November 2015

# Physically Equivalent Intelligent Systems for Reasoning Under Uncertainty at Nanoscale

Santosh Khasanvis
*University of Massachusetts - Amherst*

# PHYSICALLY EQUIVALENT INTELLIGENT SYSTEMS FOR REASONING UNDER UNCERTAINTY AT NANOSCALE

A Dissertation Presented

by

SANTOSH KHASANVIS

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 2015

Department of Electrical and Computer Engineering

**PHYSICALLY EQUIVALENT INTELLIGENT SYSTEMS FOR REASONING UNDER UNCERTAINTY AT NANOSCALE**

A Dissertation Presented

by

SANTOSH KHASANVIS

Approved as to style and content by:

_____
Csaba Andras Moritz, Chair

_____
Israel Koren, Member

_____
C. Mani Krishna, Member

_____
Jayasimha Atulasimha, Member

_____
Christopher V. Hollot, Department Head
Electrical and Computer Engineering

## ACKNOWLEDGEMENTS

I am forever indebted to my advisor Prof. Csaba Andras Moritz, for his constant encouragement, guidance, and mentorship. This dissertation would not have been possible without his direction. I am grateful to my dissertation committee members Prof. Koren, Prof. Krishna and Prof. Atulasimha for their valuable feedback and suggestions. I would like to acknowledge collaborations with the research group directed by Prof. Atulasimha and Prof. Bandyopadhyay at Virginia Commonwealth University, and thank their students Ayan K. Biswas and Mohammad Salehi Fashami for all their hard work in providing us with data for straintronic magnetic tunneling junctions used extensively in this work.

I have benefited immensely from working with several creative, intelligent and dedicated colleagues. Foremost is Mostafizur Rahman, who has been a close friend and an anchor during my PhD. I would like to thank Dr. Pritish Narayanan, who was a mentor during my initial years and continues to be a close friend. I would also like to express my gratitude to Mingyu Li for his assistance in making this dissertation possible. Finally, I would like to express my sincere gratitude to my entire family for their continued love and support through all these years.

I would like to acknowledge support from Center for Hierarchical Manufacturing at UMass Amherst and National Science Foundation grant no. 1407906 at UMass Amherst.

## ABSTRACT

**PHYSICALLY EQUIVALENT INTELLIGENT SYSTEMS FOR REASONING UNDER UNCERTAINTY AT NANOSCALE**

SEPTEMBER 2015

SANTOSH KHASANVIS

B.TECH., VELLORE INSTITUTE OF TECHNOLOGY UNIVERSITY, VELLORE, INDIA

M.S., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Csaba Andras Moritz

Machines today lack the inherent ability to reason and make decisions, or operate in the presence of uncertainty. Machine-learning methods such as Bayesian Networks (BNs) are widely acknowledged for their ability to uncover relationships and generate causal models for complex interactions. However, their massive computational requirement, when implemented on conventional computers, hinders their usefulness in many critical problem areas e.g., genetic basis of diseases, macro finance, text classification, environment monitoring, etc. We propose a new non-von Neumann technology framework purposefully architected across all layers for solving these problems efficiently through *physical equivalence*, enabled by emerging nanotechnology. The architecture builds on a probabilistic information representation and multi-domain mixed-signal circuit style, and is tightly coupled to a nanoscale physical layer that spans magnetic and electrical domains. Based on bottom-up device-circuit-architecture simulations, we show up to four orders of magnitude performance improvement (using computational resolution of 0.1) vs. best-of-breed multi-core machines with 100

processors, for BNs with about a million variables. Smaller problem sizes of ~100 variables can be realized at 20 mW power consumption and very low area around a few tenths of a $mm^2$. Our vision is to enable solving complex Bayesian problems in real time, as well as enable intelligence capabilities at a small scale everywhere, ushering in a new era of machine intelligence.

## TABLE OF CONTENTS

**Page**

## LIST OF TABLES

**Page**

## LIST OF FIGURES

Page

# CHAPTER 1

# INTRODUCTION

Machines today lack intelligence, i.e. the inherent ability to reason, make decisions, adapt, and in general operate autonomously in the presence of uncertainty. Today, all computation occurs on microprocessors based on a stored-program von Neumann computing architecture with CMOS technology. This conventional computing paradigm necessitates human intervention to "a priori tell the machine what it needs to do in a given scenario", i.e. program its behavior deterministically. We refer to such conventional computing machines as *abstraction-based engineered systems*; capable of carrying out any procedure expressed algorithmically and implemented through layers of abstraction, and engineered to perform each operation in a procedure as fast as possible given current technology. This conventional mindset of abstracted systems, driven by a desire for convenience in mapping a wide variety of algorithmically expressible problems and to have a reliable machine operation under pre-determined circumstances, has resulted in many discoveries and deterministic tasks to be automated by machines. However, while computers have evolved into fast number-crunching machines today, they are inefficient for supporting machine intelligence that requires operating under non-deterministic scenarios. Handling any new scenario requires explicit instruction by humans for the machine.

Unconventional computation models that draw inspiration from observations in nature such as probabilistic graphical models, neuromorphic computation, hierarchical temporal models using sparse data representation, etc. require immense computing resources and have orders of magnitude inefficiencies when implemented with

conventional abstraction-based engineered systems. This inefficiency spans all layers from the Boolean data representation, digital CMOS logic to the underlying microarchitecture. This is one key reason why we do not have intelligence in all things surrounding us, and why machines cannot easily handle complex decision-making problems.

### 1.1 Notion of Physical Equivalence

We believe that in order to kick-start this evolution in machines and harness the full benefits of unconventional computing paradigms for artificial intelligence, a change in implementation mindset is necessary. We propose a new mindset of architecting intelligent systems with *physical equivalence*; defined as a direct mapping from concept to physical layer, where physical implementation operates on principles defined by the conceptual framework without any abstraction.

A given computational framework can be characterized by:

- *quantum of information* (or data)
- *interaction* that specifies rules to operate on quantum of information (computation and communication), and
- *organization/architecture*, such as DAG in probabilistic graphical models, that governs the temporal/spatial hierarchy of interactions.

For example, conventional abstraction-based engineered systems operate on symbols represented using binary radix representation (quantum of information), where interactions occur as per rules of binary logic developed through switching theory (interaction), and segregate data storage or memory and computation to enable mapping a wide variety of problems (stored-program von Neumann architecture). In this

dissertation, we illustrate our physical equivalence approach through the example of Bayesian Networks, which is a computational framework using probabilistic graphical models for reasoning and decision making under uncertainty. We identify each of the aforementioned characteristics for Bayesian Networks and attempt to find physically equivalent implementation as close as possible to maximize efficiency. We will show that a physically equivalent implementation (with a resolution of 0.1) of Bayesian Networks can yield up to 4 orders of magnitude performance (runtime) benefits compared to conventional software implementations on state-of-the-art CMOS multicore processors, even when considering best-case performance assumptions for conventional approach vs. worst-case evaluation for our proposed approach.

### 1.2 Conceptual Framework Overview

Bayesian Networks (BNs) [1]-[3] represent a class of widely successful probabilistic formalism capable of modeling causal relationships between random variables in an application domain. A BN can be used for expressing the strength of belief in the state of a system given some observations on its environment. Its structure is a Directed Acyclic Graph (DAG) where every node represents a random variable and every edge is a dependency between nodes. These dependencies are quantified through conditional probabilities (parameters) associated with every node. The belief in the state of a system, specifically the probabilities associated with unobserved variables being in a particular state given the state of observed variables, can be obtained through *inference*. An inference operation is executed following periodic observations on BN variables. Any event (observed variable being assigned a state) triggers the calculation of current belief of a hypothesis, which is an unobservable system variable. Thus Bayesian Networks

operate on probabilities (quantum of information), where interactions occur as per rules of probability arithmetic for inference and learning (interaction), and organize knowledge as DAGs (organization/architecture).

Many problems can be mapped into this formalism. For example, gene expression networks are being studied extensively in order to understand the genetic basis of diseases [4][5]. Unfortunately the resulting networks are generally very complex owing to random variables representing gene-gene and gene-environment interactions. Other complex applications [6]-[9] include text classification, situational awareness for cyber-security, etc.

### 1.3 Limitations of Conventional Implementations

The high computational complexity in BNs is a result of learning from data; number of candidates is super-exponential in the number of variables. Furthermore, incomplete and limited datasets to learn from mandate a large number of inferences, which further complicates the choice of a candidate network. Additionally, cost and power efficiency aspects make adding BN capabilities impossible in embedded systems. While software implementations representing BNs are highly flexible, several limitations crop up as a consequence of all the layers of abstractions. The underlying conventional von Neumann architectures built with CMOS technology are not well suited to implementing such computational frameworks because:

(i) their emulation of an inherently non-deterministic, non-logical computing model on a deterministic Boolean logic framework is inefficient,

(ii) BN's structure and parameter learning is super-exponential in the number of variables,

(iii) conventional architectures incorporate a limited number of multiplication and division units (due to high complexity of CMOS logic implementation of multipliers and dividers),

(iv) the use of a rigid separation between logic and memory is undesirable, and

(v) the use of a radix-based representation of data is inefficient for probabilistic information and incapable of inherently supporting graceful degradation in the presence of errors.

### 1.4 Proposed Approach to Overcome Deficiencies in Conventional Implementation

Our objective is to architect an efficient machine implementation for causal learning and reasoning framework, given recent developments in nanotechnology. Therefore, our goal is to identify representations across all layers that result in *physical equivalence* with the conceptual probabilistic framework. This mindset, extending from the physical layer to architecture, can potentially address causal inference and learning problems that are computationally infeasible today, and enable such capability at smaller scale in everyday embedded systems. In this dissertation, we design a physically equivalent hardware architecture and nanoscale technology implementation of BNs based on unique magneto-electric computations that can efficiently address the aforementioned problems, as an illustration of the physical equivalence mindset. It can be extended and applied to other unconventional computation frameworks as well. For physical equivalence at all layers, we explore a technology implementation that operates directly on probabilities (quantum of information) through probability arithmetic without Boolean logic (interactions), and physically realizes a reconfigurable DAG where each node has an equivalent physical entity and communication links representing edges (organization).

At the bottom of the system stack, we use multiferroic straintronic magnetic-tunneling junctions (S-MTJs) consisting of a single-domain magnetostrictive layer with uniaxial shape anisotropy elastically coupled with a piezoelectric layer. A tiny voltage of 10 – 60 mV applied across the piezoelectric can flip the magnetization in ~1 ns. This is achieved with unprecedented energy-efficiency dissipating only 150 – 200 kT at room temperature [10]-[14] (three to four orders of magnitude reduction in energy dissipated to switch compared to state-of-the-art nanoscale transistors at 1 GHz clock speed). By appropriately "shaping" the voltage pulse, the switching error probability in S-MTJs can be reduced to $\sim 10^{-6}$ in the presence of thermal fluctuations at room temperature. In addition, these S-MTJs support non-volatility where the resistance change is persistent, which is unique.

The above-mentioned characteristics of the emerging S-MTJ devices present an opportunity for novel physically equivalent technology frameworks that is not supported by conventional CMOS technology. In this work, we leverage the physical domains that such non-volatile voltage-controlled S-MTJ devices span, for compact and efficient realization of magneto-electric computations with probabilities. They are also capable of sporting multiple magnetization states, which can enable new multi-valued redundant representation of information directly in the physical domain. In this work, we focus on two-state S-MTJs. The synergistic non-Boolean circuit style that we present is non-volatile (enabling no segregation between memory and computation), multi-domain (spanning electrical and magnetic), and mixed-signal (with emphasis on analog for computation without emulation). This leads to circuits that are *self-similar* like fractals when hierarchically composed. Bayesian structure and parameter learning, inference and

adaptation can be supported in a programmable parallel architecture framework that enables for direct mapping and adaptation of BNs.

Key contributions of this dissertation include:

(i) The idea of physical equivalence for a nanotechnology framework to realize unconventional computing models for causal inference and learning problems, using Bayesian Networks as an example.

(ii) A data representation for probabilities that has physical equivalence in electrical/magnetic domains and supports graceful degradation in the presence of faults.

(iii) A new physically equivalent multi-domain mixed-signal Probability Arithmetic Composer circuit framework for computation on probabilities, which supports memory-in-computing through the use of non-volatile devices (S-MTJs).

(iv) A reconfigurable parallel architecture based on distributed Bayesian Cell framework for implementing any desired Bayesian Network with physical equivalence.

(v) Methodology to estimate runtime performance of Bayesian inference when implemented on multi-core processors (up to 100 cores) and comparison with the proposed physically equivalent system.

(vi) A study on the propagation of errors in an example binary tree Bayesian Network due to limited numerical precision (rounding) and impact of probabilistic switching of S-MTJs on BN accuracy.

(vii) Initial projections on impact of improving computational resolution for the proposed framework.

The rest of this dissertation is organized as follows: Chapter 2 presents a brief background on Bayesian Networks. Chapter 3 discusses the S-MTJ device, the proposed data representation and Probability Composer framework for implementing Bayesian Network operations. Chapter 4 presents a reconfigurable architecture that allows implementing any Bayesian Network with the proposed hardware implementation. Chapter 5 presents the evaluation methodologies, benchmarking results against a 100-core processor implementation, and studies on error propagation in example BN. Chapter 6 concludes this dissertation.

# CHAPTER 2

## BAYESIAN NETWORKS

Bayesian Networks (BN) are probabilistic graphical models [1][2] representing uncertain domains. A BN's *structure* (e.g., a tree) captures qualitative relationships between variables. This is attractive because it is a consistent and complete representation, in addition to being modular and compact. A typical BN is a directed acyclic graph, with individual nodes representing knowledge about variables in a system. Dependencies between the variables are represented as directed links between the nodes. A node is a parent of a child if there exists a directed link from former to the latter. A

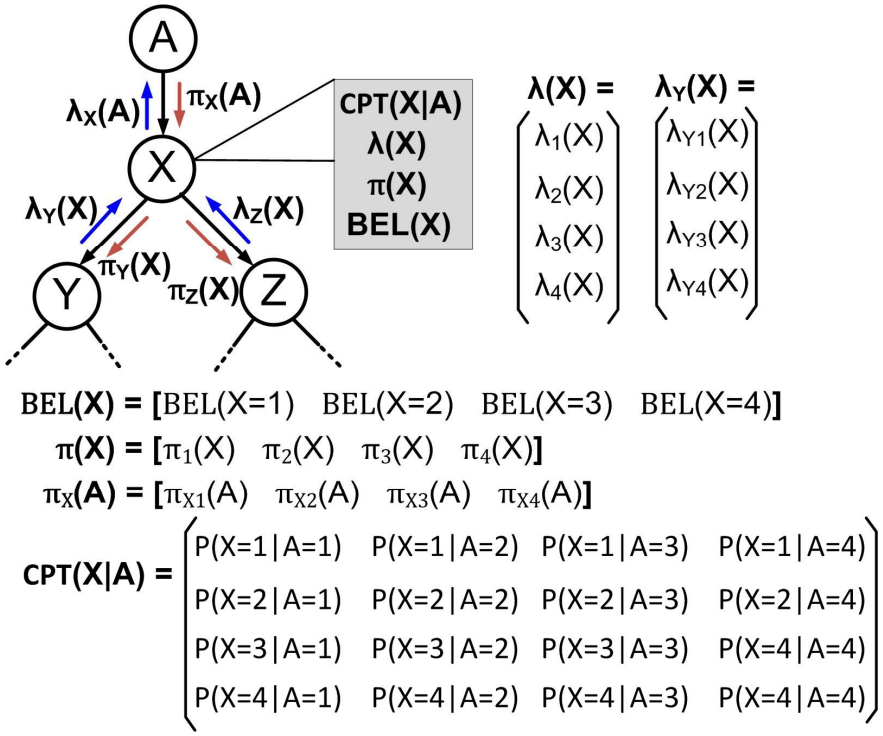



Figure 1. Part of a BN with showing node x whose child nodes are y, z and parent node is A. Outcomes of states of child nodes determine likelihood of parent. All nodes have four states in this example. Each node maintains likelihood vector (λ), prior vector (π), belief vector (BEL), and conditional probability table (CPT). The CPT information and messages from child/parent nodes are used to calculate λ, π, and BEL vectors during Bayesian inference.

node without parents is called a root node, while a node without children is called a leaf node. Each node can have several states for its corresponding variable, and a conditional probability table (CPT) stores conditional probabilities that quantify the relationship with its parents. These CPTs are the *parameters* of a Bayesian Network. The structure and parameters associated with a BN encode a joint probability distribution for all the domain variables in an efficient manner. A part of a typical BN is shown in Figure 1 with one parent node *x* and two child nodes *y, z*.

Absence of a link between a pair of variables implies conditional independency between the variables, given other intermediate variables. Due to this independence property, the joint probability distribution can be factorized into local conditional probability distributions of variables given their parent variables, using the chain rule as follows:

$$p(\boldsymbol{x}|\boldsymbol{\theta}) = \prod_{i=1}^{n} p(x_i|pa(x_i), \theta_i), \tag{1}$$

where $\boldsymbol{x} = \{x_1, x_2, \ldots, x_n\}$ are the variables or nodes in the BN and $\boldsymbol{\theta} = \{\theta_1, \theta_2, \ldots, \theta_n\}$ are the associated parameters. These parameters are CPTs, where each element in a CPT for a given node $x_i$ holds the following data:

$$CPT_{jk}(x_i) = \theta_{ijk} = p(x_i = j|pa(x_i) = k). \tag{2}$$

The factorization shown in eq. (1) reduces the number of parameters required to specify a full joint probability distribution dramatically.

When constructing a BN for a specific application, hypotheses can be expressed as BN variables and a unique probability is assigned to each hypothesis initially (e.g., based

on prior knowledge of the domain from an expert). Alternatively, the BN structure and parameters can be learned from available data on the domain, without explicit elicitation from a domain expert. Given a parameterized BN structure, an inference process requires computation of probability of a hypothesis based on current events observed (state of observed variables) and corresponding conditional probability distributions. Several algorithms exist to perform inference (both exact and approximate) and each algorithm has certain restrictions or trade-offs. We look at one algorithm to illustrate our mindset, which was proposed by Judea Pearl who invented the BN framework. This algorithm, called Belief Propagation [1], is applicable to trees and poly trees, which do not include any loops.

A belief is the probability of an unobserved variable given other observed variables and the BN. Inference is performed via belief update and message propagation through the network. The key operations in this algorithm are likelihood/prior estimation to generate these messages, belief update and diagnostic/prior support message generation. Each of these operations involves arithmetic on probabilities.

### 2.1 BN Inference using Pearl's Belief Propagation

Inference in a BN requires belief updates at all unobserved nodes based on current events observed (evidence), and is performed via message propagation (likelihoods $\lambda$ and priors $\pi$ [1] which are essentially probabilities) in the network. Belief update refers to estimating the probability that a node is in a particular state based on the states of its children/parents and current observations. The key operations at each node during inference are likelihood/prior estimation to generate messages and belief update. For

example assuming every node has four states in Figure 1, likelihood messages ($\boldsymbol{\lambda_Y(X)}, \boldsymbol{\lambda_Z(X)}$) from the child nodes are composed at node X to calculate the likelihood vector **λ(X)** as shown in eq. (3). Here symbols in bold type indicate vectors/matrices, asterisk symbol (*) represents element-wise multiplication between vectors, and ⊗ operator indicates vector/matrix multiplication.

$$\boldsymbol{\lambda(X)} = (\lambda_1(X), \lambda_2(X), \lambda_3(X), \lambda_4(X)) = \boldsymbol{\lambda_Y(X)} * \boldsymbol{\lambda_Z(X)} \quad (3)$$

where $\lambda_i(X) = \lambda_{Yi}(X) \cdot \lambda_{Zi}(X); i = \{1,2,3,4\}$.

