Implementation of Comparison Function Using Quantum-Dot Cellular Automata

Meghanad D. Wagh*, Yichun Sun* and Viswanath Annampedu**

*Department of ECE, Lehigh University, Bethlehem, PA 18015, mdw0, yis205@Lehigh.Edu
** Storage Peripherals Group, LSI Corp., Allentown, PA 18109, Vish.Annampedu@lsi.com

ABSTRACT

A comparison function is important to implement arbitrary large Boolean functions. In this paper we show that a comparison function can be directly and efficiently implemented using QCA. The resultant architecture of an n bit comparator has a delay of $\lceil log_2 n \rceil + 1$ and a complexity of $O(n)$ gates. By duplicating certain majority gates, all crossing wires in the implementation except those at the input level can be eliminated.

Keywords: quantum-dot cellular automata, majority logic, comparison function

1 INTRODUCTION

Comparison is one of the common function used in many important applications including realization of arbitrary Boolean functions. In CMOS technology, one can easily realize an n bit comparison with a carry propagation subtractor having $O(n)$ delay. This speed can be improved to $O(\log n)$ by using a more complex block carry look-ahead subtractor. However, a further dramatic improvement in speed is only possible through use of nanoelectronic technologies such as the Quantum-dot cellular automata (QCA) [1].

QCA can be implemented in many technologies including ferromagnetic and molecular. The molecular QCA is particularly interesting because of its projected density of up to 1×10^{12} devices per cm^2 [2]. A major advantage of QCA over other nanoelectronic architectural styles is that the same cells that are used for making logic gates can be used to build wires carrying logic signals. However, QCA architectures have to rely upon only two basic building blocks, namely a three input majority gate and an inverter. As a result, QCA implementations of only a few logic circuits such as binary adders, multipliers, barrel shifters, serial/parallel converters are currently available [3]–[5].

This paper describes an efficient implementation of the comparison function using QCA. The resultant architecture has optimal delay $O(\log n)$ and a low gate complexity $O(n)$ for an n bit comparison. The architecture can be easily pipelined to improve its throughput.

2 QUANTUM-DOT CELLULAR AUTOMATA

A basic QCA building block can be described as a cell with four quantum dots and two charged particles which occupy the dots. The charged particles can migrate between quantum dots when the barriers between them are lowered by an external *clock*. When the barriers are raised by removing the clock, the particles settle into two possible stable (polarized) positions because of electrostatic forces. These stable states represent logic 0 and 1 as shown in Fig. 1.

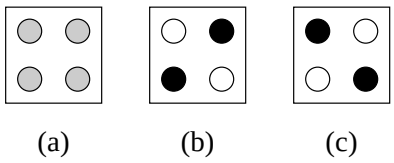

Figure 1: (a) The basic QCA cell (b) a polarized QCA cell representing logic 1 and (c) a polarized QCA cell representing logic 0.

When the clock to a QCA cell is reduced, the polarization state of its surrounding cells determines its own polarization state. This allows conduction of logic values along wires made of QCA cells as well as the design of logic elements such as an inverter and a three input majority gate. The two logic blocks made with QCA cells are illustrated in Figs. 2 and 3.

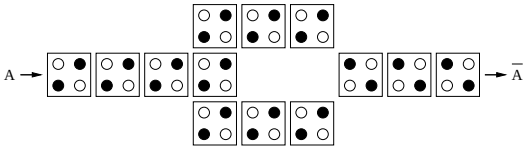

Figure 2: An inverter implementation in QCA.

A three input majority gate outputs a logic 1 when 2 or more of its inputs are 1. By fixing one of the inputs to the majority gate at 1, one can convert the majority gate into a 2-input OR gate. Similarly by fixing one of the inputs to 0, one can turn it into a 2-input AND

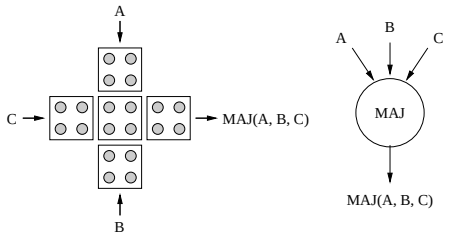

Figure 3: A 3-input majority gate implementation in QCA and its symbol.

gate. It therefore appears that it should be possible to implement any Boolean function in the QCA architecture. Unfortunately use of 2-input ANDs and ORs results in highly inefficient realizations since about 33% of the gate inputs are tied to constant values. Further, the large number of gates makes the circuit layout difficult. It is therefore important to develop designs that allow one to fully utilize the capabilities of the 3-input majority gates.

