IEEE TRANSACTIONS ON NANOTECHNOLOGY

Fine-Grained Polymorphic Circuit Framework in Crosstalk Computing

Journal:	*Transactions on Nanotechnology*
Manuscript ID	TNANO-00028-2019
Manuscript Type:	NANOARCH2018ATHENS
Date Submitted by the Author:	08-Jan-2019
Complete List of Authors:	Kumar, Naveen; University of Missouri Kansas City, Computer Science and Electrical Engineering Repalle, Bhavana Tejaswini; UMKC Rahman, Mostafizur; University of Missouri, CSEE; University of MIssouri Kansas City/ EECS
Keywords:	Crosstalk computing, reconfigurable crosstalk logic

Fine-Grained Polymorphic Circuit Framework in Crosstalk Computing

Naveen Kumar Macha, Bhavana Tejaswini Repalle, Mostafizur Rahman

***Abstract*— The functionality of Polymorphic circuits can be altered using a control variable. Owing to their multi-functional embodiment in a single circuit, polymorphic circuits find a myriad of useful applications such as reconfigurable system design, resource sharing, hardware security, and fault-tolerant circuit design etc. The polymorphic circuit approaches available in literature so far are either based on custom nonlinear circuit designs or based on special emerging devices such as ambipolar FET, configurable magnetic devices, etc. These approaches often result in inefficiencies in performance and/or realization. We have proposed a novel polymorphic circuit design approach based on Crosstalk Computing, where deterministic signal interference between nano-metal lines is leveraged for logic computation and reconfiguration purposes. In this paper, we elaborate upon Crosstalk Computing's polymorphic logic design, present a comprehensive list of polymorphic logic gates designed, and characterize and benchmark our circuits with respect to CMOS circuit implementations. A wide range of polymorphic logic circuits (basic and complex) that are compact in design and minimal in transistor count are also unique to Crosstalk Computing. This leads to benefits in circuit density, power consumption, and speed. Our circuit designs, simulation, and characterization results show that the crosstalk polymorphic circuits provide 6x improvement in density/transistor count, 2x improvement in switching energy and 1.5x improvement in speed for polymorphic logic circuits. Furthermore, a Cascaded circuit example of polymorphic Multiplier-Sorter-Adder circuit is presented and benchmarked with respect CMOS.**

***Index Terms*— Crosstalk computing, reconfigurable crosstalk logic, polymorphic logic circuits, crosstalk polymorphic logic.**

I. Introduction

POLYMORPHIC logic circuits are rich in their functional behavior, where a control variable can deterministically morph the circuit's behavior between multiple functions. For an example, an AND gate can change as OR gate and vice-versa. Thanks to their ability to transform intrinsically, polymorphic circuits find their use in a myriad of applications such as reconfigurable circuits/systems, FPGAs, resource sharing, hardware security, and fault tolerance. In addition, as scaling down of feature size in Integrated Circuits (ICs) is approaching the physical limits, the miniaturization trend of ICs (Moore's Law) is relaxing. Therefore, developing alternate techniques that try to push the horizons of Moore's Law can be of tremendous potentials. Polymorphic circuits can be one such technique that tries to sustain the Moore's Law because they increase the circuit functionality in a given footprint by reusing the circuits to execute different functions. However, the tradeoffs in achieving such polymorphic circuits make them go or no-go for applications. At the circuit level, the conventional CMOS approach to design such polymorphic circuits is to have multiple functional blocks/circuits which are then selected using multiplexers. The drawback of this approach is that it is resource intensive. Many other attempts for circuit-level polymorphism are also available in the literature [1-7]. One such straightforward approach involves the design of functionally superimposed circuits [1][2]. These design approaches face key limitations such as design complexity and circuit overhead. In another category, polymorphic circuits are designed/evolved using genetic algorithms [3][4]. In this approach, the circuit behavior is morphed using control variables such as temperature, power supply voltage, light, and control signal etc. The major downside of these circuits is that they are strongly dependent on conditions and the technology under which they are evolved. They also suffer from inefficient circuit design problems such as a lack of general circuit topologies, slow and unreliable output responses, higher power consumption etc. Another approach actively pursued is chaos computing [5], in which non-linear dynamics in transistors and circuits are captured to implement multifunctional circuits. But these circuits are custom nonlinear/mixed-signal circuit designs for digital circuits. More recently, polymorphic circuits are also designed using emerging tunable polarity transistors [6][7] which can be configured either as p-type or n-type based on a control signal. Although it is a fine-grained device and circuit level approach, it suffers from complex device engineering requirement, inefficient circuit designs, and necessity of additional circuitry to switch the power rails when the FETs morph as p-type/n-type [6][7]. The other alternate approaches using emerging spintronic devices were also proposed [8], but they rely on complex information encoding schemes through spin-polarized currents and bipolar voltages etc.

We have proposed a novel computation concept called Crosstalk Computing [9] and implemented compact and efficient polymorphic circuits in this approach [10]. In the Crosstalk Computing technique, the signal interference between adjacent metal lines is astutely engineered to a logic principle. In contrast to the traditional switch-based logic computations, a deterministic superposition of crosstalk coupled input signals produces the logic operation in Crosstalk Computing. The counterpart of connecting switches/transistors in different patterns to achieve different logic is tuning the crosstalk coupling capacitances in Crosstalk Computing. For polymorphism, an additional control signal is used which biases the circuit to alter its functional behavior [9]. In this paper, we discuss our crosstalk computing technique in detail and present a comprehensive list of crosstalk polymorphic circuit designs and their simulation responses. The polymorphic gates shown

are AND-OR, AND-AO21, AND-OA21, AND-CARRY, OR-AO21, OR-OA21, OR-CARRY, AO21-OA21, CARRY-AO21, CARRY-OA21. A large-scale polymorphic circuit example of 2-bit Multiplier-Sorter-Adder designed by cascading the above polymorphic logic gates is also presented. Finally, we characterized the performance metrics of our circuits and benchmarked them with respect to CMOS implementations. Averaging the benefits of all the gates, our circuits show 4.5x reduction in transistor count, 50% reduction in switching energy, and 24% reduction in gate delay.

The rest of the paper is organized as follows. Section II discusses the Crosstalk Computing Concept, provides intuition for logic implementation and presents practical implementations of basic and complex logic circuits. Section III discusses polymorphism in Crosstalk Computing and shows a wide range of configurable gates that can morph between different operations. A polymorphic cascaded circuit example is also presented in this section. Section IV presents the benchmarking results. Finally, Section V concludes the paper.

II. Crosstalk Computing

The Fig.1(i) shows an overview of Crosstalk Computing Fabric, which majorly comprises of four components, Crosstalk Layer, Active Devices, Interconnects and Vias. The Crosstalk layer which computes the logic is a metal layer/layers comprised of capacitively coupled metal lines called as Aggressors (inputs) and Victim (output). Interconnects and Vias serve their regular purpose, along with their contribution to coupling capacitance in Crosstalk Layer. The active devices depicted are FinFETs on SOI substrate. The purpose of transistors is for accurate control and reconstruction of signals which would be discussed in following sections. The Fig.1(ii) illustrates the aggressor-victim scenario of crosstalk-logic. It shows the capacitive interference of the signals for logic computation— the transition of the signals on two rare end aggressor metal lines (Ag1 and Ag2) induce a resultant summation charge/voltage on victim metal line (*Vi*) through capacitive coupling C_C. Since this phenomenon follows the charge conservation principle, the victim net voltage is deterministic in nature and possesses the information about signals on two aggressor nets; its magnitude depends upon the coupling strength between the aggressors and victim net. The coupling capacitance is directly proportional to the relative permittivity of the dielectric and lateral area of metal lines (which is length times the vertical thickness of metal lines) and inversely proportional to the distance of separation of metal lines. Tuning the coupling capacitance values using the variables mentioned above provides the engineering freedom to tailor the induced summation signal to the specific logic implementation.

The notion of implementing logic gates using crosstalk signal interference is depicted in Fig.1(iii) with AND and OR gate examples. Input signal transitions induce a voltage proportional to coupling capacitances. For AND gate, C_A (Fig.1(iii)) can be chosen such that the magnitude of the voltage induced is greater than a selected threshold voltage V_T (which differentiates logic levels 0 and 1) only when both inputs transition 0→1 (i.e., input combination 11). For single input transitions (input combinations 01 and 10), the voltage induced on victim net is below the V_T, hence, the output can be considered as logic 0. Thus, as shown in Fig.1(iii), AND gate functionality can be realized using the crosstalk signal interference mechanism. Similarly, OR gate functionality can be realized just by increasing the coupling capacitance, which can be easily done by appropriately tuning the physical dimensions and/or choice of high-k dielectric material. The intuition for OR gate implementation is also shown in Fig.1(iii). Compared to AND gate, for OR gate, the coupling capacitance C_O is increased ($C_O > C_A$) such that the transition of either of the input signal from 0 to 1 is now sufficient to induce a voltage above the logic threshold (V_T). Therefore, input combinations, 01, 10, and 11 computes to logic output 1, as an OR gate. Practical realization of Crosstalk logic circuits and their reliable and robust operation in cascaded topology requires additional circuit techniques to be augmented to the intuitive idea described above, which is presented next.