With the likelihood vector being computed, the node then generates messages to send to its parent node as follows:

$$\boldsymbol{\lambda_X}\big(\boldsymbol{Pa(X)}\big) = \Big(\lambda_{X1}\big(Pa(X)\big), \lambda_{X2}\big(Pa(X)\big), \lambda_{X3}\big(Pa(X)\big), \lambda_{X4}\big(Pa(X)\big)\Big) = \boldsymbol{CPT}\big(\boldsymbol{X}\big|\boldsymbol{Pa(X)}\big) \otimes \boldsymbol{\lambda(X)}. \quad (4)$$

Priors computation **π(X)** is performed at the node X based on prior support messages $\boldsymbol{\pi_X(Pa(X))}$ received from its parent as follows:

$$\boldsymbol{\pi(X)} = \big(\pi_1(X), \pi_2(X), \pi_3(X), \pi_4(X)\big) = \boldsymbol{\pi_X(Pa(X))} \otimes \boldsymbol{CPT}\big(\boldsymbol{X}\big|\boldsymbol{Pa(X)}\big). \quad (5)$$

The current belief at node X **BEL(X)** is updated as follows using computed likelihood **λ(X)** and prior **π(X)** vectors:

$$\boldsymbol{BEL(X)} = \alpha\boldsymbol{\pi(X)} * \boldsymbol{\lambda(X)} \quad (6)$$

where $BEL_i(X) = \alpha\pi_i(X)\lambda_i(X), \; ; i = \{1,2,3,4\}$.

Finally, the prior support messages to be sent to its child nodes is computed as follows:

$$\boldsymbol{\pi_Y(X)} = \big(\pi_{Y1}(X), \pi_{Y2}(X), \pi_{Y3}(X), \pi_{Y4}(X)\big) = \frac{\boldsymbol{BEL(X)}}{\boldsymbol{\lambda_Y(X)}}, and \quad (7)$$

$$\boldsymbol{\pi_Z(X)} = \left(\pi_{Z1}(X), \pi_{Z2}(X), \pi_{Z3}(X), \pi_{Z4}(X)\right) = \frac{\boldsymbol{BEL(X)}}{\boldsymbol{\lambda_Z(X)}}.$$

### 2.2 BN Learning

Search-and-score technique [3] is one of the methods used for learning a BN structure and parameters (CPTs) from observed data, even if the dataset is incomplete or has missing values. Since the search space for all possible graphs is super-exponential in the number of variables it is typically narrowed down by using heuristic techniques. Hill-Climbing (HC) algorithm is a typical heuristic approach used where a given structure is perturbed (by adding, removing or reversing edges) and a score is assigned to

**Hill-Climbing Algorithm:**

E ← ϕ ; Start with either null set or random network
T ← EM_Probability_Tables(E,D)
B← $\langle U, E, T \rangle$
*Score* ← -∞
**Do:** *Maxscore* ← *Score*
**For** each node pair (X,Y) **do**
  **For** each E'∈{E∪(X→Y), E–(X→Y), E–(X→Y)∪(Y→X)},
  T' ← EM_Probability_Tables(E',D)
  B'← $\langle U, E', T' \rangle$
  *Newscore* ← AIC(B',D)
    **If** *Newscore*> Score **then**
      B ← B'
      *Score* ← *Newscore*
**While***Score*>*Maxscore*
**Return** B

Figure 2. Pseudo-code for BN Structure Search with Hill-Climbing Algorithm.

**Expectation-Maximization Algorithm:**

**Initialize** $T^0$
**For** t=0 until termination
  E-step:
    Compute $P(X_i, pa(X_i)|D_l, T^t)$ for all $X_i$ and $D_l$
    Compute the sufficient statistics, for all i, j, k

$$m_{ijk}^t = \sum_l P(X_i = j, pa(X_i) = k|D_l, T^t)$$

  M-step:
    Compute $T^{t+1} = \frac{m_{ijk}^t}{\sum_j m_{ijk}^t}$ for all i, j, k
  **Return** $T^{t+1}$

Figure 3. Pseudo-code for BN CPT Estimation with Expectation-Maximization Algorithm.

the new structure (Figure 2). Different scoring metrics (e.g. AIC scoring) are available to determine the quality of the current BN with respect to observed data. At the end of the process, the graph with maximum score is selected.

As a part of the learning algorithm, the conditional probability tables (CPTs) also need to be estimated from data. For the general case of incomplete data, Estimation-Maximization (EM) algorithm is used to learn the CPTs [3] (Figure 3). EM involves performing iterative inference operations, and computation on conditional probabilities to estimate CPTs until convergence.

### 2.3 BN Adaptation

A BN will need to be able to adapt by reinforcing its parameters (CPTs) based on winning hypothesis at the root node. Reinforcement of a single hypothesis (that wins) can be performed by adjusting the corresponding row of the CPT at each child node slightly in the direction of the likelihood (λ) at that node for current observation. One possible scheme to compute new CPT values (for a child node *j*) is to use the count of number of times a hypothesis (node *i*) was observed as a past weight for the adjustment [7], using eq. (8). This update is performed for every element *j* in the row *i* of the CPT.

$$CPT_{ij}^{new} = \frac{count_i . CPT_{ij} + \lambda_j}{\sum_j \left( count_i . CPT_{ij} + \lambda_j \right)} \tag{8}$$

### 2.4 Summary

In this chapter, we presented a brief overview of the Bayesian Network formalism for representing knowledge, and the operations involved in inference and learning. The next

chapter presents our approach towards a physically equivalent implementation of Bayesian Networks using emerging nanotechnology.

# CHAPTER 3

# PHYSICALLY EQUIVALENT DATA REPRESENTATION AND PROBABILITY COMPOSER CIRCUIT FRAMEWORK

Our objective is to architect an efficient machine for the Bayesian Network framework. Therefore, our goal is to identify representations resulting in physical equivalency with the conceptual probabilistic framework, across all layers.

## 3.1 Data Representation

The first critical element in our approach is the underlying data representation. Since Bayesian Networks (BNs) operate on probabilities, we represent probability as a non-Boolean multi-valued flat probabilistic vector tightly tied to the physical layer. We define $n$ spatially distributed digits ($p_1$, $p_2$, … , $p_n$) such that each digit $p_i$ can take any one of $k$ values, where $k$ is the number states supported by the underlying physical device (e.g., for devices with 4-states, $k = 4$ and a given digit $p_i \in \{0,1,2,3\}$). As opposed to conventional number systems (e.g. binary, HEX etc.), in this representation *all digits carry equal weight irrespective of position*, which implies inherent redundancy and better error resilience through graceful degradation. The probability value $\boldsymbol{P}$, the basis for our architecture and the inspiration for the physical implementation, represented by an $n$-digit probability vector is given by:

$$\boldsymbol{P} = \frac{\sum_{i=1}^{n} p_i}{n(k-1)}. \tag{9}$$

In this representation, the *resolution* is defined as the unit probability at output that can be represented in this format. It is determined by number of digits $n$ and the number

of states of each digit $k$, and is given by $1/[n(k-1)]$. A higher resolution can be achieved either by having more states per device ($k$) or by increasing the number of digits ($n$). Here it is to be noted that precision in BNs has a different interpretation: it is the precision of learning and expressing the problem through supporting a large number of variables and relationships, rather than numerical precision alone. This representation also yields fault resilience supporting graceful degradation in case of faults.

### 3.2 Technology Overview: Straintronic MTJs

In this work, we use straintronic MTJs (S-MTJs) as the underlying physical technology for hardware implementation. But the proposed scheme may be implemented with any emerging non-volatile device for physical equivalence.

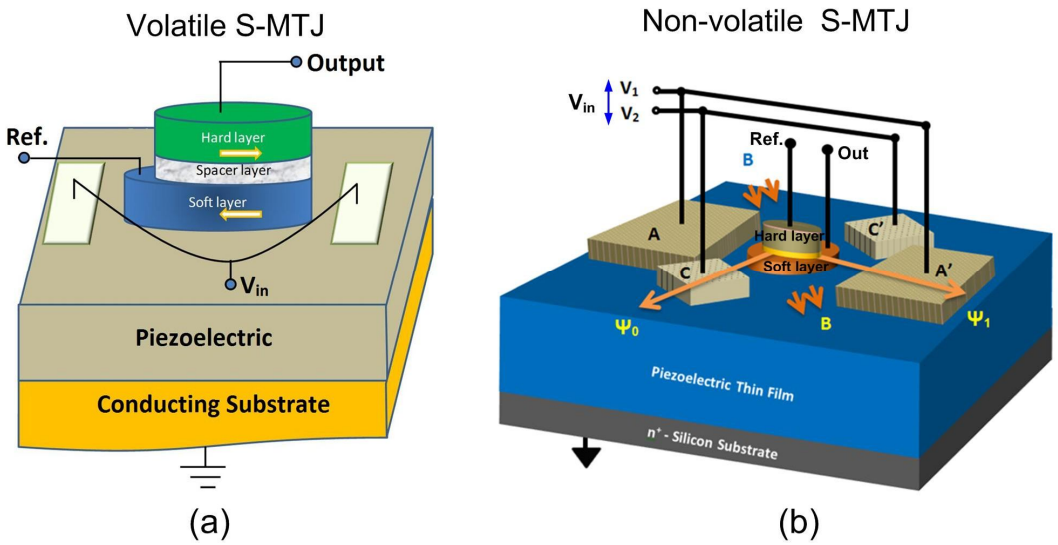



Figure 4. (a) Volatile S-MTJ device configuration: Voltage input induces strain in soft-layer layer adjusting magnetization orientation; a reference terminal (Ref.) is used for resistance readout; and (b) Non-volatile S-MTJ device: The MTJ stack is placed in between two pairs of electrode pads such that the line joining each electrodes subtends an angle of $15^0$ and $165^0$ respectively with the major axis of soft magnetic layer. A magnetic field $\boldsymbol{B}$ is applied along the minor axis of the soft magnetic layer. Voltage input persistently changes magnetization.

The concept of straintronics, where the bistable magnetization of a shape anisotropic multiferroic nanomagnet is switched with electrically generated mechanical strain, is attractive due to its extreme low energy of switching. A straintronic MTJ (S-MTJ) device is shown in Figure 4a. It consists of three layers - a "hard" ferromagnetic layer with a fixed magnetization orientation, an ultrathin spacer layer, and a "soft" ferromagnetic layer with variable magnetization orientation. The three-layered stack is fabricated on a thin piezoelectric film grown on an $n^+$-Si substrate.

Because of dipole coupling between the hard and soft layers, they tend to have mutually anti-parallel magnetizations (see Figure 4a) and in that configuration, the resistance of the S-MTJ measured between the two ferromagnetic layers is high. Application of an input voltage ($V_{in}$) at the two (shorted) contact pads generates a biaxial strain in the piezoelectric layer underneath the soft magnet (compression along the major axis of the elliptical soft magnet and tension along the minor axis) [22][23], which rotates the magnetization of the soft magnet by an angle $\Theta$ via the Villari effect, if the soft layer is magnetostrictive and has positive magnetostriction. This reduces the angular separation between the magnetization orientations of the hard and soft layers, which in turn reduces the resistance of the S-MTJ. If the input voltage is withdrawn, the stress in the soft magnetic layer relaxes and hence its magnetization will tend to return to its original orientation because of dipole coupling with the hard magnetic layer. In this case, the operation is volatile. The resistance ratio between the high- and low-resistance states as a function of applied voltage v is roughly given by [24],

$$r(v) = \frac{R_{ON}}{R_{OFF}} = \frac{R(v=V_{ON})}{R(v=0)} = \frac{1-\eta_1\eta_2}{1-\eta_1\eta_2.\cos\left[\Theta(V_{ON})\right]}, \quad (10)$$

where $\Theta(V_{ON})$ is the angle by which the magnetization of the soft layer rotates under stress generated by input voltage $V_{ON}$, assuming it starts from being exactly anti-parallel to the hard layer initially, and $\eta_1$, $\eta_2$ are the spin-injection/filtering efficiencies at the interfaces between the two ferromagnets and the spacer layer. At room temperature, these quantities are roughly 70% [25]. The maximum value of $\Theta$ is $90^0$ unless the input voltage pulse is timed in a certain way to allow reorientation by $180^0$ [26].

The magnetization rotation can be made persistent through a scheme shown in Figure 4b, resulting in non-volatile operation. The electrodes A – A' are shorted to form one input terminal, and C – C' are shorted to form the second terminal. When a voltage is applied between these terminals and the n+-substrate, electric fields are generated underneath the pads, producing a highly localized strain field in the piezoelectric film [22][23]. This results in biaxial strain (compression/tension along the line joining the electrodes and tension/compression along the perpendicular direction) since the distance between the electrode pairs is approximately equal to the PZT film thickness. This strain will then be elastically transferred to the soft layer of the S-MTJ stack despite any substrate clamping. The scheme requires a small in-plane external magnetic field (**B**) along the minor axis of the soft magnet which brings the two stable magnetization states out of the soft magnet's major axis (easy axis) and aligns them along two in-plane directions that lie between the major and minor axes with an angular separation of $\sim 132^0$. These two stable orientations ($\Psi_1$ and $\Psi_0$) of magnetization represent the low and high resistance states, respectively. The magnetization of the hard magnetic layer is parallel to $\Psi_1$, which is why the low resistance state is visited when the magnetization of the soft

magnetic layer is along $\Psi_1$. Since Terfenol-D has a positive magnetostriction coefficient, compressive stress along the line joining the electrodes A–A' will stabilize the magnetization at $\Psi_0$, while a compressive stress along C–C' electrodes will switch the magnetization back to $\Psi_1$ [30]. These magnetization orientations are stable, i.e. if the magnetization is left in either state it remains there in perpetuity even after power is switched off, which makes the device non-volatile. The change in resistance of the S-MTJ is read by using a reference voltage, which generates an output current. Thus, conversion between voltage, magnetic and current domains is achieved.

The transfer characteristics of the S-MTJ devices (Figure 5b-c and Figure 5e-f) were extracted from stochastic Landau-Lifshitz-Gilbert (LLG) simulations performed at Virginia Commonwealth University by the research group headed by Prof. Supriyo Bandyopadhyay and Prof. Jayasimha Atulasimha, and are described in refs. [14] [27]-[30]. For the volatile S-MTJ transfer characteristics, a soft layer made of Terfenol-D with dimensions 120nm x 105nm x 6.5nm was used, and 110nm x 90nm x 9 nm for non-volatile S-MTJ. The piezoelectric layer was assumed to be lead-zirconate-titanate (PZT) of thickness 100nm. The effect of room-temperature thermal noise was taken into account [14] [27]-[30] and the characteristics presented were thermally averaged characteristics. Furthermore, although the strain generated in the magnet was biaxial, it was approximated with uniaxial strain (which overestimated the voltage needed to generate a given strain). This was somewhat compensated by the fact that 100% strain transfer from the piezoelectric film to the magnetostrictive layer was assumed, leading to an underestimation of the voltage needed to generate a given strain. Every data-point in Figure 5b,e was generated by averaging 10,000 simulations. The LLG simulations also

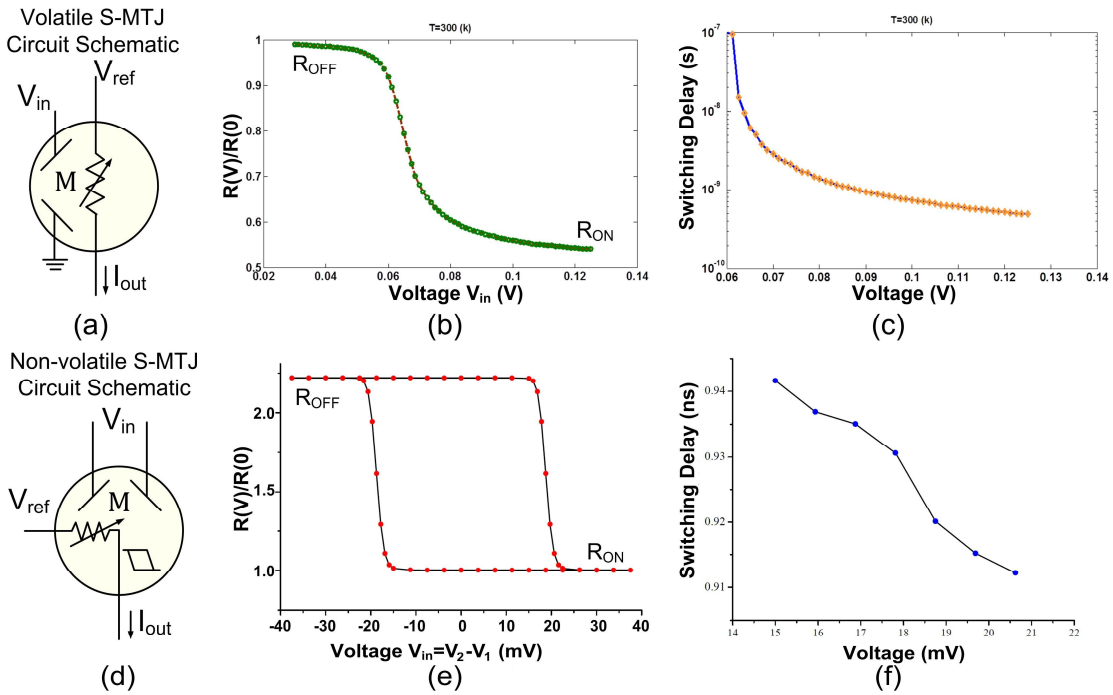



Figure 5. (a) Volatile S-MTJ circuit schematic; (b) Simulated DC transfer characteristics for volatile S-MTJ showing resistance ratio *r(v)*, as function of input voltage $V_{in}$; (c) Simulated switching delay characteristics for volatile S-MTJ; (d) Non-volatile S-MTJ circuit schematic; (e) Simulated DC transfer characteristics for non-volatile S-MTJ showing resistance ratio *r(v)*, as function of input voltage $V_{in}$. Hysteresis indicates persistence in resistance state; and (c) Simulated switching delay characteristics for non-volatile S-MTJ.

yielded the switching time needed for Θ(v) to stabilize to its final value after input voltage is abruptly switched on, shown in Figure 5c,f.

The S-MTJ device can have a number of stable states depending on the cross-sectional shape of the magnet. For example, if the cross-section is an ellipse the magnetization has two stable states. If left in one of those states, the magnetization will remain there indefinitely, making the switch non-volatile. If the cross-section is a different shape, the number of states can be increased. By orienting the hard magnet in a suitable direction, the resistance of the S-MTJ can be made to have as many states as the magnet's orientation. The number of states can be increased further by employing other

polygonal cross-sections, but not indefinitely since increasing them reduces the energy barrier between neighboring states, resulting in spontaneous switching and error.

### 3.3 Probability Representation using Straintronic MTJs

Each of the digits in a probability vector is encoded in the resistance state of a non-volatile S-MTJ (see Figure 6). For example, in the case where S-MTJs are binary with two stable magnetic orientations, the state that leads to a high resistance ($R_{OFF}$) is used to encode probability digit 0, and low resistance ($R_{ON}$) encodes probability digit 1. The probabilistic information from magnetization (and thus resistance) domain is converted to a condensed equivalent representation in the current/voltage domain (Figure 6b) through the S-MTJs for computation. We use an inverse-linear relationship between resistance ($r_i$) and the probability digit ($p_i$) being represented as shown in eq. (11).

$$r_i = \frac{\beta}{(p_i + \varepsilon)}. \tag{11}$$

Here, β and ε are constants chosen such that the above relationship holds. For binary

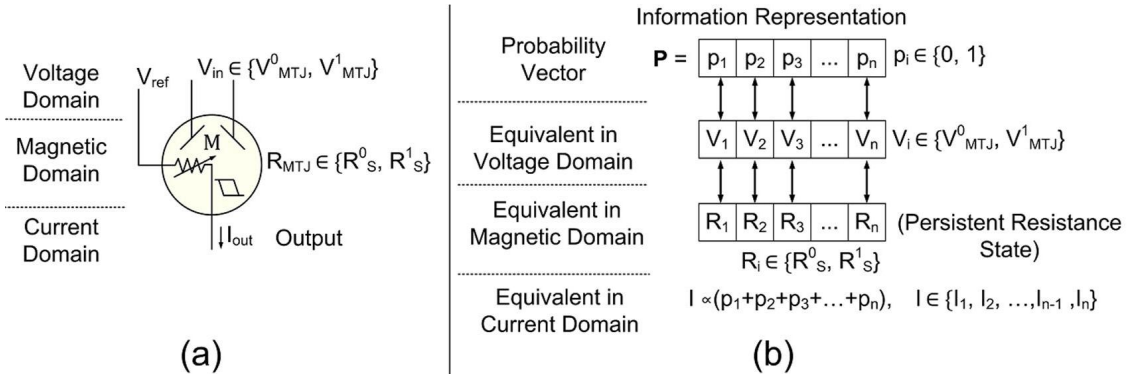



Figure 6. (a) Non-volatile S-MTJ circuit schematic with 2 states showing multi-domain representation. $V_{in}$ switches S-MTJ resistance through change in magnetization and $V_{ref}$ is used during readout; and (b) Spatial probabilistic information representation with S-MTJ with 2 states, and its physical equivalent in resistance, voltage and current domains.

devices with two resistance states ($r_i = R_{OFF}$ corresponding to $p_i = 0$ and $r_i = R_{ON}$ corresponding to $p_i = 1$), by substituting the corresponding $r_i$ and $p_i$ values we get

$$\varepsilon = \frac{1}{\left(\frac{R_{OFF}}{R_{ON}} - 1\right)}; \text{ and}$$
$$\beta = \varepsilon . R_{OFF} = \frac{R_{OFF}}{\left(\frac{R_{OFF}}{R_{ON}} - 1\right)}. \quad (12)$$

Alternative representations may also be used where the resistance is linear with respect to the probability digit. Such alternatives will require changes to the circuit implementations accordingly.

