Technology Mapping for QCA Devices

[1]Stèphano Gonçalves, [2]Mayler Martins, [1]Melissa Colvara, [2]André Reis, [2]Renato Ribas, [1]Leomar Rosa Jr., [1]Felipe Marques
{smmgoncalves,mdsrcolvara,leomarjr,felipem}@inf.ufpel.edu.br,
{mgamartins,andreis,rpribas}@inf.ufrgs.br,

[1] Universidade Federal de Pelotas
[2] Universidade Federal do Rio Grande do Sul

Abstract

The Quantum Cellular Automata (QCA) is an emerging computation technology with great potential to replace the CMOS technology. In this paper we present an algorithm to generate QCA libraries with the minimum cost. Besides, we investigate technology mapping with different QCA libraries, analyzing the impact of each library with reference to the number of inverters and majority gates. We also show the same analysis with some FPGA mapping methods. The FPGA experiments showed a lesser number of gates and no inverters since the LUTs already implement them. For the library mapping, the larger library showed the better results although poor.

1. Introduction

The complementary metal-oxide semi-conductor (CMOS) technology is reaching it's physical limits [1]. New technologies, such as Quantum Celular Automata (QCA) have been proposed as a replacement of the CMOS technology. QCA can be used to design general-purpose computational and memory circuits and is expected to achieve high device density, extremely low power consumption, and very high switching speed.

QCA provides a new method of computation and information transformation. In QCA, binary information is encoded by the configuration of electrical charges in a QCA cell. Computation is realized via the Coulombic interaction between neighboring cells. Because of the Coulombic nature of quantum cells, current does not flow between cells. Moreover, power dissipation in QCA circuits is ultra low compared with conventional CMOS circuits [2][3]. The fundamental QCA logic devices are the *three-input majority gate*, *wire*, and *inverter*. The QCA digital architectures are combinations of these cellular automata structures.

Currently, we could not find any work related to automatic generation of QCA libraries (using majority gate arrangements). Besides, there is no straightforward solution for mapping a digital circuit for this technology. Therefore, we present an investigation about the impact of using different libraries and FPGA methods in QCA technology mapping. The ABC tool [4] was used to perform the experiments using five FPGA mapping algorithms and three different libraries of containing different sets of majority gate arrangements and inverters.

This paper is organized as follow. Section 2 reviews an algorithm that can be used to automatic generate majority gate arrangements. The results of our experiments are discussed in section 3. Finally, the conclusions and future works are presented in section 4.

2. Majority-Based Library

The library concept in CMOS digital circuits is well established. Most of the digital designs today use the standard-cell flow and there are many commercial tools available to synthesize, map and place standard cells. As the QCA is a recent concept, there are no academic or commercial tools available to generate a library using only majority gates and inverters. In [5], was appointed a set of 13 functions of 3 variables mapped using only majority gates and inverters. This set is also the set of functions called 3-NPN. A NPN set is a class of functions equivalent to each other, considering the permutation of its inputs, complementation of its one or more inputs, and/or inversion of its output. In [6], some of the functions had the number of majority gates reduced. But this mapping exercise was done by hand, and this is a hard work to do with a P class (only functions that are equivalent by doing permutation its inputs). The 3-P has 80 functions, and the 4-NPN has 222 functions, making it unfeasible without computational aid.

The Functional Composition [7] (FC) is a novel synthesis paradigm that performs bottom-up association of Boolean functions as opposed to top-down functional decomposition. By performing bottom-up process, FC has a better control of the implementation cost of the final function. By relying on a canonical

representation of a Boolean function, FC can perform a more complete search of the solution space, yielding better results.

Functional composition is based on the following principles: {1} representation of logic functions as a bonded pair of functional/structural representations; {2} it starts from a set of initial functions; {3} simpler functions are associated to create more complex ones; {4} a partial order that enables dynamic programming is respected; {5} a set of allowed functions is maintained to reduce execution time/memory consumption.

