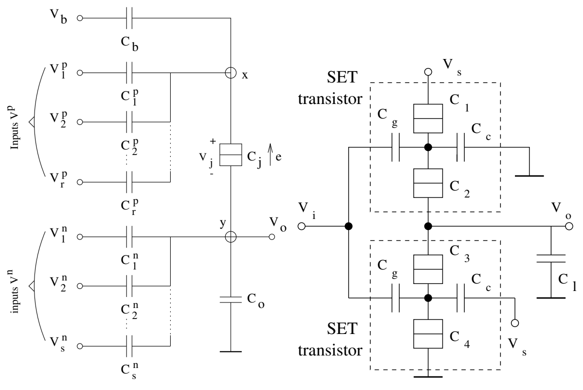
(a) n-input LTG. (b) Inverting buffer

Fig. 1. Linear Threshold Gates (LTG) and inverting buffer.

In the remainder of this paper, the following parameters will be consistently used. 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: $C_j = C_g = C_1 = C_2 = C_3 = C_4 = 0.1aF$, $C_c = 4.85aF$, $C_l = 9.8aF$, and $\Sigma C^n + C_o = 9.8aF$. We assume an error probability $P_{error} = 10^{-8}$.

The SET threshold gate combined with the inverting output buffer serves as a basic building block for the implementations discussed in the remainder of this paper.

III. Addition Building Blocks

The full adder block and the lookahead carry generator block form the basis for ripple-carry adders and carry-lookahead adders. This section presents the threshold gate based implementations of these two blocks.

A. Full Adder

The full adder (FA) calculates the addition of two input bits (a_i and b_i) and a carry-in c_{i-1}, and produces a sum bit s_i and a carry-out c_i. A threshold gate based FA can be implemented in two gates and in two logic levels [7], and is defined in correspondence with Equation (3) as follows

$$c_i = sgn\{a_i + b_i + c_{i-1} - 2\} \quad (4)$$
$$s_i = sgn\{a_i + b_i + c_{i-1} - 2c_i - 1\} \quad (5)$$

Given that the threshold gate discussed in Section II requires an inverting buffer, each of the above threshold equations is implemented as a threshold gate calculating its inverse (calculating for example $\overline{c_i}$ instead of c_i). Thus when combined with an inverting buffer the gates produce the correct output. Inverted threshold equations can be derived in a straightforward manner by inverting the sign of each weight, and subtracting 1 from the threshold value and inverting the sign of the result. Consequently, the FA implementation based on buffered threshold gates adheres to the structure displayed in Figure 2.

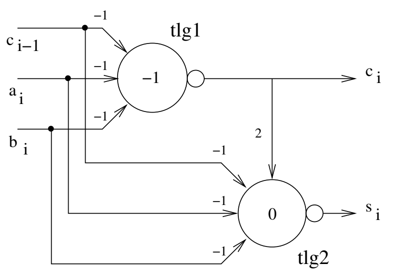

Fig. 2. Threshold gate based full adder implementation.

In order to evaluate the FA implementation the following circuit parameters are utilized (in addition to the general parameters described in Section II). For tlg1 we use $C^n_1(\omega = -1) = C^n_1(\omega = -1) = C^n_1 = (\omega = -1) = 0.5aF$, $C_b = 12.1aF$. For tlg2 we use $C^n_1(\omega = -1) = C^n_1(\omega = -1) = C^n_1 = (\omega = -1) = 0.2aF$, $C^p_1(\omega = 2) = 0.6aF$, $C_b = 12.1aF$. The FA implementation has been verified by means of simulation using the single-electron device and circuit simulator SIMON (SIMulation Of Nanostructures) [8]. The simulation results are depicted in Figure 3. As can be observed, the FA's logic function is correctly implemented.

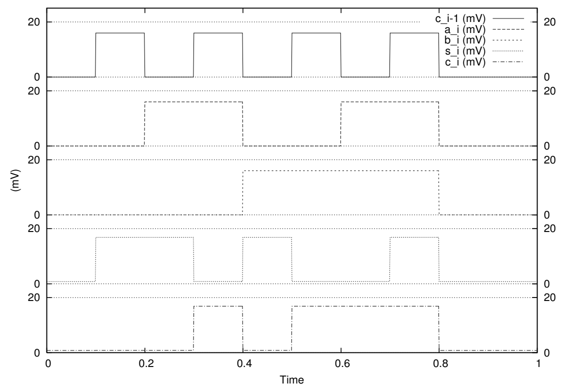

Fig. 3. Threshold gate based full adder - simulation results.