A. Basic Logic Gates

Although the logic behavior in crosstalk computing can be achieved directly through coupled nano-metal lines [8], the output net (*Vi*) which collects the crosstalk charge needs to satisfy few conditions to achieve deterministic functionality in all sorts of real circuit environments (input and output). First, the *Vi* net needs to start from a known initial state and remain floating during logic evaluation to collect the crosstalk charge.

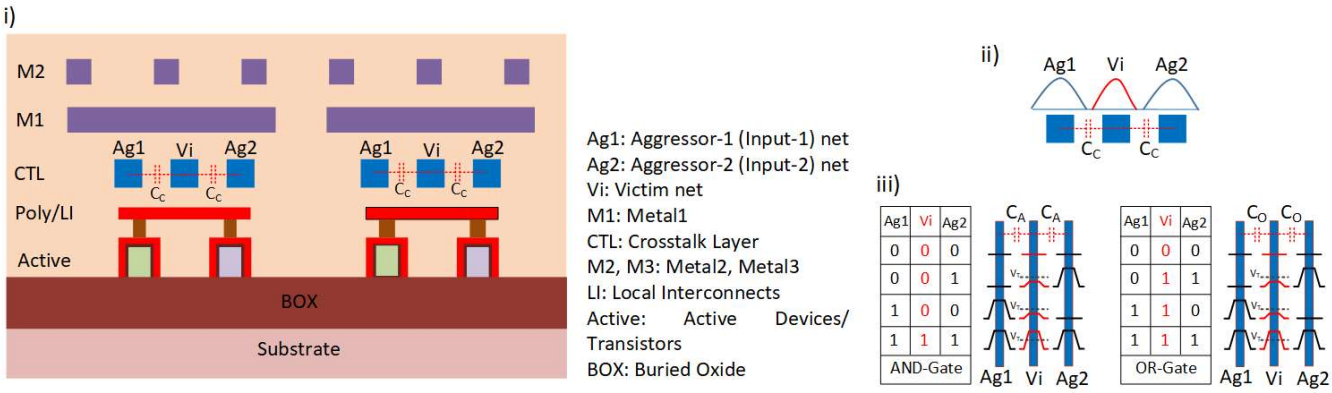

Fig.1: (i) Abstract view of Crosstalk computing fabric, (ii) Crosstalk Computing Mechanism, (iii) Implementing Logic Gates through crosstalk Computing

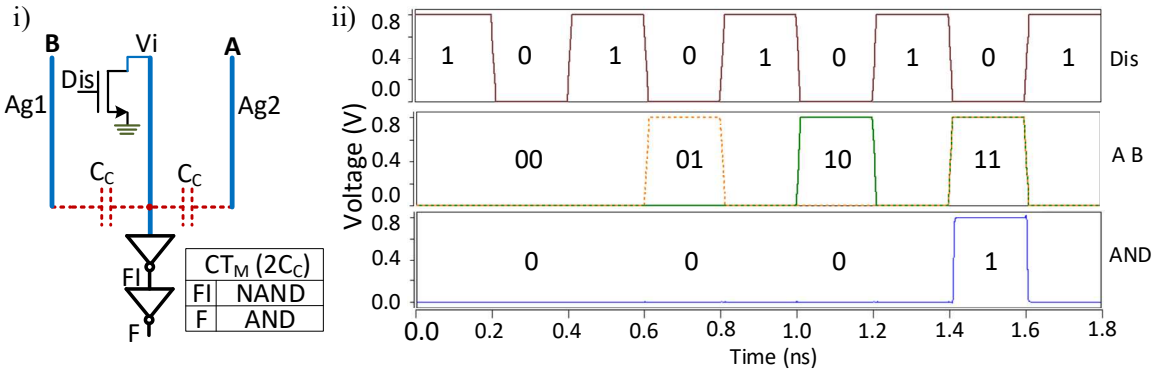

Fig.2 Crosstalk AND Gate: (i) AND Gate Circuit Schematic, (ii) AND Gate Simulation response

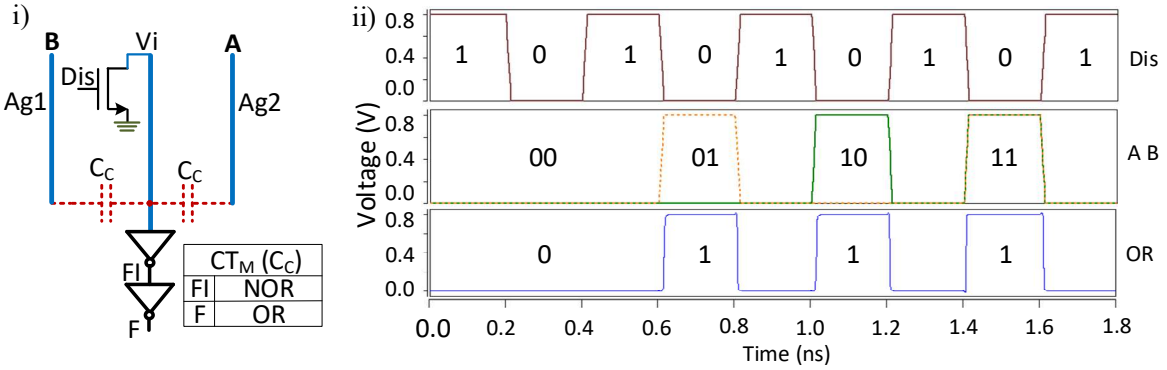

Fig.3 Crosstalk OR Gate: (i) OR Gate Circuit Schematic, (ii) OR Gate Simulation response

Second, the output node needs be able to drive the fanout gates in real circuits, and third, maintaining the signal integrity of binary voltage levels is crucial. To meet the above requirements, Crosstalk Logic circuits (Fig.2-19) are constructed by adding a discharge transistor and an inverter to the *Vi* net. Fig.2 shows the 2-input AND gate in which input aggressor nets (*A* and *B* acting as *Ag1* and *Ag2*) are coupled to victim net (*Vi*) through coupling capacitances, C_C, specific to each gate (given in Table.1). The discharge transistor is driven by the *Dis* signal. The CT-logic gates operate in two alternate states, Discharge State (DS) and Logic Evaluation (LE) state. During DS (enabled by *Dis* signal), the floating victim node is shorted to ground through the discharge transistor and thus starts with a known initial condition. The alternate DS ensures the correct logic operation during the next logic evaluation state by clearing off the charge from the previous logic operation. During LE state, the rise transitions on aggressor nets induce a proportionate linear summation voltage on *Vi* net which is connected here to a CMOS inverter. The inverter acts as a regenerative threshold function. That is, if the voltage computed on *Vi* net is above the inverter's threshold voltage (trip point), it outputs the logic level 0, and vice-versa; It actually regenerates the signals and restores them to full swing. The simulation response of the designed AND gate is shown in Fig.2(ii), where the first panel shows the discharge pulse (*Dis*), the second panel shows two input signals (*A* and *B*) with 00 to 11 combinations given through successive LE stages (when *Dis=0*), and the third panel shows the output response of AND. For all gates, FI stage gives inverting logic output (NAND etc.), and F stage gives noninverting logic output (AND etc.). Similarly, OR gate implementation and simulation response are shown in Fig.3. The difference between AND and OR gates is that the coupling strength (C_C), as given in Table.1, is greater for OR gate than AND. C_C is the quantized coupling strength/capacitance specific to each gate. The input aggressors would receive the coupling strengths in integer multiples of C_C.

The operation of CT logic gates would be represented functionally using a crosstalk-margin function, $CT_M(C_C)$, which specifies that the inverter of the CT-logic gate flips its state only when victim node sees the input transitions through the total coupling greater than or equal to C_C. For example, as shown in the table adjacent to the schematic (Fig.2(i)), AND gate CT-margin function is $CT_M(2_C)$. It states that the inverter flips its state only when the victim node sees the input transitions through total coupling greater than or equal to $2C_C$, i.e. which happens only when both inputs are high. Similarly, for OR gate (Fig.3(i)), the CT-margin function is $CT_M(C_C)$; which means the transition of any one of the aggressors is sufficient to flip the inverter, thus evaluates to OR behavior.

