Synthesis of Reversible Circuits from a Subset of Muthukrishnan-Stroud Quantum Realizable Multi-Valued Gates

Nicholas Denler *, Bruce Yen *, Marek Perkowski *, Pawel Kerntopf +
*Department of Electrical and Computer Engineering, Portland State University,
1900 SW 4th Avenue, Portland, OR 97201, USA. mperkows@ee.pdx.edu
+ Institute of Computer Science, Department of Electronics and Information Technology, Warsaw Univ. of Technology,
Nowowiejska 15/19, 00-665 Warsaw, Poland, pke@ii.pw.edu.pl

Abstract

We present a new type of quantum realizable reversible cascade. Next we present a new algorithm to synthesize arbitrary single-output ternary functions using these reversible cascades. The cascades use "Generalized Multi-Valued Gates" introduced here, which extend the concept of Generalized Ternary Gates introduced previously. While there were 216 GTGs, a total of 12 ternary gates of the new type are sufficient to realize arbitrary ternary functions. (The count can be further reduced to 5 gates, three 2-qubit and two 1-qubit). Such gates are realizable in quantum ion trap devices. For some functions, the algorithm requires fewer gates than results previously published [1, 5, 8, 14]. In addition, the algorithm also does conversion from arbitrary ternary logic to reversible logic at the cost of relatively small garbage. The algorithm is implemented here in ternary logic, but generalization to arbitrary radix is both straightforward and sees a reduction in growth of cost as the radix is increased.

1. Introduction

Reversible logic [16] is a promising approach to reduce power consumption in several emerging technologies. It is also a base of quantum circuits [12, 3]. There is a recent interest in multiple-valued quantum computing. It has been shown that most of 2x2 ternary reversible functions are universal [6]. Which family, then, of the numerous universal gates are a good choice for synthesis with respect to high processing power, low gate count cost, and simplicity of design? Picton [15] proposed reversible MV gates which were not efficient to realize, especially using quantum primitives, and lead to inefficient structures. No synthesis method was given. Several new MV reversible gates and respective circuit structures were proposed in [1, 2] but the issue of their quantum realization was not addressed and in some designs the garbage may be high. De Vos proposed two universal 2*2 ternary gates [4] together with two 1*1 permutative ternary gates. Two universal quantum gates (more general than permutative reversible gates) have been proposed by Stroud and Muthukrishnan. Their paper [11] presents realization of such gates in ion trap technology. Based on gates of De Vos and Stroud/Muthukrishnan, we proposed [13, 14] a set of gates that generalize De Vos gates and generalize one particular realization (permutative) of Stroud/Muthukrishnan gates. It was shown in [13, 14] how a ternary Toffoli gate can be build from our Generalized Ternary Gates (GTGs). Synthesis of ternary permutative quantum cascades from ternary counterparts of Toffoli and Feynman gates was discussed in [8]. However, such circuits can be highly non-minimal when the Toffoli-like multiple-valued gates (which are not directly quantum-realizable) are built using GTG gates, or using physically realizable gates from [6]. Therefore, recently we became interested in synthesis of ternary reversible cascades directly from GTG gates and their special cases, as well as with new realizable generalizations of GTG gates [7]. We believe that synthesis algorithms should be created only for gates about which we know that they are quantum-realizable and we can at least approximate their real realization costs.

In this paper, we propose an algorithm to systematically synthesize an m-valued (in particular, ternary) function with an arbitrary number of inputs, [14]. The synthesized implementation is a cascade of Generalized Multiple-Valued Gates (GMVGs) of arbitrary radices. In ternary case, the gates are special cases of GTGs. While there were 216 GTGs, a total of 12 ternary gates of the new type are sufficient to realize arbitrary ternary function. (The count can be further reduced to 5 gates, three 2-qubit and two 1-qubit). Some 1-qubit permutation gates [7, 8, 9, 14] that are more difficult to realize as quantum primitives are now avoided. We present experimental results that show the complexity and cost of the implementations on ternary benchmark functions from [9].

The paper is organized as follows: section 2 presents background on the new gates. The minimization technique for multi-valued expressions (and in particular, ternary expressions) discussed in Section 3 is entirely different from previous methods [9, 1, 2, 13, 8, 7] and is efficient. The basic algorithm is next enhanced in Section 4. In section 5 we present some experimental results of ternary benchmark functions using both the basic and the enhanced algorithms. We also discuss the complexity and cost functions of the enhanced algorithm. Finally, we draw some conclusions and discuss future work to be done in this area. This paper

assumes that the reader is familiar with the basic concepts of multiple-valued logic synthesis [5].

