A Logic Simplification Approach for Very Large Scale Crosstalk Circuit Designs

Md Arif Iqbal[1], Naveen Kumar Macha, Bhavana Tejaswini Repalle, Mostafizur Rahman[2]
Computer Science Electrical Engineering, University of Missouri-Kansas City, MO, USA
mibn8@mail.umkc.edu[1], rahmanmo@umkc.edu[2]

Abstract—Crosstalk computing, involving engineered interference between nanoscale metal lines, offers a fresh perspective to scaling through co-existence with CMOS. By capacitive manipulations and innovative circuit style, not only primitive gates can be implemented, but custom logic cells such as an Adder, Subtractor can be implemented with huge gains. In this paper, we introduce the Crosstalk circuit style and a key method for large-scale circuit synthesis utilizing existing EDA tool flow. We propose to manipulate the CMOS synthesis flow by adding two extra steps: conversion of the gate-level netlist to Crosstalk implementation friendly netlist through logic simplification and Crosstalk gate mapping, and the inclusion of custom cell libraries for automated placement and layout. Our logic simplification approach first converts Cadence generated structured netlist to Boolean expressions and then uses the synthesis tool (SIS) to obtain majority functions, which is further used to simplify functions for Crosstalk friendly implementations. We compare our approach of logic simplification to that of CMOS and majority logic-based approaches. Crosstalk circuits share some similarities to majority synthesis that are typically applied to Quantum Cellular Automata technology. However, our investigation shows that by closely following Crosstalk's core circuit styles, most benefits can be achieved. In the best case, our approach shows 36% density improvements over majority synthesis for MCNC benchmark circuits.

Keywords—Crosstalk Computing, Capacitive Coupling, Crosstalk Logic, Majority Network, Logic Synthesis

I. INTRODUCTION

As the traditional way of CMOS scaling becomes difficult, Crosstalk computing provides an alternative solution while leveraging CMOS devices and interconnect technologies [1]-[5]. In Crosstalk circuits, computation is realized by embracing the increasing signal interference at advancing technology nodes and astutely engineering it to obtain logic function. For operation, the transition of signals on input metal lines called as aggressor nets, induce a resultant summation of charge on output metal line, called as victim net, through capacitive couplings. This induced signal serves as an intermediate signal to control thresholding devices like an inverter to get the desired logic output.

All the elementary gates, as well as many multi-level logic functions, can be implemented by a single Crosstalk gate [3]. To implement a multi-level logic function, two different circuit styles are followed which are homogeneous and heterogeneous. In homogenous circuits, the coupling capacitance between input and output nets are equal, whereas in heterogeneous, the capacitances are unequal. Crosstalk circuits use these homogeneous and heterogeneous Crosstalk circuits as primitive cells with other basic gates like AND/OR. Due to the innovation in circuit style and physical principle of Crosstalk computing, the traditional synthesis flow for large circuits is not directly applicable.

Majority logic, where the summation of signals determines logic output through thresholding function, can resemble some of the Crosstalk's logic principles. However, existing majority synthesis approaches in the literature mostly concentrate on Quantum Cellular Automata technology where primitive cells are only inverter and majority gates [6]-[8]. Though some benefits can be obtained by using majority synthesis methods, fundamentally, obtaining simplified expressions for Crosstalk circuits require a different approach that utilizes fabric's native functionalities.

We propose in this paper, Crosstalk implementation friendly logic simplification approach that takes advantage of both the CMOS and majority synthesis methods for simplified Boolean expressions. First, we take an arbitrary network in Verilog form and use Cadence RTL Compiler [9] to generate a netlist of the network with logic constraints (e.g., limit the tool to use NAND/NOR, AOI, OAI gates only) to benefit from Crosstalk implementations. Then the netlist is converted to Boolean expressions and fed to the SIS [10] tool to obtain 3-input Boolean expressions. Finally, these expressions are used in our logic simplifier tool iteratively for obtaining Crosstalk friendly expressions. Our results show that for three different circuits' cm85, mux and pcle from MCNC benchmark suits [11], there are 11%, 27%, and 32% transistor count reduction compared to majority synthesis approach and 58%, 62%, and 24% transistor count reduction compared to CMOS based approach, respectively.