To further elucidate the relationship between crosstalk margin function and working mechanism of CT logic gates, considering a generic crosstalk capacitive network with 'n' number of input aggressors as shown in the Fig.4. The voltage induced on victim net can be calculated, by applying KVL, as follows

$$V_{Vi} = \left(\frac{c_1}{c_T}V_1 + \frac{c_2}{c_T}V_2 \ldots + \frac{c_n}{c_T}V_n\right) \ldots \text{(I)}$$

Where, $C_1, C_2 \ldots C_n$, are capacitance from respective aggressors to *Vi* net. C_T is the total capacitance on *Vi* net, which is,

$$C_T = C_1 + C_2 \ldots + C_{INV} + C_{ds};$$

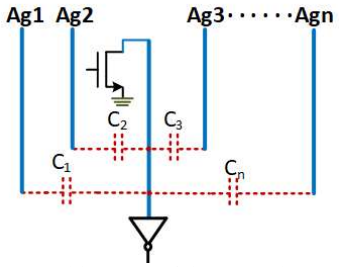

Fig.4 Capacitive Network in a Generic Crosstalk Gate

C_{INV} = Inverter Gate Capacitance,
C_{ds} = Discharge transistor drain-source capacitance

The logic levels on input aggressors, which are given by $V_1, V_2 \ldots V_n$ in equation (I), can be fomulated as voltage sources, given by

$$V_i = L_i V_{DD};$$
$$L_i = \begin{cases} 0 & if\ logic\ 0 \\ 1 & if\ logic\ 1 \end{cases}$$

The capacitance given to aggressors are in integer multiples of a constant C_C specific to each gate. $C_i = w_i * C_C$; where, w_i is the integer multiplying factor representing weighted strength of the each aggressor. The equation (I) now modifies to

$$V_{Vi} = \frac{C_C}{C_T}.V_{DD}.m$$

Where, $m = w_1L_1 + w_2L_2 \ldots + w_nL_n$.

The CT-margin function of each gate can be related to Vi net voltage equation as follows. If a given logic gate is characterized by a margin-function, $CT_M(\mathrm{k}.C_C)$, for all input combinations producing logic output 0, m ≥ k and $V_{Vi} > V_{INV}$. Similarly, for all input combinations producing logic output 1, m < k and $V_{Vi} < V_{INV}$. Table I gives the logic design table for AND2 and OR2 gates, which lists the CC values, aggressor weights, and margin function. This logic design table summarizes the mechanism and circuit aspects of crosstalk logic gates.

B. Complex Logic Gates

By increasing the fan-in (the number of input aggressors), more interesting complex logic functions can be realized because of the increased coupling capacitance and CT margin-function choices. The input aggressors can be assigned equal or unequal coupling capacitances. Gates with equally coupled aggressors are called homogeneous CT-Logic gates and unequally coupled

TABLE I
CROSSTALK LOGIC DESIGN TABLE FOR BASIC GATES

Gate	C_C (fF)	w1	w2	Margin Fuction
AND2	0.8	1	1	$CT_M(2C_C)$
OR2	4	1	1	$CT_M(C_C)$

aggressors are called heterogeneous CT-Logic gates. These homogenous and heterogeneous coupling choices further enhance the scope of complex logic functions that can be implemented efficiently through CT-Computing mechanism. Starting with three homogeneous input aggressors, a margin function of $CT_M(3C_C)$ implements 3 input AND gate, a margin-function of $CT_M(2C_C)$ implements CARRY (AB+BC+CA) logic, and margin-function of $CT_M(C_C)$ implements 3 input OR gate. The circuit schematic and simulation response of AND3, CARRY and OR3 logic are shown in Fig.5, Fig.6, and Fig.7, respectively. The first panel shows *Dis* pulse, the second panel shows the three input signals (*A*, *B* and *C*) feeding all combinations from 000 to 111 in successive LE states. The third panel shows the simulation response. Next, by giving weighted/heterogeneous couplings to the input aggressors such that one input has stronger capacitance than the other two, the functions like AO21 and OA21 can be realized as shown in Fig.8&9. Logic expression of AO21, i.e., (AB+C), evaluates to 1 when either AB or C, or both are 1. That means the output is biased towards the input C, i.e., irrespective of A and B values, the output is 1 when C is 1. Therefore, as in Fig.8(i), input C has twice coupling capacitance than A and B, i.e., $2C_C$. Similarly, for OA21 function, (A+B)C, the output is biased towards input C i.e., for output to be 1, C should be 1 along with A+B. Therefore, as in the previous case, C receives twice the coupling than A and B. To satisfy the above relations, the margin functions for AO21 and OA21 gates are $CT_M(2C_C)$ and $CT_M(3C_C)$, respectively. The simulation responses of the AO21 and OA21 gates satisfying the logic for all input combinations (000 to 111) are shown in Fig.8(ii)&9(ii). Table.2 gives the crosstalk logic design table for above complex logic gates. It can be observed from the table that a verity of complex logic functions can be generated by engineering with aggressor weights and margin functions in crosstalk logic.

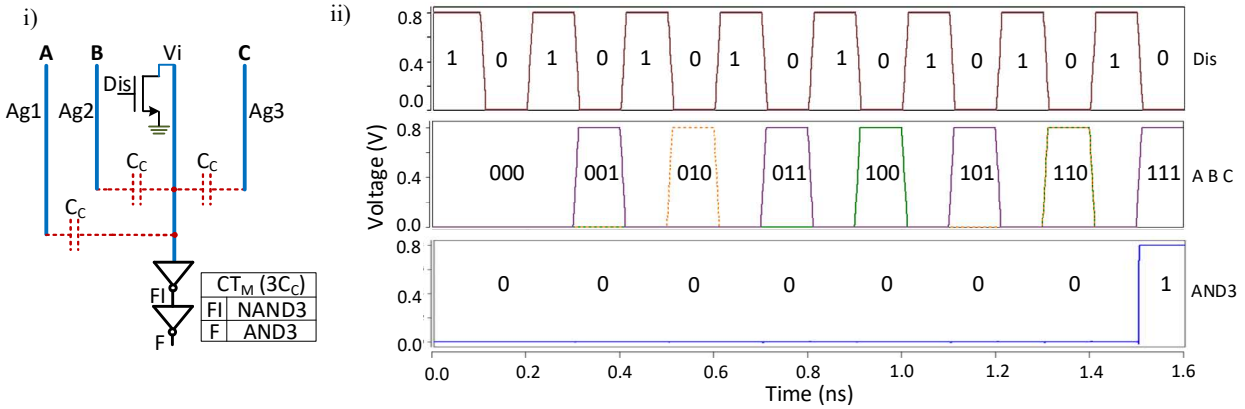

Fig.5 Crosstalk AND3 Gate: (i) AND3 Gate Circuit Schematic, (ii) AND3 Gate Simulation response

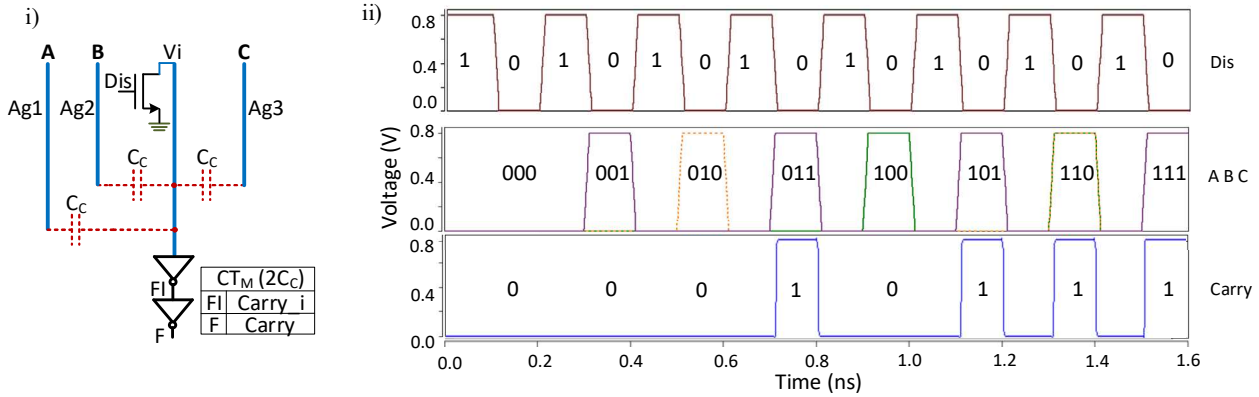

Fig.6 Crosstalk CARRY Gate: (i) CARRY Gate Circuit Schematic, (ii) CARRY Gate Simulation response

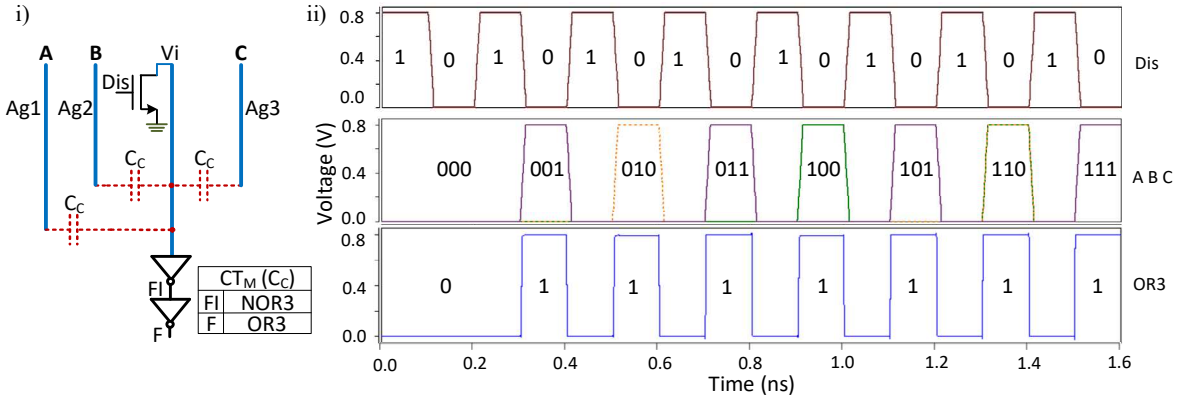

Fig.7 Crosstalk OR3 Gate: (i) OR3 Gate Circuit Schematic, (ii) OR3 Gate Simulation response