### 3.4 Resolution Scaling with Probability Composer

A single S-MTJ with 2 states is very limited since it can only express probability 0 or 1. In order to increase the resolution, we use a parallel configuration of several S-MTJs to be able to express other probability values between 0 and 1 (see Figure 7a). We call this topology as Probability Composer (for scaling resolution), which accepts inputs represented in probability vector format of *n*-digits. The effective resistance ($R_{PC}$) has $n$+1 discrete states, given by the following expression (see Figure 7b):

$$\frac{1}{R_{PC}} = \sum_{i=1}^{n} \frac{1}{r_i} = \sum_{i=1}^{n} \frac{(p_i + \varepsilon)}{\beta} = \frac{1}{\beta}\left[\sum_{i=1}^{n} p_i\right] + \left\{\frac{n\varepsilon}{\beta}\right\}. \quad (13)$$

In general, if each device in this Probability Composer topology has *k* states, then the effective resistance of the circuit has $n(k-1)+1$ distinct states with a resolution of $1/[n(k-1)]$. By using a common reference voltage, the probability digits represented by S-MTJ resistance are added up in the Probability Composer via the electrical current flowing

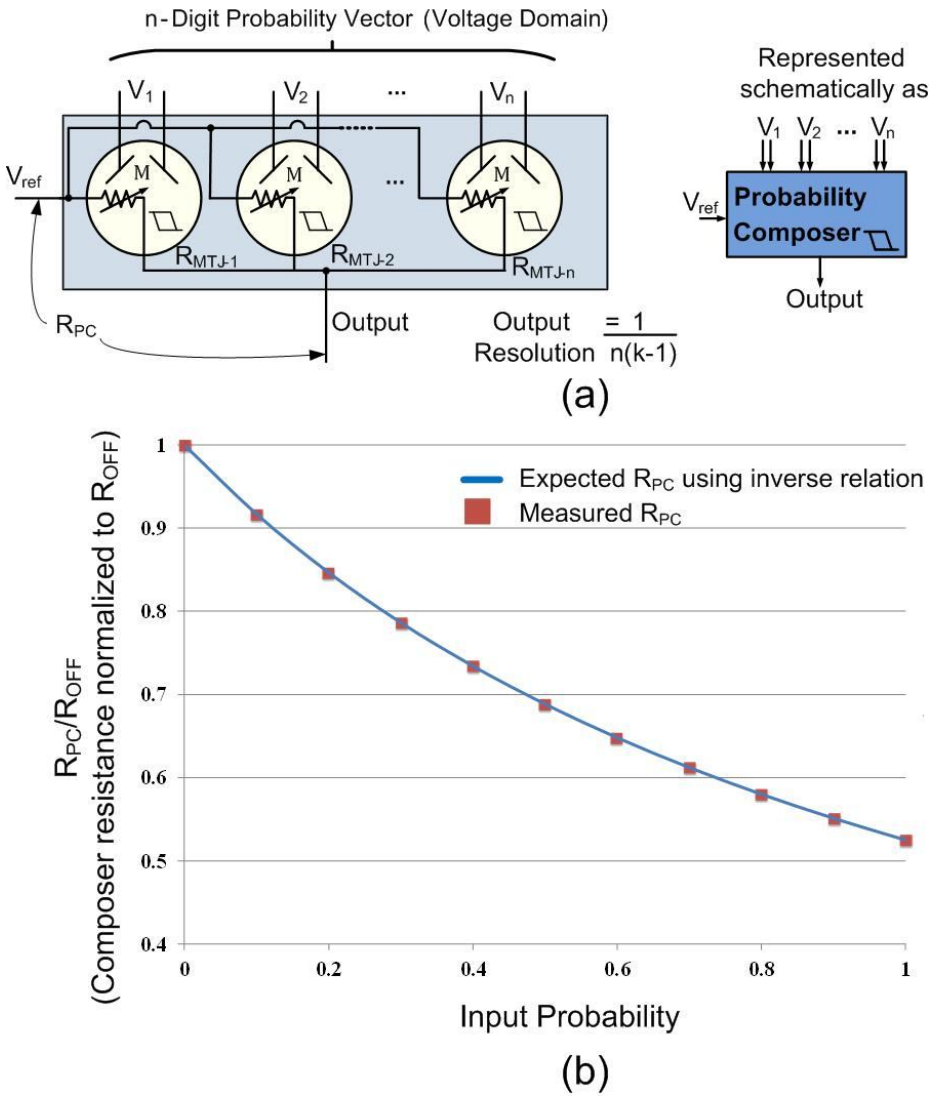



Figure 7. (a) Circuit and schematic representation of Probability Composer element using S-MTJs to increase output resolution; and (b) The effective resistance vs. input probability value (represented using probability digits and stored in each S-MTJ resistance state) of the Probability Composer normalized to its OFF state resistance.

through each device. Thus, the Probability Composer essentially converts the discrete probability vector to a compressed form in analog electric domain.

When using a load resistance $R_L$ much smaller than the S-MTJ resistance connected between the output terminal of Probability Composer and ground, the output current flowing through this load resistor is given by:

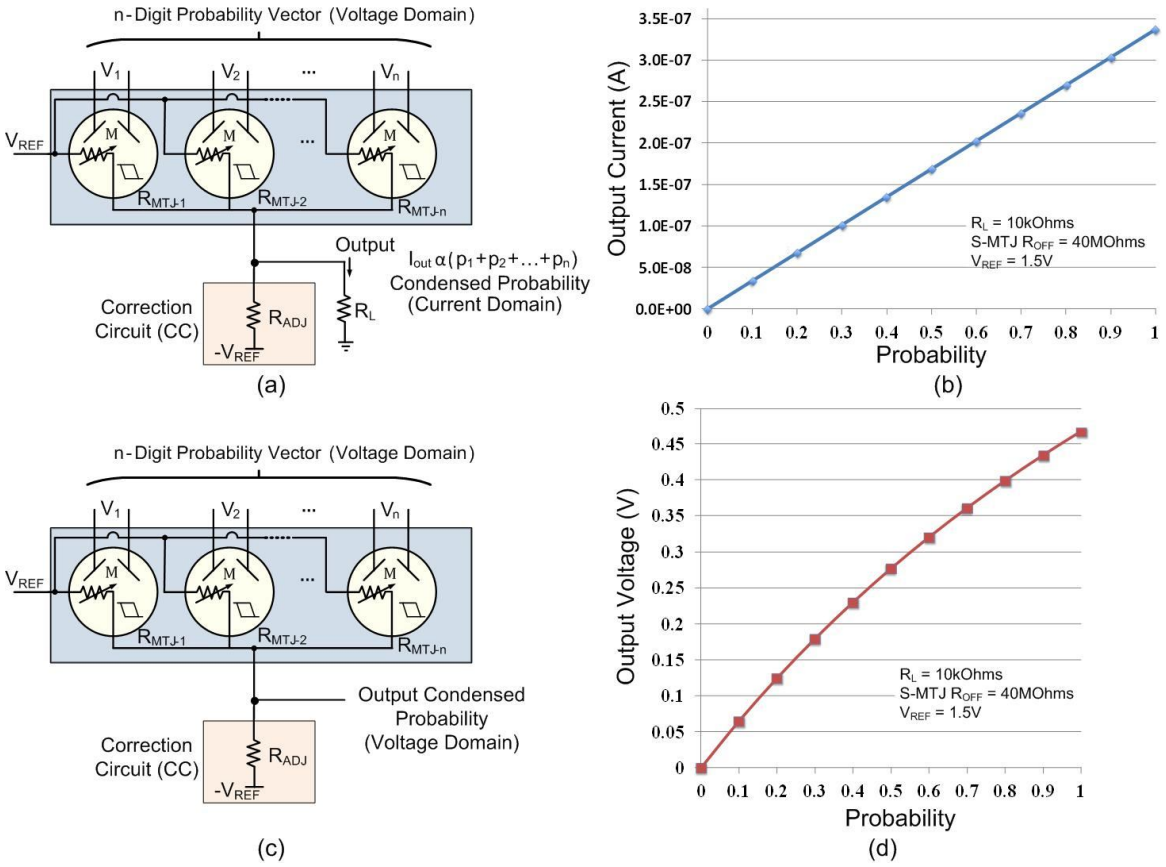



Figure 8. Read-out schemes for Probability Composer Element. (a) Current read-out with corresponding output values shown in (b); and (c) Voltage read-out with corresponding values shown in (d).

$$I_{out} = \frac{V_{REF}}{(R_{PC} + R_L)} \approx \frac{V_{REF}}{R_{PC}} = \frac{V_{REF}}{\beta}\left[\sum_{i=1}^{n} p_i\right] + \left\{\frac{n\varepsilon V_{REF}}{\beta}\right\}. \qquad (14)$$

The term in {.} represents the additional current that needs to be corrected for output linearity. This can be done with a Compensation Circuit (see Figure 8a), such that the output current is given by:

$$I_{out} \approx \frac{V_{REF}}{\beta}\left[\sum_{i=1}^{n} p_i\right] + \left\{\frac{n\varepsilon V_{REF}}{\beta}\right\} + \frac{V_{ADJ}}{R_{ADJ}} = \frac{V_{REF}}{\beta}\left[\sum_{i=1}^{n} p_i\right] = \frac{nV_{REF}\boldsymbol{P}}{\beta}. \qquad (15)$$

Here, $V_{ADJ} = -V_{REF}$, $R_{ADJ} = \beta/(n\varepsilon)$ and $\boldsymbol{P}$ is the probability value represented by the digital probability vector as defined in eq. (1). Thus for every probability value there is a corresponding current domain output.

However, we are interested in a voltage output since S-MTJs are voltage-controlled. The current signal can be converted to analog voltage domain by using the resultant voltage across the load resistance, given by $V_{out} = I_{out}.R_L = \frac{V_{REF}.R_L.\boldsymbol{P}}{\beta}$. However, since the value of $R_L$ has to be necessarily low relative to S-MTJ resistance for the approximation in eq. (13), the range of output voltages using this scheme needs amplification. But, if the output voltage non-linearity can be tolerated while read-out, then the analog voltage output with a larger range can be obtained by simply eliminating the load resistance $R_L$ (see Figure 8c). The output voltage is given by the following expression:

$$V_{out} = V_{REF}.\left[\frac{\frac{1}{R_{PC}} - \frac{1}{R_{ADJ}}}{\frac{1}{R_{PC}} + \frac{1}{R_{ADJ}}}\right] = V_{REF}.\left[\frac{\sum_{i=1}^{n} p_i}{\sum_{i=1}^{n} p_i + 2n\varepsilon}\right] = V_{REF}.\left[\frac{\boldsymbol{P}}{\boldsymbol{P} + 2\varepsilon}\right]. \qquad (16)$$

Here $\boldsymbol{P}$ is the probability value represented by the digital probability vector, defined in eq. (9). This topology results in a non-linearity in the output; for probability close to 0 the output voltage is proportional to sum of individual probability digits, but degrades for probability close to 1. As long as different output levels can be differentiated, the above topology may be used. This represents a trade-off between using sub-threshold CMOS analog support circuits for amplifying the low output voltage range exhibiting linearity as in the first case, vs. tolerating non-linearity in output for wider voltage range with a potentially simpler circuit implementation.

### 3.5 Decomposer Element

We need a way to convert the analog voltage output back to a digital probability vector representation. To achieve this we design a Decomposer circuit with volatile S-MTJs as follows. The Decomposer has the following requirements:

i) For converting to an *n*-digit probability vector, it requires *n* decomposer elements; each decomposer element is designed to trigger at a different input voltage value, i.e. they have different threshold voltages.

ii) When triggered, each decomposer element needs to generate a pair of differential output voltage signals, so as to switch a non-volatile S-MTJ in the successive stage.

Drawing inspiration from flash analog-to-digital converters, we use a resistive ladder (tuned for low-power operation since it does not contribute directly to critical path delay after startup) to setup varying threshold voltages for each decomposer element. Alternatively, the S-MTJ device may be designed to have varying thresholds by changing the device parameters (such as PZT thickness, etc.). Here, the volatile S-MTJs in each decomposer element act as a voltage comparator; if the input voltage is below the reference voltage (setup with the resistance ladder) the S-MTJ switches its resistance state, else it remains in its previous state. To generate differential voltage output when triggered, each decomposer element consists of two branches, one with S-MTJ in pull-up

Table 1. Decomposer Element Operation

| Operating Condition | S-MTJ Resistance | Voltage Output1 | Voltage Output2 | Probability Digit |
|---|---|---|---|---|
| $V_{app} < V_{th}$ | $R_{OFF}$ | 0 | $V_{REF}/3$ | 0 |
| $V_{app} \geq V_{th}$ | $R_{ON}$ | $V_{REF}/3$ | 0 | 1 |

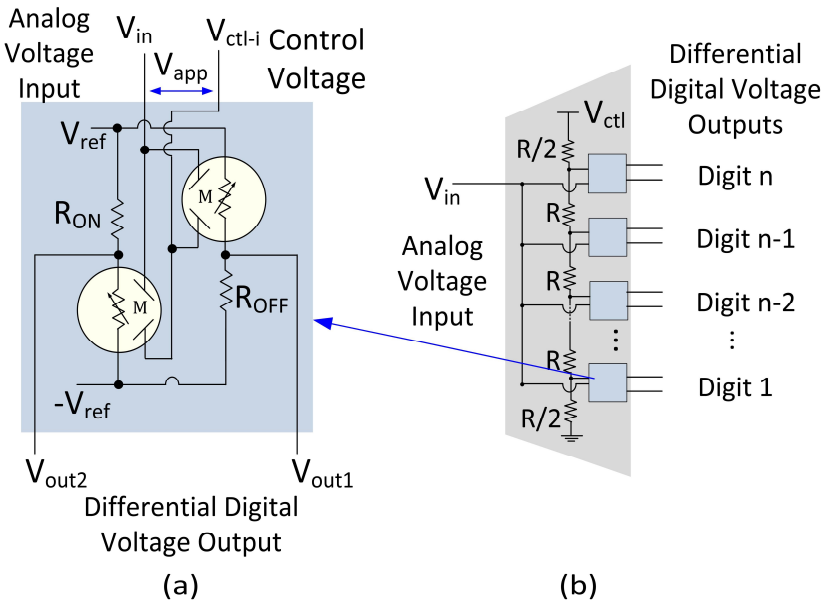



Figure 9. Decomposer Circuit Design: (a) Decomposer Element used to generate differential digital voltages based on analog input voltage for a given threshold voltage; and (b) Full Decomposer circuit consisting of n Decomposer Elements to convert analog voltage signal to n-digit probability vector using discrete voltage representation. Here, $V_{ctl-i}$ controls the threshold voltage for the *i*-th element and is determined bv the resistance ladder network.

and the other with S-MTJ in pull-down (see Figure 9). The possible states of the S-MTJs and the corresponding output voltages are shown in Table 1 for this configuration.

### 3.6 Fault Resilience (Supporting Graceful Degradation)

Information representation is inherently fault resilient in both electrical and magnetic domains. Consider two possible single-fault scenarios: (i) an input voltage at any position is shifted by a single level, and (ii) a magnetization vector in an S-MTJ is offset to a neighboring state of the 'intended' value. Given that the representation is redundant with all digits carrying equal weight, either fault would cause the overall value to be erroneous by $1/[n(k\text{-}1)]$, i.e., the resolution of the computation. This is in direct contrast to conventional *m*-digit radix-based representations (e.g., binary, HEX) where a single fault

can cause up to a $2^{m-1}$ error in the value being stored/computed based on the position. The proposed approach thus supports a graceful degradation, which is linear with increasing number of faults. Furthermore, the number of digits used ($n$) can be adjusted depending on the precision and fault-resilience required by the application.

### 3.7 Probability Arithmetic Composer Circuit Framework

The proposed circuit framework achieves physical equivalence by directly implementing arithmetic functions operating on probabilities, rather than emulating with Boolean logic functions. An Arithmetic Composer can be recursively defined as a hierarchical instantiation of other Arithmetic Composer functions until Elementary Arithmetic Composer functions with S-MTJs are reached, as shown in Figure 10. To this end, we defined four Elementary Composers: '+', '–', 'x', '÷'. Details on circuit designs are presented in the subsequent section. Thus, an Arithmetic Composer $\boldsymbol{f^n}$ consisting of $n$ levels of operations to be performed can be recursively expressed as:

$$for\ n > 1, \qquad \boldsymbol{f^n} = f^{n-1}\left(f_1^{n-2}, f_2^{n-2}, f_3^{n-2}, \dots, f_j^{n-2}\right) \tag{17}$$

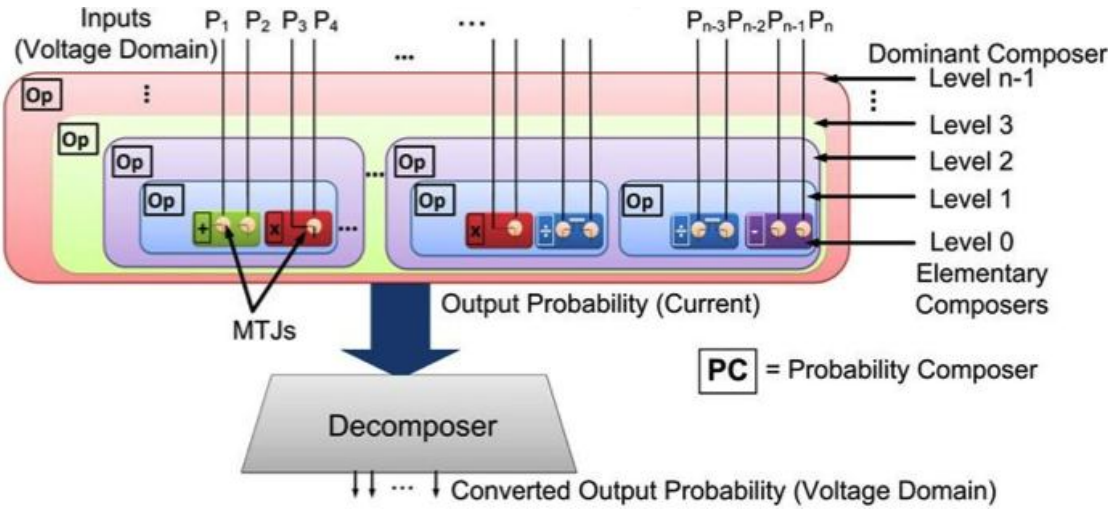



Figure 10. Probability Composer Circuit Framework

$$for\ n = 1, \qquad f^1 = f^0(primary\ inputs); \qquad where f^0\ is\ an\ Elementary\ Composer.$$