2. Background: GMVG gates

Research in the synthesis of multiple-valued quantum reversible circuits remains relatively immature. In our work, we adapted the paradigms from both EXOR logic and evolutionary algorithms [9, 1, 2, 13, 8, 10, 7, 14]. We use EXOR-logic rather than OR-logic since EXORs are the combining operators in binary quantum gates. Analogously, Modulo-Sums are the best choice for combining operators in ternary reversible logic. It is clear that some circuits synthesized by this method are far from minimum; they create high garbage, and/or use costly methods to realize gates, such as 3*3 Toffoli, n-input Toffoli, or Swap.

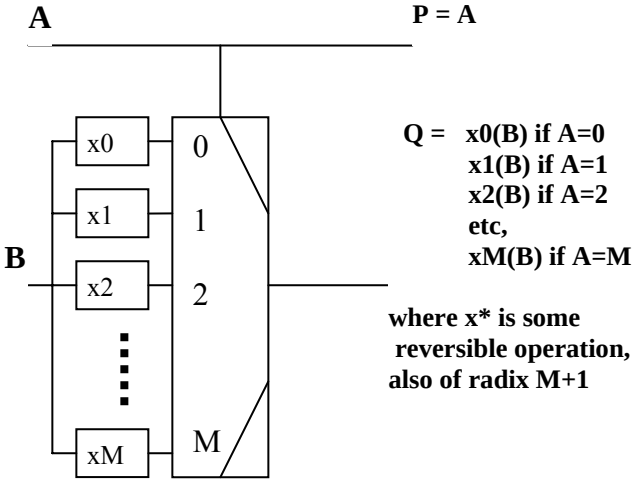

Figure 1. Generalized Multi-Valued Gate of Radix M+1.

Below we propose a synthesis algorithm, which, for ternary functions, utilizes a subset of quantum-realizable [11] generalized ternary gates (GTGs), as defined in [9]. For higher radix multi-valued functions, we apply the same algorithm using an extension of the GTG, the Generalized Multi-Valued Gate (GMVG). The GMVG is a multiplexing (conditional) gate analogous to the GTG with n-inputs. The GMVG is depicted in Figure 1. Only gates in which one-qubit xi operations are adding of constants (standard shift literals [8]) are used in this paper. In the following section we present the basic algorithm for combining these GMVGs into a cascade to realize any function in a form which is somehow similar to the well known sum-of-products and exclusive-or-sum-or-products forms, and especially to their special case – the-sum-of-disjoint-products form (called DSOP – disjoint sum of products in the literature).

The gate from Figure 1 can be denoted by [A, x0, x1, x2, … xM]. Using this notation and denoting by +I operations of adding a value of I to the argument, it can be easily proved that the following set of 12 ternary gates is universal: [X, +1, +0, +0], [X, +0, +1, +0], [X, +0, +0, +1], [X, +1, +1, +0], [X, +1, +0, +1], [X, +0, +1, +1], [X, +2, +0, +0], [X, +0, +2, +0], [X, +0, +0, +2], [X, +2, +2, +0], [X, +2, +0, +2], [X, +0, +2, +2]. There exists also a 5 gate universal set with 2-qubit gates: [X, +1, +0, +0], [X, +0, +1, +0], [X, +0, +0, +1], and 1-qubit gates +1 and +2 (these are the so-called cyclical shifts that add 1 or 2 modulo three). This can be further reduced to one 1-qubit gate from quantum realizable gates [11] and two ternary shift-gates.[1]

3. Basic Cascade Mapping Algorithm

The algorithm generates a cascaded implementation of reversible generalized multi-valued gates (GMVGs). A ternary function *f* can be specified with two sets of DSOPs: one set for value 2 of function *f* and another for value 1 of function *f*. We will call them "value DSOPs" and will denote them DSOP-2(*f*) and DSOP-1(*f*), respectively. Those neither specified by DSOP-2(*f*) or DSOP-1(*f*) are assumed to be of value 0. The algorithm generates a cascade of GMVGs. Choosing disjoint sum of product implicants for many logic functions results in poor implementation. A method to relax this constraint and improve the resultant implementation is discussed in section 5.

The basic algorithm applies to arbitrary functions F, of N variables with radix M+1 (where M = 2 implies ternary, etc). First a GMVG cascade is created for each product group in DSOP-*k*, where $1 \leq k \leq M$. This product group may or may not be a prime implicant of the output value of F. However, it is always a product implicant of the standard DSOP corresponding to the replacement of an output value *k* by Boolean value 1. Each resulting cascade can be implemented separately. Each of the product groups are mutually exclusive, and they are combined using another GMVG cascade, called the OR-cascade. A complete implementation of function F can be realized by connecting the OR--cascade serially after the longest product cascade. This joining cascades is referred to as a cascade of cascades or combined cascade.