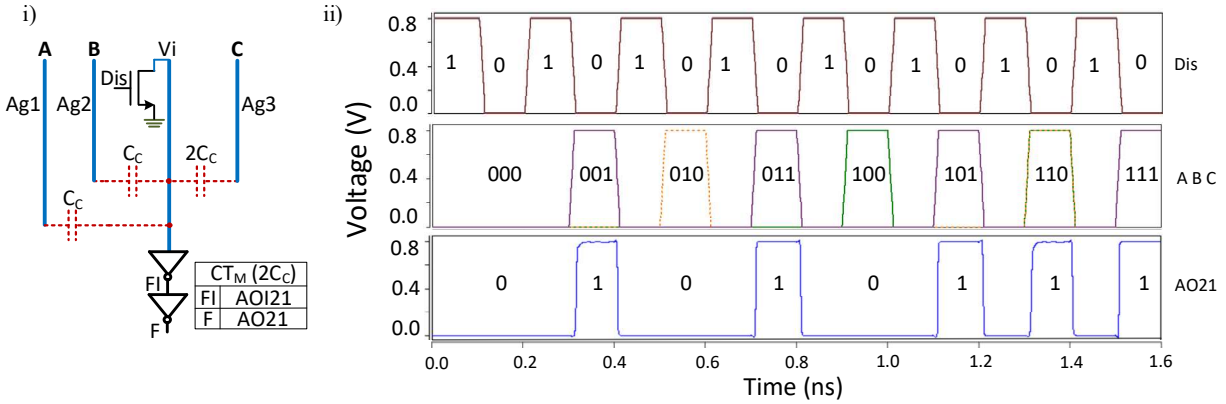

Fig.8 Crosstalk AO21 Gate: (i) AO21 Gate Circuit Schematic, (ii) AO21 Gate Simulation response

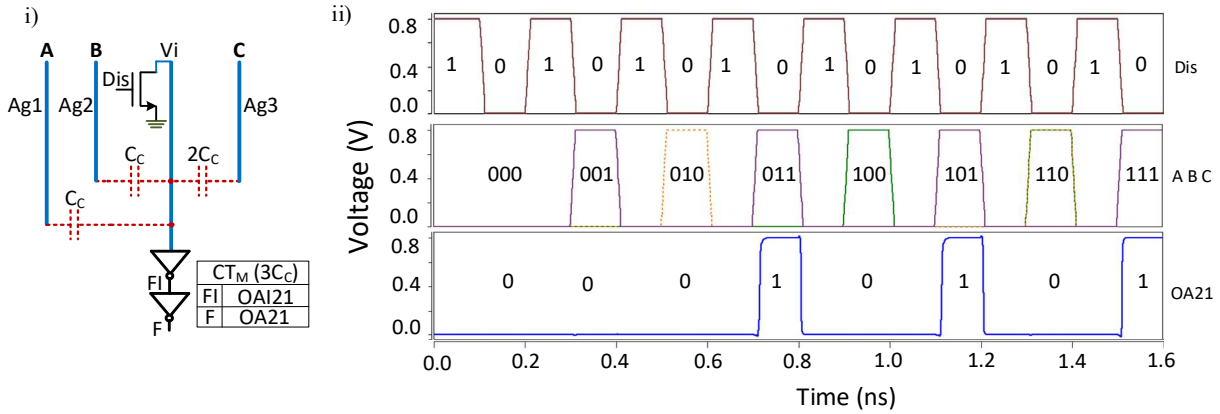

Fig.9 Crosstalk OA21 Gate: (i) OA21 Gate Circuit Schematic, (ii) OA21 Gate Simulation response

TABLE II
CROSSTALK LOGIC DESIGN TABLE FOR COMPLEX GATES

Gate	w_1	w_2	w_3	Margin Fuction
AND3	1	1	1	$CT_M(3C_C)$
CARRY	1	1	1	$CT_M(2C_C)$
OR3	1	1	1	$CT_M(C_C)$
AOI21	1	1	2	$CT_M(2C_C)$
OAI21	1	1	2	$CT_M(3C_C)$

III. CROSS-TALK POLYMORPHIC LOGIC GATES

It can be observed from the circuit schematics, Table.1 and Table.2 that unlike fixed and hardwired patterns of switches (transistors) for each logic in CMOS, CT-logic circuits are of uniform pattern with the only difference in their coupling capacitances. That means, if the coupling capacitances from inputs to the *Vi* net can be altered at runtime, the logic behavior of the gate can also be altered. This is the idea which paves the way to design reconfigurable logic circuits through Crosstalk Computing mechanism. Instead of trying to achieve this by a run-time control of material properties or constructing novel devices for this purpose, an alternate path can be taken where *Vi* net is given an additional control aggressor (*Ct*) coupled to it. When the signal transitions on the control aggressor (*Ct*) it augments an extra charge/voltage on to the *Vi* net which is equivalent to changing capacitance coupled to *Vi* net in run time. This extra voltage induced on the *Vi* net would actually disturb the intended behavior of the gate. However, if extra voltage induced is engineered properly, a new functional pattern can emerge, giving rise to polymorphic gates. By clever design choices, the polymorphism is possible between various logic functions, which are presented in this section. The polymorphism is shown between all the logic functions discussed above; homogeneous to homogeneous logic: AND-OR, AND-CARRY, OR-CARRY; heterogeneous to heterogeneous logic: AO21-OA21; homogeneous to heterogeneous logic: AO21-AND3, AO21-OR3, AO21-CARRY, OA21-AND3, OA21-OR3, OA21-CARRY.

The margin-functions in CT logic are formulated to reflect the gate's logic behavior. Therefore, the transformation of crosstalk logic gates from one function to the other function would also mean that there is an effective transformation in their margin-functions. Let, $CT_M(k.C_C)$ be the margin function of a polymorphic gate with '$w_{Ct}.C_C$' capacitance given to control aggressor (*Ct*). When *Ct*=0, the inverter can flip its state only when it receives the voltage through a total coupling capacitance of $k.C_C$; therefore, the gate's logic behavior corresponds to the margin function $CT_M(k.C_C)$. However, when *Ct*=1, an extra voltage would be induced through capacitance $w_{Ct}.C_C$, leaving only $(k\text{-}w_{Ct})C_C$ capacitance margin; i.e., the inverter can now start flipping its state just with the voltage induced due to capacitance greater than or equal to $(k\text{-}w)C_C$. Therefore, the margin function and its corresponding logic behavior now transform to $CT_M((k\text{-}w_{Ct})C_C)$. Table.3 presents the crosstalk logic design table for CT polymorphic gates. The margin functions are given both for Ct=0 and Ct=1 in Table.3.

Fig.10(i) shows the CT polymorphic AND3-OR3 circuit schematic. The inputs A, B, C, has the same coupling C_C (the coupling capacitance values are detailed in Table.3). *Ct* aggressor receives $2C_C$ capacitance. A table adjacent to the circuit diagram lists the margin function and the circuit operating modes. The margin function for AND3-OR3 cell is $CT_M(3C_C)$, which makes it behave as AND3 gate when control Ct=0. Whereas, when *Ct*=1, it augments an extra charge through coupling capacitance $2C_C$ and effectively manipulates the margin function to $CT_M(C_C)$, making it an OR3 gate. Following the function $CT_M(C_C)$, the transition of either *A* or *B* or *C* is now sufficient to flip the inverter; thus, the gate would be now biased to operate as OR3 gate. The same response can be observed in the simulation response plots as shown in Fig.10(ii): the first panel shows *Dis* and *Ct* signals; the second panel shows the input combinations fed through *A*, *B* and *C*; the third panel shows the response at stage F. It can be observed that the circuit responds as AND3 when Ct=0 for first eight input combinations (000 to 111), whereas, it responds as OR3 when Ct=1 during next eight combinations (000 to 111). For AND3 gate, if control aggressor is given just C_C coupling strength instead of $2C_C$ in the previous case, $CT_M(3C_C)$ manipulates to $CT_M(2C_C)$, which becomes polymorphic AND3-CARRY gate as shown in Fg.11 (schematic and simulation). Similarly, Fig.12(i) shows the polymorphic CARRY-OR3 circuit. Its margin function is $CT_M(2C_C)$. The control aggressor is given C_C strength, which alters the function to $CT_M(C_C)$ (see also Table.3) and behaves as an OR3 gate. The above three gates are homogeneous logic types, where input aggressors receive equal coupling strength (Table.3). Fig.13(i) shows the AO21- OA21 gate which is a heterogeneous to heterogeneous logic. The aggressors' weights, (w_1, w_2, w_3; w_{ct}), are (1, 1, 2; 1) (See Table.3). The margin function $CT_M(3C_C)$ alters to $CT_M(2C_C)$

