Architecture and Optimization of Associative Memories used for the Implementation of Logic Functions based on Nanoelectronic 1S1R Cells

Arne Heittmann and Tobias G. Noll
Chair of Integrated Digital Systems and Circuit Design, RWTH-Aachen University, Germany
{heittmann,tgnoll}@ids.rwth-aachen.de

***Abstract*— A neuromorphic architecture based on Binary Associative memories and nanoelectronic resistive switches is proposed for the realization of arbitrary logic/arithmetic functions. Subsets of non-trivial code sets based on error detecting 2-out-of-n-codes are thoroughly used to encode operands, results, and intermediate states in order to enhance the circuit reliability by mitigating the impact of device variability. 2-ary functions can be implemented by cascading a mixer memory, a correlator memory, and a response memory. By introduction of a new cost function based on class-specific word-line-coverage, stochastic optimization is applied with the aim to minimize the overall number of active amplifiers. For various exemplary functions optimized architectures are compared against solutions obtained using a standard-cost function. It is especially shown that the consideration of word-line-coverage results in a significant circuit compaction.**

Keywords—Resistive Switches, Selector Device, Associative Memory, Clipped Hebbian Synaptic Rule, Reliability, logic functions, arithmetic function, nanoelectronics, BiNAM

I. Introduction

With the advent of emerging nanoelectronic devices such as resistive switches (RS) [1] the associative memory concept becomes interesting for the implementation of logic functions. For many material systems RSs can be realized with the smallest possible footprint, i.e. an area occupation of $4F^2$ (F: lithographic feature size) which allows for the area-efficient realization of architectures that implement functions in a connection-centric way. So far, binary neural associative memories (BiNAMs) have been extensively studied in the context of information retrieval, pattern recognition, and brain modelling [2]. The input/output patterns of the BiNAM are considered to be binary vectors, which designates BiNAMs as a potential candidate also for the implementation of logic/arithmetic functions: the *n*-ary combination of *n* independent operands is considered as a generalized *address* used to look up a desired functional output.

Due to the regular matrix architecture of BiNAMs, standard-memory structures can be used for a circuit implementation. With respect to nanoscaled RS, it is only necessary to provide a so-called low-resistive-state (LRS) and a sufficiently large high-resistive-state (HRS). Under the assumption, that the electrical behavior of nanoscaled devices becomes (generally speaking) unreliable, dedicated - and potentially circuit-specific - measures have to be taken into account in order to achieve capabilities for transient error-detection, fault tolerance, and error correction. Most concepts found in the literature for the implementation of logic functions based on nanoelectronic RSs (e.g. [3-6]) provide no particular build-in features that contribute to reliability.

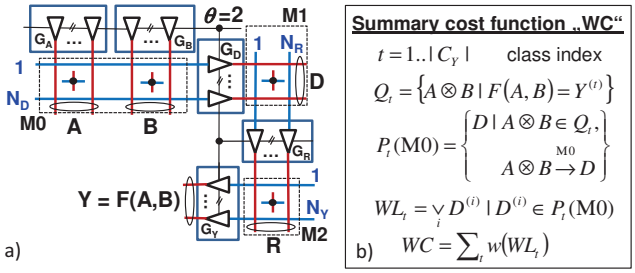

Figure 1: a) 3-layered BiNAM-architecture for a connection-centric implementation of *Y=F(A,B)*, b) proposed cost function.

In this paper it is proposed to encode operands and intermediate symbols using so-called *binary-constant-weight*-codes (*m-out-of-n*-codes). In principle, *m-out-of-n*-codes can detect all *unidirectional* errors [7] and all 1-bit *bidirectional* errors as well. In addition to that, these codes turned out to be well suited to be processed by the BiNAM [2]. However, one of the yet unsolved problems is the preparation of design principles that provide a guarantee for an error-free mapping based on BiNAMs working near their storage capacity limits. In this work a three-stage BiNAM-based architecture is proposed which allows for the realization of a connection-centric implementation of arbitrary logic/arithmetic functions operating in an error-free way.