3 COMPARISON FUNCTION

An n bit comparison function compares two n bit binary strings and outputs a logic one if the value represented by the first string is greater than or equal to that of the second string. The comparison function is important in its own right. But we illustrate here its use in implementing an arbitrary Boolean function. Consider a Boolean function $f(x_{n-1}, x_{n-2}, \ldots, x_0)$ which outputs a logic one only when the n-bit input string $X = x_{n-1}x_{n-2}\cdots x_0$ has a value between X and Y. Let function $C(X)$ denote a comparison function that compares value of string X with X and outputs a 1 only if value of X is greater than or equal to that of X. Since f is 1 only when $X \geq X$ is true but $x \geq Y+1$ is false, it can be expressed as $f = C(X)\overline{C}(Y+1)$. Note that since an inverter as well as a 2-input AND gate can be realized in the QCA technology, so can the function f if one can design the comparison function $C(\cdot)$ in QCA. Similarly if f is 1 anytime the input string X is either between X_1 and Y_1 or between X_2 and Y_2, then that f may be described as $f = C(X_1)\overline{C}(Y_1+1)+C(X_2)\overline{C}(Y_2+1)$ and realized by the QCA. Fig. 4 shows this realization.

It is obvious that any arbitrary Boolean function can be implemented using the strategy described here. Creating such an implementation requires one to determine contiguous groups of 1's in the truth table of the function. The end points of each group are then compared with the input variable string, and the results ANDed. Finally, outputs of all the ANDs are added together to get the function. Clearly, All the comparisons can be done concurrently and all the ANDs evaluate at the same time. Thus the resultant Boolean function realization may have a fairly small depth provided one can find a small depth comparison function implementation.

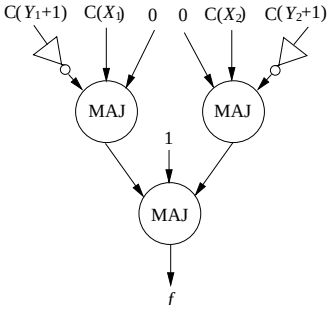

Figure 4: QCA realization of function f which is 1 only when its argument is between X_1 and Y_1 or between X_2 and Y_2. Note that $C(\cdot)$ is a comparison function.

The comparison function $C(B)$ that compares two n bit strings $B = b_{n-1}b_{n-2}\cdots b_0$ and $X = x_{n-1}x_{n-2}\cdots x_0$ can be implemented by subtracting B from X while keeping track of the carry only. Fig. 5 shows a QCA implementation of this strategy.

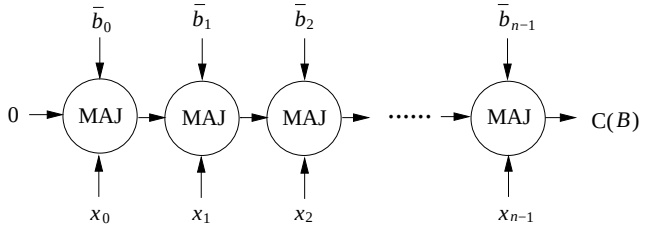

Figure 5: A serial realization of comparison $C(B)$ which outputs a 1 when $x_{n-1}x_{n-2}\cdots x_0 \geq b_{n-1}b_{n-2}\cdots b_0$.

Unfortunately, the architecture in Fig. 5 has an $O(n)$ delay. Minimizing this delay in QCA architectures is important because even in the combinational logic, unlike CMOS, all the gates in QCA implementations need to be clocked. Further, in applications such as the Boolean function implementation, the comparator delay can directly impact the delay of the Boolean function.

In order to achieve the optimal delay, we build the comparison output recursively. Suppose operands X and B are partitioned as $X = [X_1|X_0]$ and $B = [B_1|B_0]$. $X \geq B$ is true if $X_1 > B_1$ or if $(X_1 = B_1)$ and simultaneously $(X_0 \geq B_0)$. However, computing equality of X_1 and B_1 requires XOR gates which are expensive in QCA technology. We therefore define intermediate logical variables $p_i \equiv (X_i > B_i)$ and $q_i \equiv (X_i \geq B_i)$, $i = 0$ or 1. In this new notation, $(X_1 = B_1)$ has the same truth value as $(q_1\overline{p}_1)$. Thus the output of comparison $X \geq B$ can be written as a Boolean expression $p_1 + (q_1\overline{p}_1)q_0 = p_1 + q_1q_0$. Further, using the fact that $p_1 = 1$ implies $q_1 = 1$, one gets $p_1 = p_1q_1$ and $q_1 = p_1 + q_1$. Thus $p_1 + q_1q_0$ can be rewritten as $p_1q_1 + p_1q_0 + q_1q_0$, which is precisely the output of a