For each product cascade, the quantum line begins with constant input "0". There is a GMVG corresponding to each literal in the product term, plus additional roll-over GMVGs as needed, described below. Thus it is advantageous to have large prime implicants in every value expression for the original function F, not only because it may reduce the total number of product implicants and thus the number of product cascades, but also because the

[1] In [11] it is not explicitly stated that GMVGs generalized to the form from Figure 1 are directly quantum realizable in ion trap. Whether or not gates such as these will be realizable directly is not certain to these authors based on [11] and discussions with physicists. However, in ternary case, from the gates as presented by Muthukrishnan and Stroud, and two shift operators, we can create all ternary GMVG gates as illustrated on an example in Figure 10. Thus the GMVGs can be treated as macros or high level gates.

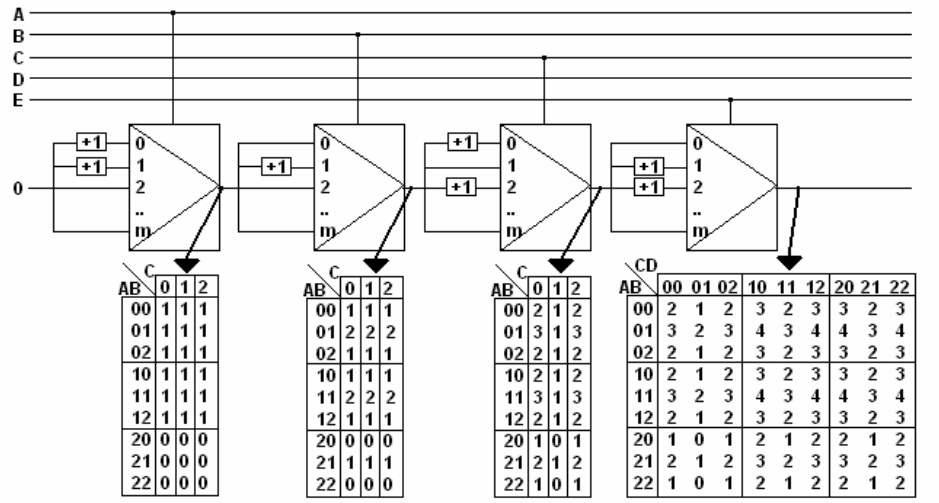

Figure 2. Example 1 of m+1 valued cascade for a disjoint product implicant

product cascades will be shorter. Each GMVG corresponding to a literal will have its select line driven by that literal. The GMVG will have +1 operations at all GMVG inputs corresponding to that literal's coverage in the product implicant.

Example 1. Assume function F of five variables A, B, C, D, E with product implicant A $^{\{0,1\}}$ B $^{\{1\}}$ C $^{\{0,2\}}$ E $^{\{1,2\}}$, the product cascade will include a GMVG with control input driven by A, with +1 operations at the 0 and 1 selects of the corresponding GTG. B, C, and E make contributions in like form. Input D controls no GTG as it makes no contribution to the product term. The cascade is shown in Figure 2. In the rightmost MV Karnaugh map in Figure 2, the value 4's give the DSOP implicant given above. All other values are considered nonsensical and are discarded in the following steps. Converting this highest value (and eliminating the lesser values) is accomplished by the OR-cascade.

Realizing the product cascade is, in general, non-trivial. Realization depends on the radix and the number of literals. It should be clear that, with a low value of M or with many inputs in the product term, the +1 combining operators executed in the gates will eventually cause values of M-1 to "roll-over" as +1 is considered to be modulo-M addition. In the event of "roll-over," we no longer accurately track the maximum value. If the example above represented a ternary function, values "3" become non-distinguishable from the unchanged "0" values.