B. Lookahead Carry Generator

Carry-lookahead addition is based on unrolling the recurrence in the calculation of the carry. Each carry c_{i+1} is calculated as $c_{i+1} = g_i + p_i c_i$, where $g_i = a_i b_i$ and $p_i = a_i + b_i$. After a single unrolling step we find $c_{i+1} = g_i + p_i g_{i-1} + p_i p_{i-1} c_{i-1}$. The unrolling process can be continued further, such that all carries are generated in parallel. However, given that the number of gate inputs is limited, 4-bit lookahead carry generators, as depicted in Figure 4, are typically used as building blocks for larger networks. The output signals $g_{[i,i+3]}$ and $p_{[i,i+3]}$ are the generate and propagate signals of the entire 4-bit block.

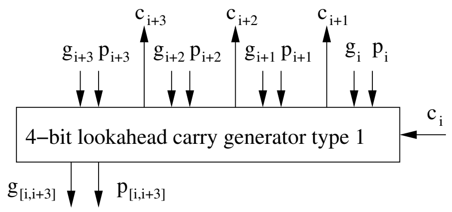

Fig. 4. 4-bit lookahead carry generator.

Each output signal of the 4-bit lookahead carry generator can be formulated as a single threshold logic equation [9]. However, in order to reduce the size of the weights, we utilize the intermediate signal $g_{[i,i+2]}$. Resulting, the calculation of the output signals of the 4-bit lookahead carry generator ($type$ 1) can be implemented by 6 threshold gates and in 2 logic levels as follows.

$$\begin{aligned} g_{[i,i+2]} &= sgn\{5g_{i+2} + 3p_{i+2} + 2g_{i+1} + p_{i+1} \\ &\quad +g_i - 5\} && (6) \\ c_{i+1} &= sgn\{2g_i + p_i + c_i - 2\} && (7) \\ c_{i+2} &= sgn\{5g_{i+1} + 3p_{i+1} + 2g_i + p_i \\ &\quad +c_i - 5\} && (8) \\ c_{i+3} &= sgn\{4g_{[i,i+2]} + p_{i+2} + p_{i+1} + p_i \\ &\quad +c_i - 4\} && (9) \\ p_{[i,i+3]} &= sgn\{p_{i+3} + p_{i+2} + p_{i+1} + p_i - 4\} && (10) \\ g_{[i,i+3]} &= sgn\{2g_{i+3} + p_{i+3} + g_{[i,i+2]} - 2\} && (11) \end{aligned}$$

Each of the above threshold equations is implemented as a threshold gate calculating the inverse (derived by means of the method described in Section III-A) and an inverting buffer. In order to evaluate the implementation the following circuit parameters are utilized (in addition to the general parameters described in Section II). For tlg1 ($\overline{g_{[i,i+2]}}$) and tlg3 ($\overline{c_{i+2}}$) we use $C_1^n(\omega = -5) = 0.75aF$, $C_2^n(\omega = -3) = 0.45aF$, $C_3^n(\omega = -2) = 0.3aF$, $C_4^n(\omega = -1) = C_5^n(\omega = -1) = 0.15aF$, $C_b = 11.9aF$. For tlg2 ($\overline{c_{i+1}}$) and tlg6 ($\overline{g_{[i,i+3]}}$) we use $C_1^n(\omega = -2) = 1aF$, $C_2^n(\omega = -1) = C_3^n(\omega = -1) = 0.5aF$, $C_b = 12.1aF$. For tlg4 ($\overline{c_{i+3}}$) we use $C_1^n(\omega = -4) = 1aF$, $C_2^n(\omega = -1) = C_3^n(\omega = -1) = C_4^n(\omega = -1) = C_5^n(\omega = -1) = 0.25aF$, $C_b = 12.5aF$. For tlg5 ($\overline{p_{[i,i+3]}}$) we use $C_1^n(\omega = -1) = C_2^n(\omega = -1) = C_3^n(\omega = -1) = C_4^n(\omega = -1) = 0.5aF$, $C_b = 16aF$. The implementation has been verified by means of simulation (using SIMON [8]). The simulation results are depicted in Figure 5. The inputs g_{i+1}, g_{i+2} and g_{i+3} have not been depicted as they remain 0. As can be observed, the logic function of the 4-bit lookahead carry generator is correctly implemented by the threshold gate network.

