Structural Approach for Performance Driven ECC Circuit Synthesis

Chauchin Su, Kathy Y. Chen, and Shyh-Jye Jou
Department of Electrical Engineering
National Central University
Chung-Li, Taiwan 32054, R.O.C.

Abstract

ECCGen is a logic synthesizer for error control coding circuits. It takes H matrices as inputs and produces circuit schematics in two steps, literal minimization and gate/pin assignment. Different from conventional logic synthesis tools, it takes a structural approach to avoid the combinatorial explosion problem in Boolean function and/or true table representations of ECC circuits. Moreover, the structural approach also reduce the complexity of timing and area optimization significantly when multiple-input exclusive-or gates are used. The test results show that ECCGen achieves a reduction of 57% in transistor count and 15% in delay time on thirteen industrial ECC circuits.

1 Introduction

Error Control Coding (ECC) is an effective technique for the control of errors in digital communication and data storage systems. Today, it becomes even more important because of the trend of digitization. Both digital communication systems and computer storage devices have ECC circuits built in. The popular ones include Reed-Soloman and convolution codes [2]. For circuit design, the use of logic synthesis tools is indispensable. Logic synthesis tools, such as MIS II [4], BOLD[7], and SOCRATES[8], examine the logic equations, minimize the logic, and create a netlist in terms of library gates. The advantages include correct by construction, fast design cycle, and performance optimization. These tasks are difficult to achieve by human engineers. Unfortunately, the conventional logic synthesis tools have difficulties handling ECC circuits.

Conventional synthesis tools use binary cubes or sum-of-product terms to represent logic functions [3,4,7,8]. The synthesis task is transformed to and solved as set covering problems. Such a representation is impractical for the representation of ECC functions. ECC circuits are composed of binary exclusive-or trees. Every *minterm* in a ECC function is also a *prime implicant*. Therefore, the number of cubes to represent a n-input exclusive-or function requires the use of 2^{n-1} cubes. As a result, these synthesis tools will suffer a serious combinatorial explosion problem for large n. Moreover, the XOR is not a primitive operator in Boolean algebra. For instance, $AABC$ equals ABC while $A \oplus A \oplus B \oplus C$ does not equal $A \oplus B \oplus C$. Hence, Boolean algebra based synthesizers are not suitable for ECC circuits either.

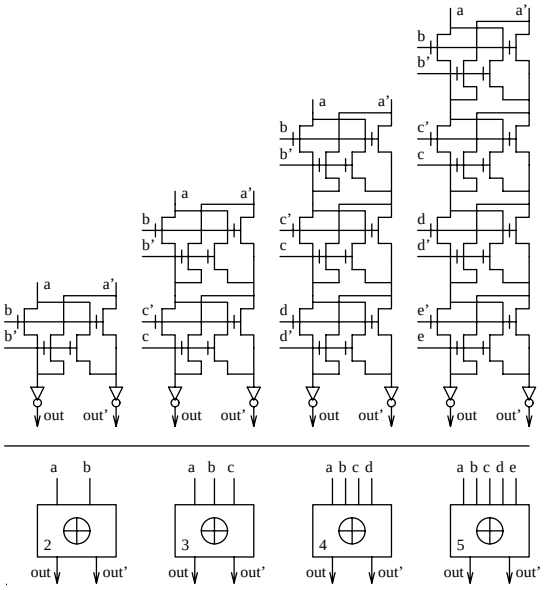

Figure 1: The Multiple-inputs CPL XOR Gates

There are many works focus on the minimization of XOR and logical equivalence switching circuits [9-13]. However, they attempt to solve the problem by expressing an arbitrary switching function in a *modulo-2 sum-of-product* term form or *Reed-Muller* form. A ECC circuit is a special case of the Reed-Muller representation. Every product term contains one only one variable. Moreover, in complementary pass-transistor logic (CPL) structure [6] a XOR gate can have more than two inputs. The circuit diagrams of the multi-input CPL XOR gates are shown in Figure 1. Although the minimization of Reed-Muller form can also handle ECC circuits, it is not sufficient for performance optimization, especially when muti-input XOR gates are used. With multi-input XOR gates, not only literal minimization, pin/gate assignment plays a role of same importance. Therefore, in this paper, we will propose a methodology for performance driven synthesis of ECC circuits, ECCGen.