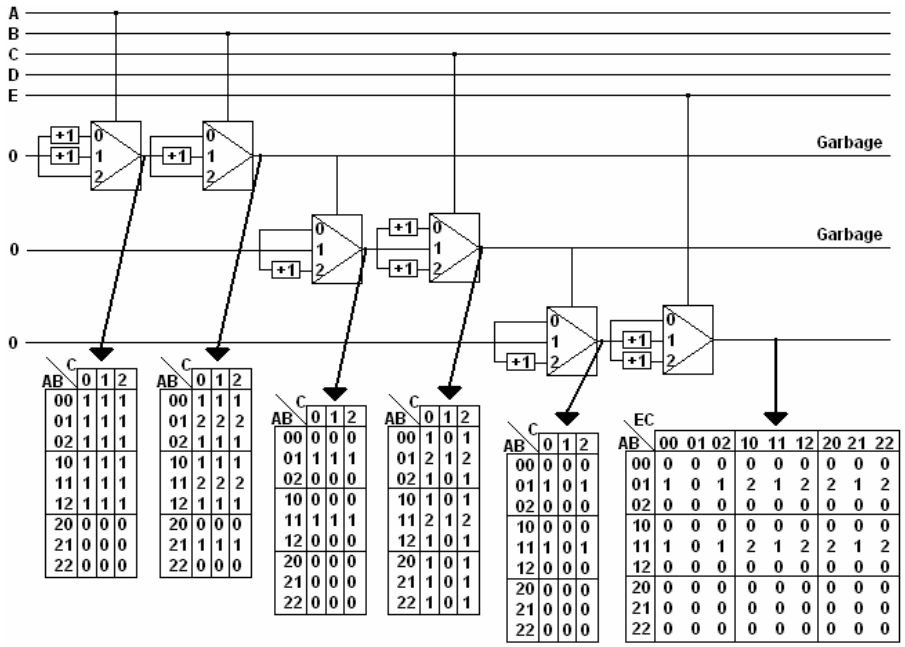

Figure 3. Ternary cascade of cascades to realize a product cascade for Example 1.

Figure 4. A four variable, ternary function Karnaugh map for example 1.

AB\EC	00	01	02	10	11	12	20	21	22
00	0	0	0	1	1	1	1	1	1
01	1	0	1	2	1	2	2	1	2
02	0	0	0	1	1	1	1	1	1
10	0	0	0	1	1	1	1	1	1
11	1	0	1	2	1	2	2	1	2
12	0	0	0	1	1	1	1	1	1
20	0	0	0	1	1	1	1	1	1
21	0	0	0	1	1	1	1	1	1
22	0	0	0	1	1	1	1	1	1

When "roll-over" occurs, additional constant inputs are introduced. X represents the number of GMVGs on a given line. The value of X is incremented for each GMVG that is placed in the cascade. When X = M-1, an additional GMVG is introduced on a new line (again starting with constant "0"). This is denoted by GMVG-X. The control line of GMVG-X is provided by the output of the preceding, "rolled-over," cascade. The "+1" operation is placed at the X-th input of the GMVG-X. All other inputs to GMVG-X have no operation (i.e. they are wires). By this method X is reset to value 1 and the cascade is continued.

Therefore, to implement a ternary function (M=2), a GMVG-X is required every two gates. Consequently, GMVG-X cost does not increase as number of states M increases, because GMVG-X gates are needed less often. Using the product implicant from Example 1 for the case M=2, the circuit shown in Figure 3 is created. The desired implicant output is realized by the highest values in the lowest wire in Figure 3. In Figure 4, the maximum values of 2 are correct, and the values of 1 have not yet been changed back to 0's.

Figure 3 depicts the complete product cascade for one value. The other products for each value are implemented successively as cascades. Once all the product implicants for all output values have been synthesized, the OR-cascade, which combines these output values, is placed serially after the longest product cascade. The OR-cascade is a single line, again starting from a constant zero, shown by Figure 5.

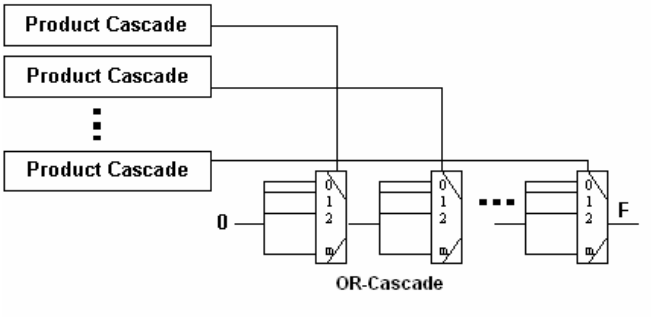

Figure 5. A naïve construction of a cascade of cascades. This OR-cascade combines the results from the product cascades.