Carry generator blocks are used as components of carry-lookahead adders. A 16 bit carry-lookahead adder for example consists of a tree structure of 5 4-bit lookahead

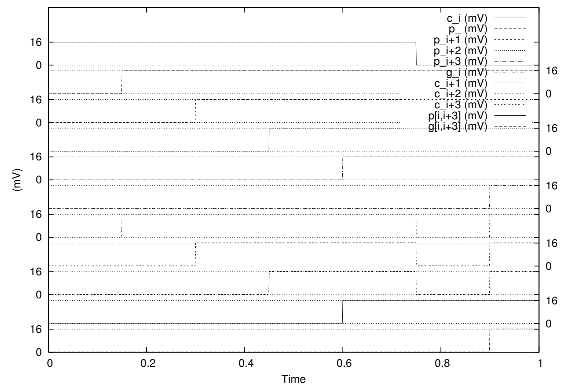

Fig. 5. Type 1 lookahead carry generator - simulation results.

carry generator blocks. The g_i and p_i inputs are either the group generate and propagate outputs signals of another carry generator ($g_{[i,i+3]}$ and $p_{[i,i+3]}$) or calculated directly from the adder's input signals ($g_i = a_i b_i$ and $p_i = a_i + b_i$). In the second case this would require an additional 8 gates per carry generator block. However, with threshold logic these calculations can be embedded within the block. This results in the $type$ 2 lookahead carry generator block, whose outputs can be calculated by 5 threshold gates and in 1 logic level as follows.

$$\begin{aligned} c_{i+1} &= sgn\{a_i + b_i + c_i - 2\} && (12) \\ c_{i+2} &= sgn\{2a_{i+1} + 2b_{i+1} + a_i + b_i \\ &\quad +c_i - 4\} && (13) \\ c_{i+3} &= sgn\{4a_{i+2} + 4b_{i+2} + 2a_{i+1} + 2b_{i+1} \\ &\quad +a_i + b_i + c_i - 8\} && (14) \\ p_{[i,i+3]} &= sgn\{8a_{i+3} + 8b_{i+3} + 4a_{i+2} + 4b_{i+2} \\ &\quad +2a_{i+1} + 2b_{i+1} + a_i + b_i - 15\} && (15) \\ g_{[i,i+3]} &= sgn\{8a_{i+3} + 8b_{i+3} + 4a_{i+2} + 4b_{i+2} \\ &\quad +2a_{i+1} + 2b_{i+1} + a_i + b_i - 16\} && (16) \end{aligned}$$

Each of the above threshold equations is implemented as a threshold gate calculating the inverse and an inverting buffer. The following parameters are used for evaluation. For tlg1 (c_1) we use $C_1^n(\omega = -1) = C_2^n(\omega = -1) = C_3^n(\omega = -1) = 0.6aF$, $C_b = 12.5aF$. For tlg2 (c_2)we use $C_1^n(\omega = -2) = C_2^n(\omega = -2) = 0.6aF$, $C_3^n(\omega = -1) = C_4^n(\omega = -1) = C_5^n(\omega = -1) = 0.3aF$, $C_b = 13aF$. For tlg3 (c_3) we use $C_1^n(\omega = -4) = C_2^n(\omega = -4) = 0.6aF$, $C_3^n(\omega = -2) = C_4^n(\omega = -2) = 0.3aF$, $C_5^n(\omega = -1) = C_6^n(\omega = -1) = C_7^n(\omega = -1) = 0.15aF$, $C_b = 13.3aF$. For tlg4 ($p_{[0,3]}$) and tlg5 ($g_{[0,3]}$)we use $C_1^n(\omega = -8) = C_2^n(\omega = -8) = 0.56aF$, $C_3^n(\omega = -4) = C_4^n(\omega = -4) = 0.28aF$, $C_5^n(\omega = -2) = C_6^n(\omega = -2) = 0.14aF$, $C_7^n(\omega = -1) = C_8^n(\omega = -1) = 0.07aF$, $C_b(tlg4) = 15aF$, $C_b(tlg5) = 15.3aF$.

The implementation has been verified by means of

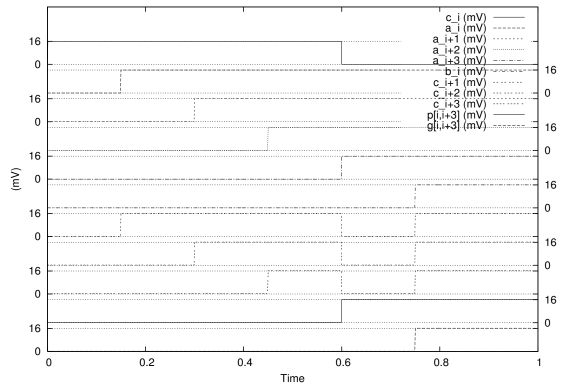

Fig. 6. Type 2 lookahead carry generator - simulation results.

simulation (using SIMON [8]). The simulation results are depicted in Figure 6. The inputs b_{i+1}, b_{i+2} and b_{i+3} have not been depicted as they remain 0. As can be observed, the logic function of the $type$ 2 lookahead carry generator is correctly implemented by the threshold gate network.