The rest of the paper is organized as follows: Section II describes the fundamentals of Crosstalk computing (CT) and implementation of logic gates. Section III presents the overview of logic simplification methodology. Section IV compares and benchmarks proposed simplification methods with majority based synthesis and CMOS based synthesis methods and Finally, Section V presents the conclusion.

II. CROSSTALK CIRCUIT STYLE & POTENTIAL FOR DENSITY, POWER AND PERFORMANCE GAINS

We utilize deterministic interference between adjacent interconnects for logic computing [1]. In this approach, metallic nano-lines are organized in a compact manner such that whenever signal transitions take place in these lines, the sum of their Crosstalk interference gets induced through virtual coupling capacitance in another metal nano-line that was floating; the transitioning signals are inputs and the net induced charge is the output. The coupling strength between the input

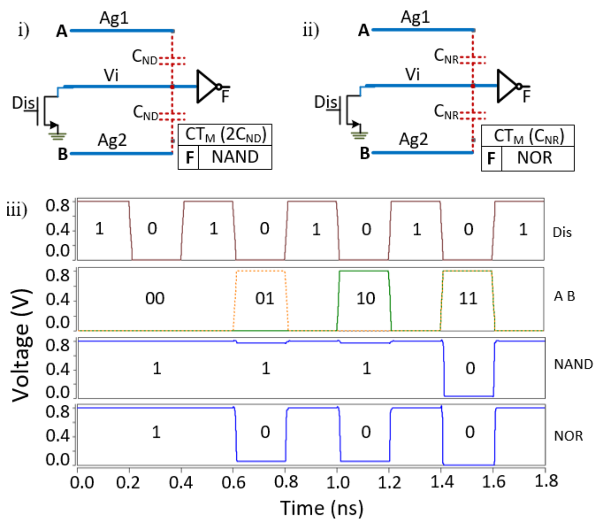


Fig.1. Fundamental Crosstalk Logic Gates: i) NAND ii) NOR

and output nano-lines and the net charge induced determines what logic is being computed. Fig.1 shows the implementation of primitive (NAND and NOR) gates in Crosstalk. During logic computation, the victim net (*Vi*) voltage is controlled electrostatically through coupling capacitances between two aggressors (*Ag1* and *Ag2*) and victim (*Vi*) net. To drive the *Vi* node for the next round of logic evaluation, its voltage is discharged to ground through a transistor controlled by dis signal after every round of logic evaluation. Thus, the *Vi* node is connected to an inverter acting as a threshold function on one end and connected to the drain side of the discharge transistor on the other end. This principle is used while implementing both NAND and NOR gates with the only difference of coupling strengths (C_{ND} & C_{NR}) between inputs and *Vi* net.

By assigning different coupling capacitances (equal or unequal) among the input aggressors and increasing input fan-in, many complex logic functions can be achieved. Based on this principle, two different circuit styles is applied; homogeneous Crosstalk logic gates where aggressors are equally coupled and heterogeneous Crosstalk logic gates where aggressors are unequally coupled. Fig.2 shows the implementation of Crosstalk homogeneous and heterogeneous logic gates with three input fan-ins. In Figs.2 (i&ii), show implementation of heterogeneous Crosstalk gate, AOI21 (AND-OR-Inverter), i.e., F= (AB+C)' where the functionality is achieved by varying coupling capacitances between the aggressors. Figs.2(iii&iv) shows the example of homogeneous Crosstalk logic where Carry function, F = MAJ_3 (A,B,C) = AB +BC+CA is achieved by keeping same coupling strength among the input aggressors. Noticeably, when we have homogeneous functionality with multiple inputs, this has some similarity to Majority logic where majority threshold functions are generally used to obtain max/min functions. However, the key difference is that we can achieve not only Boolean logic gates (NAND, NOR) and Majority logic but also heterogeneous logic. For us, the flexibility is much more than just majority gates or that of CMOS which provides more opportunities to compress logic using Crosstalk logic cells.