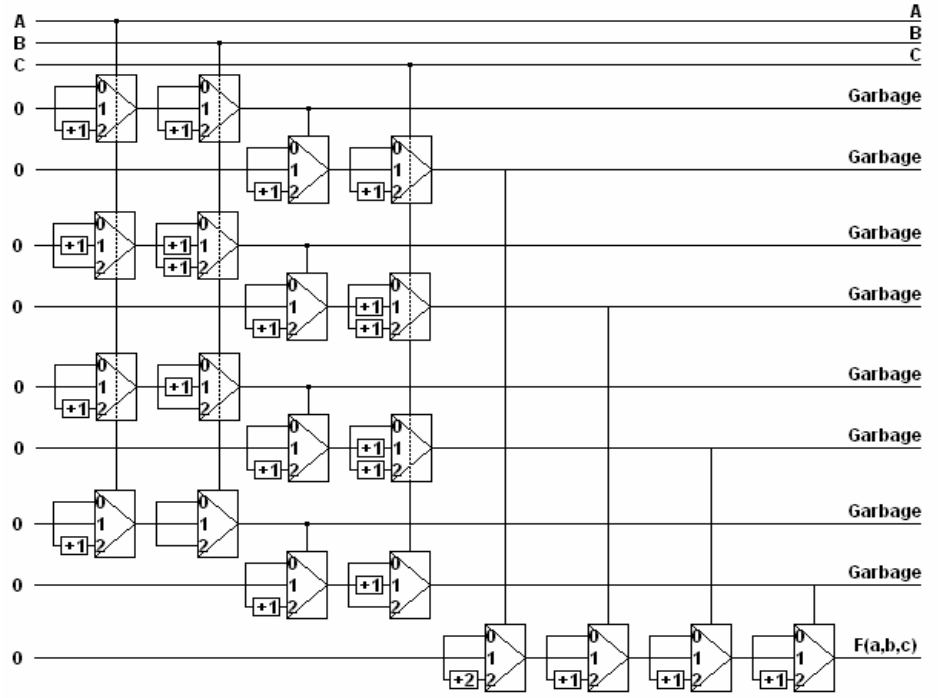

Figure 6. A naïve method of constructing a cascade for function: Min(A, B, C) = 2A {2} B {2} C {2} + 1 A {1} B {1,2} C {1,2} + 1 A {2} B {1} C {1,2} + 1A {2} B {2} C {1}.

The output of each DSOP product cascade drives the control line of one of the GMVG's in the OR-cascade. The OR-line is similar in operation to the GMVG-X "roll-over" gate, except in the sense that we have several GMVG-X gates in series, each of which contributing a disjoint covering for a particular value. That is, shift operators (in this example +1 or +2) will be used as the M-input to the GMVG-X gate to realize each DSOP value (value 1s, +1; value 2s, +2).

Example 2. A complete example is shown in Figure 6 for ternary function Min(A, B, C) = $2A^{\{2\}} B^{\{2\}} C^{\{2\}} + 1\ A^{\{1\}} B^{\{1,2\}} C^{\{1,2\}} + 1\ A^{\{2\}} B^{\{1\}} C^{\{1,2\}} + 1A^{\{2\}} B^{\{2\}} C^{\{1\}}$.

In Figure 6 the first gate of the OR-cascade (lowest line) has +2 in its input 2. This is because of constant 2 in product term $2\ A^{\{2\}} B^{\{2\}} C^{\{2\}}$. The garbage created by this method is large. Reducing this garbage is a goal of subsequent work.

This preliminary algorithm will see good improvement with optimizations under development, which we discuss in the following sections.

4. Improved Cascade Algorithm

AB\C	0	1	2
00	0	0	0
01	0	0	0
02	0	0	0
10	0	0	0
11	0	1	1
12	0	1	1
20	0	0	0
21	0	1	1
22	0	1	2

AB\C	0	1	2
00	0	0	0
01	0	0	0
02	0	0	0
10	0	0	0
11	0	1	1
12	0	1	1
20	0	0	0
21	0	1	1
22	0	1	2

Figure 7. (a) Disjoint groups for DFSOP-1(F) and DFSOP-2(F), (b) Non-disjoint groups for DFSOP-1(F) and DFSOP-2(F).

It is evident from Min (A, B, C) in Example 2 that the constraint of requiring product groups to be mutually exclusive results in an excessive number of groups. This requirement is not necessary, but was convenient for straightforward implementation of the OR-cascade. However, the OR-cascade can be created in such a way as to effectively layer coverings to create the proper output, similar to EXOR logic.

Figure 7 shows two possible coverings for MIN(A,B,C). As realized previously, (a) is composed of DSOP coverings. However, in (b), and the subsequent realization of (b) in Figure 8, we see that a more effective covering would be to choose coverings that conveniently cover implicants some Y number of times, Y being the particular value of the logic to be implemented, assuming only +1 operators across the OR-cascade. Even more freedom can be achieved by introducing all available 5 shift operators [8,9] in the OR-cascade. Ultimately this results in fewer, larger product implicants, and thus in fewer product cascades, shorter combined cascades, and a smaller garbage.

5. Experimental Results and Future Work

The algorithms have been implemented in C++. The following results have been obtained for solutions both with and without the overlapping product implicant improvement (Table 1). The test-bench functions used here are the same or similar to those used in [9] (All benchmarks that have letter G (for Galois) are the same as in [9], other are new).
The details of the functions are defined in Appendix A. In several cases an improved algorithm (right part of the Table) gives a large savings over the basic algorithm (i.e., benchmark 4cyM3). Several observations can be made about the above data. It can be easily shown that the number of garbage outputs is always one less than the number of constant inputs.