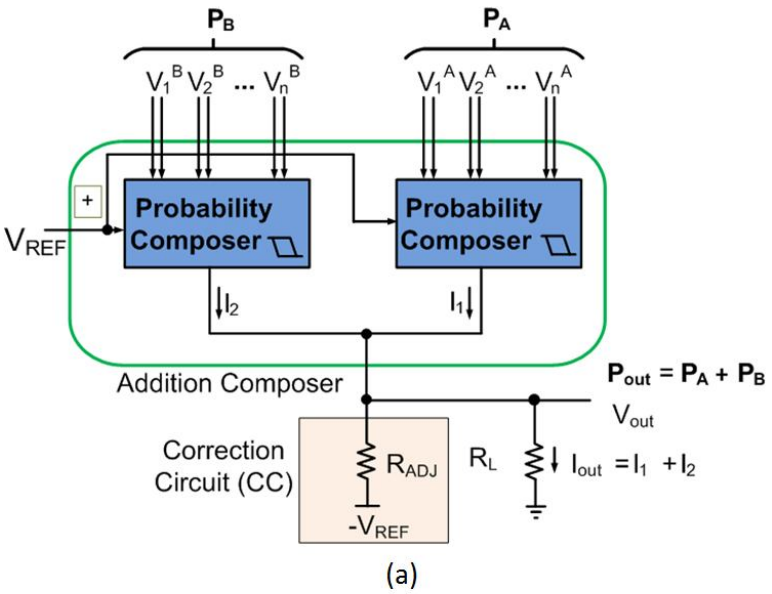
The top-level operation to be performed ($f^{n-1}$) is called the Dominator Composer since it determines the overall Composer circuit structure, where each node is either another Arithmetic Composer or an Elementary Composer. This approach is easily scalable since any function can be hierarchically built by plugging Arithmetic Composer nodes in a Dominator Composer, without changing the circuit style. For example, a function $F = (P_a.P_b)+(P_c.P_d)$ can be hierarchically represented as

$$F = \boldsymbol{f^2} = f^1(f_1^0, f_2^0) = \mathrm{SUM}[\mathrm{MUL}(P_a, P_b), \mathrm{MUL}(P_c, P_d)]. \tag{18}$$

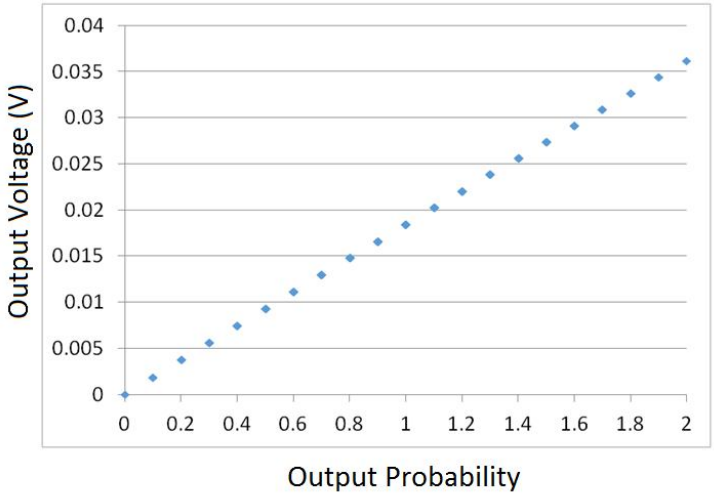
Here $n = 2$ since there are two levels of operations to be performed, $f^1$=SUM and $f^0$=MUL. Thus, at any given level, the Arithmetic Composer is Self-Similar to its corresponding Elementary Composer, exhibiting fractal-like behavior.

### 3.8 Elementary Arithmetic Composers

The Composers at the lowest level of hierarchy perform fundamental arithmetic operations on probabilities, and are called Elementary Arithmetic Composers. Three of the four fundamental arithmetic operations, viz. multiplication, addition, subtraction, are physically realized based on fundamental laws of circuit physics. While division operation may also be envisioned for physically equivalent implementation, the S-MTJ device limitations (particularly the low $R_{OFF}/R_{ON}$) preclude S-MTJ based direct divider implementation. Hence, we use a physically equivalent circuit based on approximation with addition, subtraction and multiplication with correction techniques to implement a probability divider for our framework. However, a different non-volatile device that does not have S-MTJ limitations may enable a direct physical divider implementation.




(a)




(b)

Figure 11. (a) Elementary addition composer using voltage mode read-out; and (b) Corresponding output voltage vs. probability characteristics as calculated by eq. (19) after correction, and validated using HSPICE simulations for all possible input combinations.

Kirchhoff's current law enables elegant physically equivalent implementation for addition and subtraction operations. This is well known in analog CMOS circuits. Here, we illustrate how to implement non-volatile probability adders and subtractors using S-MTJs. Current addition can be implemented by using a parallel configuration of

Probability Composer elements, as shown in Figure 11a. This is an extension of the Probability Composer element itself where each S-MTJ was arranged in parallel to be able to sum the probability digits represented using the resistance states. By using a single reference voltage $V_{REF}$ and load resistor $R_L$ (of the order of 10-100KOhms) with value much smaller than Probability Composer element resistance (in the order of tens of MOhms), the parallel topology of two *n*-digit Probability Composer elements produces an output current as follows:

$$\begin{aligned} I_{out} &= \frac{V_{REF}}{\left(\frac{R_{PC-A}.R_{PC-B}}{R_{PC-A}+R_{PC-B}}+R_L\right)} \approx V_{REF}\left(\frac{1}{R_{PC-A}}+\frac{1}{R_{PC-B}}\right) \\ &= \frac{V_{REF}}{\beta}\left[\left(\sum_{i=1}^{n} p_i\right)_A + \left(\sum_{i=1}^{n} p_i\right)_B\right] + \left\{\frac{2n\varepsilon V_{REF}}{\beta}\right\} \\ &= \frac{nV_{REF}}{\beta}[\boldsymbol{P_A}+\boldsymbol{P_B}] + \left\{\frac{2n\varepsilon V_{REF}}{\beta}\right\}. \end{aligned} \tag{19}$$

Correction Circuits (CC) can be used as before in Probability Composer element to extract the current given by the term in {.} in eq. (19). To get a voltage output, we use the voltage across the load resistor (see Figure 11b), which can be amplified using CMOS op-amps. Alternatively, we can simply use the same topology with correction circuits, while removing the resistor $R_L$ (if the non-linearity in output can be tolerated) for larger voltage range as follows:

$$V_{out} = V_{REF}.\left[\frac{\frac{1}{R_{PC-A}}+\frac{1}{R_{PC-B}}-\frac{2}{R_{ADJ}}}{\frac{1}{R_{PC-A}}+\frac{1}{R_{PC-B}}+\frac{2}{R_{ADJ}}}\right] = V_{REF}.\left[\frac{\boldsymbol{P_A}+\boldsymbol{P_B}}{\boldsymbol{P_A}+\boldsymbol{P_B}+4\varepsilon}\right]. \tag{20}$$

Output Decomposers can be designed to differentiate the voltage levels such that output non-linearity is tolerated.

Subtraction is achieved by reversing one of the branches such that it supplies a $-V_{REF}$ to the Probability Composer Element, as shown in Figure 12. The voltage output is given by:

$$V_{out} = I_{out}R_L = \frac{nV_{REF}}{\beta}[\boldsymbol{P_a} - \boldsymbol{P_b}], \qquad R_L \ll R_{PC.} \tag{21}$$

We implement multiplication based on Ohm's law, $V = I.R$, rewritten as $I = V/R$. By representing one of the inputs as voltage $V$, and the other as resistance of Probability Composer Element, we directly implement a multiplication operation. The circuit topology is shown in Figure 13a. The first Probability Composer element converts the digital probability vector from magnetic (resistance) domain to analog voltage domain. This voltage needs to be adjusted so that the loss in the first stage is compensated through

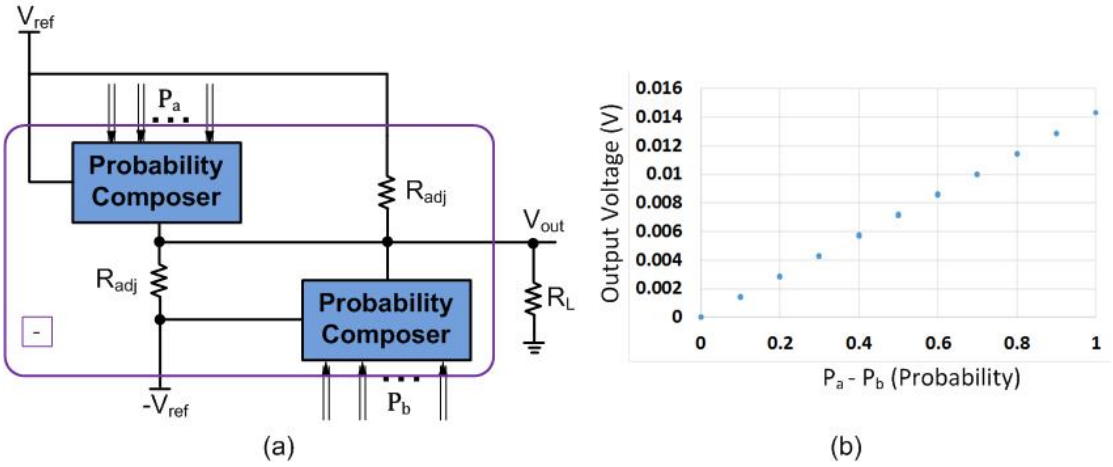
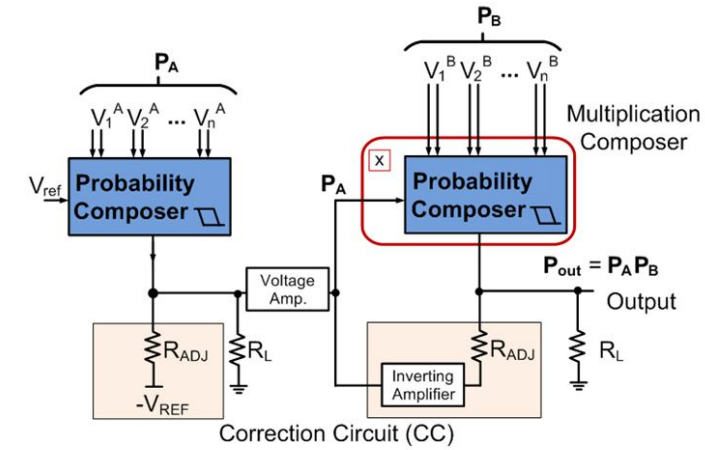
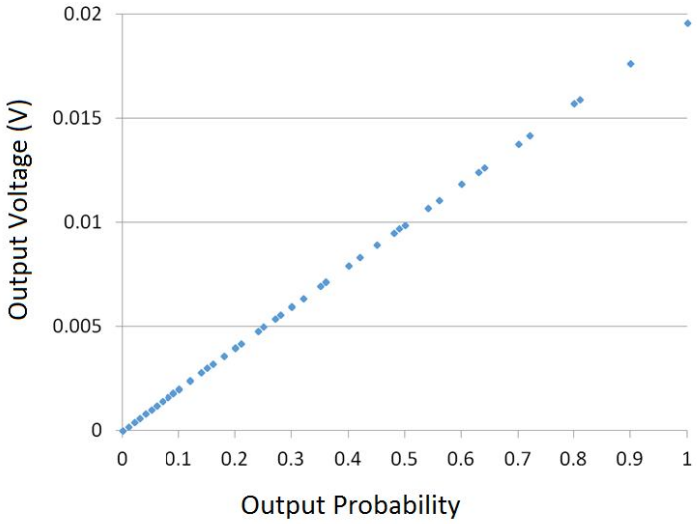



Figure 12. (a) Elementary subtraction composer using voltage mode read-out; and (b) Corresponding output voltage vs. probability characteristics for $P_a \geq P_b$ as calculated by eq. (21), and validated using HSPICE simulations for all possible input combinations.




(a)




(b)

Figure 13. (a) Elementary multiplication composer topology; and (b) Output probability vs. voltage characteristics in continuous analog domain validated using HSPICE simulations for all possible input combinations.

amplification using CMOS support circuits (such as op-amps). The amplified voltage is used as an input voltage to the next Probability Composer element, whose resistance is inversely related to the encoded probability value. The current through the second Probability Composer element achieves multiplication of the two probabilities (with Correction Circuits), given by:

$$I_{out} \approx \frac{n}{\beta}[V_A.\boldsymbol{P_B}] = \frac{n^2}{\beta^2}.g.R_L.V_{REF}.(\boldsymbol{P_A}\boldsymbol{P_B}), \qquad R_L \ll R_{PC}. \tag{22}$$

The current output can be converted to voltage mode by simply using the voltage across the load resistance $R_L$ (see Figure 13a). Alternatively, voltage domain output with larger range can also be obtained by simply removing the load resistance $R_L$, given by:

$$V_{out} = g.V_{REF}.\left[\frac{\boldsymbol{P_A}.\boldsymbol{P_B}}{(\boldsymbol{P_A} + 2\varepsilon)(\boldsymbol{P_B} + 2\varepsilon)}\right]. \tag{23}$$

Similar to previous case, the denominator causes non-linearity in the output, and is affected when $\boldsymbol{P_A}$ or $\boldsymbol{P_B}$ takes a value close to 1. As long as a Decomposer can be designed to tolerate this non-linearity, this topology may be used.

A direct division may be implemented based on the above mindset, through Ohm's law. However, the limited $R_{OFF}/R_{ON}$ for the S-MTJ devices means that such topologies will have error factors which are difficult to eliminate. A different non-volatile device with a higher $R_{OFF}/R_{ON}$ may enable such circuit implementations. Hence, in this work we attempt to implement a physically equivalent division through approximation using multiplication, addition and subtraction. We use the following expression using addition, subtraction and multiplication based on assumptions to be stated in the following:

$$\boldsymbol{P_{out}} = \left[\frac{\boldsymbol{P_a}}{\boldsymbol{P_b}}\right] \approx [k_0.\boldsymbol{P_a} + k_1.\boldsymbol{P_b} + k_2] + f_c(\boldsymbol{P_a}, \boldsymbol{P_b}), \tag{24}$$

where $k_0$, $k_1$, and $k_2$ are constants between -1 and 1 with a resolution of 0.1, and $f_c$ is a correction factor as a function of the two input arguments. The square brackets [.] in eq. (24) indicate rounding function. The assumptions here are:

i) The output of a division will always result in a valid probability as per the algorithm being used. This implies $\boldsymbol{P_a \leq P_b}$.

ii) The probability $\boldsymbol{P_b}$ will never take a value 0. Control circuits may be designed to detect a violation of this condition.

iii) Computational resolution is 0.1.

Based on the conditions above, we evaluated all possible combinations of constants $k_0$, $k_1$ and $k_2$ that result in the best fit for equation (24) (see Figure 14, Figure 15, Figure 16). Expressions resulting in least number of unique error cases were selected as candidate solutions. We found two expressions that resulted in a minimum of 4 unique error cases to be corrected (see Figure 14). Ideally, the final expression would be chosen based on ease of implementation. In our case, we found that both expressions had similar complexity of implementation and any one of the two expressions could be used. The

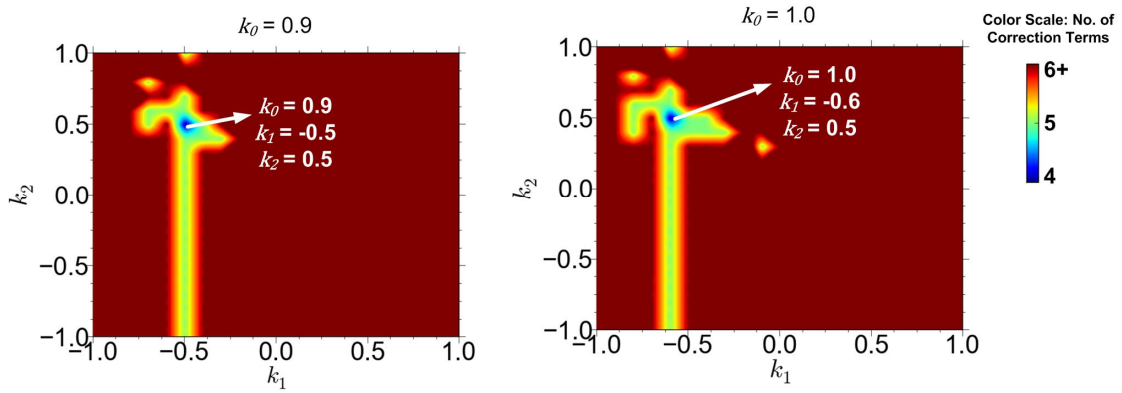



Figure 14. Contour plots showing the count of unique error cases, which is equal to the number of correction circuits required, for division implementation through approximation using eq. (24). All possible combinations with resolution of 0.1 for coefficients $k_0$, $k_1$, and $k_2$ were tested and the best results are shown here. The minimum number of correction circuits (indicated by blue color) required were found to be 4, for two expressions with coefficients (a) $k_0$ = 0.9, $k_1$ = -0.5, $k_2$ = 0.5; and (b) $k_0$ = 1.0, $k_1$ = -0.6, $k_2$ = 0.5. Here, negative coefficients indicate the use of subtraction.

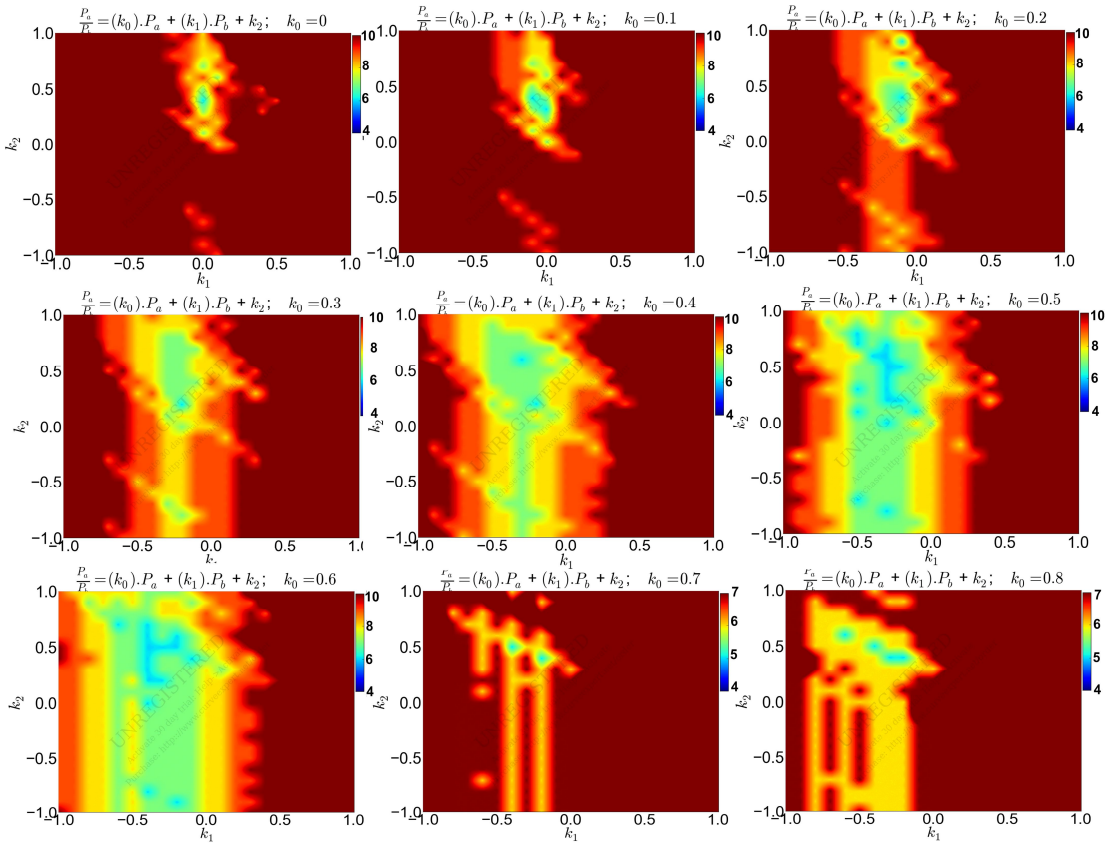


Figure 15. Contour plots showing the count of unique error cases, which is equal to the number of correction circuits required, for division implementation through approximation using eq. (24). Coefficient $k_0$ ranges from 0 to 0.8.

values used were $k_0$ = 0.9, $k_1$ = -0.5 and $k_2$ = 0.5.