The proposed synthesis methodology follows the similar route as most synthesis tools, namely, literal

ASP-DAC '97
0-89791-851-7$5.00

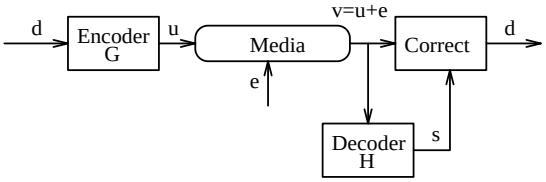

Figure 2: The Block Diagram of Error Control Coding Techniques

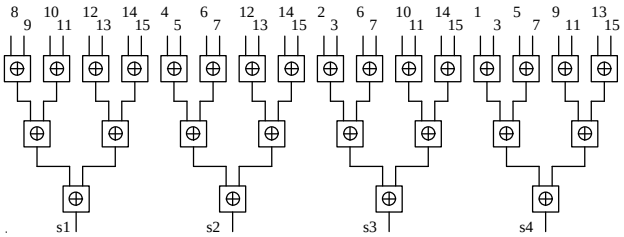

Figure 3: The Original Implementation of H(15,11,3)

minimization and gate/pin assignment. In Section 2, We will first study the ECC circuit structure and its minimization. In Section 3, the set partitioning algorithms for literal minimization are presented in detail. In Section 4, the gate/pin assignment for CPL XOR structure are discussed. In Section 5, the test results on 13 ECC circuits are presented. Finally, the conclusions are given in Section 6.

2 ECC Circuit Design and Minimization

Before the introduction of the synthesis algorithm, let us discuss the encoding and decoding mechanisms of the ECC technique first. The block diagram of a generalized ECC system is shown in Figure 2. Before data d is transmitted, it is encoded by taking an inner product with a *generation matrix* G in GF(2). The result vector u, $u = d \times G$, is called a *codeword*. During the transmission, the codeword may be contaminated by an error vector e and become $v = u+e$. Such an error can be a stuck-at fault in memory systems or noise in communication channels. At the receiving end, the received data v is multiplied by a *parity matrix* H to check if it is a codeword or not. The result parity or syndrome s, $s = H \times v$, indicates the correctness of the received word v. If s is a zero vector then v is an error-free codeword. Otherwise, v is an erroneous non codeword.

The encoding and decoding blocks are similar. They both implement inner products of a vector and a matrix in GF(2). Let us take the decoding circuit of the well known (15,11,3) Hamming code as an example to illustrate the synthesis and minimization of ECC circuits. The **H** matrix of Hamming code is given as follows.

$$\mathbf{H} = \begin{bmatrix} 0\,0\,0\,0\,0\,0\,0\,1\,1\,1\,1\,1\,1\,1\,1 \\ 0\,0\,0\,1\,1\,1\,1\,0\,0\,0\,0\,1\,1\,1\,1 \\ 0\,1\,1\,0\,0\,1\,1\,0\,0\,1\,1\,0\,0\,1\,1 \\ 1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1 \end{bmatrix}$$

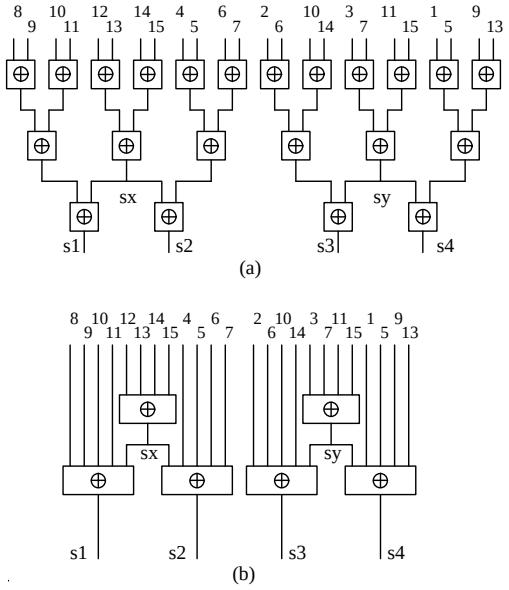

Figure 4: The Shared Implementation of H(15,11,3)

The corresponding syndrome equations are listed below.