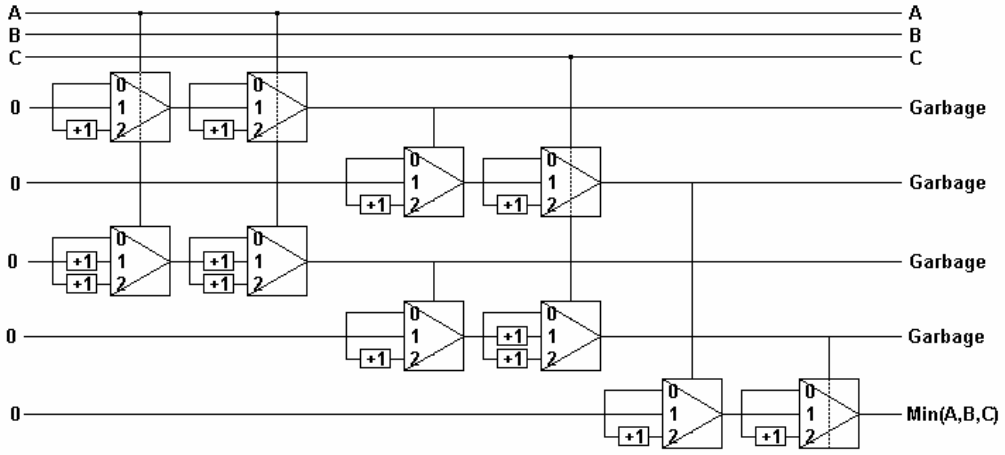

Figure 8. An improved cascade for the function from Example 2: Min(A, B, C) = $2\ A^{\{2\}} B^{\{2\}} C^{\{2\}} + 1\ A^{\{1,2\}} B^{\{1,2\}} C^{\{1,2\}}$. Of interest is the overlap between the DSOP-1 and DSOP-2.

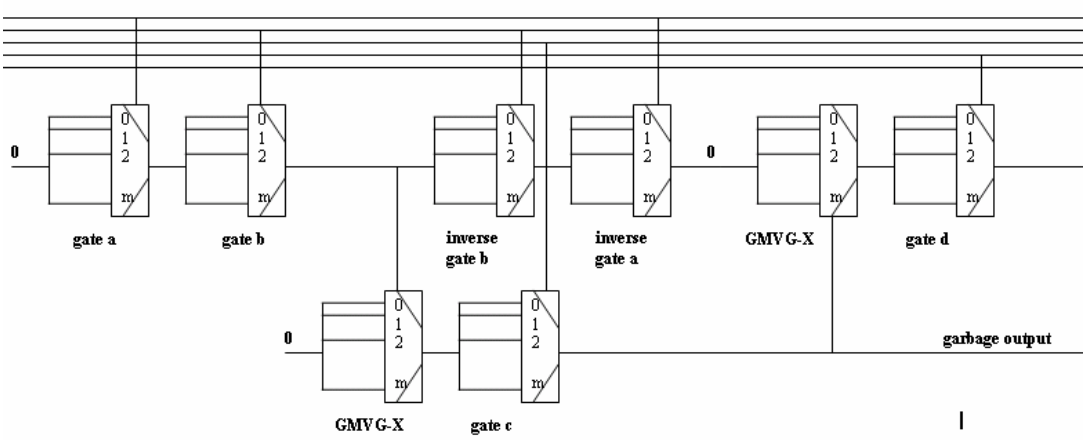

Figure 9. Illustration of the "return to constant" approach to decrease the garbage bit count.

One obvious drawback of the algorithm, as presented, is the use of at least one constant input per product group. The maximum cost function for constant inputs for a function of N inputs, power M, and PI product groups is

Max Constant Inputs = PI[(1)+(max # of GMVG-Xs)] + (1 for OR-cascade) = PI[(1)+(int N/(M-1))] + 1

Similarly, the number of garbage outputs equals the number of constant inputs used for the product group cascades.

Max Garbage Outputs = PI[(1)+(int N/(M-1))] + 1

Often one needs less than the maximum number of GMVG-Xs per prime implicant, due to some primary inputs not being present in a given product implicant. Still, for functions with a large number of product implicants, there are a high number of constant inputs. In the future, the algorithms that expand the ideas from ESOP and GFSOP logic [7,8,9] should be created to minimize the total number of overlapping product groups that are not necessarily the product implicants.

One approach to reduce the garbage cost [13, 18] is to reuse lines by changing garbage outputs into a more useful form. Because the GMVGs are all reversible, then we can apply the inverse GMVGs (once we are done with that line) to return to constant zero. We can then reuse this line instead of introducing a new constant. This concept is illustrated in Figure 9.

Another cost function to consider is the maximum length of the cascade, which is effectively the length of the longest prime-implicant cascade, plus the length of the combining cascade. This can be expressed as follows:

Max Length = (Max Length of PI Cascade) + (Length of OR Cascade) = (N + (int N/(M-1))) + (PI)

The upper bound on both the length and total gate cost functions can be reduced by having fewer, larger, prime implicants. The algorithm is most affective for large values of M and low values of N.