The correction factor was determined by taking every possible input combination, and hard-wiring the required corrections for cases where the relationship in eq. (24) causes error. For each correction case, we used an enable logic circuit that switched the correction term ON based on the input values. This was a logic-based implementation and used S-MTJs in conjunction with digital CMOS logic circuits in our approach (see Figure 17).

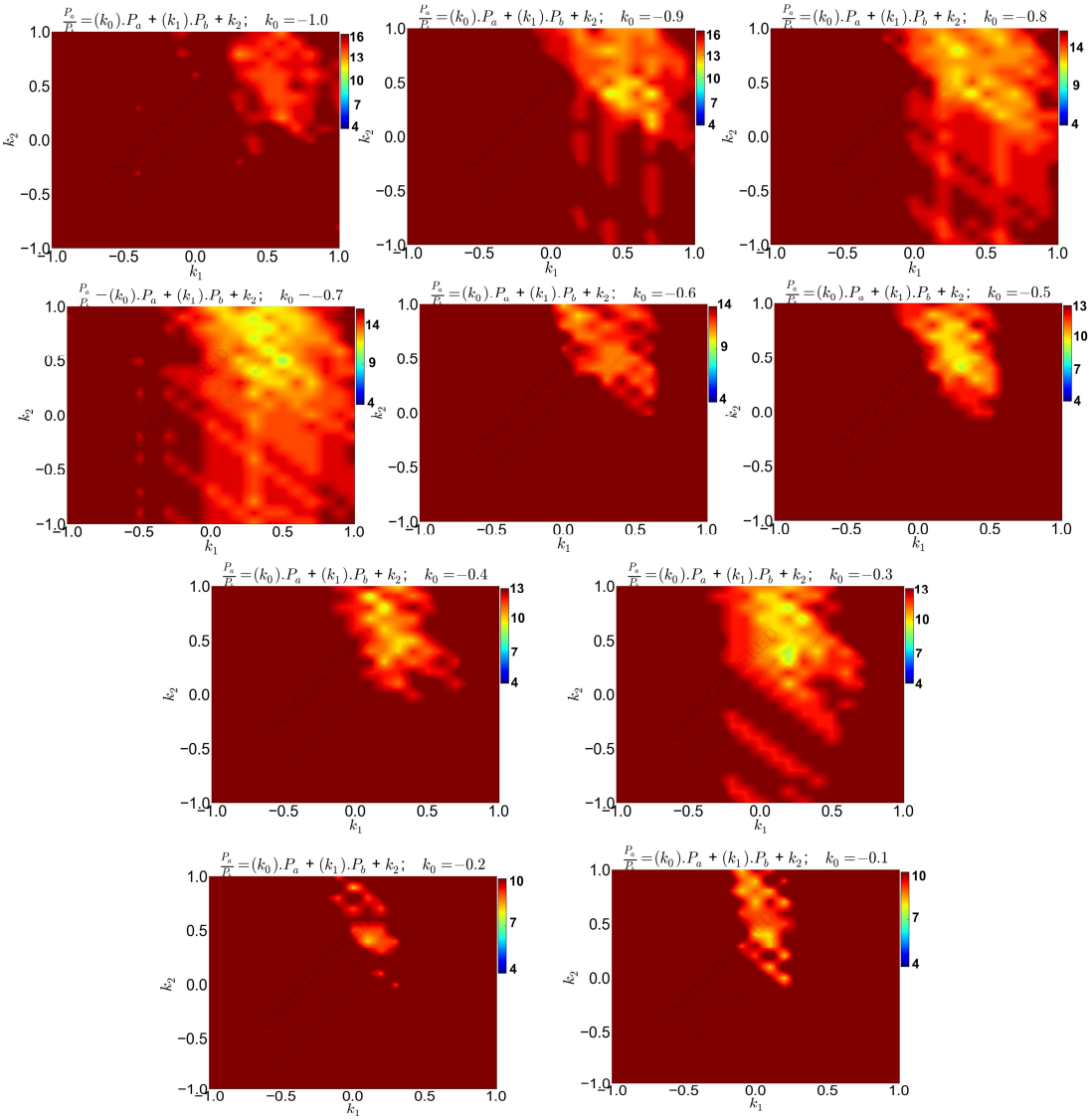
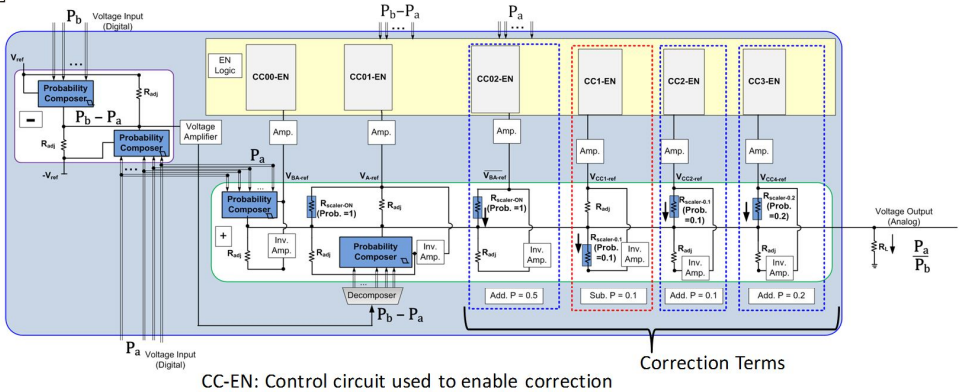


Figure 16. Contour plots showing the count of unique error cases, which is equal to the number of correction circuits required, for division implementation through approximation using eq. (24). Coefficient $k_0$ ranges from -1.0 to -0.1.

A more complex arithmetic operation such as sum-of-products (used frequently in BN inference) can be composed using these Elementary Addition and Multiplication Composers. We illustrate an example to compose an operation of the form $(P_A.P_B)+(P_C.P_D)$. One way to implement it is to use Elementary Addition and Multiplication Composers and connect them serially. However, the Probability

$$\left[\frac{P_a}{P_b}\right] = [0.9P_a - 0.5P_b + 0.5] + f_c(P_a, P_b) = [0.4P_a - 0.5(P_b - P_a) + 0.5] + f_c(P_a, P_b)$$




CC-EN: Control circuit used to enable correction terms; implemented with CMOS digital logic

(a)

**Conditions for Enabling Correction Factors:**

**CC00-EN:** If $P_a$=$P_b$ Out = 0V, else Out = 1V

**CC01-EN:** If $P_a$=0, Out = 0V, else Out = 1V

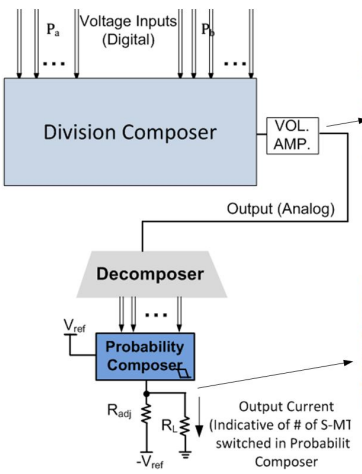
**CC02-EN:** If $P_a$=$P_b$ Out = 1V, else Out = 0V

**CC1-EN:** Out = 1V if [$P_a$=0.1 & 0.3≤$P_b$≤0.8] OR [$P_a$=0.2 & $P_b$=0.6]

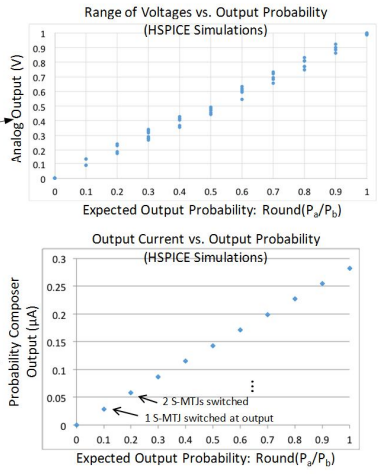
**CC2-EN:** Out = 1V if [0.4≤$P_a$≤0.5 & 0.6≤$P_b$≤0.7] OR [$P_a$=0.3 & $P_b$=0.5] OR [0.5≤$P_a$≤0.8 & $P_b$=0.9] OR [0.6≤$P_a$≤0.9 & $P_b$=1.0]

**CC3-EN:** Out = 1V if [0.2≤$P_a$≤0.4 & $P_b$-Pa=0.1] OR [0.6≤$P_a$≤0.7 & $P_b$-Pa=0.1] OR [$P_a$=0.6 & $P_b$=0.8]

(b)




(c)




(d)

Figure 17. (a) Division schematic (through approximation using addition, multiplication and subtraction Composers and correction circuits); (b) Conditions for enabling correction circuits; (c) Test cascade for functional validation using HSPICE; and (d) HSPICE simulation output.

Arithmetic Composer framework allows us to implement it efficiently for parallel computation by hierarchically composing an Add-Multiply composer as follows.

Each product term implemented with an elementary Multiplication Composer is arranged in a topology of the Addition Composer (see Figure 18a). Thus the Dominator Composer structure is that of the adder, which uses elementary Multiplication Composers

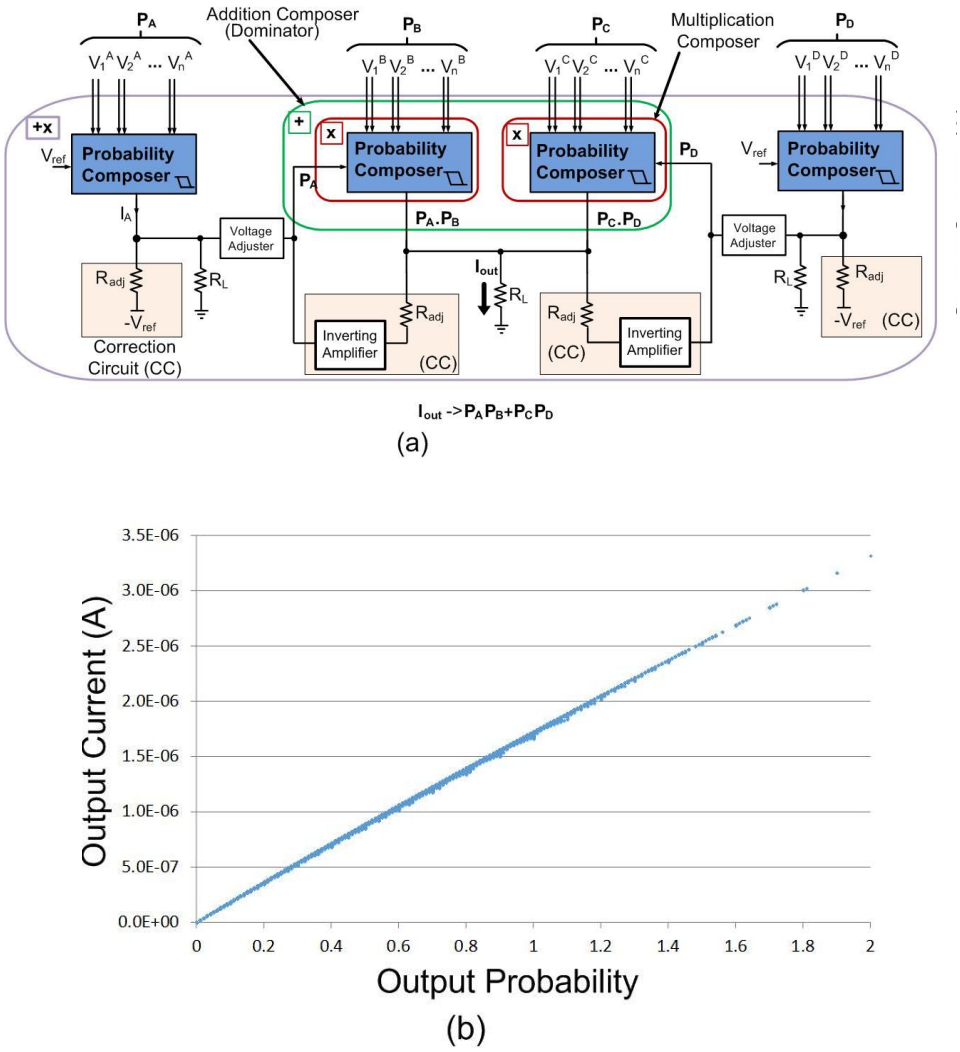



Figure 18. (a) Add-Multiply Composer for calculating sum-of-products on input probabilities. The output is in analog current-domain, and corresponds to the function, $P_A.P_B+P_C.P_D$. The voltage adjusters are used to amplify the voltage from first Probability Composer stage, which is then used as input voltage for read-out at the second stage. These adjusters and other support circuits such as the inverting amplifiers can be implemented using CMOS analog circuits (e.g. op-amps); and (b) Output characteristics showing probability output for all possible input combinations and the corresponding output current value, which are obtained using HSPICE simulations.

as the basic building blocks. This topology realizes the add-multiply operation in a single step (simulated output characteristics in Figure 18b).

In addition to these basic arithmetic operations on probabilities, normalization operation is used after computing updated beliefs at every node to ensure that resulting

beliefs are probabilities. This can be implemented using current-mode CMOS analog circuits based on Gilbert normalizer circuit [15], as shown in Figure 19. If the input currents $I_{in\text{-}1}$, $I_{in\text{-}2}$, etc. are in sub-threshold region of the MOSFET, then

$$I_{in-i} = I_0 e^{\kappa \frac{V_{di}}{V_T}}$$
$$I_{out-i} = I_0 e^{\kappa \frac{V_{di}}{V_T} - \frac{V_c}{V_T}} \quad (25)$$

where $i$ is the index of the input cell ($i \in \{1, 2, \ldots, n\}$ in this example), $V_T$ is thermal voltage and $\kappa$ is the subthreshold slope coefficient of the MOSFETs. Using Kirchhoff's current law at the common node $V_C$, we get

$$\sum_{i=1}^{n} I_{out-i} = I_b \quad (26)$$

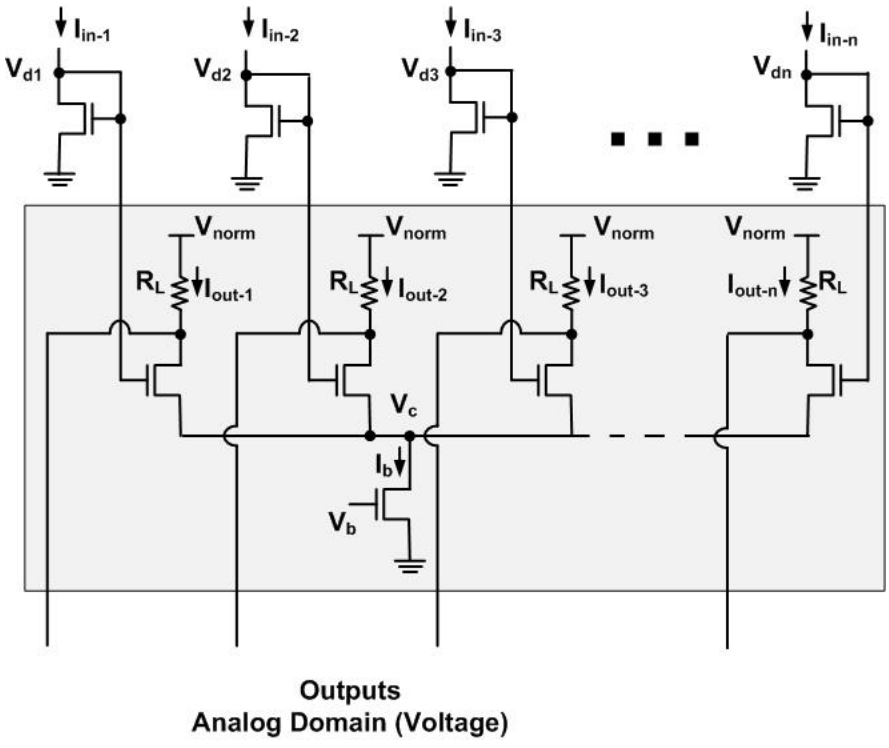



Figure 19. Normalization circuit for $n$ inputs using MOSFETs.

where $I_b$ is the constant current set by the reference voltage $V_b$. Substituting eq. (26) in (25) gives us the following relation, which is a normalization operation of the input currents. The input currents can be set using S-MTJ based Probability Composers.

$$I_{out-i} = I_b \frac{I_{in-i}}{\sum_{j=1}^{n} I_{in-j}} \tag{27}$$

### 3.9 Summary

In this chapter, we presented our approach towards using physically equivalent data representation for probabilities in the case of Bayesian Networks computing framework. We also discussed implementing elementary arithmetic operations on probabilities using physical laws in keeping with the mindset of physical equivalence for circuit implementation, through the Probability Arithmetic Composer framework. In the next chapter, we will introduce a physically equivalent architecture for Bayesian Networks.

# CHAPTER 4

# PHYSICALLY EQUIVALENT ARCHITECTURE FOR REASONING UNDER UNCERTAINTY

In keeping with the overarching philosophy of physical equivalence, the proposed architecture is designed such that it supports BNs intrinsically; i.e. there is a direct relationship to the structure of a BN graph and its physical implementation. Drawing inspiration from Field-Programmable Gate Arrays (FPGAs) that provide a reconfigurable hardware platform for mapping any digital Boolean logic function, we propose a

Table 2. Comparison: von Neumann Approach vs. Physical Equivalence Approach

| | von Neumann Computing | Physical Equivalence Paradigm |
|---|---|---|
| Information Representation | Radix Boolean (Voltage) | Flat Probability Vectors (Resistance, voltage, current) |
| Approach | Digital Logic, Pipelines, Arithmetic, Memory Hierarchy, Multi Core | Non-volatile Probability Arithmetic Composer Circuits (memory-in-computing), Programmable Switch Boxes |
| Architectural State | Registers, Memory | Probability Tables, Beliefs, Likelihoods, Priors incorporated into non-volatile Composer Circuits; Network structure in switch-boxes |
| Operations Defined by | Instruction Set Architecture (ISA) | Roles – Learning, Inference, Adaptation |
| Plasticity | Explicit software update | Autonomous learned behavior, reconfigurability |
| Machine Execution | Explicitly timed instruction execution, data sharing | Event-based message propagation in network |
| Failure Tolerance | None – Susceptible to single fault | Graceful degradation with faults |
| Technology / Primary Device | CMOS (charge-based) / MOSFET | Hybrid of CMOS and S-MTJs (charge, magnetic)/ S-MTJ: Voltage controlled rotation of magnetization; Resistance change is persistent and can be readout |
| Target Applications | High precision arithmetic, interactive applications, deterministic behavior | Applications requiring causal learning and inference in various domains, under uncertainty |

reconfigurable distributed Bayesian Cell architecture to map any given Bayesian Network structure (see Figure 20). This is a significant departure from conventional von Neumann architecture (Table 2). Each Bayesian Cell (or a cluster of multiple Bayesian Cells) implements computation for BN operations in a node. The network consists of several such Bayesian Cells interconnected in a mesh network, through a heterogeneous integration with CMOS metal routing stack for message propagation.

Each Bayesian Cell incorporates state information and conditional probability tables (CPT) intrinsically within non-volatile Probability Composer circuits, for inference and learning operations. Updates to the CPTs can be performed during learning and adaptation, by changing the resistance state of corresponding S-MTJs in the Probability

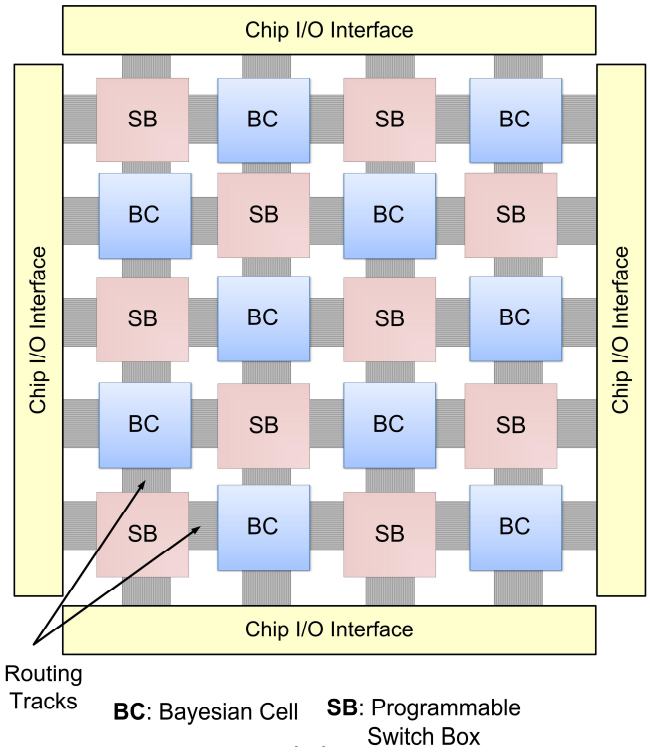



Figure 20. Proposed Reconfigurable Bayesian-Cell (BC) architecture. Each module in a BC is implemented with non-volatile Probability Composers (no separate memory needed). Routing tracks implemented with CMOS metal stack.