II. The BiNAM Concept

A. The Clipped Hebbian Synaptic Rule

The proposed associative memory is based on an architecture which is similar to conventional memory structures. It comprises *q* word lines (WLs) and *p* bit lines (BLs), and comparators which transform a bit line signal into a binary output signal. For a given mapping of binary vectors $X^{(k)} \rightarrow Y^{(k)}$, $k=1..N_M$, *connections* can be established between WL and BL according to the so-called clipped Hebbian synaptic rule [2]

$$c_{ij} = \bigvee_{k=1}^{N_M} X_j^{(k)} \cdot Y_i^{(k)} \quad i=1..p \quad j=1..q \ . \qquad (1)$$

In (1) $X_j^{(k)}$ denotes the (binary) state of the j-th component in the binary vector $X^{(k)}$, $Y_i^{(k)}$ the state of the i-th component in the binary vector $Y^{(k)}$, N_M the total number of pairs $(X^{(k)},Y^{(k)})$, and "$\vee$" the logical OR function. A connection between a particular word line j and a bit line i exists if $c_{ij}=1$ holds.

B. Pattern Mapping

The mapping of a given input pattern X to an output pattern (or symbol) Y is realized by the determination of the sum

$$S_i = \sum_{j=1}^{q} X_j \cdot c_{ij} \quad , Y_i = 1 \Leftrightarrow S_i \geq \theta \tag{2}$$

for any bit line $i=1...p$. S_i is compared with a threshold θ which results in a particular state for the output bit Y_i (2). If a given input pattern X is contained in the pattern set used in (1), an appropriate choice of θ is given by

$$\theta = \sum_{j=1}^{q} X_j \,. \tag{3}$$

III. Logic Functions Realized by BiNAM

An arbitrary bivariate operation $Y=F(A,B)$ is considered which involves two operands A and B. A and B can take on symbols from a given (finite) code set C encoding states (or numbers) using *2-out-of-n* codes. Then, any desired function F can be realized by a particular bivariate mixing operation $A\otimes B\in C^2 \rightarrow D\in C_D$, a so-called monadic correlation operation $D\rightarrow R\in C_R$, and a monadic response operation $R\rightarrow Y\in C_Y$. A particular set C_X (X=D, R, or Y) holds *2-out-of-*N_X codes which can be used for encoding states of the operand X. The entire sequence $A\otimes B\rightarrow D\rightarrow R\rightarrow Y$ is a connection-centric implementation of the function F. The mixing operation, the correlation operation, and the response operation are implemented by the connection matrices M0, M1, and M2 respectively (cf. Fig.1) using threshold gates (comparators).

A. Preliminary Work

The proposed architecture builds on some preliminary works. First, in [9] a fundamental circuit concept based on passive crossbars has been proposed and analyzed with regard to variability. Here, a crossbar is used to realize a connection matrix $\{c_{ij}\}$. Any particular connection c_{ij} (2) is represented by the state of a memory cell which is composed of resistive switch RS and a *selector device* S [8] connected in series (Fig.2). A selector device allows for the implementation of so-called *unidirectional* connections [9]: electrical current can only flow in an unidirectional way from word lines to bit lines when they are *connected*. Specifically, a word line WL j is considered to be connected to a BL i in the case that the RS of cell c_{ij} is in a LRS ($c_{ij}=1$). In turn, a high resistive state (HRS, $c_{ij}=0$) electrically isolates the WL j from the BL i. If states of individual bits of a given input vector X are assigned to voltage levels V_R (representing a logical '1') and ground (representing a logical '0') respectively, the sum S_i (2) is presented by I_{BL} and/or V_{BL} [9]. It turns out that the signal margin of S_i can be maximized (and the circuit reliability as well) if the number of cells contributing significant current to I_{BL} is limited to 2. In this case, (I) the choice θ=2 constitutes the most reliable threshold operation, and (II) any monadic (unary) mapping operation should involve *2-out-of-*N_x-codes only.