$$\begin{aligned} s_1 &= v_8 \oplus v_9 \oplus v_{10} \oplus v_{11} \oplus v_{12} \oplus v_{13} \oplus v_{14} \oplus v_{15} \\ s_2 &= v_4 \oplus v_5 \oplus v_6 \oplus v_7 \oplus v_{12} \oplus v_{13} \oplus v_{14} \oplus v_{15} \\ s_3 &= v_2 \oplus v_3 \oplus v_6 \oplus v_7 \oplus v_{10} \oplus v_{11} \oplus v_{14} \oplus v_{15} \\ s_4 &= v_1 \oplus v_3 \oplus v_5 \oplus v_7 \oplus v_9 \oplus v_{11} \oplus v_{13} \oplus v_{15} \end{aligned}$$

A straight forward implementation of the above syndrome equations is shown in Figure 3. However, if we reorganize the Boolean equations as follows.

$$\begin{aligned} s_x &= v_{12} \oplus v_{13} \oplus v_{14} \oplus v_{15} \\ s_1 &= v_8 \oplus v_9 \oplus v_{10} \oplus v_{11} \oplus s_x \\ s_2 &= v_4 \oplus v_5 \oplus v_6 \oplus v_7 \oplus s_x \\ s_y &= v_3 \oplus v_7 \oplus v_{11} \oplus v_{15} \\ s_3 &= v_2 \oplus v_6 \oplus v_{10} \oplus v_{14} \oplus s_y \\ s_4 &= v_1 \oplus v_5 \oplus v_9 \oplus v_{13} \oplus s_y \end{aligned}$$

The new implementation is shown in Figure 4.a. As a result, the number of XOR gates reduced from 28 to 22. As one can see, s_x is a sharable subtree of s_1 and s_2. So, finding as many sharable subtrees as possible is an alternative way to minimize ECC circuits. Our approach to achieve literal minimization is detailed in the next section.

3 Literal Minimization

Instead of considering the cube covering of the Boolean functions, we achieve literal minimization by finding as much sharable subtrees as possible. In this section, a set modeling for H matrices is proposed, set partitioning for literal minimization is illustrated, and finally, the test results on thirteen EEE circuits are presented.

The Set Modeling and Partitioning of H Matrix

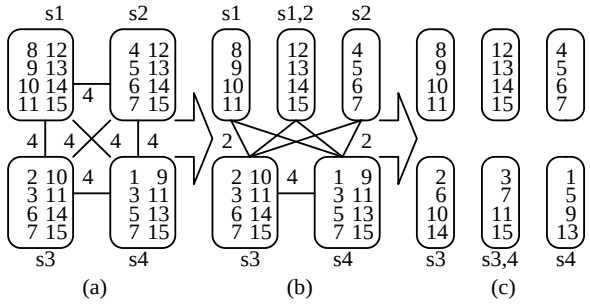

Figure 5: A Step-by-Step Set Modeling and Partitioning for H(15,11,3).

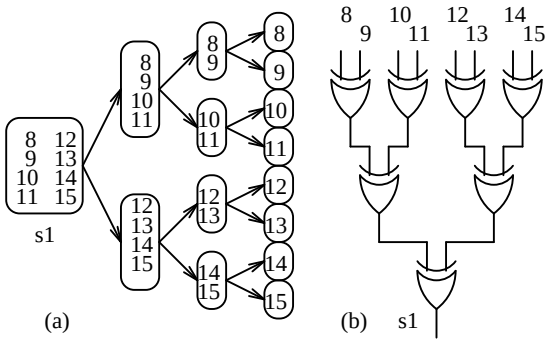

Figure 6: A Partitioning and Implementation of s_1.

The H matrix is modeled as a number of syndrome sets. Each *syndrome set* represents a row of the H matrix. The members of a syndrome set are the non zero bits of the corresponding row. For example, the H(15,11,3) matrix is represented as the four syndrome sets shown in Figure 5.a.

The implementation from a syndrome set to a XOR tree can be regarded as a recursive partitioning process. A set can be partitioned into two subsets. The physical meaning of the partitioning is to create a root XOR gate and determine the fanins to both inputs. Note that, the detail structures of the subtrees are not decided yet. To determine the detail structure of the subtrees, one must keep partitioning the subsets with two or more elements until no more partitioning is possible. After that, the circuit structure is fully defined. For example, for the complete partitioning shown in Figure 6a, the corresponding tree structure is shown in Figure 6b.