In [8], was used the FC algorithm to implement a Boolean factoring. Using the literals as a partial order set, the dynamic programming can be done associating functions associated with a optimal literal count to create more complex functions. The idea was adapted to use majority gates.

The exact factored form of a majority gate is:

$$MAJ = a * (b + c) + b * c \qquad (1)$$

A method to compose a majority port using Boolean functions needs four operations, the same number of operators in (1). Since the majority gate is positive unate (e.g. positive unateness can be considered as passing the same slope (no change in the input) and negative unate is passing the opposite slope), the input assignment cannot generate an inverted polarity output, thus the necessity of inverters. The order of inputs is not important, since the majority function is symmetric.

Following the principles, in {1} is used a pair {Boolean function, majority port}. The partial order set used to define {4} is the logic depth, e.g. max no. of majority gates a signal need to travel from the input to output. Since the objective is to minimize the area in library, it is needed the exact results, so the set in {5} is all functions. To generate the initial functions to attend principle {2}, there are 3 steps: Generate all majority ports with one variable assigned, and the other pins assigned in zero and one (the output function is the input function); Generate all 2-combination of all inputs, and the other input assigned in zero or one (the output function is a AND or OR function of the inputs, respectively); and at last, generate all 3-combination of all inputs. This composes all functions allocated in the first depth. In principle {3}, to generate a n-th depth function, is necessary at least one function representing the (n-1)-th depth, and the two other functions can be of any level. All bonded-pairs are stored in memory. If a bonded-pair is generated, the function associated is allocated in memory and its structure has less majority gates than the allocated in memory, the structural representation is replaced, to ensure the minimal number of majority gates. Using this methodology, we are able to build libraries containing cells with three or more inputs.

3. Experimental Results

Currently, the main goal of QCA synthesis is to achieve smallest number of majority gates and inverters needed to implement a given digital circuit. In order to analyze how good is the synthesis achieved by the state-of-art algorithms for technology mapping considering QCA devices, we have run a set of experiments on the ABC tool. For these experiments, a subset of the ISCAS benchmarks were used.

First, the five FPGA mapping methods were run for each circuit. Since a majority gate relies on three variables, each circuit was mapped considering *Look-Up Tables* (LUTs) of 3 inputs. The FPGA methods available on ABC are implemented by the commands *if*, *fpga*, *ffpga*, *imfs* and *lutpack*. The *fpga* command [9] implements an algorithm that uses improved cut computation (stores the cuts only in the mapping frontier), two complementary heuristics (Area Flow and Exact Area) for area recovery, and lossless synthesis. The *ffpga* command [10] implements an simple algorithm based in priority cuts. Priority cuts, Area Flow and Exact Area and cut expanding, are used on the method *if* [11] to improve area recovery. The *imsf* command [12] is a SAT-based re-synthesis package which relies on windowing, re-substitution, SAT solving, and interpolation. The approach is based on several heterogeneous algorithms, which include structural analysis, random and constrained simulation, and manipulation of Boolean functions using a SAT solver. The *lutpack* command [13] is a re-synthesis algorithm based on co-factoring and disjoint-support decomposition and is capable of finding the smallest network of k-LUTs (where *k* is the number of variables) needed to implement the function. This method is several orders of magnitude faster than previous algorithms that relies on BDDs for functional decomposition or on Boolean Satisfiability for FPGA architecture evaluation since it exploits the Boolean structure of the function being mapped and uses truth-tables to represent functions.

As a second step, the library mapping method were run using three different libraries for each circuit. The library mapping is implemented by the command *map*. The library mapping algorithm [14] is based on a simplified cut-based Boolean matching, lossless synthesis and supergates. To run the experiment, we have built three libraries. The QCALib is a very simple library composed by a 2-input AND gate, an inverter and teh constants ZERO and ONE. The QCALib2 is an expanded library which contains the thirteen functions defined in [5] The QCALib3 has forty primitive functions which can be implemented by a single majority gate [15]. Each gate in the libraries can be implemented by a certain number of majority gates. Each majority gate

arrangement has a area cost that is proportional to the number of majority gates and inverters used into the arrangement.