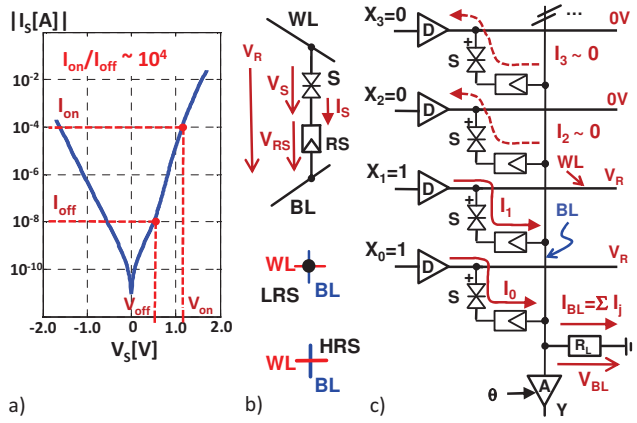

Figure 2: a) exemplary I/V characteristics of a selector device [8]. b) Cell composed of a selector S connected to a word line WL and an RS connected to a bit line BL. A HRS isolates WL and BL while a LRS establishes an *unidirectional connection* between WL and BL, c) bit line architecture and current summation.

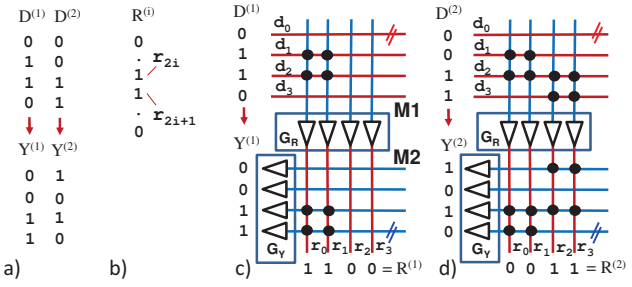

Figure 3: Raw-matrix construction of M1/M2. a) exemplary mapping, b) definition of a non-overlapping two-hot code, c-d) interference-free programming based on rule (1).

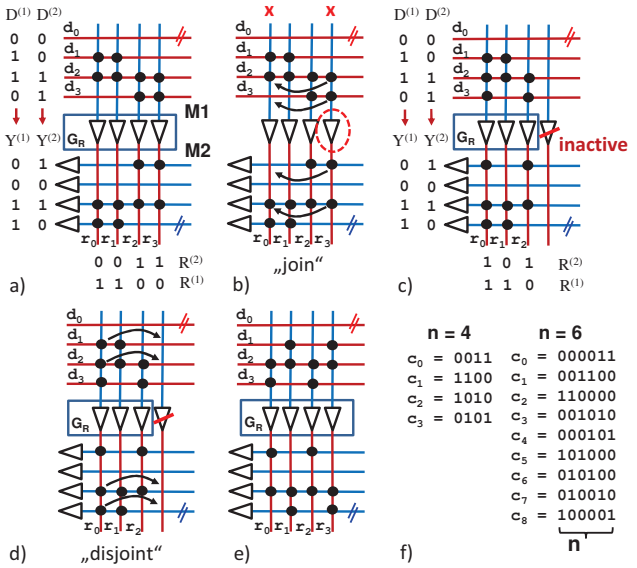

Figure 4: a-e) Reorganization of M1/M2 based on a join-operation and a disjoint-operation, respectively; f) example code sets for C.

If *2-out-of-*N_x-codes are used to encode operands, a bivariate function $F(A,B)$ receives *4-out-of-*(N_A+N_B)-encoded input symbols $A\otimes B$. If the codes for A and B are drawn from a

code set C which fulfills ("⊕" denotes the bit-wise EXOR function):

$$\forall i, j : c_i, c_j \in C \Rightarrow c_i \oplus c_j \notin C . \quad (4)$$

it is possible to show that a connection matrix exists such, that (I) for any input $A \otimes B$ at most 2 connections deliver significant current to a comparator of the output layer, (II) - unlike (4) - the choice of θ=2 for the comparators is still appropriate, (III) any output pattern D is represented by a *2-out-of-*N_D-code, and (IV) the mapping $A \otimes B \rightarrow D$ is strictly one-to-one [9]. This operation $A \otimes B \rightarrow D$ is called *mixing operation M0*. In Fig.4f some example code sets for C are shown (n=4 and n=6).

If the operands A and B originate from a subset $C_A, C_B \subset C$ it is possible to achieve a compaction of the connection matrix by execution of so-called *join()* and *disjoint()* operations driven by stochastic optimization [9]. Both *join()* and *disjoint()* reorganize the connection structure of M0 by merging and splitting individual connections with the ultimate goal to reduce the number of comparators N_D, and to make the emerging code set C_D more suitable for the mapping operation $D \rightarrow R \rightarrow Y$.