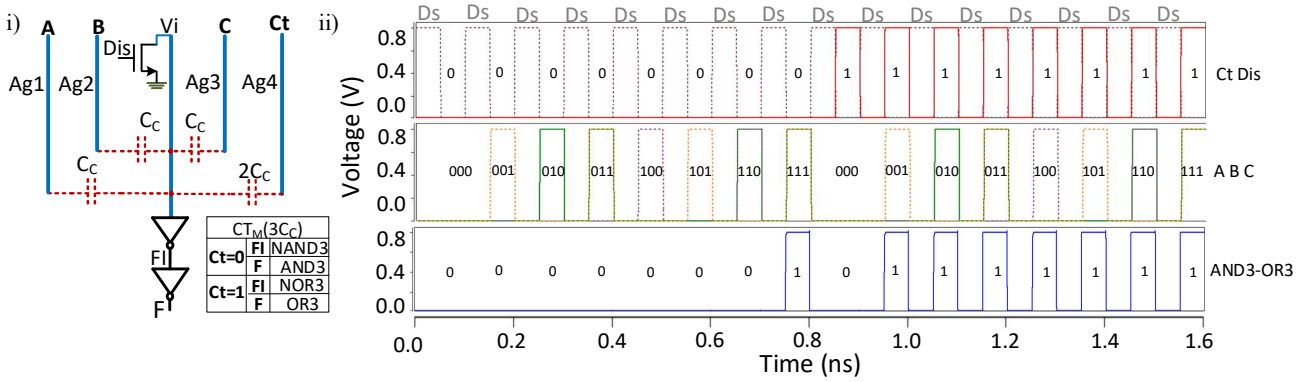

Fig.10 Crosstalk Polymorphic AND3-OR3 Gate: (i) AND3-OR3 Circuit Schematic, (ii) AND3-OR3 Simulation response

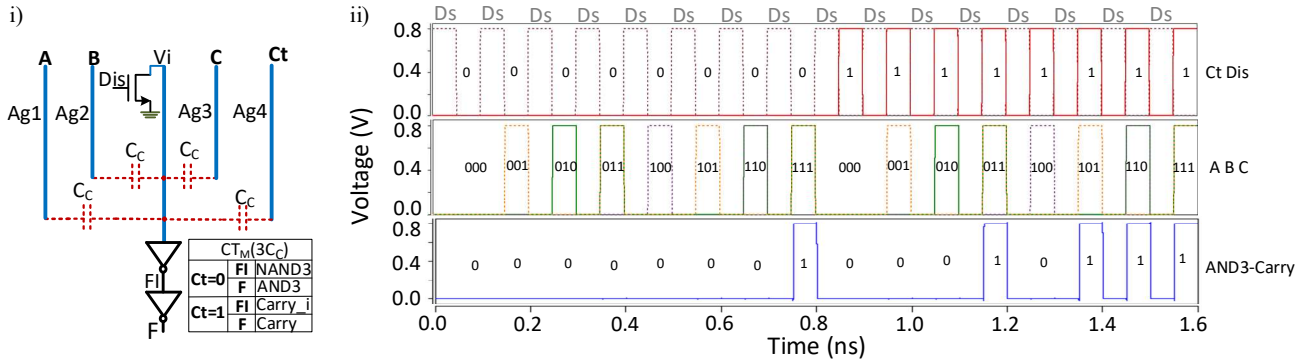

Fig.11 Crosstalk Polymorphic AND3-CARRY Gate: (i) AND3-CARRY Circuit Schematic, (ii) AND3-CARRY Simulation response

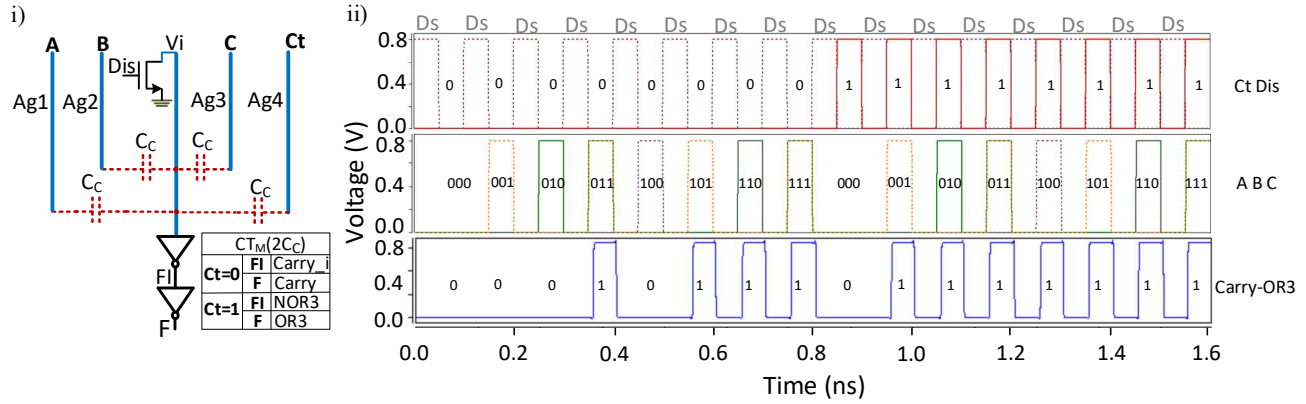

Fig.12 Crosstalk Polymorphic CARRY-OR3 Gate: (i) CARRY-OR3 Circuit Schematic, (ii) CARRY-OR3 Simulation response

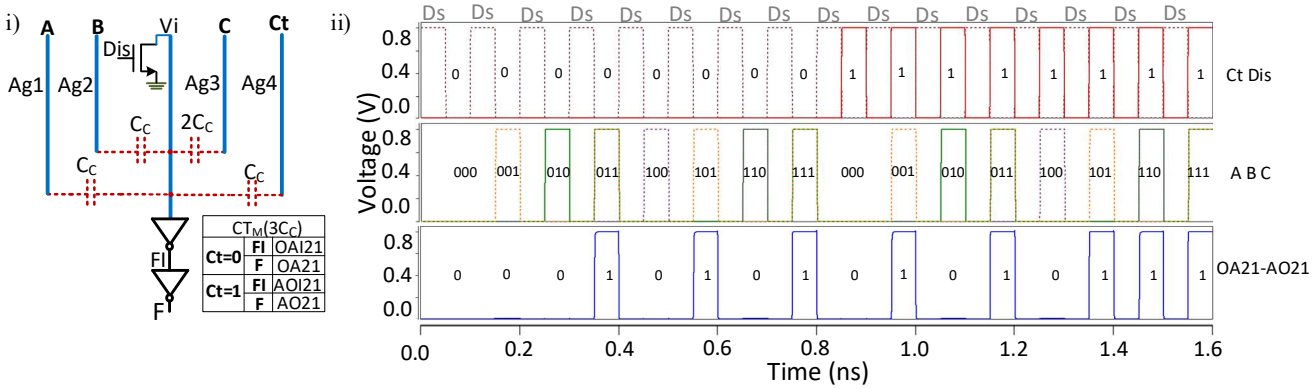

Fig.13 Crosstalk Polymorphic OA21-AO21 Gate: (i) OA21-AO21 Circuit Schematic, (ii) OA21-AO21 Simulation response

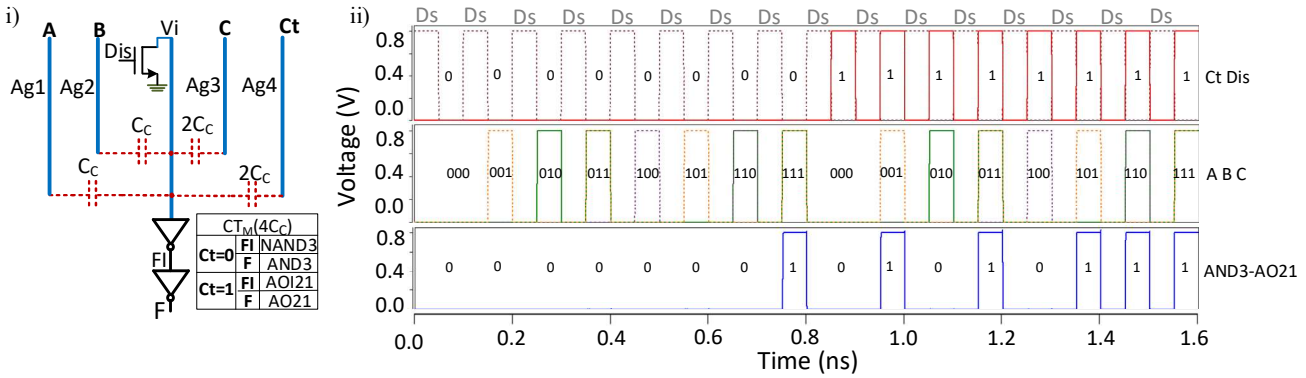

Fig.14 Crosstalk Polymorphic AND3-AO21 Gate: (i) AND3-AO21 Circuit Schematic, (ii) AND3-AO21 Simulation response