3.1 FPGA Mapping

The commands *imfs* and *lutpack* were run ten times (as suggested in the tool manual), followed by a sweep. The *sweep* method [4] performs the following tasks: removes nodes without fanouts, collapses buffers and inverters into their fanouts, propagates constants, and removes duplicated fanins. For *if, fpga* and *ffpga* methods, the following command line were run ten times, followed by a sweep: *choice; method –K 3*. The command *choice* converts the currently stored AIG snapshots into a FRAIG and sets it to be the current network in order to apply technology mapping [4]. The option *–K 3* restricts the number of inputs of a LUT to three. Table 1 shows the results of the FPGA mapping. The methods *imfs* and *lutpack* obtained a smaller number of LUTs in most of the circuit. Both methods use Boolean algorithm and functional decomposition to decompose the circuit on 3-input LUTs. On the other hand, they are not so good on circuits that have internal nodes with large fanout.

Table 1 – Number of LUTs for each FPGA method

	Number of LUTs				
Benchmarks	if	fpga	ffpga	imfs	lutpack
C17	4	4	4	6	4
decod	24	24	24	18	18
C499	118	115	134	110	122
9symml	111	110	122	26	40
C1355	116	116	136	354	122
C1908	145	144	151	266	179
C3540	532	515	612	537	388
C6288	735	736	749	2334	960
alu2	206	209	228	58	59
vda	354	374	527	123	123
ttt2	76	76	106	45	42
frg1	52	51	71	3	3
mux	23	23	46	6	6
i2	142	143	201	35	35
k2	787	720	1125	223	223
uneg	48	49	48	32	32
apex6	360	359	414	220	208
cht	82	82	91	36	36
pcler8	38	36	38	24	24

Table 2 – Number of inverters and majority gates for three QCA libraries

	Inverters			Majority Gates		
Benchmarks	QCALib	QCALib2	QCALib3	QCALib	QCALib2	QCALib3
C17	6	4	0	36	30	23
decod	4	4	4	102	102	102
C499	390	48	131	2331	1089	1589
9symml	146	50	12	1014	744	606
C1355	388	47	125	2325	1032	1586
C1908	296	60	105	1992	1125	1425
C3540	655	342	157	4742	3759	3212
C6288	1833	241	644	11220	4575	8089
alu2	252	92	60	1779	1311	1185
vda	155	142	19	1959	1956	1536
ttt2	106	30	28	726	513	515
frg1	86	46	23	519	408	34
pcler8	56	35	8	357	294	218
mux	34	14	4	240	180	15
i2	217	204	1	1296	1248	639
k2	338	287	42	4386	4191	3374
unreg	116	18	19	639	390	348
apex6	486	330	72	3258	2766	2024
cht	122	4	6	810	456	462

3.2 Library Mapping

The command *map* were also run ten times for each circuit. Table 2 shows a comparison of the mapped circuits, considering three different libraries. The library mapping presents a great number of inverters. It is not good for QCA designs since the inverter area cost is almost two times the majority gate area cost. The QCALib presented the larger number of inverters and majority gates. This is expected since it is a very simple library. In most cases the QCALib3 showed a smaller number of inverters in comparison with QCALib2. This can be explained by the fact that the QCALib3 has many cells with inverters in the primary inputs (PIs) of the cell while QCALib2 has only two cells with some of the PIs negated. In two thirds of the benchmarks, the QCALib3 showed a greater reduction of majority gates, which is expected since it is a library with a greater variety of cells.

4. Conclusions and Future Work

This work investigated the differences of using three QCA libraries on ABC library mapping as well as five FPGA mapping methods. A group of benchmarks were run for each method and library, and results regarding number of inverters and majority gates were obtained. The number of gates (LUTs) resulted from FPGA mapping is significantly smaller than the number of majority gates plus inverters obtained through library mapping. However, these numbers are not comparable since the FPGA mapping results in a set of LUTs that implements a set of functions. In order to achieve comparable numbers, the method presented in section 2 could be used to generate minimal majority gate arrangements to implement each of the functions expressed in the LUTs. Furthermore, this method can also be used to build larger QCA libraries. In general, just like CMOS library mapping, the use of a library with more cells results in area saving.