B. Construction of Matrices M1/M2

The operation $D \rightarrow R \rightarrow Y$ (cf. Fig.1) is divided into two unary mapping operations executed in succession. The mapping $D \rightarrow R$ is realized by M1, the mapping $R \rightarrow Y$ is realized by M2. Initially, a (trivial) code set C_R is defined which contains exactly $|C_D|=|C_R|$ non-overlapping 2-hot-endoded symbols: any $D^{(k)} \in C_D$ is associated to a particular $R^{(k)} \in C_R$. Then, the intermediate matrix M1 (mapping $D \rightarrow R$) can be programmed using (1). In the same way the matrix M2 is programmed (cf. Fig.3). Note, that for given $Y=F(A,B)$ the mapping $A \otimes B \rightarrow D$ has to be known in advance in order to establish $D \rightarrow R \rightarrow Y$. Therefore, the optimization of M0 and M1/M2 are executed consecutively.

The choice of the initial code set C_R results in an exaggerated use of comparators in the layer G_R. Just like for the optimization of M0 [9] *join()* and *disjoint()* operations can be defined which can be used to reduce the number of comparators in M1. If a comparator r_y is selected to be removed, the respective bit line connections are moved to a target comparator r_x which takes on the responsiveness of r_y: any input pattern D which forces the comparator r_y to be active (before *join()* has been executed) necessarily must evoke comparator r_x after *join()* has been executed. If this operation is successful, the code set C_R certainly changes. Hence, also the word line connections have to be moved to r_x which is equivalent to a re-programming operation of M2 using (1) with changed input patterns. However, in some cases the *join*-operation generates connections which unintentionally cause an interference with different input patterns: the comparator r_x might become responsive to additional input patterns D which were not able to evoke neither r_y nor r_x before *join()* was executed. An interference shows up by (I) having more than two connections delivering current to a bit line, and (II) the existence of *m-out-of-N* codes in C_R (or C_Y) with $m>2$. If a *join*-operation causes such an interference (to be checked by a function *joinable*) the *join*-operation must be abandoned

Algorithm 1 Stochastic optimization of M0 and M1/M2

```
set all rows to active, T = 1
repeat until (exit_condition())
    current_costs=COSTS()
    select randomly two active rows r_x, r_y
    if (joinable(r_x, r_y))
          r_x ← join(r_x, r_y) ; r_y ← inactive
       if (expected_costs=COSTS() > current_costs)
                recover(r_x, r_y)
       else if (log(RAND() > -T0/T)
                disjoint(r_x); disjoint(r_y)

    T=T+1
output active rows
```

Figure 5: Algorithm to minimize architectural costs. RAND() generates a random number 0 < RAND() < 1, T0 = 7.0.

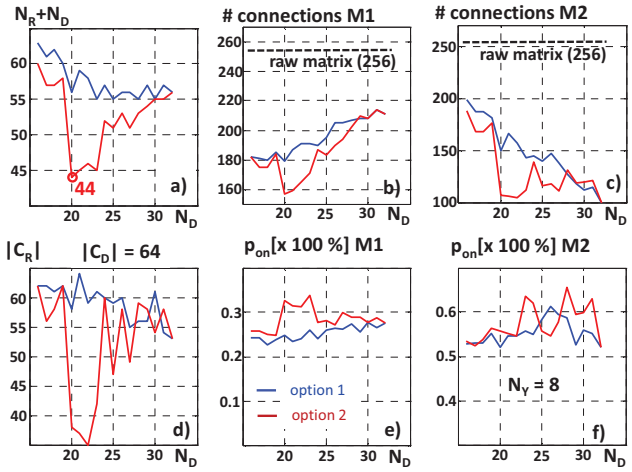

Figure 6: a-f) Performance figures of M0, M1, and M2 dependent on N_D, given Y=(A+B)mod z, z=8, n=8, T=10^6 .

which is realized by a *recover*-operation.

In contrast to the *join*-operation a *disjoint*-operation is also defined. The *disjoint*-operation splits a selected bit line r_x into its parts by reversing *all* preceding *join*-operations of r_x. Here, inactive bit lines will adopt the fragments obtained from splitting, see Fig.4d-e.