For large-scale circuits, logic cascading and maintaining signal integrity is a critical issue. In this regard, the crosstalk computing approach provides opportunities as well as challenges. Since by utilizing crosstalk, we can implement both fundamental logic gates and reduce complex combinational logic blocks, any function can be implemented. The logic

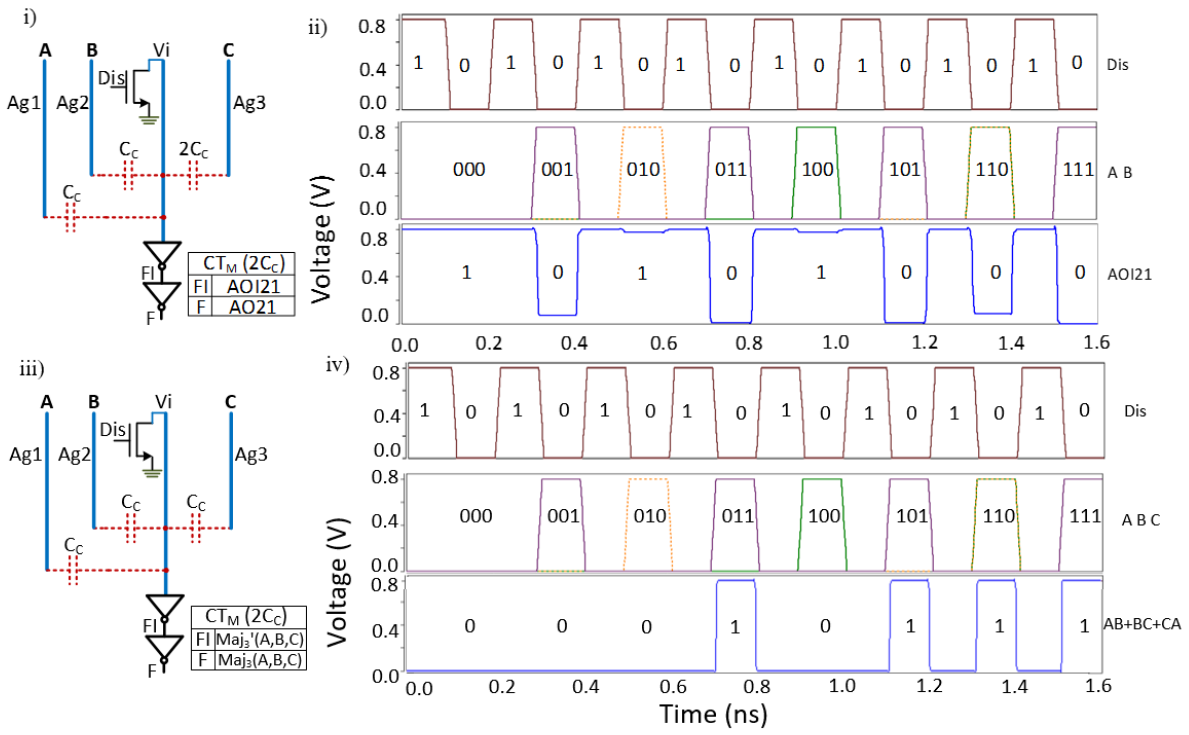


Fig.2. CT Heterogeneous and Homogenous gates, i) Heterogeneous gate: AOI21 (f_1= AB+C), ii) Simulation results of AOI21, iii) Homogenous gate, f_2=AB+BC+CA, iv) Simulation results of function f_2.

functions that require hierarchical implementation will be implemented by cascading outputs through the coupling. In this regard, we can use a constructive-destructive topology that is if a non-inverted gate is implemented first, we can cascade the output to an inverting gate or vice-versa. While cascading outputs at several levels, maintaining signal integrity becomes a challenge, since with each stage of coupling the induced voltage in the next level reduces compared to the previous stage. We resolved this issue in different ways by placing buffers or by using a Pass-Gate solution, where, the inverting and non-inverting gate interfaces are connected through a transmission gate which is controlled by clock cycles [2]. The other solution is by using a different set of Crosstalk logic gates which operate on falling edge transition also. Thus, a fully working large-scale compact circuits, with reduced size, improved performance and power can be achieved using Crosstalk logic style.