No specific method was proposed here to find a minimal covering expression. We believe that our general approach will allow for cost-effective design of large functions under condition that a good choice of prime implicants in all layers will be done. This is our current research.

6. Conclusion

We introduced a new type of multi-valued quantum cascade that is realizable in ion trap technology following [11]. This is in contrast to previous papers that were not concerned with the realization ability of their multi-input gates. The algorithm is implemented here in ternary logic, but generalization to arbitrary radix is both straightforward and sees a reduction in growth of cost as the radix is increased.

We proposed two algorithms to synthesize single-output functions with arbitrary number of inputs and arbitrary radix. We have evaluated the algorithm using ternary benchmark functions from [9]. The algorithms are very fast and allow to minimize large functions, but their quality is not yet satisfactory, although the second algorithm is a substantial improvement over the first one.

We are working on several additional improvements that can be made to the presented second algorithm. Further effort should be made to reduce the impact of, or ideally remove, the constraint of mutually exclusive product groups in a covering. The method should be also extended to multi-output functions. We work on minimization techniques based on exhaustive search, evolutionary approaches and extensions of the algorithms presented in

this paper. New algorithms to find best covering of incomplete MV functions to be used as preprocessors to the current algorithms should be also created.

7. References

1. Al-Rabadi, "Synthesis and Canonical Representations of Equally Input-Output Binary and Multiple-Valued Galois Quantum Logic," *Technical Report* #2001/008,ECE Dept., PSU, August 2001.
2. A. Al-Rabadi, "Novel Methods for Reversible Logic Synthesis and Their Application to Quantum Computing", *Ph. D. Thesis*, PSU, Portland, Oregon, USA, October 24, 2002.
3. J. L. Brylinski and R. Brylinski, "Universal Quantum Gates", *Mathematics of Quantum Computation*, CRC Press, 2002, LANL e-print quant-ph/010862.
4. A. De Vos, B. Raa, and L. Storme, "Generating the group of reversible logic gates", *Journal of Physics A: Mathematical and General*, Vol. 35, 2002, pp. 7063-7078.
5. E. Dubrova, "Multiple-Valued Logic Synthesis and Optimization," in *Logic Synthesis and Verification* (edited by S. Hassoun and T. Sasao), Kluwer Academic Publishers, 2002, pp. 89-114.
6. P. Kerntopf, M. Perkowski, M.H.A. Khan, "On Universality of General Reversible Multiple-Valued Logic Gates," *Proc. ISMVL 2004*.
7. M.H.A. Khan and M.A. Perkowski, "Genetic Algorithm Based Synthesis of Multi-Output Ternary Functions Using Quantum Cascade of Generalized Ternary Gates," *Proc. Congress of Evolutionary Computation*, 2004.
8. M.H.A. Khan, M.A. Perkowski, and P. Kerntopf, "Multi-Output Galois Field Sum of Products Synthesis with New Quantum Cascades", *Proc. 33rd IEEE Int. Symp. On Multiple-Valued Logic*, Tokyo, May 16-19, 2003, pp. 146-153.
9. M.H.A. Khan, M.A. Perkowski, M.R. Khan, and P. Kerntopf, "Ternary GFSOP Minimization using Kronecker Decision Diagrams and Their Synthesis with Quantum Cascades", Accepted to *Journal of Multiple-Valued Logic and Soft Computing: Special Issue to Recognize T. Higuchi's Contribution to Multiple-Valued VLSI Computing*.
10. M. Lukac, M. Perkowski, H. Goi, M. Pivtoraiko, C.H. Yu, K. Chung, H. Jee, B-G. Kim, and Y-D. Kim, "Evolutionary approach to Quantum and Reversible Circuits synthesis", *Artificial Intelligence Review*, 20, pp 361-417, 2003.
11. A. Muthukrishnan, and C. R. Stroud, Jr., "Multivalued logic gates for quantum computation", *Physical Review A,* Vol. 62, No. 5, Nov. 2000, 052309/1-8.
12. M. Nielsen and I. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press, 2000.
13. M. Perkowski, A. Al-Rabadi, and P. Kerntopf, "Multiple-Valued Quantum Logic Synthesis", *Proc. of 2002 International Symposium on New Paradigm VLSI Computing*, Sendai, Japan, December 12-14, 2002, pp. 41-47.
14. M. Perkowski, P. Kerntopf, A. Al.-Rabadi, M. H. A. Khan, "Multiple-Valued Quantum Computing. Issues, Open Problems, Solutions," *Technical Report*, Korea Advanced Institute of Science and Technology (KAIST), Taejeon, Korea, 2002
15. P. Picton, "A Universal Architecture for Multiple-Valued Reversible Logic," *Multiple-Valued Logic Journal*, Vol. 5, pp. 27-37, 2000.
16. T. Toffoli, „Reversible Computing", in *Automata, Languages and Programming* (edited by de J. W. Bakker and J. van Leeuwen), Springer Verlag, pp. 632-644, 1980.