C. Stochastic Optimization and Cost Functions

The stochastic optimization procedure (cf. Fig.5 and [9]) incorporates the evaluation of costs which characterize a concurrent architectural implementation. A prospective *join()* operation is executed if the overall costs decrease after *join()*, otherwise it is abandoned. Simple cost functions are given by the number of comparators N_D in layer G_D (optimization of M0) or the number of comparators N_R in layer G_R (optimization of M1/M2). However, it is subsequently shown that it is possible to incorporate an empirical measure for the risk of interference during the optimization of M1/M2 already during the reorganization of M0 (note that M0 is optimized first). In many cases the size of the input space C^2 is significantly larger than the size of the output space $|C_Y|$. Therefore, it is useful to establish a class-centric view of F: the sets Q_t (cf. Fig.1b)

TABLE I: COMPARISON OF ARCHITECTURAL PARAMETERS OBTAINED FOR OPTION 1 (BLUE, STANDARD) AND OPTION 2 (RED, WORD LINE COVERAGE).

Y=F(A,B)	$\|C_Y\|$	$N_{D,opt}$	$\|C_D\|$	$\|C_R\|$	$P_{on,M0}$	$P_{on,M1}$	$P_{on.M2}$	N_R+N_D	ε
(A + B) mod z	8	20	64	38	15.0%	32.7%	55.7%	44	68.8%
„0“ - „7“		23	64	61	15.3%	25.9%	55.9%	55	85.9%
(A x B) mod z	8	16	48	26	17.2%	46.1%	63.3%	32	50.0%
„0“ - „7“		19	64	53	17.4%	27.8%	54.4%	50	78.1%
((A x B)/z) mod z	7	18	53	34	16.7%	31.1%	53.2%	41	64.1%
„0“ - „6“		21	64	48	16.4%	29.1%	54.0%	46	71.9%
MAX(A,B)	8	20	64	31	15.0%	36.7%	53.6%	41	64.1%
„0“ - „7“		20	62	46	16.8%	32.7%	59.4%	44	68.8%
MIN(A,B)	8	20	56	43	15.6%	32.8%	56.5%	43	67.2%
„0“ - „7“		16	64	52	18.8%	35.7%	53.4%	45	70.3%
COMPARE(A,B)	3	16	48	15	17.1%	57.3%	54.1%	28	43.8%
„<“ „=“ „>“		16	40	33	19.2%	41.2%	63.2%	33	51.6%
Carry-lookahead	3	18	42	20	17.6%	48.7%	53.4%	30	46.9%
„D“ „G“ „P“		18	50	34	18.0%	42.3%	59.0%	34	53.1%

include the particular input patterns $A \otimes B$ which are mapped to a specific $Y^{(t)} \in C_Y$. The index t is called class-index. If a join-operation is possible, *join()* is executed and the sets $P_t(M0)$ (cf. Fig.1b) are gathered for every class index t. The vector WL_t (cf. Fig.1b) specifies the set of word lines in M1 which would participate (with respect to the whole input space) in the generation of $Y^{(t)}$ later on. Then, the number

$$WC = \sum_{t} w(WL_t) \tag{5}$$

represents the so-called accumulated *class-specific-word-line-coverage* using the hamming weight $w(.)$. If (5) is minimized the optimization process prefers solutions of M0 which tend to group codes of C_D (if possible) that (I) have the same output Y anyway and are (II) similar (i.e. have a common active component) among each other. A specific side effect of minimizing (5) is that the size $|C_R|$ is significantly reduced; the matrix M1 is able to partly correlate patterns of the input space which cause the same output.