III. Overview of the Proposed Logic Simplification Methodology

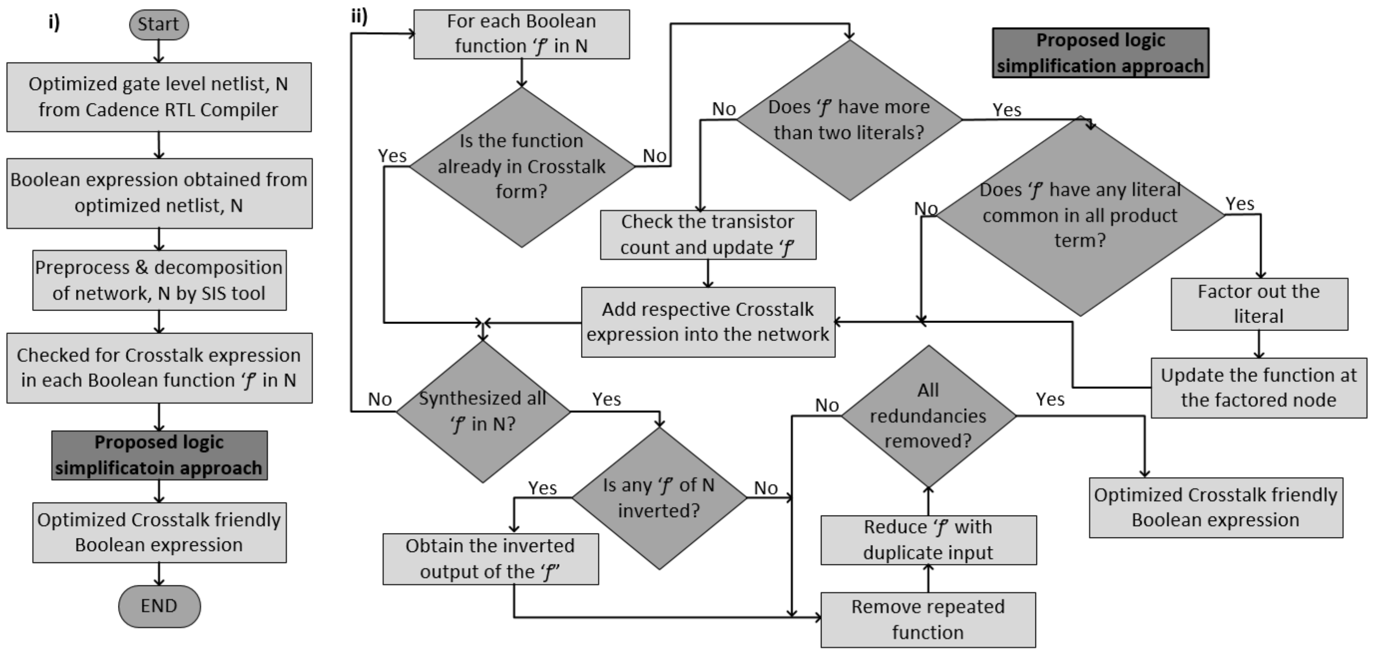
In this section, we introduce our simplification approach for Crosstalk circuit friendly expression and detail implementation steps. We take advantage of the compressibility feature that Crosstalk presents through custom logic, CMOS logic, Majority logic and explain in our approach that how we can combine all of them to obtain the best result.

Figs.3(i&ii) give a flow diagram of the Crosstalk logic synthesis methodology. Our process starts with having Cadence RTL Compiler that generates a netlist from Verilog code with constraints such that the netlist use gates like NAND, NOR, AOI, or OAI which are Crosstalk friendly. It is noticeable though we cannot constrain the tool to use majority gates (AB+BC+CA) or other heterogeneous logics that are especially suitable for Crosstalk computing. Because of this, after obtaining the netlist from Cadence tool then we convert it back to Boolean expressions and feed it again through SIS tool such that the SIS tool already works on an optimized Boolean expression and further tries to simplify it in terms of majority gates (Fig.3(i)). The new expression already has majority expressions and some custom expressions which can be implemented using universal gates like NAND/NOR gate, however, we look for further opportunities for simplification as given in Fig.3(ii) to get expressions for heterogeneous logic. If the heterogeneous logics cannot be found we use Crosstalk NAND/NOR gate and complete the Boolean expression. Finally, we obtain an expression that can be converted into structural netlist and that structural netlist can be used in conjunction with cell libraries to obtain full layout and parametric results like area, power, and performance.