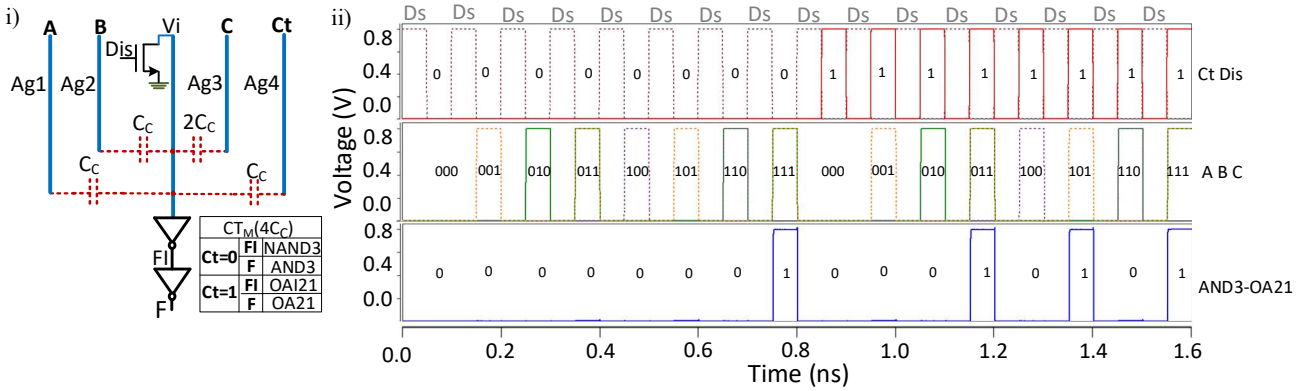

Fig.15 Crosstalk Polymorphic AND3-OA21 Gate: (i) AND3-OA21 Circuit Schematic, (ii) AND3-OA21 Simulation response

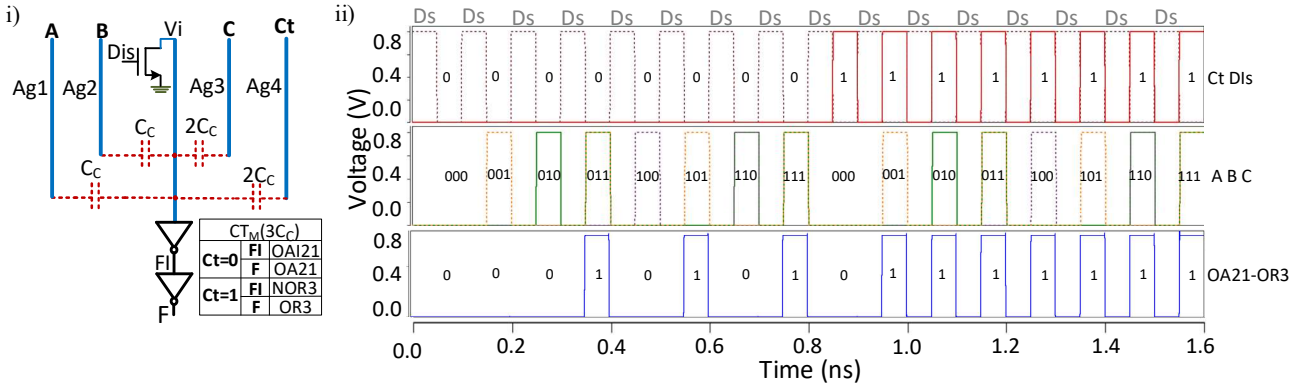

Fig.16 Crosstalk Polymorphic OA21-OR3 Gate: (i) OA21-OR3 Circuit Schematic, (ii) OA21-OR3 Simulation response

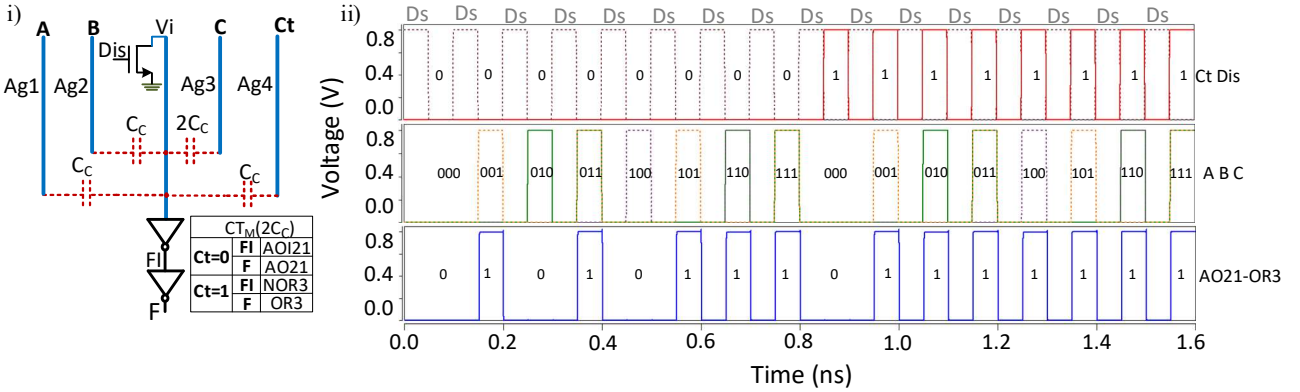

Fig.17 Crosstalk Polymorphic AO21-OR3 Gate: (i) AO21-OR3 Circuit Schematic, (ii) AO21-OR3 Simulation response

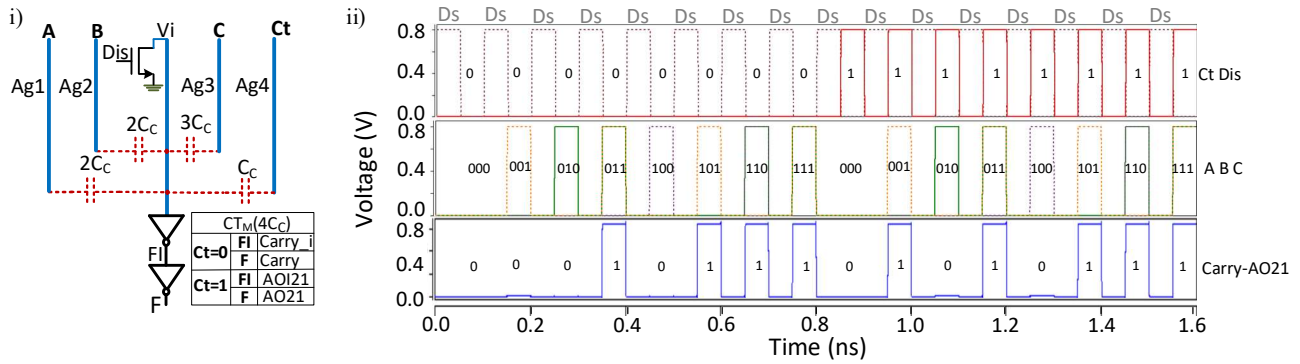

Fig.18 Crosstalk Polymorphic CARYY-AO21 Gate: (i) CARYY-AO21 Circuit Schematic, (ii) CARYY-AO21 Simulation response

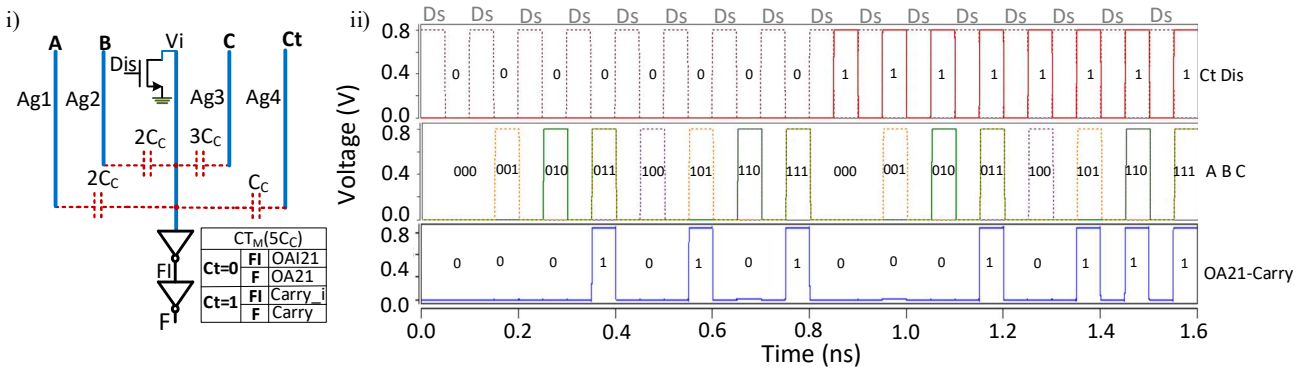

Fig.19 Crosstalk Polymorphic OA21-CARRY Gate: (i) OA21-CARRY Circuit Schematic, (ii) OA21-CARRY Simulation response

and gives AO21-OA21 polymorphism. The next six gates are homogeneous to heterogeneous logic type. Fig.14 shows the schematic and simulation of AND3-AO21 gate. The aggressor weights are (1, 1, 2; 2) (note that inputs' weights are heterogeneous). The margin function for AND3, in this case, is $CT_M(4C_C)$. The control aggressor biases it to $CT_M(2C_C)$ and operates the gate as AO21. In the previous case, if *Ct* is given C_C strength instead of $2C_C$, the margin function manipulates from $CT_M(4C_C)$ to $CT_M(3C_C)$, giving rise to AND3-OA21 gate as shown in Fig.15. Circuits in Fig.16-19 show the polymorphic OA21-OR3, AO21-OR3, and OA21-CARRY, and AO21-CARRY gates and their simulation response. Their CC, coupling weights, and margin functions are given in the crosstalk logic design table (Table.3).