IV. RESULTS AND SUMMARY

For the evaluation of the proposed optimization the parameters n=8 and z=$|C_A|$=$|C_B|$=8 were chosen representing 3 bit per symbol. This particular choice of parameters allows for the optimization of architectural parameters with reasonable effort. A selected set $C_A(=C_B)$ realizes a true subset of C which allows for different implementations of M0 with different sizes N_D ranging from $N_{D,min}$=16 up to $N_{D,max}$=32. Several 2-ary arithmetic functions (cf. Table I) were examined. Two different options were examined for the optimization of M0. Option 1 (blue) represents a cost function which is based on the comparator count N_D (the “standard cost function”), option 2 (red) represents the proposed accumulated *class-specific-word-line-coverage* (5). For $N_{D,min} \le N_D \le N_{D,max}$ and both cost functions various realizations of M0 were generated using the stochastic optimization approach. While the optimization of M0 for option 1 was relatively straight-forward (i.e. the optimization loop immediately stops as soon as the target value for N_D was obtained), T=2x10^6 iteration were allowed to obtain a stable cost-minimal solution of M0 for given F and option 2. Afterwards, M1/M2 were optimized (also T= 2x10^6 iterations) based on the various realizations of M0 using the count N_R as single cost function. Both optimization runs start at N_R=128. After optimization, the number of connections was reduced in all cases, both for M1 and M2 (Fig.6b-c); apparently several connections typically get *shared*. For the given example, also the size of C_R is reduced which demonstrates the ability of M1 to represent correlations between various input patterns that map to the same output pattern. Clearly, the option 2 demonstrates a higher optimization potential. It is noteworthy, that for both options the rate of improvement significantly decreases as soon as the value of p_{on} (i.e. the percentage of realized connections in relation to the matrix size) for matrix M2 reaches 50%. This seems to be in accordance with the theoretical results presented in [2] which state that for sparsely coded patterns the BiNAM reaches its storage capacity limit at p_{on}=0.5 (=50%). Clearly, as soon as the full capacity of M2 is used up, further optimizations are unlikely. However, larger values of p_{on} seem to be feasible for M2, cf. Fig.6f. The value of p_{on} for matrix M1 is smaller than 50% which indicates a certain amount of free memory space which might be due to sub-optimal code sets given by C_D with respect to C_R. In particular, the overall comparator count N_{RD}=N_R+N_D appears to be minimal for a specific $N_{D,min}$, cf. Fig 6a. Interestingly, the optimal value for N_{RD} is significantly lower than the size of the input space z^2=64, which indicates that the required overall complexity in terms of word lines is less than that of a plain look-up-table, cf. Table I. The factor ε=N_{RD}/z^2 is significantly smaller than 1.0 in every case.

As a summary, the proposed BiNAM-architecture can be seen as an atomic interconnection scheme to be used in a hierarchy of memory structures that realize complex computational tasks. An overall architectural picture is still to be developed which includes interconnected hierarchies of BiNAMs. Although the underlying connection-centric structures are comparatively simple and modular as well, various degrees of freedom are available to optimize parameters and the performance.

REFERENCES

[1] Waser, R., Dittmann, R., Staikov, G. and Szot, K., “Redox-Based Resistive Switching Memories – Nanoionic Mechanisms, Prospects, and Challenges”, Adv. Mater., 21: 2632–2663, 2009

[2] Palm, G., “on Associative Memory”, Biol. Cybernetics 36, 1980

[3] A. Siemon et al., "In-memory adder functionality in 1S1R arrays," ISCAS, Lisbon, 2015, pp. 1338-1341, 2015

[4] P.-E. Gaillardon et al.:”GMS: Generic memristive structure for non-volatile FPGAs”, VLSI-SoC 2012, pp. 94-98, 2012

[5] A. Velasquez and S. K. Jha, "Automated synthesis of crossbars for nanoscale computing using formal methods," , NANOARCH 2015, pp. 130-136, 2015

[6] Z. Alamgir et al, "Flow-based computing on nanoscale crossbars: Design and implementation of full adders," ISCAS 2016, pp. 1870-1873, 2016

[7] Parag Lala, “Self-checking and Fault-Tolerant Digital Design”, 1st. Edition, Morgan Kaufmann, July 2000

[8] R. Aluguri and T. Y. Tseng, "Overview of Selector Devices for 3-D Stackable Cross Point RRAM Arrays," in IEEE Journal of the Electron Devices Society, vol. 4, no. 5, pp. 294-306, Sept. 2016

[9] A.Heittmann, T.G.Noll, “Mixing Circuit based on Neural Associative Memories and Nanoelectronic 1S1R Cells”, IEEE/ACM Nanoarch’17, DOI 10.1109/NANOARCH.2017.8053707, 2017