To minimize the circuits, we can partition the sets in such a way that one of the subsets is also the subset of the other sets. Such a subset is called a *sharable subset* and such a partitioning, a *sharable partitioning*. For two sets, the maximal sharable subset is the intersection of the sets. To facilitate the sharable partitioning algorithm, we now formally model the ECC matrix as a graph. Each *node* represents a row in H matrix. The weight of an edge is the number of columns common to its two terminal nodes. For H(15,11,3) code, the graphical representation is shown in Figure 5a. With the graphical model, the ECC logic minimization algorithm is abstracted as follows.

LiteralMin()
while ($\exists e(i,j) \geq 2$) {
e(i,j) = the edge of the largest weight
$n_{new} = n_i \cap n_j$; /* Sharable Subset */
$n_i = n_i$ - n_{new}; /* Sharable Partitioning */
$n_j = n_j$ - n_{new}; /* Sharable Partitioning */
update the edges related to n_i, n_j, and n_{new}.
}

This is a greedy algorithm which works on the most profitable partitioning one at a time until no more sharable partitioning is possible. For H(15,11,3), if $e(1,2)$ is selected first, the new graph is shown in Figure 5b. Now, the next choice is $e(3,4)$. The final graph is in Figure 5c. From now on, further partitioning will yield no improvement because there is no sharable subset or no edge with weight greater than 2. The final logic implementation is shown in Figure 4.a.

Test Results

The test results on thirteen ECC circuits obtained from [2] are shown in Table 1. Many of them are practical ECC circuits implemented in commercial machines. The first three columns are the properties of the ECC circuits. Column four and after are the numbers of XOR operators required after literal minimization. The forth column is the original design without minimization. The fifth column is the results by [1]. The sixth column is the results of a random search algorithm with a CPU time limit of 100,000 seconds. The last column are the results by our set partitioning algorithm. The last row shows the total number of XOR operators used for all the cases. Overall, ECCGen achieve 44.7% reduction over original design, 12.4% over [1], and 4.7% over 100,000 seconds of random searches.

There is a very interesting observation here. Conventionally, we believe that, when selecting an error control code, we must select the code with the fewest 1's in order to minimize the size of the circuit. However, the results in Table 1 suggest that this is not always true. Comparing case 8 with 4, 5, 6, and 7, case 8 has the most 1's before optimization. After minimization, it requires only 44 XOR operators which is the smallest of all. The same observation can be seen in case 2 and 3 and in case 11 and 12. So, the key issue is the sharing capability of the code.

After literal minimization, there is gate/pin assignment to obtain the circuit. Figure 4 shows two designs for the sharable partitioning in Figure 4.c. By the use of 4-input and 5-input XOR gates, the circuit size is minimized. The differences between these two extreme implementations are area and timing. Moreover, there are more implementations lie between them. In the next section, we will present a gate/pin assignment method to obtain an optimal trade-off between area and timing.

4 Gate/Pin Assignment

Gate/pin assignment accepts the results of literal minimization, selects an optimal circuit structure for

Table 1: Test Results - Literal Minimization

Circuit Property			Number of XOR Operators			
Case	Size	Function	Original	ECCSyn[1]	Random	ECCGen
1	(15,11)	DEC-SED	28	22	22	22
2	(24,18)	SEC-DEC-S4ED	54	43	42	42
3	(24,18)	SEC-DED-S3ED	84	48	44	43
4	(32,25)	SEC-DED-S4ED	81	58	57	57
5	(40,32)	S2EC	97	73	76	73
6	(45,34)	SEC-DED-BED	108	90	83	84
7	(45,39)	SEC-S3ED	150	92	83	86
8	(48,40)	HD-3	184	47	44	44
9	(60,53)	SEC-S4ED	163	119	106	106
10	(72,64)	SEC-DED-S4ED	208	159	122	142
11	(72,64)	SEC-DED-S4ED	240	158	186	146
12	(72,64)	S2EC	272	177	152	146
13	(207,198)	SEC-DED-S3ED	819	484	424	385
		Total	2488	1570	1444	1376