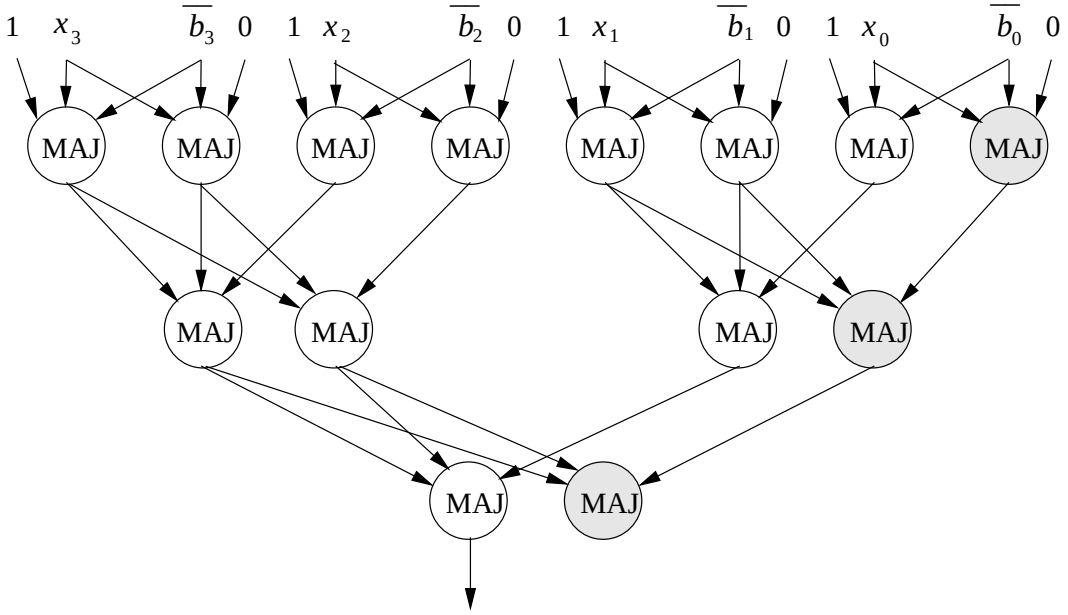

Figure 6: A 4-bit comparator architecture which produces a 1 when $x_3x_2x_1x_0 \geq b_3b_2b_1b_0$. (The majority gates in gray need not be implemented.)

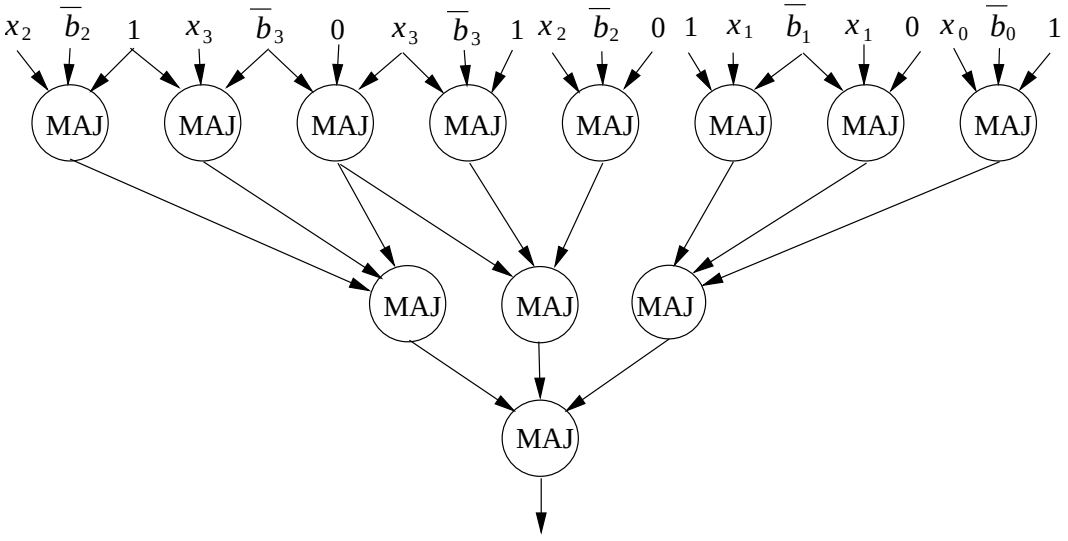

Figure 7: The same comparator as in Fig. 6 after elimination of wire crossings.

three input majority gate with inputs p_1, q_1 and q_0.