8. Appendix A: Ternary Benchmark Functions

n*cyG*r: input x0, x1, ... xn ; output consists of n outputs of r input variables in cyclic order (e.g. 3cyG2: y(a,b,c) = ab + bc + ca), using Galois mod3 multiplication.
n*cyM*r: input x0, x1, ... xn ; output consists of n outputs of r input variables in cyclic order (e.g. 3cyG2: y(a,b,c) = ab + bc + ca), using Min/Max operations.
a2bccG: input a, b, c; output y = (a^2 + bc + c), using Galois mod3 operations.
a2bccM: input a, b, c; output y = (a^2 + bc + c), using Min/Max operations.
avgG*n*: input x0, x1, ... xn ; output y = int [(x0 + x1 + ... + xn) / n], using Galois mod3 operations.
prodG*n*: input x0, x1, ... xn ; output y = (x0 * x1 * ... * xn), where * is Galois mod3 multiplication.
prodMin*n*: input x0, x1, ... xn ; output y = (x0 * x1 * ... * xn), where * is the Minimum operation.
sqsumG*n*: input x0, x1, ... xn ; output y = (x0^2 + x1^2 + ... + xn^2), using Galois mod3 operations.
sqsumM*n*: input x0, x1, ... xn ; output y = (x0^2 + x1^2 + ... + xn^2), using Min/Max operations. This is omitted from the data taken in this paper, because this is the same as sumMaxn!
sumG*n*: input x0, x1, ... xn ; output y = (x0 + x1 + ... + xn), where + is Galois mod3 addition.
sumMax*n*: input x0, x1, ... xn ; output y = (x0 + x1 + ... + xn), where + is the Maximum operation.

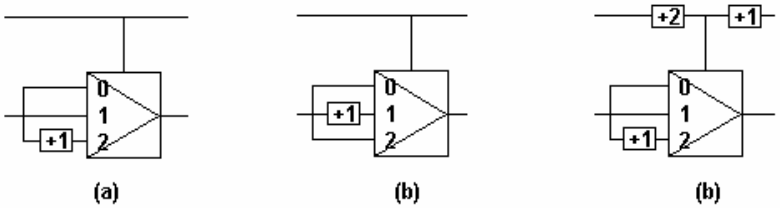

Figure 10. (a) A Conceptual Ternary Muthukrishnan-Stroud Quantum Realizable Gate, represented as a MUX. (b) A more general GTG and (c) its possible realization from a Muthukrishnan-Stroud Quantum Gate (if it cannot be realized directly).

Table 1: Experimental Results

	Mutually Exclusive Products				Overlapping Products			
	Garbage	Constant	Length	Total Gates	Garbage Outputs	Constant Inputs	Length	Total Gates
2cyG2	4	5	6	12	3	4	5	9
2cyM2	3	4	5	9	2	3	4	6
3cyG2	22	23	16	56	20	21	15	51
3cyG3	16	17	12	40	10	11	9	25
3cyM2	18	19	13	38	10	11	9	22
3cyM3	8	9	8	20	4	5	6	10
4cyG3	94	95	38	220	73	74	31	171
4cyG4	48	49	22	112	27	28	15	63
4cyM2	28	29	18	66	16	17	14	40
4cyM3	53	54	25	125	21	22	14	50
4cyM4	15	16	11	35	6	7	8	14
a2bccG	15	16	12	38	13	14	11	33
a2bccM	15	16	12	38	12	13	11	31
avgG2	3	4	5	9	3	4	5	9
avgG3	14	15	12	36	14	15	12	36
avgG4	54	55	26	128	50	51	25	119
prodG2	4	5	6	12	3	4	5	9
prodG3	16	17	12	40	10	11	9	25
prodG4	48	49	22	112	27	28	15	63
prodMin2	3	4	5	9	2	3	4	6
prodMin3	8	9	8	20	4	5	6	10
prodMin4	15	16	11	35	6	7	8	14
sqsumG2	3	4	5	9	3	4	5	8
sqsumG3	16	17	12	40	14	15	12	36
sqsumG4	33	34	17	77	29	30	17	69
sqsumM2	4	5	6	11	4	5	6	10
sqsumM3	10	11	10	25	8	9	10	20
sqsumM4	19	20	14	45	14	15	14	34
sumG2	6	7	8	18	6	7	8	18
sumG3	36	37	22	90	36	37	22	90
sumMax2	4	5	6	11	4	5	6	10
sumMax3	10	11	10	25	8	9	10	20
sumMax4	19	20	14	45	14	15	14	34