As future works, we intend to extend the experiments with other libraries and compute the cost of the gates resulted from FPGA methods. Based on these analysis we could be able to propose a new method for QCA mapping. Due to computational complexity, a method to perform QCA mapping should use Boolean techniques and functional decomposition present in FPGA methods and a rich cell library.

5. References

[1] International Technology Roadmap for Semiconductors (ITRS). [Online]. Available: http://www.itrs.net

[2] P. D. Tougaw and C. S. Lent, “Logical devices implemented using quantum cellular automata,” **J. Appl. Phys.**, vol. 75, no. 3, pp. 1818–1825, Feb. 1994.

[3] Lent, C.; Tougaw,P. “A device architecture for computing with quantum dots,” **Proc. IEEE**, vol. 85, no. 4, pp. 541–557, Apr. 1997.

[4] Mishchenko, A.; Chatterjee, S.; Brayton, R.; Wang, X.; Kam, T. “Technology Mapping with Boolean Matching, Supergates and Choices”. **ERL Technical Report**, [S.l.], 2005. http://www.eecs.berkeley.edu/ alanmi/abc/abc.htm.

[5] Rumi Zhang; Walus, K.; Wei Wang; Jullien, G.A.; , "A method of majority logic reduction for quantum cellular automata," Nanotechnology, IEEE Transactions on , vol.3, no.4, pp. 443- 450, Dec. 2004

[6] Momenzadeh, M.; Jing Huang; Tahoori, M.B.; Lombardi, F.; , "Characterization, test, and logic synthesis of and-or-inverter (AOI) gate design for QCA implementation," Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on , vol.24, no.12, pp. 1881- 1893, Dec. 2005

[7] M.G.A.Martins, R.P. Ribas and A.I.Reis. Functional Composition: A New Paradigm to Perform Logic Synthesis. Accepted in ISQED 2011.

[8] M.G.A.Martins, L.S.Rosa Jr, A.B. Rasmussen, R.P.Ribas, A.I. Reis. "Boolean factoring with multi-objective goals," ICCD 2010, pp.229-234.

[9] Mishchenko, A.; Chatterjee, S.; Brayton, R. “Improvements to Technology Mapping for LUT-based FPGAs”. In: IEEE TCAD, 2007. **Proceedings. . .**[S.l.: s.n.], 2007. p.41–49

[10] Cho, S.; Chatterjee, S.; Mishchenko, A.; Brayton, R. “Efficient FPGA mapping using priority cuts”. In: FPGA 07, 2007. **Proceedings. . .** [S.l.: s.n.], 2007.

[11] Mishchenko, A.; Cho, S.; Chatterjee, S.; Brayton, R. “Combinational and sequential mapping with priority cuts”. In: ICCAD, 2007. **Proceedings. . .** [S.l.: s.n.], 2007. p.354–361if

[12] Mishchenko, A.; Brayton, R.; Jiang, J. H.; JANG, S. “SAT-based logic optimization and resynthesis”. In: IWLS ’07, 2007. **Proceedings. . .[**S.l.: s.n.], 2007. p.358–364.

[13] Mishchenko, A.; Chatterjee, S.; Brayton, R. “**Fast Boolean Matching for LUT Structures**”. 2007

[14] Chatterjee, S.; Mishchenko, A.; Brayton, R.; Wang, X.; Kam, T. “Reducing Structural Bias in Technology Mapping”. In: PROC. IWLS 05, 2005. **Proceedings. . .** [S.l.: s.n.], 2005. p.519–526.

[15] Kong, K.; Lu, R.; Shang, Y. “An Optimized Majority Logic Synthesis Methodology for Quantum-Dot Cellular Automata”. **IEEE TRANSACTIONS ON NANOTECHNOLOGY**, [S.l.], v.9, n.2, p.170–183, 2010.