Fig.4 represents the pseudo-algorithm of our simplification approach where we check for each function of the network to be simplified as Crosstalk friendly expression. Variables that are used in the algorithm are defined as follows:

f_1,f_2,f_3	A function in network N
S	Set of Crosstalk homogeneous & heterogeneous function
f_n	Fan-in to network N
S'	Inverted Crosstalk homogeneous & heterogeneous function
l_i	The ith literal in the expression for function f
p_j	The jth product term for function f
n_l	No. of literals in function f
T	No. of the transistor in the function
I	No. of the inverter in the function
f_{dm}	Function f after applying De Morgan's Law

The corresponding pseudo algorithm takes in preprocessed and decomposed network as input and returns a more simplified



i)
Start
Optimized gate level netlist, N from Cadence RTL Compiler
Boolean expression obtained from optimized netlist, N
Preprocess & decomposition of network, N by SIS tool
Checked for Crosstalk expression in each Boolean function 'f' in N
Proposed logic simplificatoin approach
Optimized Crosstalk friendly Boolean expression
END
ii)
For each Boolean function 'f' in N
Is the function already in Crosstalk form?
Yes
No
Does 'f' have more than two literals?
No
Yes
Proposed logic simplification approach
Check the transistor count and update 'f'
Does 'f' have any literal common in all product term?
No
Yes
Factor out the literal
Add respective Crosstalk expression into the network
Update the function at the factored node
Synthesized all 'f' in N?
No
Yes
All redundancies removed?
No
Yes
Is any 'f' of N inverted?
Yes
No
Optimized Crosstalk friendly Boolean expression
Obtain the inverted output of the 'f'
Reduce 'f' with duplicate input
Remove repeated function

Fig.3. Overview of proposed logic simplification methodology. i) Top-level simplification approach, ii) Detail steps of proposed logic simplification approach

```
Input: Optimized Network $N$
Output: Crosstalk expression corresponding to $N$
begin
1     Convert the netlist to Boolean expression
2     Preprocess and decompose network $N$ by SIS
3     for each $f$ in $N$ do
4         if $f \notin S$ then
5             if $n_l > 2$ then
6                 if $\exists l_i$ so that $\forall_j$, $l_i \in p_j$ then
                      $f_1 = f|_{li=1}$
                      $f = l_i f_1$
7                 else
                      Add Crosstalk expression to the $f$
8             else
                  //check the transistor count
                  Count $f_{old} = T+2*I$
                  Apply De Morgan's Law to function $f$
                  $f_x = f_{dm}'$
                  Count $f_{new} = T+2*I$
9                 If Count $f_{new}<$ Count $f_{old}$ then
                      $f = f_x$
10                else
                      Keep the original f and add
                      Crosstalk expression
11        else
              Add Crosstalk expression to the $f$
12    if $f_n$ in $N$ such that $f_n = f'$ do
          $f_n = S'$ where $S'$ is the inverted form of $S$
13    else
          break
14    Do redundancy check
15 End
```

Fig. 4. Pseudo algorithm for Crosstalk logic simplification approach

network that is Crosstalk friendly. After preprocessing and decomposing, each function f of the network N is checked to determine whether the function f is in homogeneous or heterogeneous Crosstalk form. If so, we proceed to simplify the next function. Otherwise, as shown in Fig.3(ii), we check to see if there exist more than two literals in the function. If there exist only two literals in the function, we check for transistor count. First, we calculate a number of transistors need to implement function f. Then, we take an inverted function (fx) of f and calculate the required transistor. If the transistor count for fx is lower than the original function f, we update the function with fx. For example, consider a function $f=a+b'$. Crosstalk mapping would require seven transistors including an inverter for literal *'b'* to map the function f. However, inverted function $f' = fx = (a'b)'$ would require only five transistors. If there are more than two literals present in the function, we look for any common literal that is present in all the product terms of the function. If a common literal exists, we factor out this literal and map with heterogeneous Crosstalk circuits. Consider, function $f = bc + ca$. If we are to map crosstalk gates directly, it would take three Crosstalk gates whereas if we factor out the common literal *'c'* from both product terms, function f, therefore, can be presented as $f = cf_1$, where $f_1 = (b+a)$, thus requiring only one Crosstalk gate. If there are no common literals, we check whether all the functions are synthesized or not. After simplifying all the functions in the network N, we further investigate if there exist any function that is in inverted form. If so, by using Crosstalk fabric inherent feature, we can save an additional inverter, required for making function f inverted. The final process is to remove all the redundancies, if exist, otherwise terminate. For redundancy removal, we follow the procedure explained in [8].