Composers. An incoming message would trigger an Activity Controller to power up the Bayesian Cell. This mesh network can scale to large problem sizes since message propagation is near neighbor in BNs. I/O requirements would be typically sparse since not all evidence variables need to be observed simultaneously; even single evidence triggers inference.

### 4.1 Bayesian Cell Description

A Bayesian Cell is designed to be capable of implementing all operations required for BN inference (see Figure 21). The main operational component is the Inference Engine. Architectural support components include Activity Monitor, Role Management Unit, and Switch Box Interfacing. The Inference Engine incorporates all operations occuring during

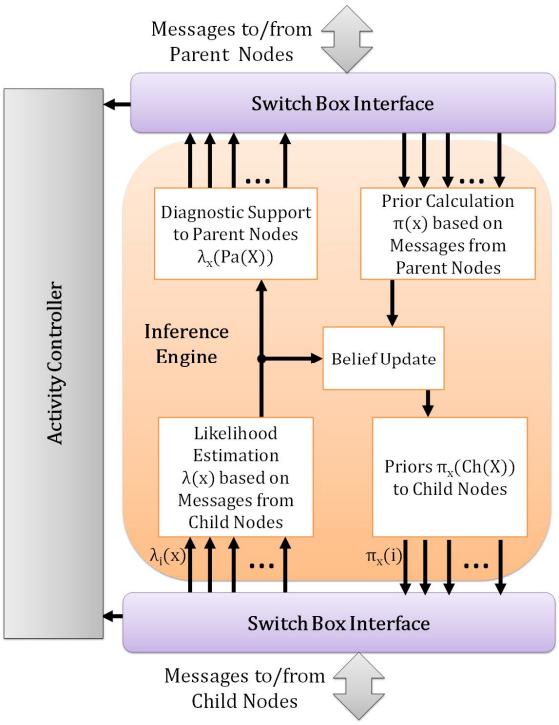



Figure 21. Bayesian Cell architectural schematic showing modules for inference and learning operations. Each module is implemented with non-volatile Probability Composers.

a Bayesian Inference process. Several algorithms exist in literature for Bayesian Inference [16]. As a starting point, we use Pearl's Belief Propagation algorithm that is amenable for local message passing impementations using a Bayesian Cell based architecture. While this algorithm performs exact inference in trees and polytrees, it is not applicable to networks where the graphs incorporate loops. For more general networks, approximate algorithms have been developed such as the loopy Belief Propagation algorithm. Future work will investigate implementing such generalized algorithms applicable for any given Bayesian Network structure.

The Inference Engine implements the Bayesian Inference operations using physically-equivalent Probability Arithmetic Composers, described in the previous chapter. In addition, non-volatility in resistance state of the S-MTJs implies all required arguments/parameters for Bayesian computations are stored locally within the Composers themselves. In stark contrast to von Neumann architecture, there is no separate memory store to read data from, thus leading to a memory-in-computing architecture. Future work can extend this architecture to support leanrning and adaptation as well.

We describe the inference operations and their corresponding Composer implementations next. We use an example scenario where each node has one parent node and two child nodes to illustrate the approach. In addition, each node is assumed to support a maximum of four states. This can be extended to accommodate more states and parents/children per node as defined by the underlying computations. The complete schematic for the Inference Engine is shown in Figure 22.

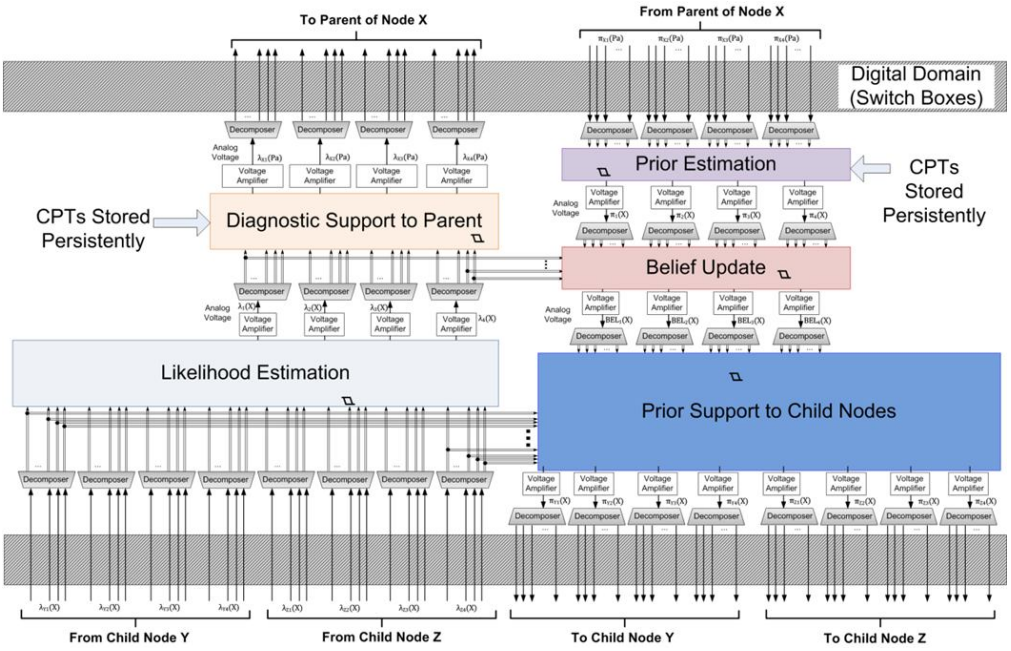



Figure 22. Inference Engine schematic showing various modules and CMOS analog support circuits involved during Bayesian Inference operation.

A BN inference process using Pearl's Belief propagation is iterative. The key operations at each node during inference are (i) likelihood/prior estimation for the current node based on messages received from child/parent nodes, (ii) belief update to estimate the probability of each state of the current node given the observed evidence, and (iii) diagnostic/prior support to generate messages for communicating with child/parent nodes.

Consider a node X with parent node A and child nodes Y and Z (see Figure 1). Messages are received at node X either from parent A (top-down), or child nodes (bottom-up), or both depending on where the evidence is observed. When evidence is observed in a node that is a descendent of node X, a bottom-up message propagation is triggered in the network form the evidence node and messages eventually reach node X through its children Y and Z. These messages are called diagnostic support messages. On

the other hand, evidence observed at a node that is an ascendant of node X, top-down messages are triggered that eventually reach X through A. These are called prior support messages to node X. Evidence may be observed in both directions as well, triggering both kinds of message propagation. All these messages are assimilated at node X through the computations defined by Pearl's belief propagation algorithm, and updates to the probability of each state of node X are performed. After this, prior and diagnostic support messages are triggered from node X to other neighboring nodes that communicate the changes due to observed evidence.

The following operations occur at node X when diagnostic support messages (bottom-up propagation) are received from its child nodes Y and Z:

(a) Diagnostic support messages from the child nodes are composed to calculate the likelihood vector $\boldsymbol{\lambda(X)}$ for node X. This operation requires 4 multiplications as follows –

$$\boldsymbol{\lambda(X)} = \boldsymbol{\lambda_Y(X)} * \boldsymbol{\lambda_Z(X)}, \tag{28}$$

where each element in $\boldsymbol{\lambda(X)}$ is given by

$$\lambda_i(X) = \lambda_{Yi}(X) \cdot \lambda_{Zi}(X); i = \{1,2,3,4\}. \tag{29}$$

This is implemented with multiplication composers discussed in the previous chapter.

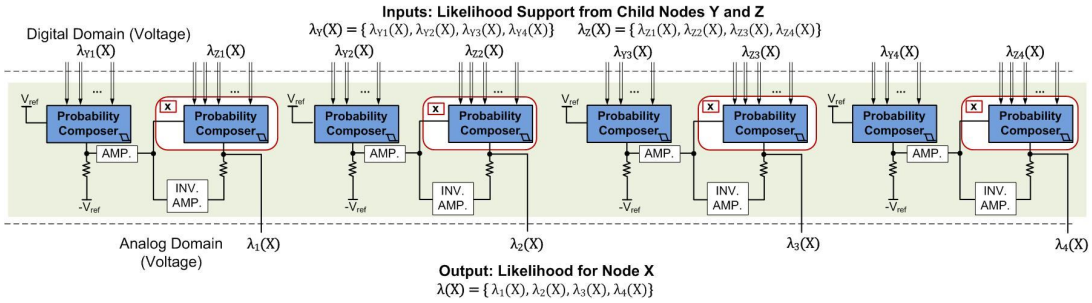



Figure 23. Probability Composers for Likelihood Estimation for Bayesian Inference. Amplifiers are implemented with analog CMOS circuits.

The corresponding circuit is shown in Figure 23.

(b) Diagnostic support to parent of node X (say node A) is computed based on λ(X) and the CPT. This requires 16 multiplications and 12 additions as follows –

$$\boldsymbol{\lambda_X(A)} = \boldsymbol{CPT(X|A)} \otimes \boldsymbol{\lambda(X)} \tag{30}$$

where each element in $\boldsymbol{\lambda_X(A)}$ is given by

$$\lambda_{Xi}(A) = \sum_{j=1}^{4} CPT_{ij}(X|A) \cdot \lambda_j(X)\,; i = \{1,2,3,4\}. \tag{31}$$

This is a composed arithmetic operation. One way to implement it is to use elementary addition and multiplication Composers and connect them serially. However, the Probability Composer framework allows us to implement it efficiently for parallel computation by hierarchically composing an add-multiply composer as follows. Each product term implemented with an elementary multiplication Composer is arranged in a topology of the addition Composer. Thus the dominator Composer structure is that of the adder, which uses elementary multiplication composer blocks. This topology realizes the add-multiply operation in a single step. The corresponding circuit is shown in Figure 24. Prior Estimation is similar and shown in Figure 25.

(c) Based on new evidence, a belief update is performed at node X using likelihood and prior vectors. Computing the elements of the belief vector involves 4 multiplications, 3 additions and 4 divisions as follows –

$$\boldsymbol{BEL(X)} = \alpha \boldsymbol{\pi(X)} * \boldsymbol{\lambda(X)} \tag{32}$$

where each element in $\boldsymbol{BEL(X)}$is given by

$$BEL_i(X) = \frac{\pi_i(X) \cdot \lambda_i(X)}{\sum_{i=1}^{4} \pi_i(X) \cdot \lambda_i(X)}\,; i = \{1,2,3,4\}. \tag{33}$$

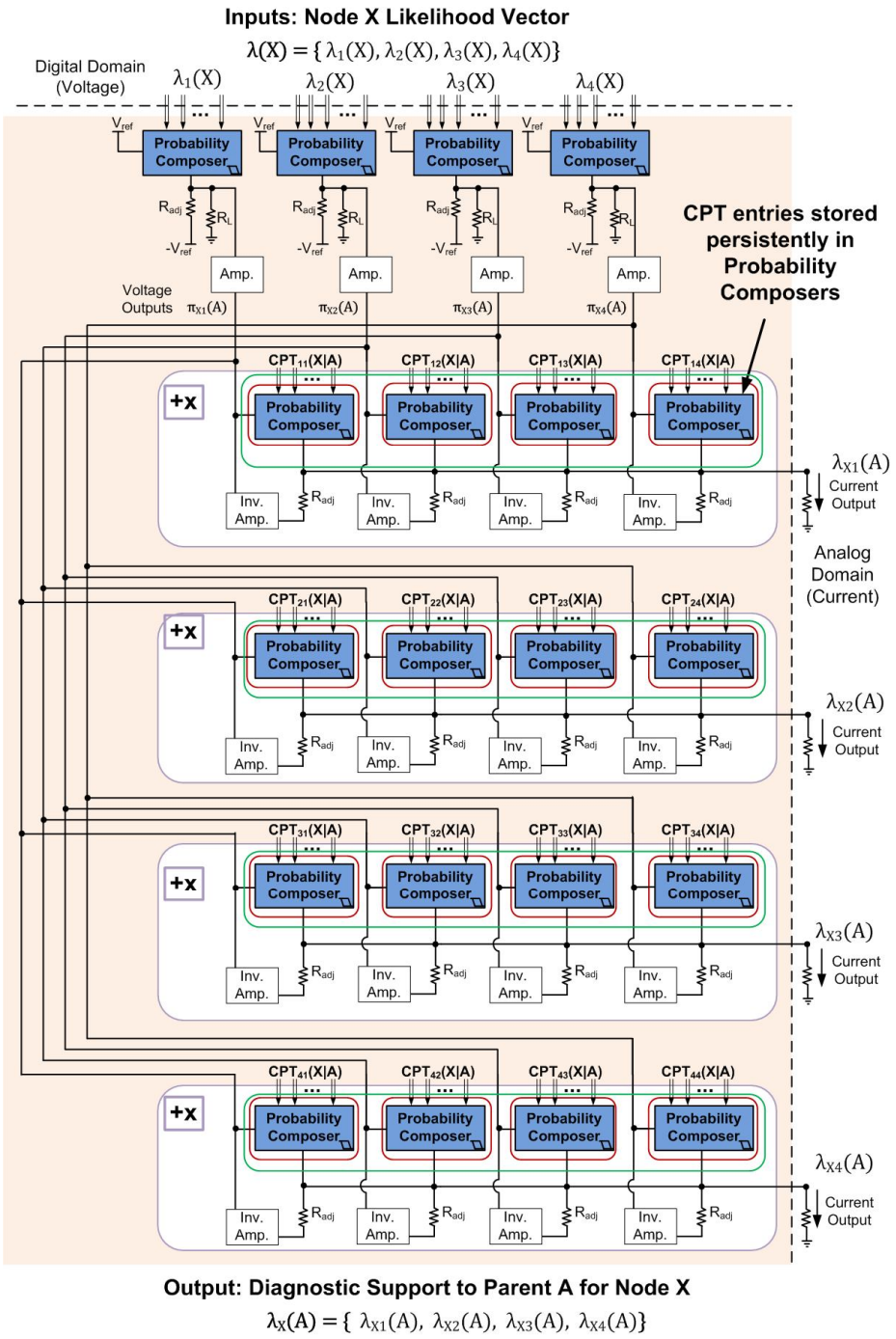



Figure 24. Module for diagnostic support to parent node during Bayesian Inference. All composers are implemented with non-volatile S-MTJs and do not require a separate memory store. The CPT entries are stored in the resistance states of the S-MTJs.

This represents a normalization operation performed after multiplication. Again, while

this may be achieved with elementary Arithmetic Composers cascaded in series, a parallel implementation is achieved through the hierarchical Composer framework.

(d) Predictive support to each child of node X is computed based on computed belief BEL(X) and the likelihood support from the child node. This requires 4 division operations per child node as follows –

$$\boldsymbol{\pi_Y(X)} = \frac{\boldsymbol{BEL(X)}}{\boldsymbol{\lambda_y(X)}}, and\ \boldsymbol{\pi_Z(X)} = \frac{\boldsymbol{BEL(X)}}{\boldsymbol{\lambda_Z(X)}}. \quad (34)$$

Here, $\pi_{Yi}(X) = \frac{BEL_i(X)}{\lambda_{Yi}(X)}; \pi_{Zi}(X) = \frac{BEL_i(X)}{\lambda_{Zi}(X)}; i = \{1,2,3,4\}$.

This is implemented with division composers described in the previous chapter.

For a given node X, the following operations occur when predictive support message is received from its parent node A:

(a) Based on new predictive support, the prior vector for node X is calculated as follows –

$$\boldsymbol{\pi(X)} = \boldsymbol{\pi_X(A)} \otimes \boldsymbol{CPT(X|A)} \quad (35)$$

The circuit implementation is based on add-multiply composers (see Figure 25).

(b) Belief update is performed at node X involving 4 multiplications, 3 additions and 4 divisions as follows –

$$\boldsymbol{BEL(X)} = \alpha\boldsymbol{\pi(X)} * \boldsymbol{\lambda(X)} \quad (36)$$

where each element in $\boldsymbol{BEL(X)}$is given by

$$BEL_i(X) = \frac{\pi_i(X) \cdot \lambda_i(X)}{\sum_{i=1}^{4} \pi_i(X) \cdot \lambda_i(X)}; i = \{1,2,3,4\}. \quad (37)$$

(c) Predictive support to each child of node X is computed based on computed belief

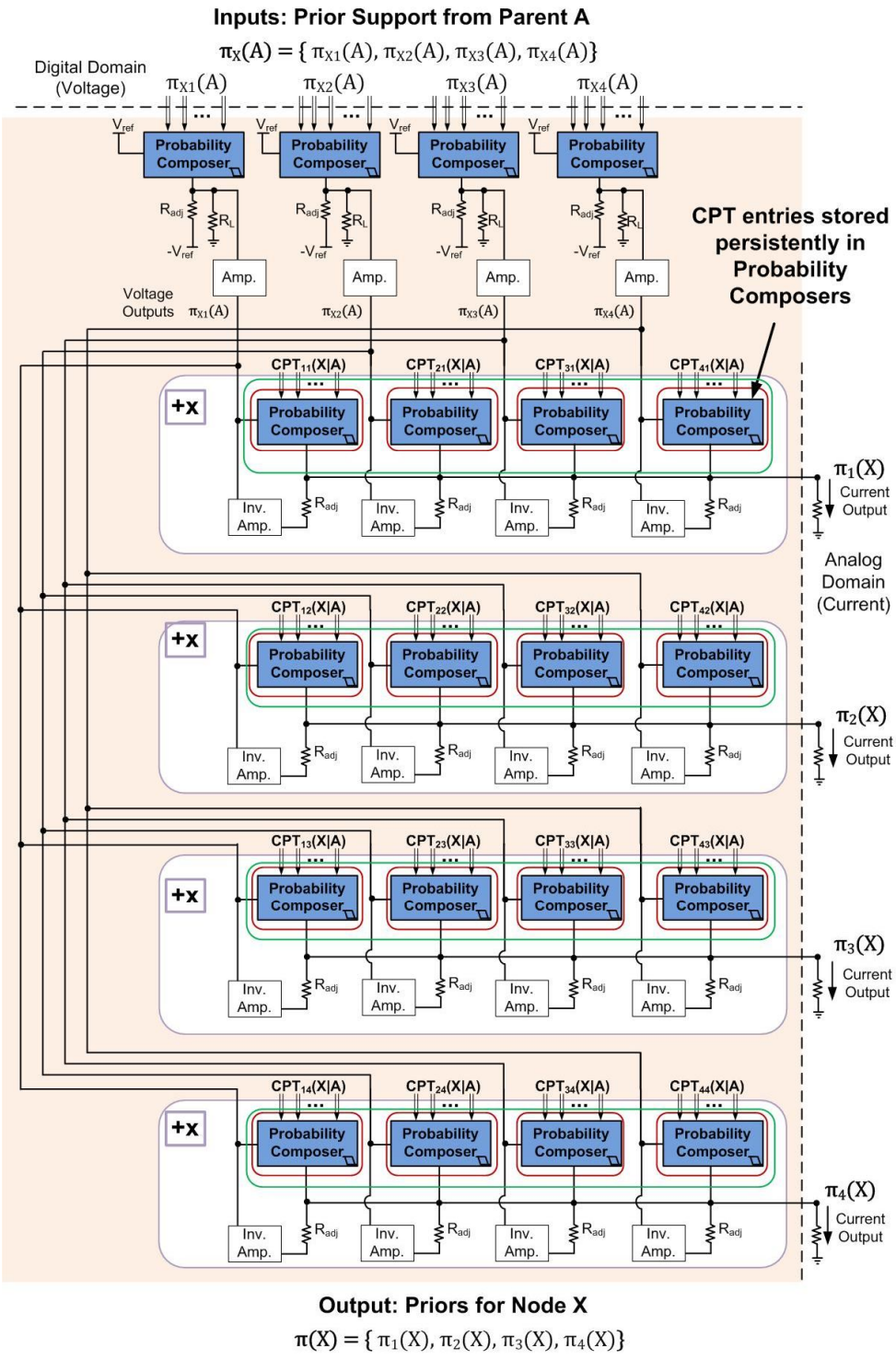



Figure 25. Composer implementation for estimating priors based on support received from parent node during Bayesian Inference.