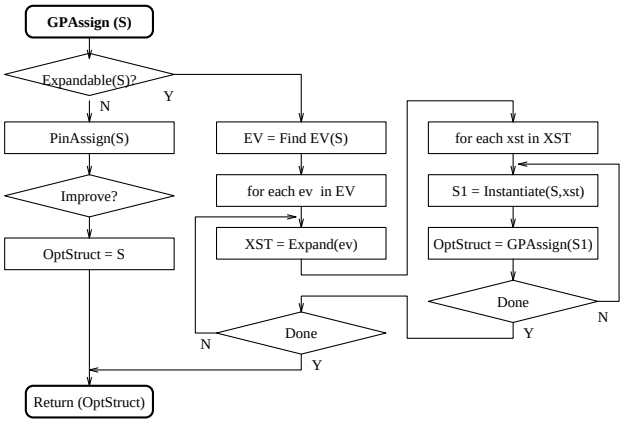

Figure 7: The Gate/Pin Assignment Algorithm

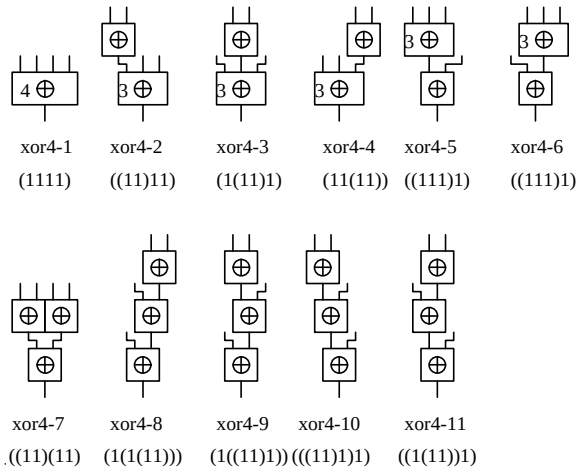

Figure 8: All Possible 4-input XOR Trees.

each subset, and produces the netlists in circuit level. The netlist can be fed to layout synthesizers to produce physical layouts. Here, we chose CPL XOR gates to demonstrate the feasibility. Gate/pin assignment is able to handle other multi-input XOR gates, such as static, dynamic, and cascode voltage switch logic (CVSL) [5,6] providing that the gates are characterized.

The gate/pin assignment (**GPAssign**) is a recursive depth-first search algorithm as shown in Figure 7. It searches for the optimal circuit structure for a given subset according to an user-define cost function. Upon receiving a tree structure (S), it determines if the tree is fully expanded. If it is, an optimal pin assignment ($PinAssign$) will proceed. If S has improvement over the previous optimal structure ($OptStruct$), $OptStruct$ is updated If S is not fully expanded, it finds the expandable variables (EV) and expands every expandable variables (ev) to a subtree $S1$. For each instantiation of $S1$, it calls PGAssign recursively for further expansion. In the rest of the section, we will discuss the tree expansion, pin assignment, and the search pruning that is not shown in the flowchart.

XOR Tree Expansion

In order to explain the tree expansion, let us look at all possible 4-input XOR tree structures shown in Figure 8 first. The XOR trees are represented in the syntax of LISP, as shown at the bottom of the figure. A pair of parentheses represents a gate. The ith element in the list are corresponding to the ith input. To obtain all possible implementation of a k-input tree, we start from (k), for example (4) for k of 4. Any element larger than 1 in the list is regarded expandable. We must decompose k into a nested list of 1's. Again, (4) can be expanded into {(1111), ((2)11), (1(2)1), (11(2)), ((3)1), (1(3)), ((2)(2))}. Let us take (1(3)) as an example. (1(3)) indicates that the first level is a 2-input XOR gate with the first input a primary input and the second input the fanout of a 3-input XOR

tree. We can further expand the subtree (3) into (111), (1(11)), and ((11)1). After substituting (3) by these subtree structures, the fully expanded tree structures are xor4-6, xor4-8, and xor4-9 as shown in Figure 8. The expansion algorithm is rather simple, we will not detail it here.

Pin Assignment

Once the tree structure is determined, the next step is to assign pins. Since the circuit structure is decided, transistor count and tree delay from each tree input to the tree output are also determined. The purpose of pin assignment is to produce an assignment with the shortest delay. This is a straightforward task. We simply order the signal inputs according to their path delays and order the tree input pins according to their tree delays. Here, the *path delay* of a signal is the maximal delay from primary inputs. The *tree delay* of a pin is the delay from the input pin to the tree output. The input with longest path delay is assigned to the pin with the smallest tree delay, and vise versa. Such a pin assignment is able to yield the minimal total path delay for that particular.