Next, we present Boolean expressions of the different network to explain the flowchart. First, the Boolean expression is obtained from a 4-bit ALU and later one is the network for the 2-bit multiplier. We represent the Crosstalk functions by denoting as function $X_{gate}(a,b,c)$ where a,b,c are the sub-functions and subscript *'gate'* defines what type of logic the function will be implemented.

A. First example

Step 1: Boolean expression obtained from 4-bit ALU:

$((((A_1A_2+A_1B_2')+A_2(B_2'+B_1'))+B_1'B_2')A_3+(((A_1A_2+A_1B_2')+A_2(B_2'+B_1')+B_1'B_2')B_3'+(A_3+(B_0'B_3'))')+A_3B_2B_3'$

Step 2: By using SIS [10] tool to preprocess and decompose, we obtain the following expression,

$$N = f_2 + f_3' + f_5$$
$$f_1 = A_1 + B_1' \quad (1)$$
$$f_2 = A_3B_0'B_3' \quad (2)$$
$$f_3 = A_3B_2B_3' \quad (3)$$
$$f_4 = f_1A_2 + f_1B_2' + A_2B_2' \quad (4)$$
$$f_5 = f_4A_3 + f_4B_3' \quad (5)$$

Step 3: For each function of network N, presented in equation (1)-(5), we check if the function is in Crosstalk homogeneous or heterogeneous form.

- The first function, f_1 is neither in homogeneous nor in heterogeneous form. Next, we find that it has only two literals. Then, we check for fewer transistor count which we get after applying De Morgan's law and then taking inverter of the function f. Therefore, the updated function is $f_1 = (A_1'B_1)'$. Since there are still three other functions to be simplified, we proceed to the next function, f_2.
- Function f_2 is directly in Crosstalk homogeneous form $X_{and}(A,B,C)$. We proceed to simplify the next function.
- Function f_2 is also in Crosstalk homogeneous form $X_{and}(A,B,C)$. Therefore, we update the function with Crosstalk homogeneous expression and check if there is any other function to be simplified.
- Function f_4 is in Crosstalk homogeneous form $X_{homo}(AB+BC+CA)$ too, so, we update the function with Crosstalk homogeneous gate.
- Function f_5 cannot be mapper with heterogeneous or homogeneous form. Next, we check to see if the function has any common literals. We find that f_4 is the common literal in both of the product terms of function f_5. Therefore, we factor out the common term and update the function as $f_5 = (A_3+B_3')\, f_4$ which is in Crosstalk heterogeneous form $X_{hetero}((A+B)C)$.

TABLE I. COMPARISION OF DIFFERENT BOOLEAN NETWORKS

Standard Function	**I/O**	**CMOS**		**Synthesis using existing method**		**Synthesis using proposed method**		**R% w.r.t CMOS**		**R% w.r.t existing method**	
		Transistor Count	***Gate Count***	***Transistor Count***	***Gate Count***	***Transistor Count***	***Gate Count***	***Transistor Count***	***Gate Count***	***Transistor Count***	***Gate Count***
F=ab+bc+a'b'c'	3/1	30	7	20	6 [7]	13	3	56%	57%	35%	50%
F=d(c+(b'+a)')	4/1	18	4	25	6 [8]	12	3	33%	25%	52%	50%
Example1	7/1	94	21	62	16 [8]	44	11	53%	48%	29%	31%
Arithmetic Block											
Full Adder	3/2	18	9	17	4 [8]	10	2	44%	77%	41%	50%
2-bit Multiplier	4/4	56	15	67	17 [8]	43	13	23%	13%	36%	23%
MCNC Benchmark											
cm85a	11/3	264	64	125	31 [8]	111	27	58%	58%	11%	13%
mux	21/1	404	72	209	49 [8]	152	37	62%	49%	27%	12%
pcle	19/9	246	56	276	66 [8]	186	42	24%	25%	32%	36%

- Next, we proceed to simplify other functions.
- From equation (1), we can see that both function f_2 and function f_3 have common literals A_3B_3' between them which we can factor out and get the expression as $A_3B_3'(B_0'+ B_2)+f_5$. A_3B_3' term can be obtained by Crosstalk AND gate which we can map with $(B_0'+ B_2)$ to get Crosstalk heterogeneous form.