***BEL(X)*** and the likelihood support from the child node. This requires 4 division operations as discussed earlier.

### 4.3 Switch Box Description

The BN structure consists of links that describes dependencies between variables. In our approach, we use direct physical connections between Bayesian Cells to encode the links between nodes with physical equivalence. These connections are made reconfigurable through the use of Switch Boxes (see Figure 26), similar to those used for programmable routing in FPGAs. The reconfigurability allows adding/removing connections as required for adaptability, as well as the capability to map any given BN in

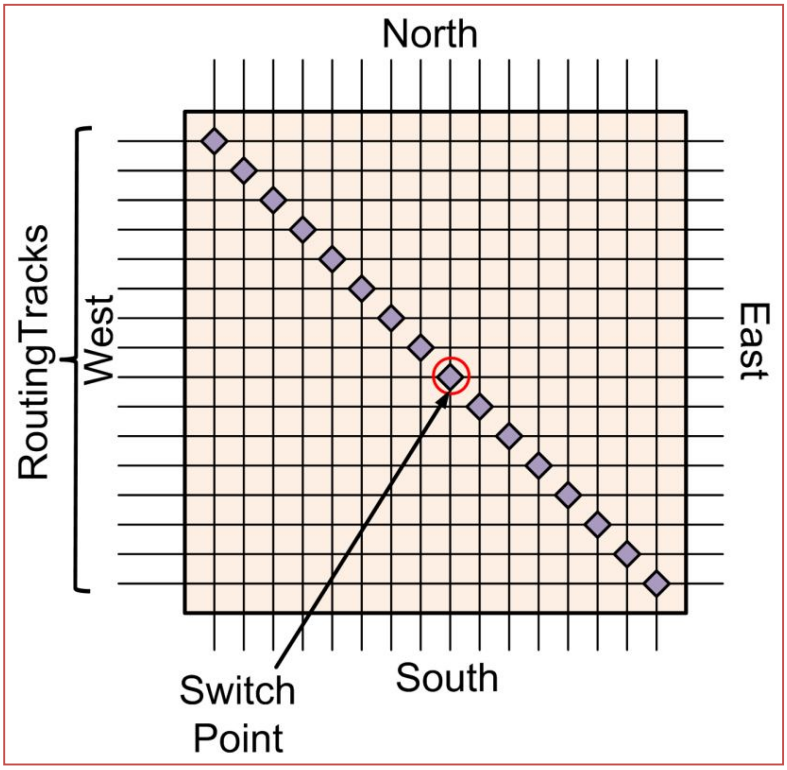



Figure 26. Programmable switch box schematic showing routing tracks and switch points. Routing tracks are implemented using conventional CMOS metal routing layers.

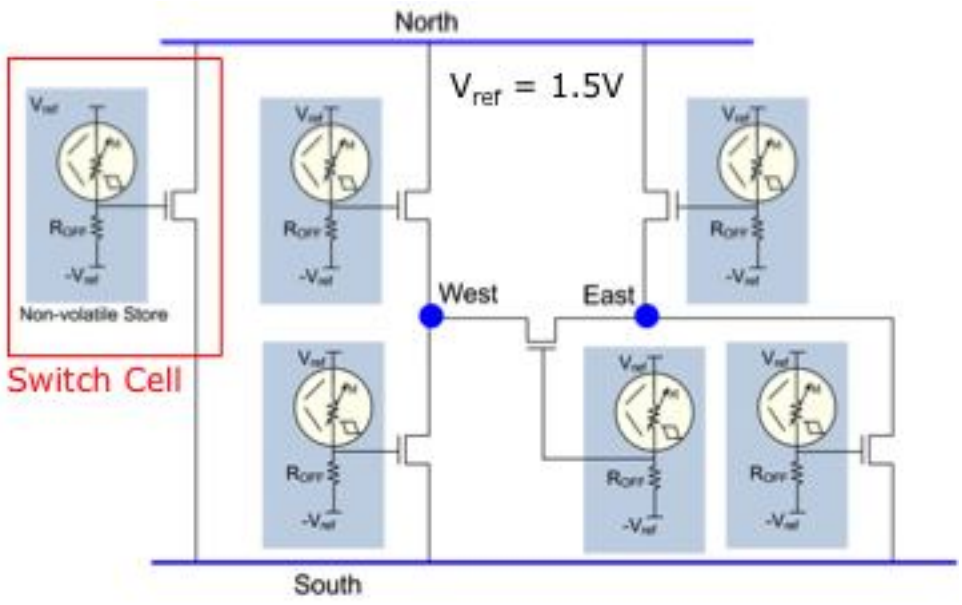



Figure 27. Switch-point schematic showing pass-transistors gated by S-MTJs. The pass-transistors enable/disable a particular connection between two points, controlled by the voltage output of the S-MTJs. Since the resistance state of the S-MTJs is non-volatile, the pass-transistors are programmed persistently.

hardware. Here we implement the switch boxes using non-volatile S-MTJs, which makes it persistent.

The programmable switch-box provides a pathway to connect Bayesian Cells in a reconfigurable manner, and has the ability to route signals from a given input to any of three outgoing directions through programmable *switch-points*. Each switchpoint connects one incoming wire to three outgoing wires, through six pass transistors. This is similar to FPGAs. However, the connection in our approach is made persistent through the use of non-volatile S-MTJ for state storage (see Figure 27). Since it is capable of holding two states; low resistance representing ON and high resistance representing OFF, it can be programmed to enable or disable a particular link simply by storing the corresponding data in the S-MTJ resistance.

The messages through the network are sets of probability vectors associated with diagnostic support for bottom-up and prior support for top-down messages and the propagation supported is through switch-boxes. In our example, if each node supports 4 states, then each of these messages contains 4 sets of probability vectors. Thus each switch-box has to accommodate sufficient switchpoints to allow transmission of all the elements of probability vector sets in parallel. Alternatively, a serial implementation can be realized with a narrow bus sending single probability vectors at a given time. The trade-off involved depends on the area requirement for a switch-box and the resolution requirement of data representation (number of probability digits) vs. performance (latency in communication).

### 4.3 Summary

In this chapter, we introduced a programmable Bayesian Cell-based architecture for physically implementing Bayesian Networks. Each Bayesian Cell directly implements a node in the network, and the links between nodes are physically implemented using physical connections. These connections are made programmable through the use of reconfigurable switch-boxes. All modules in the Bayesian Cell and Switch Box are implemented using non-volatile S-MTJs, leading to persistent circuits without the need for a separate memory store. In the next chapter, we evaluate the proposed architecture and benchmark against software implementations running on CMOS multicore processors.

# CHAPTER 5

# EVALUATION AND BENCHMARKING

## 5.1 HSPICE Device Models

We use HSPICE circuit simulations for validating and evaluating the proposed approach. In order to do this, we first develop HSPICE behavioral device macromodels for the volatile and non-volatile S-MTJ devices. Such macromodels have been used before for other emerging non-volatile resistive devices such as those based on phase-change materials [17]. These macromodels essentially describe the static and dynamic electrical characteristics of the device. Binary S-MTJ devices were used for evaluation in this dissertation. Device characteristics, such as resistance vs. input voltage and switching delay, were extracted with extensive macrospin simulations at room temperature (when thermal noise can disrupt magnetization dynamics) using the stochastic Landau-Lifshitz-Gilbert equation treating thermal agitation as a Gaussian magnetic field by VCU group [10][13]. The behavioral macromodels need to capture the change in S-MTJ resistance for a given range of input programming voltages, as well as the switching delay associated with a given input voltage. For the non-volatile S-MTJs, they need to simulate the persistence in resistance state as well. In the following subsections, we describe our macromodels used to meet these requirements.

### 5.1.1 Volatile S-MTJ HSPICE Macromodel

The DC characteristics of the volatile S-MTJ showing resistance vs. input voltage are shown in Figure 28a. The switching delay vs. input voltage is shown in Figure 28b. HSPICE offers several behavioral constructs to model these characteristics, such as

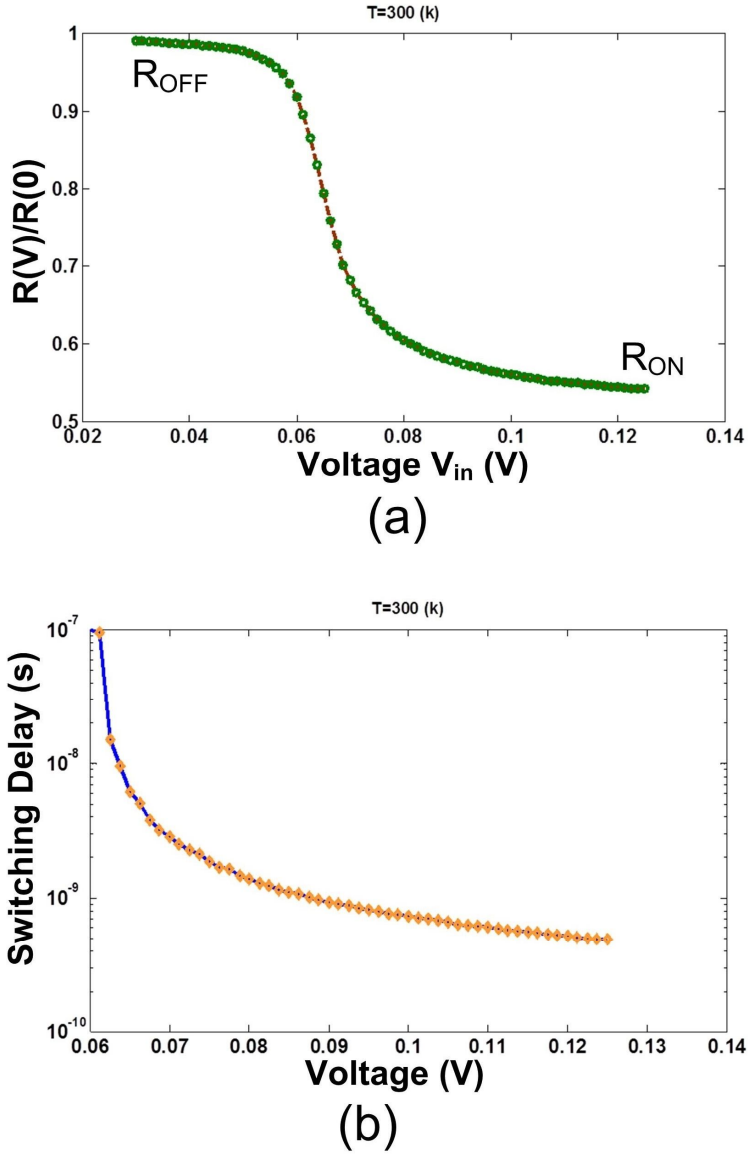



Figure 28. Simulated DC characteristics for volatile S-MTJ device [13]. (a) Resistance vs. input voltage showing two resistance states; and (b) Switching delay vs. input voltage.

voltage/current controlled sources (voltage as well as current sources), voltage controlled resistances, etc. Since we are developing macromodels for voltage-controlled S-MTJs, we use voltage-controlled resistors (VCR) in HSPICE through the use of G-elements [18].

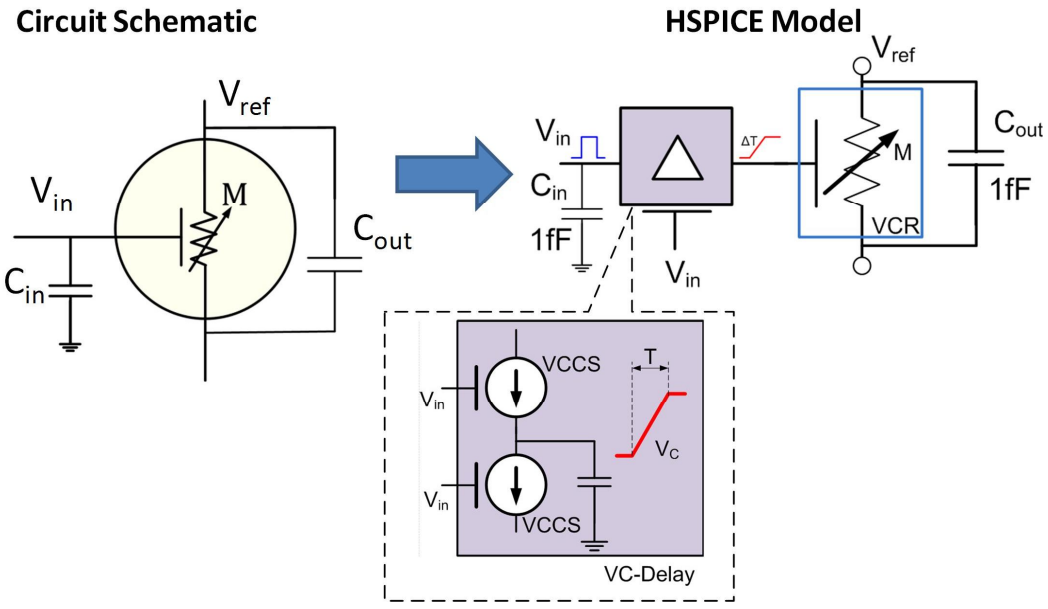



Figure 29. HSPICE behavioral macromodel describing volatile S-MTJ device characteristics for circuit simulation.

The VCR is a behavioral description (in our case, a tabular description) that assigns a resistance value for a given input voltage, in order to model the MTJ resistance. HSPICE then uses a piece-wise linear approximation between the data points provided in the table to complete the behavior for the full voltage range. In order to model the variable switching delay, we define a custom voltage-controlled delay element (VC-Delay) inserted between the input terminals and the control terminal for VCR, using HSPICE voltage-controlled current sources (VCCS) and fixed capacitances as shown in Figure 29. The VCCSs are described using a table that assigns a current value for a given input voltage. The load capacitance at the control node through which this current flows is fixed. Thus the rise-time (delay) of voltage at the control node is linear in relation to the amount of current flowing through the capacitor, which is in turn a function of the applied input voltage.

$$C.\frac{dv}{dt} = I$$
$$\Rightarrow T = CV/I \tag{38}$$

Ideal switches (not shown) are used to control the flow of currents as required. Finally, the parasitic capacitances at the input and between output terminals are added to complete the device macromodel.

### 5.1.2 Non-volatile S-MTJ HSPICE Macromodel

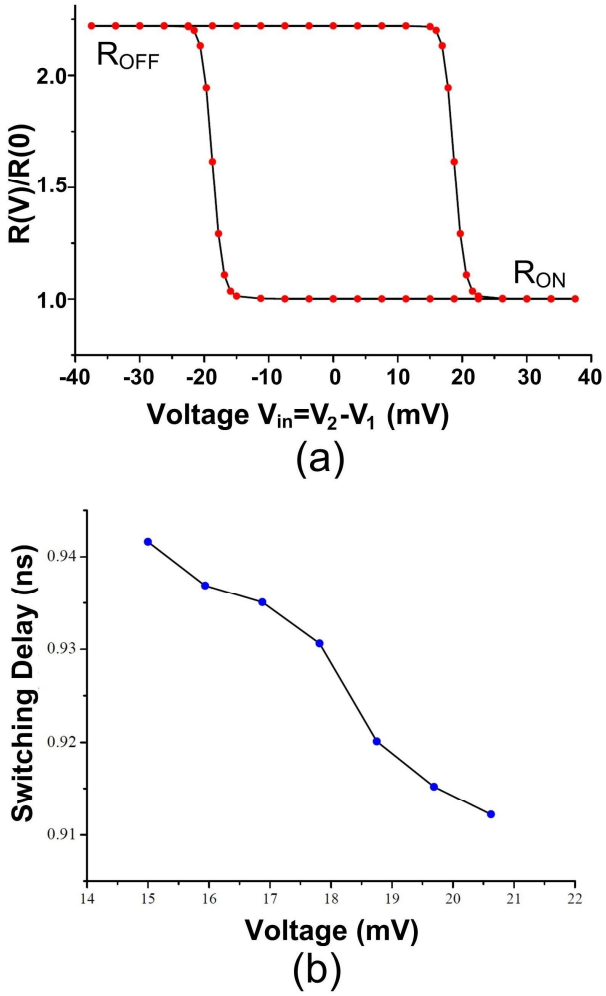



Figure 30. Simulated DC characteristics for non-volatile S-MTJ device [14]. (a) Resistance vs. input voltage showing two stable resistance states and switching threshold voltages; and (b) Switching delay vs. input voltage.

The non-volatile S-MTJ device characteristics are shown in Figure 30. While the resistance and switching delay are similar to volatile device, we need to model the persistence in resistance state (DC curve for resistance vs. input voltage shows hysteresis in Figure 30a). We include this behavior in the behavioral macromodel through several custom constructs described next.

The non-volatile S-MTJ HSPICE macromodel is shown in Figure 31 schematically. First, the resistance vs. input voltage is modeled using two VCRs: the first one models the curve that tracks the switching behavior from high resistance to low resistance state, given that the initial state was high resistance (i.e. when applied input voltage is increasing), and the second VCR models the other case. Each VCR is controlled by a node connected to the output of a voltage controlled delay element, described previously

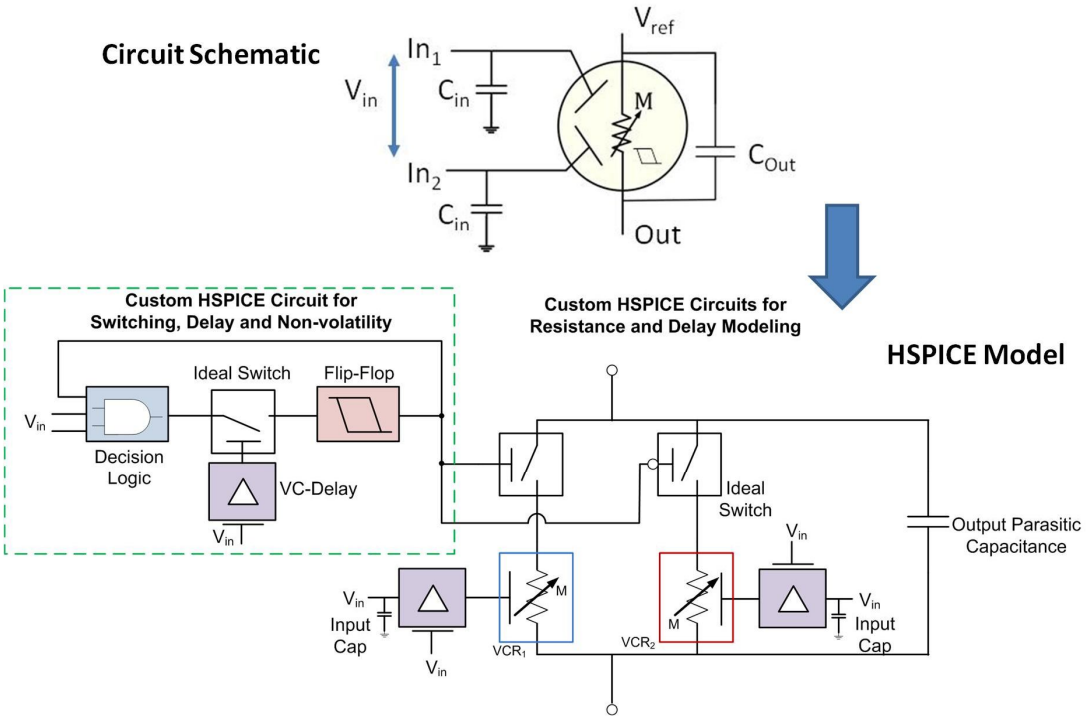



Figure 31. HSPICE behavioral macromodel describing non-volatile S-MTJ device characteristics for circuit simulation.

for volatile device model. In addition, each VCR is connected in series with ideal switches; only one of them is active at any given time and the active switch selects the VCR for the given operating condition.