Cost Function

In ECCGen, the optimality is judged by a user-define criterion. Such a criterion is quantized by a mathematical equation called *CostFunction*.

$$CostFunction = W_t * TxCount + W_d * Delay$$

If the task is to minimize area, W_t can be set to 1 and W_d to 0, while to optimize timing, W_t to 0 and W_d to 1. With the cost function, we can determine the optimality of the structure.

Search Pruning

From the tree structure expansion mechanism, we know that the complexity is non polynomial. Therefore, we need to prune those futile searches. For this, an estimation function **EstimateCost** is outlined as follows. For an unexpanded element (n), the lower bound of the transistor count is $n \times 4$, the use of a n input XOR gate, as shown in Figure 1. However, the estimation of the lower bound of the delay is more difficult. Assume that there are m all possible structure for a n-input tree. Let the tree delays of jth structure are denoted from t_1^i to t_n^j in ascending order. The estimated delays are from t^1 to t^n also in ascending order also. Then, we use the following equation to calculate t^i.

$$t_i = \min_{j=1}^{m} t_i^j$$

In other words, t^i is the smallest among the ith shortest delay of all possible n-input tree structure.

It can be shown that the previous calculation lead to a theoretical lower bound on delays and transistor count. ECCGen estimates the lower bound of the cost by the lower bounds of the unexpanded elements and the actual data of the expanded elements, If the estimated cost **EstimateCost(Struct)** is greater than the current best cost **Cost**, **S** is pruned. For example, if only transistor count is considered, all the other unexpanded structures will be pruned once (1111) is explored, because it uses only 16 transistors. Note that, the lower bounds on trees of different number of inputs can be calculated beforehand. So, it is a simple table look up in ECCGen.

Table 3: Test Results - CPU Time

2-Xor	Min-A	Min-T	Min-T8
1-10 Sec.	1-10 Sec.	10-1000 Sec.	10-100 Sec.

5 Test Results

The test results on thirteen ECC circuits are listed in Table 5. Since there are many ECC circuits with sets of more than 16 inputs, in some case we have to set a limit on the search to control the execution time. So, users are able set a limit on the number of inputs to a gate. The first column of Table 5 is the result of the original design by the use of only 2-input XOR gates. The third column, 2-XOR, are the designs which use 2-input gates after literal minimization. The fourth columns, Min-A, are the designs for the criterion of minimal area. The fifth columns, Min-T, are the designs for the criterion of minimal delay without limitation. The final columns, Min-T8, are the designs for the criterion of minimal delay with an input limit of 8. Note that, Min-T could not finish on case 6, 7, 9, and 13 because there are subsets with too many inputs. The results are collected after 50,000 seconds of CPU time. ECCGen has been tested on TwinHead SS2 Workstation, a SUN Sparc 2 compatible machine. The CPU time for different cases are listed in Table 5.

As expected, the criterion for minimal area yields the smallest transistor count, a 65% reduction over original design. Even the criteria for minimal delay, Min-T and Min-T8, achieve 57% reduction in transistor count. As compare to the conventional 2-input gate design, 2-XOR, Min-T achieves not only 23% reduction in transistor count but also 15% improvement in delay time. This is in accord with our hypothesis that the use of balanced binary tree structure does not necessarily yield the shortest delay. So, we conclude that the use of multiple input CPL XOR gates reduce not only the area but also the delay.

6 Conclusions

In this paper, we have designed and implemented a dedicated logic synthesizer, ECCGen, for the synthesis of ECC circuits. ECCGen takes H matrices as inputs and produces physical layouts in two steps, literal minimization and gate/pin assignment. For literal minimization, we model H matrices as sets and use partitioning algorithm to achieve 44.7% reduction in literal count. For circuit implementation, we use CPL XOR gates as the basic circuit structure. Through the exploration of different XOR tree structures, gate/pin