TABLE 3
CROSSTALK LOGIC DESIGN TABLE FOR POLYMORPHIC GATES

Gate	C_C (fF)	w_1	w_2	w_3	w_{Ct}	Ct	Margin Fuction	Function
AND3-OR3	1	1	1	1	2	0	$CT_M(3C_C)$	AND3
						1	$CT_M(C_C)$	OR3
AND3-CARRY	0.9	1	1	1	1	0	$CT_M(3C_C)$	AND3
						1	$CT_M(2C_C)$	CARRY
CARRY-OR3	4.5	1	1	1	1	0	$CT_M(2C_C)$	CARRY
						1	$CT_M(C_C)$	OR3
OA21-AO21	0.7	1	1	2	1	0	$CT_M(3C_C)$	OA21
						1	$CT_M(2C_C)$	AO21
AND3-AO21	0.28	1	1	2	2	0	$CT_M(4C_C)$	AND3
						1	$CT_M(2C_C)$	AO21
AND3-OA21	0.21	1	1	2	1	0	$CT_M(4C_C)$	AND3
						1	$CT_M(3C_C)$	OA21
OA21-OR3	0.97	1	1	2	2	0	$CT_M(3C_C)$	OA21
						1	$CT_M(1C_C)$	OR3
AO21-OR3	3	1	1	2	1	0	$CT_M(2C_C)$	OA21
						1	$CT_M(1C_C)$	OR3
CARRY-AO21	2.2	2	2	3	1	0	$CT_M(4C_C)$	CARRY
						1	$CT_M(3C_C)$	AO21
OA21-CARRY	0.6	2	2	3	1	0	$CT_M(5C_C)$	OA21
						1	$CT_M(4C_C)$	CARRY

A. CT-P Cascaded Circuit Example

This section demonstrates cascading polymorphic gates to implement a block level polymorphic circuit. Fig.20 is a 2-bit Multiplier-Sorter-Adder circuit. The circuit uses 31 gates in total, out of which 25 are crosstalk gates, and 6 are inverters. 16 out of 25 crosstalk gates are polymorphic gates, which are efficiently employed to switch the circuit between the multiplier, sorter and adder operations. Two control signals, C1 and C2, are given to a control circuitry shown in the inset figure which generates C3-C5 signals. C1-C5 signals are employed in the circuit to switch the circuit between three functions. Fig.21 shows the simulation response of the circuit; different operation modes of the circuit are annotated on top, which are, Multiplier (M), Sorter (S), and Adder (A). The first panel in the figure shows Dis signal; *Dis*=1 is the discharge state (DS) and *Dis*=0 is the Logic Evaluation (LE) state. The second panel shows the control signals *C1* and *C2*, whose values as 01, 11 and 10 corresponds to the multiplier, sorter, and adder operations, respectively. Third and fourth panels show the 2-bit inputs A[1:0] and B[1:0].. The subsequent four panels show the 4-bit response of the circuit, Y[3:0]. In order to effectively demonstrate the transformation of the circuit, control signals are given such that the circuit switches alternately

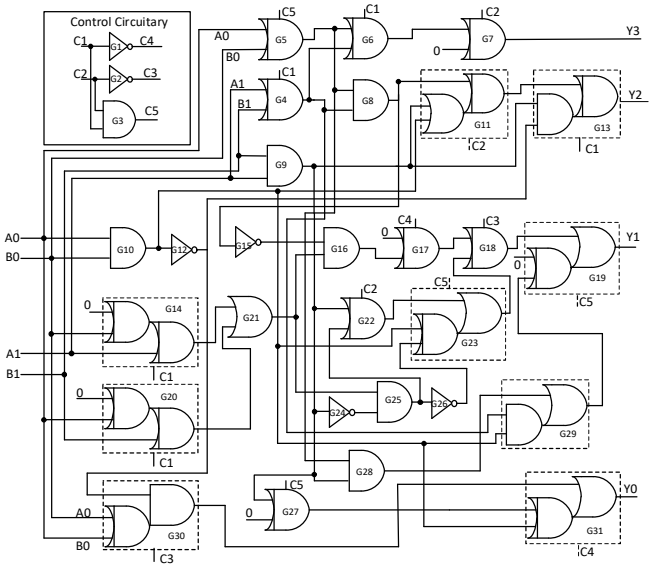

Fig.20 Crosstalk Polymorphic Multiplier/Adder/Sorter circuit

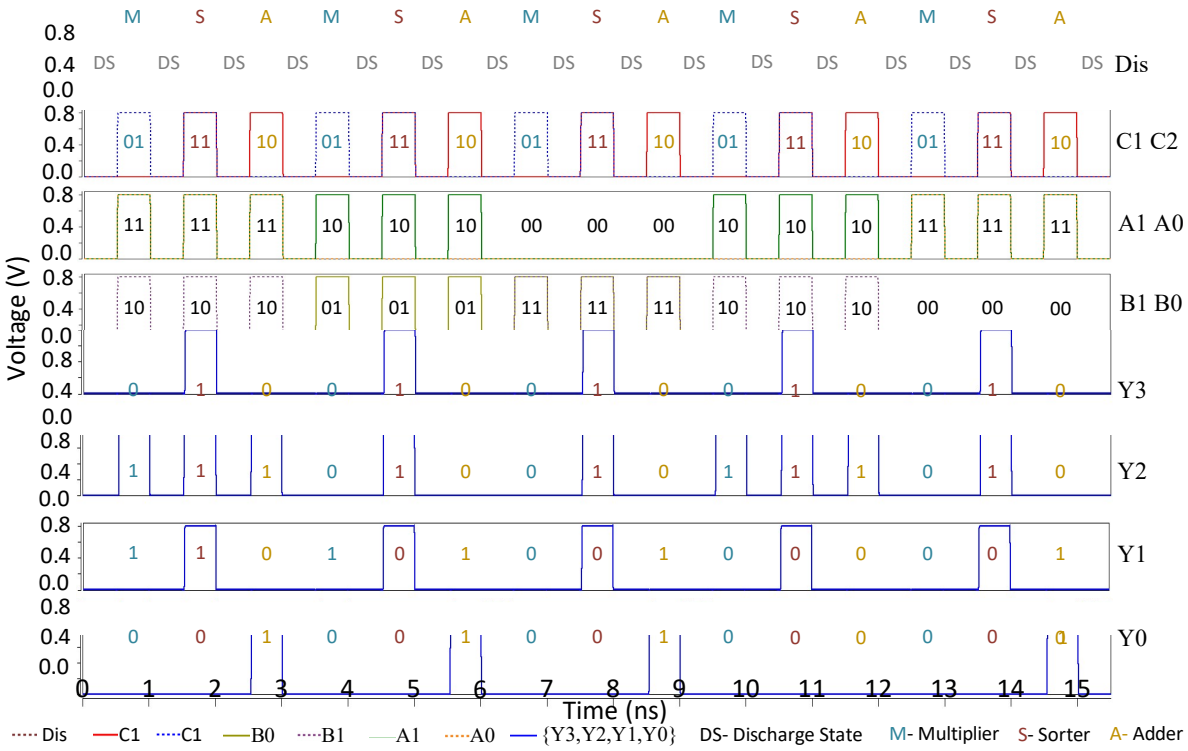

Fig.21 Crosstalk Polymorphic Multiplier/Adder/Sorter circuit simulation response

between multiplier, sorter, and adder modes, and in each set of these modes, common input values are fed through A1A0 and B1B0. For example, for the first input combinations, 11 and 10, the multiplier operation gives 0110 as output while the succeeding sorter and adder operations give 1110 and 0101 outputs, respectively. Similarly, for the second inputs, 10 and 01, M, S, and A operations give 0010, 1100 and 0011 outputs, respectively. In a similar fashion, few other combinations are shown in the next stages. The circuit consumes only 155 transistors in total.

B. Discussion about Signal Integrity

Although the actual computation in CT-logic happens in the nano-metal lines, maintaining Signal Integrity and drive-strength for next stage gates is crucial to construct the larger circuits. Therefore, the output signals from the logic gates need to be robust, full-swing and possess enough drive-strength to drive the fan-out loads. These issues are addressed by connecting the victim net to a CMOS inverter. It acts as a regenerative boolean threshold function, where, it detects the logic levels computed on victim net and restores them to full swing, i.e., the victim voltages below the low logic threshold are restored to a logic high and voltages above the high logic threshold are restored to logic low. Also, the inverters provide good noise margins to the signals.