Step 4: Update the node function for inverted output. We check if there is any function in an inverted form. If so, we can avoid additional inverter by using Crosstalk fabric feature, which can apply for the literal f_3'.

Step 5: Check for redundant functions and also redundant input to any single function. We have checked and found no redundancy for the first example.

Step 6: Complete the process. Finally, we update the network N with simplified Crosstalk friendly Boolean expression which is,

$$N=X_{or}(X_{hetero}(X_{and}(A_3,B_3'),B_0',B_2),X_{hetero}(X_{homo}(X_{nand}(A_1',B_1), A_2, B_2'),A_3,B_3'))$$

B. Second example:

Step 1: Input an arbitrary network: In 2-bit multiplier, there are four outputs and four inputs to the network.

$$Y_0 = A_0B_0$$
$$Y_1 = A_1A_0'B_0 + A_1B_1'B_0 + A_1'A_0B_1 + A_0B_1B_0'$$
$$Y_2 = A_1A_0'B_1 + A_1B_1B_0'$$
$$Y_3 = A_1A_0B_1B_0$$

Step 2: By using the SIS [10] tool to preprocess and decompose, we obtain the following expression:

$$Y_3 = Y_0A_1B_1 \quad (1)$$
$$Y_2 = f_1B_1 \quad (2)$$
$$Y_1 = f_2\ A_1 + f_3 \quad (3)$$
$$Y_0 = A_0B_0 \quad (4)$$
$$f_1 = A_0'A_1 + A_1B_0' \quad (5)$$
$$f_2 = A_0'B_0 + B_0B_1' + A_0'B_1 \quad (6)$$
$$f_3 = A_0\ A_1'B_1 \quad (7)$$

Step 3: For each function of network N, presented in equation (1)-(7), we have to check if the function is already in Crosstalk homogeneous or heterogeneous form.

- First function f_1 is neither in homogeneous nor in heterogeneous form. Next, we check to see if the function has any common literals. We find that A_1 is the common literal in both of the product terms in function f_1. Therefore, we factor out the common term and update the function as $f_1 = (A_0'+B_0')\ A_1$ which is in Crosstalk heterogeneous form $X_{hetero}((A+B)C)$. Next, we proceed to simplify other functions.
- Function f_2 is directly in crosstalk homogeneous form $X_{homo}(ab+bc+ca)$, therefore we update the function with Crosstalk homogeneous gate and check whether all the functions are simplified or not.
- As function f_3 is also directly in Crosstalk homogeneous form $X_{and}(ABC)$ and there is no other function left to be simplified, we move on to step 4.

Step 4: Update the node function for inverted output. We check if there is any function in an inverted form. If so, we can avoid additional inverter by using Crosstalk fabric feature. We have found no function to be in inverted form.

Step 5: Check for redundant functions and also redundant input to any single function. We have checked and found no redundancy in the simplified network.

Step 6: Complete the process. Finally, we update the network N with simplified Crosstalk friendly Boolean expression.

$$Y_3 = X_{and}(Y_0,A_1,B_1) \quad (1)$$
$$Y_2= X_{and}(X_{hetero}(A_1,A_0',B_0'),B_1) \quad (2)$$
$$Y_1= X_{hetero}(X_{and}(A_0,A_1',B_1),X_{homo}(A_0',B_0,B_1'),A_1) \quad (3)$$
$$Y_0 = X_{and}(A_0,B_0) \quad (4)$$
$$f_1 = A_0'A_1 + A_1B_0' \quad (5)$$
$$f_2 = A_0'B0 + B_0B_1' + A_0'B_1' \quad (6)$$
$$f_3 = A_0\ A_1'B_1 \quad (7)$$