Similarly, the truth value of $X > B$ can also be computed from the intermediate logical variables p_i and q_i. In particular, $X > B$ if either $X_1 > B_1$ or $X_1 \geq B_1$ and simultaneously $X_0 > B_0$. This can be expressed by the Boolean expression $p_1 + q_1 p_0$. Once again using the fact that $p_1 = p_1 q_1$ and $q_1 = p_1 + q_1$, the expression for $X > B$ can be rewritten as $p_1 q_1 + (p_1 + q_1)p_0$. But this says that $X > B$ can be computed with a 3-input majority gate with inputs p_1, p_0 and q_1.

To compute p_i and q_i, the operands X_i and B_i can be partitioned recursively and the same procedure used. This can be continued till each partition is a single bit. Fig. 6 shows an implementation of a 4 bit comparator using only inverters and 3-input majority gates derived by this strategy. In every pair of majority gates in this figure, the gate on the left computes q_i and the one on the right, p_i.

The number of majority gates used in this realization of an n bit comparator is $4n - \lceil log_2 n \rceil - 3$ and the delay of the structure is $\lceil log_2 n \rceil + 1$. The tree-like structure allows for easy separation of clocking zones in QCA architectures. Further, if one of the strings, say B, is constant, then all the $\bar{b}_i$ are known in the design and one can apply 1's and 0's, as appropriate at the inputs that expect $\bar{b}_i$'s.

A two bit comparator architecture layout is illustrated in Fig. 8. It should be noted that for n as small as

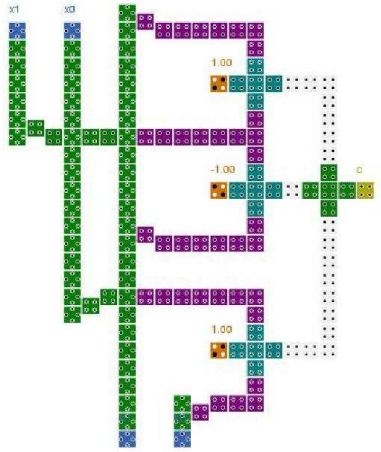

Figure 8: Layout of a 2 bit comparator in QCA technology.

2, the proposed strategy actually will have the same delay but more gates than the sequential strategy of Fig. 5. One may note that the advantage of the new architecture in Fig. 6 can be realized only for larger values of n. It reduces the comparator delay from n to $\lceil log_2 n \rceil + 1$, while increasing the number of majority gates from n to $4n - \lceil log_2 n \rceil - 3$.

However, as can be seen from Fig. 6, the tree architecture has wire crossings at every level of the tree. QCA being a planer architecture, minimizing these wire crossings is important. To achieve this objective, we duplicate certain majority gates as shown in Fig. 7. Note that this conversion does not affect the architecture but it completely eliminates the crossing of wires from all tree levels except the first. The resultant comparators of 2, 4, 8 and 16 bits use only 4, 12, 33 and 89 majority gates respectively.

4 CONCLUSION

This paper provides a new QCA architecture for the comparison function with an $O(\log n)$ delay and $O(n)$ gate complexity for an n bit comparator. Using this comparison realization, one may be able to obtain better (low depth) implementations of some Boolean functions.

REFERENCES

[1] C. S. Lent, P. D. Tougaw, W. Porod, and G. H. Bernstein, "Quantum cellular automata," *Nanotechnology*, vol. 4, pp. 49–57, Jan. 1993.

[2] "The international technology roadmap for semiconductors: Emerging research devices." http://www.itrs.net/, 2005.

[3] K. Walus, G. A. Jullien, and V. S. Dimitrov, "Computer arithmetic structures for quantum cellular automata," in *Proc. 37th Asilomar Conf. Signals, Systems and Computers*, (Pacific Grove, CA), pp. 9–12, Nov. 9–12 2003.

[4] I. Hanninen and J. Takala, "Binary multipliers on quantum-dot cellular automata," *Facta Universitatis Ser.: Elec. Energ.*, vol. 20, pp. 541–560, Dec. 2007.

[5] H. Cho and E. E. Swartzlander, "Adder designs and analyses for quantum-dot cellular automata," *IEEE Trans. Nanotechnology*, vol. 6, pp. 374–383, May 2007.