Table 2: Test Results - Transistor Count

	Transistor Counts / Delay (ns)				
Case	Original	2-Xor	Min-A	Min-T	Min-T8
1	224/2.79	176/2.79	112/3.34	160/2.79	160/2.79
2	432/3.23	336/3.23	212/4.34	300/2.90	300/2.90
3	656/3.41	456/3.41	284/5.61	368/3.03	368/3.03
4	648/3.98	344/3.98	220/4.71	300/3.65	300/3.65
5	776/3.46	572/3.46	444/5.97	576/3.33	552/3.33
6	864/5.02	688/5.02	424/7.06	604/4.15	616/4.15
7	1200/4.80	848/4.80	508/6.99	748/4.00	772/4.00
8	1472/5.47	352/5.47	260/9.96	316/4.97	320/4.97
9	1304/4.69	1136/4.69	704/7.57	960/3.94	972/3.94
10	1664/5.57	1168/5.57	728/7.86	1040/4.28	1000/4.28
11	1920/5.60	1168/5.60	752/8.07	976/4.31	1008/4.31
12	2176/3.97	584/3.97	356/5.28	480/3.10	480/3.10
13	6552/7.04	3080/7.04	1920/9.83	2656/5.52	2656/5.52
Total	19904/59.0	11008/59.0	6924/86.6	8532/50.0	8544/50.0
Compare	1.00/1.00	0.55/1.00	0.35/1.47	0.43/0.85	0.43/0.85

assignment achieves not only 57% reduction in transistor counts but also 15% reduction in delay over the original designs. Moreover, gate/pin assignment is able to handle other multi-input XOR gates by a simple change of delays and areas in the library. ECCGen is not limited to ECC circuits only. It can be used as the technology mapping for Reed-Mullar circuit synthesis as well. After the logic synthesis in the Reed-Mullar form, the modulo-2 part can be minimized by ECCGen.

From literal minimization, we learned that to minimize the size of an ECC circuit, select the code with smallest number of 1's is not the only guideline. Select the codes with the most sharable rows can lead to better results after optimization. For circuit implementation, the use of balanced binary tree structure does not necessarily have the shortest delay. Through careful gate/pin assignment, the use of multi-input XOR gates does not only reduce the circuit size but also shorten the delay.

References

[1] C. Su, and J. Wang, "ECCSyn - A synthesis tool for ECC circuits," Proc. ISCAS'93, pp. 1706-1709, Chicago, U.S.A., 1993.

[2] T.R.N. Rao and E. Fujiwara, *Error-control coding for computer systems*, Prentice-Hall Int, Inc., Englewood Cliffs, New Jersey, U.S.A., 1989.

[3] R. Brayton, G. Hachtel, C. McMllen, and A. Sangiovanni-Vincentelli, *Logic minimization algorithms for vLSI synthesis*, Kluwer Academic Publishers, Hingham, Maryland, U.S.A., 1984.

[4] R. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang, "MIS: A multiple-level logic optimization system," *IEEE Trans. Computer Aided Design*, Nov. 1987, pp. 1062-1081.

[5] N. Weste, and K. Eshraghian, *Principles of CMOS VLSI design a systems perspective 2nd ed.*, Addison-Wesley, New York, U.S.A., 1993.

[6] K. Yano and et al., "A 3.8-ns CMOS 16X16 multiplier using complementary pass-transistor logic," IEEE J. of Solid-State Circuits, vol. 25, no. 2, April, 1990, pp. 388-395.

[7] D. Bostic and et al., "The boulder optimal logic design system," Proc. Int'l Conf. on Computer-Aided Design, Nov. 1987, pp.62-65.

[8] D. Gregory and et al., "Socrates: a system for automatically synthesizing and optimizing combinational logic," Prof. 23th Design Automation Conference, June 1985.

[9] S. Even, I. Kohavi, and A. Paz, "On minimal modulo 2 sums of products for switching functions," IEEE trans. on Electronic Computers, Oct. 1967, pp. 671-674.

[10] A. Mukhopadhyah and G. Schmitz, "Minimization of exclusive OR and logical equivalence switching circuits," IEEE Trans. on Computers, vol. C-19, No. 2, Feb. 1970, pp. 132-140.

[11] G. Papakonstantinou, "Minimization of modulo-2 sum of products," IEEE Trans. on Computers, Vol. C-28, No. 2, Feb. 1979, pp. 163-167.

[12] M. Helliwell and M. Perkowski, "A fast algorithm to minimize multi-output mixed-polarity generalized Reed-Muller forms," Proc. 25th ACM/IEEE Design Automation Conf., 1988, pp.427-432.

[13] C.C. Tsai and M. Marek-Sadowska, "Multilevel logic synthesis for arithmetic functions," Proc. 33th ACM/IEEE Design Automation Conf., 1996.