The other issue that cascaded circuits faces is CT-logic specific monotonicity problem; which is, the CT-logic gates need the input signal transitions from 0 to 1 during each logic evaluation state for correct logic operation. If a logic high is retained on the victim node from the previous operation it leads to logic failure. For example, when a given CT-logic gate is driven by another inverting CT-logic gate (NAND, NOR etc..), it receives a logic high during DS. This logic high would be carried to the next evaluation state, which prevents the 0 to 1 signal transitions and leads to logic failure. This issue is resolved in this paper by using a Pass-Gate (PG) solution [11]. In PG, the inverting and non-inverting gate interfaces are connected through a transmission gate. The aggressors connected to these transmission-gates are discharged to ground in every DS (during which the transmission gates passing the inputs are shut off), and input signals are passed afresh during each LE state; thus, fixes the monotonicity problem. The other solution is by using a different set of CT-logic gates that operate on inputs' falling edge transitions, which are not presented here. Thus, a fully working large-scale compact polymorphic circuits, with reduced size, improved performance and power can be achieved using CT-logic style.

IV. Benchmarking

The Crosstalk polymorphic circuits are better compared to existing polymorphic approaches in terms of technology, device and circuit metrics such as working mechanism, process node dependency, control variable, scalability, performance trade-offs and transistor count [9]. Despite of its radically different logic and reconfigurability aspects, the working mechanism in crosstalk computing is based on well-known capacitive electrostatics, which makes it easily realizable through the existing process setups. Interestingly, the crosstalk circuits address the impediments advanced technology nodes face in terms of interconnect crosstalk by astutely leveraging it for the computational purpose; thus, it is friendly to lower technology nodes. The polymorphism is achieved by using the same aggressor-victim technique performing the logic computation, which enables very fast reconfiguration of the

gates. In addition, the crosstalk polymorphic approach consumes very fewer transistors than any other approach, and a wide range of compact single-stage polymorphic complex logic implementations like in Crosstalk logic were not reported in other approaches [9]. Moreover, as the crosstalk polymorphic circuits are uniform, modular and functionally rich, they can be scalable to larger polymorphic systems.

The density, switching energy and performance for all the crosstalk gates presented above are characterized and benchmarked with their counterpart CMOS implementations. Table.4 presents these results. The CMOS implementation is multiplexer based, where independent stand-alone circuits are designed and selected through a multiplexer. Both the circuits are implemented and benchmarked using 16nm PTM tri-gate transistor models. The benefits are huge in all aspects of Crosstalk logic based implementations. As shown in Table.4, the polymorphic logic gates have over 6x density, ~1.5x performance, and ~2x power benefits. The benefits are primarily due to reduced transistor count and are projected to be higher for large-scale designs. A comparison of CMOS vs Crosstalk circuit can illustrate the source of these benefits. For an example, the AND3-CARRY polymorphic circuit, with its boolean expression, ABCS'+ S(AB+BC+CA), requires just 5 transistors compared to 30 transistors in CMOS based implementation. For the polymorphic Multiplier-Sorter-Adder unit, the benefits were 3.4x, 62% in terms of density and power with comparable performance with respect to CMOS at 16nm. It is to be noted that owing to the reduced circuit density the interconnection requirements would also be considerably less.

V. Conclusion

Crosstalk Logic is a very novel and radically different way of doing the logic computation. The paper develops a detailed framework for polymorphic logic circuits in Crosstalk Logic and implements a wide range of crosstalk polymorphic logic gates. The gates presented are reconfigurable AND-OR, AO21-OA21, AND3-AO21, AND3-OA21, OR3-AO21, OR3-OA21, AND3-CARRY, CARRY-OR3, CARRY-AO21 and OA21-CARRY. It also characterizes the circuit implementations and benchmarks them with respect to CMOS implementations. At gate level, the benefits observed are, 6x in density, 2x in power and 1.5x in performance. Our circuit evaluation and benchmark comparisons show that CT-P logic approach is very compact (i.e less device count) and efficient than any other polymorphic approach. The benefits observed are due to the reduced transistor count in all the circuits. The large number of polymorphic circuits implemented, and the benefits observed make them a potential solution to many hurdles ICs face at advanced technology nodes. Fortunately, the Crosstalk Computing requires known materials and devices, and comply to existing process/fabrication setups. This motivates us to purse for experimentation and practical realization of the Crosstalk Logic circuits as future work.

VI. References

[1] McDermott, M.W., and Turner, J.E.: 'Configurable NAND/NOR element'. United States Patent 5,592,107, January 1997

TABLE 4
Benchmarking of Crosstalk Logic Gates With Respect to CMOS

	Transistor Count		Switching Energy (aJ)			Performance (ps)		
GATES	CMOS	Cross-talk	CMOS	Cross-talk	%Reduction	CMOS	Cross-talk	% Reduction
NAND2	4	3	232.1	122.3	47.31	4.12	4.06	1.4325
NOR2	4	3	202.7	260.5	-28.5	5.61	5.86	-4.492
AOI21	6	3	154.4	207	-34.07	5.73	5.51	3.9373
OAI21	6	3	229.3	135.2	41.03	4.36	5.17	-18.52
NAND3	6	3	347.7	112.5	67.65	4.98	4.18	16.11
Carry	16	5	1198.8	326.592	72.757	14.5822	8.6923	40.39102
NAND2-NOR2	14	3	796.14	139.03	82.537	13.46	4.32	67.887
NAND3-NOR3	18	3	1472.6	172.02	88.319	13.21	5.12	61.224
AOI21-OAI21	18	3	698.42	190.52	72.721	9.52	5.39	43.379
NAND3-AOI21	18	3	1091.3	641.38	41.228	14.08	14.14	-0.4211
NAND3-OAI21	18	3	874.99	959.44	-9.651	11.69	19.4	-65.922
NOR3-AOI21	18	3	1030.4	661.67	35.785	17.78	12.65	28.864
NOR3-OAI21	18	3	938.88	546.89	41.75	18.14	11.47	36.807
CARRY-OR3	30	5	4258.6	420.15	90.134	15.02	8.3	44.74035
Carry-AND3	30	5	3059.9	289.6908	90.533	16.7759	7.3955	55.91593
Carry-AO21	30	5	2332.9	481.3124	79.368	28.9174	10.56	63.48219
OA21-Carry	30	5	2004.2	366.9129	81.693	15.67	9.9741	36.34907
MUL-SORT-ADD	408	155	16.2 fJ	6.104 fJ	62.41	61.5	54.4	11.56

[2] R. Ruzicka, “New Polymorphic NAND / XOR Gate 2 Known Polymorphic Gates,” pp. 192–196, 2007.

[3] A. Stoica, R. Zebulum, and D. Keymeulen, “Polymorphic Electronics,” Evolvable Syst. From Biol. to Hardw., vol. 2210, pp. 291–302, 2001.

[4] A. Stoica et al., "Taking evolutionary circuit design from experimentation to implementation: some useful techniques and a silicon demonstration," in IEE Proceedings - Computers and Digital Techniques, vol. 151, no. 4, pp. 295-300, 18 July 2004.

[5] W. Ditto, A. Miliotis, K. Murali, S. Sinha, and M. Spano, “Chaogates: Morphing logic gates designed to exploit dynamical patterns,” Chaos 20, 037107 (2010)

[6] M. De Marchi et al., “Configurable logic gates using polarity controlled silicon nanowire gate-all-around FETs,” IEEE Electron Device Lett., vol. 35, no. 8, pp. 880–882, 2014.

[7] J. Zhang, P. E. Gaillardon, and G. De Micheli, “Dual-threshold-voltage configurable circuits with three-independent-gate silicon nanowire FETs,” Proc. - IEEE Int. Symp. Circuits Syst., pp. 2111–2114, 2013.

[8] S. Rakheja and N. Kani, "Polymorphic spintronic logic gates for hardware security primitives — Device design and performance benchmarking," 2017 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH), Newport, RI, 2017, pp. 131-132.

[9] Naveen kumar Macha, et al., “A New Concept for Computing Using Interconnect Crosstalks,” 2017 IEEE International Conference on Rebooting Computing (ICRC), Washington, DC, USA, December 2017.

[10] Naveen kumar Macha, Sandeep Geedipally, Bhavana Tejaswee Repalle, Md Arif Iqbal, Wafi Danesh, Mostafizur Rahman “Crosstalk based Fine-Grained Reconfiguration Techniques for Polymorphic Circuits,” IEEE/ACM NANOARCH 2018.

[11] Naveen kumar Macha, Bhavana Tejaswini Repalle, Sandeep Geedipally, Rafael Rios, Mostafizur Rahman “A New Paradigm for Fault-Tolerant Computing with Interconnect Crosstalks,” 2018 IEEE International Conference on Rebooting Computing (ICRC), Washington, DC, USA, December 2018.