IV. Addition Schemes

Given the methodology described in Section II and the parameters derived in Sections II and III, the calculated area, delay and energy consumption per (output) switching of the full adder (FA) and the $type$ 1 and 2 carry generator (CG) are summarized in Table I. For the FA $delay1$ is the delay of tlg1 (carry) and $delay2$ is the delay of tlg2 (sum). For the CG blocks $delay1$ is the delay of the $p_{[i,i+3]}$ and $g_{[i,i+3]}$ signals, and $delay2$ is the delay of the carry signals (c_{i+1}, c_{i+2} and c_{i+3}).

Block	Area	Delay1	Delay2	Energy
FA	31	1.7 ns	2.8 ns	1.1 meV
CG type1	84	4.7 ns	5.0 ns	5.7 meV
CG type2	91	3.4 ns	6.2 ns	4.6 meV

TABLE I
Area delay and power of FA and CG blocks.

Ripple carry addition (RC) and Carry-LookAhead Addition (CLA) are two commonly used schemes for n-bit addition. An n-bit RC adder requires n FA blocks, with the carry out of the ith FA connected to the carry-in input of the $i+1$th FA. The 16-bit and 64-bit CLA adders can be constucted with a network of $type$ 1 and $type$ 2 CG blocks as depicted in Figure 7, combined with 1 threshold gate for each bit position to calculate the final sum bits (tlg2 of the FA block). We estimated the area, delay and energy consumption of 16-bit and 64-bit RC and CLA adders by utilizing the results obtained from FA and CG blocks, and summarized the results in Table II. It can be observed that, when compared with their RC counterparts, the CLA adders require approximate 40% more area and energy, but result in a significant reduction in delay (75% for 64-bit addition).

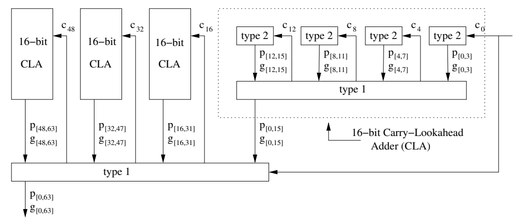

Fig. 7. 16-bit and 64-bit carry-lookahead adders.

AdderType	Area	Delay	Energy
16-bit ripple carry	496	28.3 ns	17.6 meV
16-bit carry-lookahead	704	17.4 ns	24.1 meV
64-bit ripple carry	1984	109.9 ns	70.4 meV
64-bit carry-lookahead	2900	27.1 ns	102.1 meV

TABLE II
Area delay and power of adder schemes.

V. Conclusions

This paper investigated the implementation of binary addition based on single electron encoded threshold logic gates. We first proposed implementations of full adder (FA) and 4-bit lookahead carry generator (CG) blocks and verified the designs by means of simulation. We then evaluated the area, delay and power consumption of 16 and 64 bit ripple carry and carry-lookahead adders based on the results obtained for the FA and CG blocks.

References

[1] K.K.Likharev, "Single-Electron Devices and Their Applications," *Proceeding of the IEEE*, vol. Vol. 87, no. No. 4, pp. pp. 606–632, April 1999.

[2] A.N.Korotkov, "Single-Electron Logic and Memory Devices," *International Journal of Electronics*, vol. Vol. 86, no. No. 5, pp. pp. 511–547, 1999.

[3] C.Lageweg and S.Cotofana and S.Vassiliadis, "A Linear Threshold Gate Implementation in Single Electron Technology," in *IEEE Computer Society Workshop on VLSI*, April 2001, pp. 93–98.

[4] ——, "Achieving Fanout Capabilities in Single Electron Encoded Logic Networks," in *6th International Conference on Solid-State and IC Technology (ICSICT)*, October 2001, pp. 1383–1386.

[5] R.H.Chen, A.N.Korotkov, and K.K.Likharev, "Single-electron Transistor Logic," *Applied Physics Letters*, vol. Vol. 68, no. No. 14, pp. pp. 1954–1956, April 1996.

[6] C.Lageweg and S.Cotofana and S.Vassiliadis, "Static Buffered SET Based Logic Gates," in *2nd IEEE Conference on Nanotechnology (NANO)*, August 2002, pp. 491–494.

[7] S.Cotofana and S.Vasiliadis, "Low Weight and Fan-In Neural Networks for Basic Arithmetic Operations," in *Congress on Scientific Computation, Modelling and Applied Mathematics, Volume 4: Artificial Intelligence and Computer Science*, August 1997, pp. 227–232.

[8] C.Wasshuber, H.Kosina, and S.Selberherr, "SIMON - A Simulator for Single-Electron Tunnel Devices and Circuits," *IEEE Transactions on Computer-Aided Design*, vol. Vol. 16, no. No. 9, pp. pp. 937–944, September 1997.

[9] S.Vassiliadis, S.Cotofana, and K.Bertels, "2-1 addition and related arithmetic operations with threshold logic," *IEEE Transactions on Computers*, vol. Vol, 45, no. No. 9, pp. pp. 1062–1068, September 1996.