IV. Comparison results

Comparison between the proposed approach and majority based synthesis approaches [6]-[8] is presented in this section. We have simplified different functions, arithmetic blocks and also three MCNC benchmark circuits [11]. Table I lists all the results for benchmarks. For CMOS, all the primitive cells are considered and for majority based approach, primitive cells are replaced with equivalent Crosstalk gates. For gate count comparison, the inverter is accounted for wherever needed for all three different approaches. Our results show significant improvement in a density benefit with respect to CMOS. The average reduction (R%) in gate count with respect to CMOS approach is 44%, with the maximum reduction being 77%. For MCNC benchmarks, the average gate reduction is 44%, with the maximum reduction being 58%. This is mostly due to traditional logic reduction approaches for CMOS are constrained to use a limited set of standard cell functions, where, more complex logic functions are not implemented because of the performance concerns that arise in CMOS logic circuits as they would require long pull-up and pull-down branches of switch (transistor) patterns. We also compared our results with majority based simplification approaches due to the similarity between logic reduction approaches. The average reduction (R%) in the gate with respect to other majority synthesis approach is 33%, with the maximum reduction being 50%. For MCNC benchmarks, the average gate reduction is 20%, with the maximum reduction being 36%. This is mostly due to majority logic approaches are inefficient in logic reduction as they provide a very limited number of primitive gates (majority-three, majority-five, and inverter) and any logic function needs to be transformed to these gates. However, for all the cases, the Crosstalk computing provides holistic logic-reduction opportunities owing to its ability to effectively implement all three, traditional standard cell functions, majority-logic gates, and additional complex functions.

V. Conclusion

We have presented a logic simplification approach for large scale Crosstalk circuit integration. We have simplified different Boolean networks like complex logic networks obtained from 4-bit ALU, Multiplier, Adder and also three MCNC benchmark circuits. Our results show significant density benefits over CMOS and majority based approach; for the best case, there is 58% and 36% reduction in density over CMOS based and Majority based logic reduction approach, respectively. The logic simplification approach presented in this work is a vital step towards the full-scale synthesis of Crosstalk circuits leveraging existing EDA tools.

References

[1] N. K. Macha, V. Chitturi, R. Vijjapuram, M. A. Iqbal, S. Hussain and M. Rahman, "A New Concept for Computing Using Interconnect Crosstalks," 2017 IEEE International Conference on Rebooting Computing (ICRC), Washington, DC, 2017, pp. 1-2.

[2] N. K. Macha, S. Geedipally, B. T. Repalle, M. A. Iqbal, W. Danesh and M. Rahman, "Crosstalk based Fine-Grained Reconfiguration Techniques for Polymorphic Circuits," 2018 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH), Athens, 2018, pp. 1-7.

[3] N. K. Macha, B. T. Repalle, S. Geedipally, R. Rios and M. Rahman, "A New Paradigm for Fault-Tolerant Computing with Interconnect Crosstalks," 2018 IEEE International Conference on Rebooting Computing (ICRC), McLean, VA, USA, 2018, pp. 1-6.

[4] N. K. Macha, S. Geedipally, B. T. Repalle, M. A. Iqbal, W. Danesh and M. Rahman, "A New Paradigm for Computing for Digital Electronics under Extreme Environments," 2019 IEEE Aerospace and Electronic Systems Society, Big Sky, Montana, USA, 2019, in press.

[5] R. Desh, N. K. Macha, S. Hossain, R. B. Tejaswini and M. Rahman, "A Novel Analog to Digital Conversion Concept with Crosstalk Computing," 2018 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH), Athens, 2018, pp. 1-3.

[6] Rui Zhang, P. Gupta and N. K. Jha, "Synthesis of majority and minority networks and its applications to QCA, TPL and SET based nanotechnologies," 18th International Conference on VLSI Design held jointly with 4th International Conference on Embedded Systems Design, Kolkata, India, 2005, pp. 229-234.

[7] K. Kong, Y. Shang and R. Lu, "An Optimized Majority Logic Synthesis Methodology for Quantum-Dot Cellular Automata," in IEEE Transactions on Nanotechnology, vol. 9, no. 2, pp. 170-183, March 2010.

[8] P. Wang, M. Niamat, S. Vemuru, M. Alam and T. Killian, "Comprehensive majority/minority logic synthesis method," 2013 13th IEEE International Conference on Nanotechnology (IEEE-NANO 2013), Beijing, 2013, pp. 694-697.

[9] Cadence Conformal LEC, http://www.cadence.com/products/

[10] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and Al. L. Vincentelli, “SIS: A System for Sequential Circuit Synthesis,” EECS Department, University of California, Berkeley, 1992.

[11] S. Yang, Logic synthesis and optimization benchmark user guide: version 3.0. Microelectronics Center of North Carolina (MCNC), 1991, pp.502-508.