A decision circuit that incorporates a flip-flop to store the previous state of the device controls these ideal switches. The decision logic takes the current set of program inputs and previous state of the device as inputs, and determines based on these if the state of the device should switch. If the conditions do not meet the switching criteria, the resistance of the S-MTJ does not change, thus showing persistent behavior maintained through the flip-flop. The switching criteria are shown in Table 3. If the conditions allow resistance switching, then the decision logic outputs a switch signal that passes through a voltage-controlled delay element to the flip-flop for state retention. The delay element is

Table 3. Switching Criteria Encoded in Decision Circuit for HSPICE Macromodeling of Non-Volatile S-MTJ

| Input Voltage $V_{in} = (V_1 - V_2)$ | Previous S-MTJ Resistance State | Current S-MTJ Resistance State |
|---|---|---|
| $0 \leq V_{in} < V_{th}$ | $R_{ON}$ | $R_{ON}$ |
| | $R_{OFF}$ | $R_{OFF}$ |
| $V_{th} \leq V_{in} \leq V_{set}$ | $R_{ON}$ | $R_{ON}$ |
| | $R_{OFF}$ | |
| $-V_{th} < V_{in} \leq 0$ | $R_{ON}$ | $R_{ON}$ |
| | $R_{OFF}$ | $R_{OFF}$ |
| $V_{reset} \leq V_{in} \leq V_{th}$ | $R_{ON}$ | $R_{OFF}$ |
| | $R_{OFF}$ | |

used to synchronize the VCR element switch with the dynamic change in resistance of the S-MTJ. Finally, the flip-flop stores the new resistance state. Adding the parasitic capacitances at input and output terminals completes the behavioral model.

### 5.2 Evaluation of Composers used in Bayesian Inference Operations

We use the HSPICE S-MTJ macromodels to validate the functionality and evaluate the proposed Composers (using computational resolution of 0.1) for Bayesian inference operations in terms of power dissipation and latency. The evaluation results are shown in Table 4. For each Composer, we evaluate the worst-case latency that occurs during largest output voltage swing. Using HSPICE simulations, we measure the total settling time at the output as the latency. This is the time measured from the instant the input finishes 90% of its switching transition, to the instant when output settles to within 10% of its final voltage value. For the analog CMOS support circuits, we estimate the delay based on the maximum delay of a minimum-sized voltage follower driving a load of 20pF (equivalent to a Decomposer circuit with 10 digit representation). For cascaded data paths, these latencies are then added up to estimate the total latency for a given path. We estimate the worst-case latencies for all possible paths in a Bayesian Cell, and consider the largest latency as the total delay of a Bayesian Cell. For switchbox, we evaluate the delay of communication through a pass transistor driving a 2fF load (Decomposer input capacitance) through HSPICE simulations.

For area estimation, we use magnet dimensions of 100nm x 90nm to find the area of a single S-MTJ. In order to magnetically isolate neighboring S-MTJs, we include a spacing of 410nm along the minor axis and 400nm along the major axis. These numbers were

Table 4. Evaluation of Composer Circuits for Bayesian Inference (Resolution is 0.1)

| Module | Critical Path Delay (ns) | Area ($\mu m^2$) | Worst-case Power ($\mu W$) |
|---|---|---|---|
| Likelihood Estimation (Multiplication Composers x4) | 144 | 20 | 4.57 |
| Belief Update (Multiplication Composers x4) | 144 | 20 | 4.57 |
| Prior Estimation (Add-multiply Composers x4) | 137 | 50 | 11.24 |
| Diagnostic Support (Add-multiply Composers x4) | 137 | 50 | 11.24 |
| Prior Support (Division Composers x8) | 541.86 | 316 | 90.36 |
| Decomposer (x60) | 132.9 | 240 | 11.37 |
| CMOS Op-Amp (x176) | 100 | 95.4 | 89.32 |
| Bayesian Cell (Critical Path Delay) | 1396.06 | 791.4 | 222.67 |
| Switch Box | 10 | 398.8 | 0.85 |

derived from micromagnetic simulations at VCU that showed the distance at which magnetic interaction is low enough to be ignored for neighboring S-MTJs. This gives us an area of 500nmx500nm for one S-MTJ accounting for spacing requirements as well. The area of CMOS analog support circuits were estimated based on the number of transistors required and the area of a single CMOS transistor in 45nm technology node, accounting for spacing requirements between transistors. Similarly, the area of switch box is estimated based on the number of switch-points and the area of each switch-point (6 S-MTJs + 6 MOSFETs). The total number of switch-points required to accommodate 10 digit messages with 4 states per node was estimated to be 240 per switch-box.

For active power estimation, we measure the total power dissipation for all Composers in a Bayesian Cell using HSPICE simulations. Static power was found to dominate the total power dissipation, since $R_{OFF}/R_{ON}$ is very low for S-MTJs and switching times are relatively long. The worst-case static power dissipation occurs when all S-MTJs are switched ON, also leading to worst-case dynamic power dissipation since output voltage swing is the largest. Due to non-volatility of S-MTJs, a Bayesian Cell can be switched OFF during inactive periods completely. This means there is no stand-by power consumed.

### 5.3 Comparison of BN Inference on Physically Equivalent Implementation vs. Implementation on Multi-core Processors

We use an example of a binary tree BN for benchmarking to illustrate the potential benefits of physically equivalent implementation for BNs vs. conventional abstraction

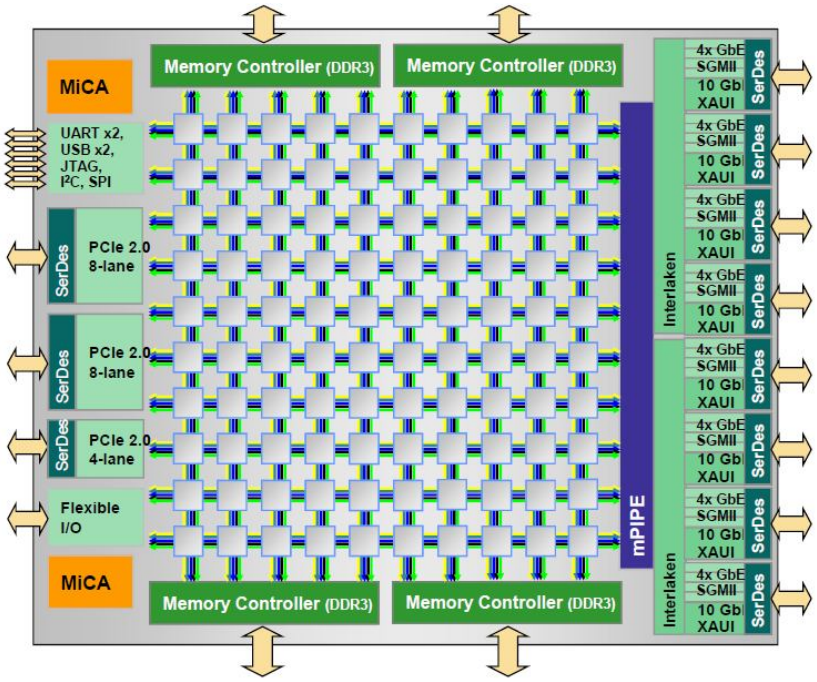



Figure 32. Architecture of a Tilera 100-Core Processor [19].

Table 5. Hardware Specifications for CMOS Multi-core Processors*

| | Notation Used | Parameter Values | |
|---|---|---|---|
| No. of Cores | $C$ | 64 | 100 |
| Clock Speed | $T_{clock}$ | 1.33ns | 0.67ns |
| No. of Arithmetic Pipelines | $p$ | 2 | 2 |
| Size of L2 Cache Line | $S$ | 64B | 64B |
| DRAM Bus Width | $B$ | 64 Bits | 72 Bits |
| DRAM Data Rate | $R$ | 51.2Gbps | 136.5Gbps |
| No. of DRAM Ports | $k$ | 4 | 4 |
| Latency of Cache Miss | $L$ | 80 Clock Cycles | |

*Based on data-sheets from Tilera Corp. [19].

based software implementations on von Neumann machines. The software implementation is expected to run on state-of-the-art multi-core CMOS processors, such as those designed by Tilera Corp. [19]. These processors represent the cutting-edge trend in multi-core processing featuring up to 100 cores on a single chip (see Figure 32), and are well suited to leverage the inherent parallelism available in inference applications.

We estimate the best-case performance of 64-core and 100-core processors for Bayesian operations. For a given BN size in terms of variables (or nodes), the runtime of an inference operation on CMOS processors is estimated based on hardware characteristics and algorithmic requirements (Pearl's Message Propagation algorithm) for computation and memory. Hardware characteristics considered for multi-core processors considered in this work are listed in Table 5. Algorithmic computation requirements for an inference operation are extracted by considering the total number of arithmetic

operations occurring at a node in the BN, and multiplying by the total number of nodes. Operations that can be parallelized are distributed among the cores while the rest of them that depend on results of other operations are serialized, to compute the arithmetic execution time $T_{arith}$. Data memory requirements are identified for each node in the BN and total overhead in servicing the memory requirements, $T_{mem}$, is estimated. The on-chip communication time between cores $T_{comm}$ is neglected, thus taking the best-case scenario for the CMOS microprocessor implementation. CMOS runtime $T_{CMOS}$ is given by the sum of these components. Power and area are taken from datasheets by Tilera [19].

### 5.3.1 Example Bayesian Network

We use a binary tree (shown in Figure 33) for analytical estimation of run time for inference. Each node (or variable) in the BN has 4 states, and each child node has a single parent. This is selected such that target applications like gene expression networks [20], typically requiring 3 states for discrete gene expression levels can be supported. Each node maintains a CPT in addition to belief, likelihood and prior vectors. All the leaf

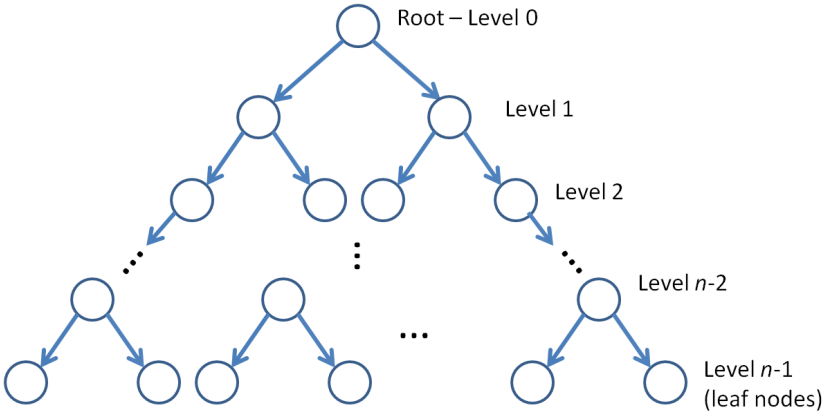



Figure 33. Binary tree with *n*-levels as an example Bayesian Network used for benchmarking proposed physically-equivalent architecture vs. CMOS. Each parent node has 2 child nodes and every node can support 4 states.

nodes are assumed to be the evidence variables in the BN, such that any new evidence triggers an upward propagation of likelihood messages from the leaf nodes all the way to the root of the tree for an inference operation. Through the course of the inference, several messages are propagated in both bottom-up and top-down directions and every node performs several iterations before the process is completed.

### 5.3.2 Analytical Model for Runtime Estimation of BN Inference on CMOS Multicore Processor

**Arithmetic Computation Requirements**

Our multi-core processor analysis is under ideal assumptions for parallelism, resource contention, and performance in general, so it is very optimistic and reflective of best-case performance. In a binary tree, the operations occurring in nodes at a given time step can be executed in parallel. Operations occurring across different time steps cannot be parallelized since belief update at a node depends on the messages propagated from its child/parent nodes. Two regions are identified for a given time-step $l$ based on the number of active nodes $N_l$ in that time step, and the number of processing cores $C$.

Region I: $N_l \geq C$

All cores are active with multiple BN nodes mapped to each core. Assuming that operations are scheduled such that maximum instruction level parallelism is achieved, the arithmetic execution time for this level with $x$ operations per node is given by

$$T_{arith}^{l} = \frac{x.N_l}{C.p} \times T_{clock}. \tag{39}$$

Region II: $N_l < C$

A single BN node is mapped to a core, and the number of active cores is equal to the number of nodes at this level. Using same assumptions as before, the arithmetic execution time is given by

$$T_{arith}^{l} = \frac{x}{p} \times T_{clock}. \tag{40}$$

Given the time for execution of arithmetic operations per time-step, we map out active levels at every step of the algorithm for the example BN. For example, when levels are labeled starting from 1 to *n* for a binary tree with *n* levels, at step 1 only the bottom-most level with leaf nodes (level *n*) is active assuming all evidence variables are leaf nodes. In the second step, level *n*-1 is active. Third step sees levels *n*-2 and *n* as active due to both to-down and bottom-up message propagation. This sequence is mapped out until new messages cease to propagate. An example of this is shown in Table 6 for a binary tree BN with 127 nodes (7 levels; where levels are labeled starting with level 1 at root node through level 7 incorporating all leaf nodes). The arithmetic execution time across different time steps is additive since operations are serialized in time.

The number of operations per node is determined by considering the events occurring at that node. At each node, there are two scenarios considered: (i) Bottom-up message propagation - diagnostic support received from child node(s); and (ii) Top-down message propagation - predictive support received from parent node. For a given node X, the following operations occur when diagnostic support messages are received from its child nodes Y and Z:

Table 6. Sequence of steps for a BN binary tree with 7 levels (127 nodes)

| 127 Nodes | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Step Sequence | Active Level ID | | | | | | | #Active Nodes |
| 1 | 7 | | | | | | | 64 |
| 2 | 6 | | | | | | | 32 |
| 3 | 5 | 7 | | | | | | 80 |
| 4 | 4 | 6 | | | | | | 40 |
| 5 | 3 | 5 | 7 | | | | | 84 |
| 6 | 2 | 4 | 6 | | | | | 42 |
| 7 | 1 | 3 | 5 | 7 | | | | 85 |
| 8 | | 2 | 4 | 6 | | | | 42 |
| 9 | | | 3 | 5 | 7 | | | 84 |
| 10 | | | | 4 | 6 | | | 40 |
| 11 | | | | | 5 | 7 | | 80 |
| 12 | | | | | | 6 | | 32 |
| 13 | | | | | | | 7 | 64 |

(a) Diagnostic support messages from the child nodes are composed to calculate the likelihood vector λ(X) for node X. This operation requires 4 multiplications as follows –

$$\boldsymbol{\lambda(X)} = \boldsymbol{\lambda_Y(X)} * \boldsymbol{\lambda_Z(X)}, \tag{41}$$

where each element in $\boldsymbol{\lambda(X)}$ is given by

$$\lambda_i(X) = \lambda_{Yi}(X) \cdot \lambda_{Zi}(X); i = \{1,2,3,4\}. \tag{42}$$

Based on new evidence, a belief update is performed at node X using likelihood and priors. Computing the elements of the belief vector involves 4 multiplications, 3 additions and 4 divisions as follows –

$$\boldsymbol{BEL(X)} = \alpha\boldsymbol{\pi(X)} * \boldsymbol{\lambda(X)} \tag{43}$$

where each element in $\boldsymbol{BEL(X)}$is given by

$$BEL_i(X) = \frac{\pi_i(X) \cdot \lambda_i(X)}{\sum_{i=1}^{4} \pi_i(X) \cdot \lambda_i(X)}; i = \{1,2,3,4\}. \tag{44}$$

(c) Diagnostic support to parent of node X (say node A) is computed based on λ(X) and the CPT. This requires 16 multiplications and 12 additions as follows –

$$\boldsymbol{\lambda_X(A) = CPT(X|A) \otimes \lambda(X)} \tag{45}$$

where each element in $\boldsymbol{\lambda_X(A)}$ is given by

$$\lambda_{Xi}(A) = \sum_{j=1}^{4} CPT_{ij}(X|A) \cdot \lambda_j(X)\,; i = \{1,2,3,4\}. \tag{46}$$

(d) Predictive support to each child of node X is computed based on computed belief BEL(X) and the likelihood support from the child node. This requires 4 division operations per child node as follows –

$$\boldsymbol{\pi_Y(X) = \frac{BEL(X)}{\lambda_y(X)};\quad \pi_Z(X) = \frac{BEL(X)}{\lambda_Z(X)}.} \tag{47}$$

Here, $\pi_{Yi}(X) = \frac{BEL_i(X)}{\lambda_{Yi}(X)}$; $\pi_{Zi}(X) = \frac{BEL_i(X)}{\lambda_{Zi}(X)}$; $i = \{1,2,3,4\}$.

For a given node X, the following operations occur when predictive support message is received from its parent node A:

(a) Based on new predictive support, the prior vector for node X is calculated as follows:

$$\boldsymbol{\pi(X) = \pi_X(A) \otimes CPT(X|A).} \tag{48}$$

This involves 16 multiplications and 12 additions similar to diagnostic support calculation discussed previously.

(b) Belief update is performed at node X involving 4 multiplications, 3 additions and 4 divisions as discussed earlier.

(c) Predictive support to each child of node X is computed based on computed belief ***BEL(X)*** and the likelihood support from the child node. This requires 4 division operations per child node as discussed earlier.

**Data Memory Requirements**

In order to execute the operations outlined in previous sub-section, each node requires access to data maintained in its CPT, belief vector, likelihood and prior vectors. The data memory requirement per node, *M* (measured in bytes), is then given by

$$M = E_{CPT} \times S_{CPT} + E_{BEL} \times S_{BEL} + E_{\lambda} \times S_{\lambda} + E_{\pi} \times S_{\pi}. \quad (49)$$

Here, $E_i$ denotes the number of entries and $S_i$ denotes the number of bytes per entry for component *i*. Since in our example each node supports 4 states, the CPT has 16 entries (all possible state combinations of the node and its parent), while the likelihood ($\boldsymbol{\lambda}$), prior ($\boldsymbol{\pi}$) and belief (***BEL***) vectors have 4 entries each. The size of each entry is assumed to be 2 bytes.

At every time-step in the algorithm, we determine the data memory required for computations occurring in that step (see Table 5 for example of BN with 127 nodes). This data has to be retrieved from the main memory (DRAM) due to misses in the cache. For every cache miss, a cache line (of size *S* bytes) is brought in from main memory. If latency is *L* cycles, *B* is the DRAM bus-width in bytes and *R* is the data rate in bytes per second, the time to service a cache miss is given by

$$T_{miss} = L \times T_{clock} + \frac{(S - B)}{R}. \quad (50)$$

Here, the data rate *R* is given by *Min(DRAM_data_rate, Chip_network_data_rate)*. If *k* cores can be serviced by the main memory in parallel, the total time to service memory requests for a given time step is estimated by

$$T_{mem}^{l} = \frac{1}{k} \times No.of\ cache\ misses \times T_{miss} \quad (51)$$

$$= \frac{1}{k} \times \frac{N_l \times M}{S} \times T_{miss}.$$

Here $N_l$ is the number of nodes active in a given time-step. Thus total arithmetic execution time for a BN with $n$ levels, and $2n$-1 time steps is given by

$$T_{Exec} = \sum_{l=1}^{2n-1} (T_{arith}^{l} + T_{mem}^{l} + T_{comm}^{l}). \quad (52)$$

For simplicity, the communication cost of transferring the data over the on-chip network is not considered, i.e. $T^{l}_{comm}$ = 0. This is a best-case scenario for CMOS assuming maximum instruction level parallelism can be achieved and does not take into account effects such as network contention, conflict misses, etc.

### 5.3.3 Runtime Estimation of Inference on Proposed Physically Equivalent Architecture

For the proposed physically equivalent architecture, runtime estimation for inference is based on critical path analysis, and area is estimated based on total number of Bayesian Cells and switch boxes for a given size of BN. Worst-case power dissipation is determined by the maximum number of active nodes and power dissipated per node.

We directly implement the BN in hardware and the message propagation algorithm is implemented with the Probability Composer framework. Every node is mapped to a Bayesian Cell that uses Composers to realize the computations for inference and internally maintains the required data using non-volatile S-MTJs. Thus for a binary tree, inference proceeds in an event-driven manner; each level executes the required operations and propagates messages to the neighboring levels. All the computations among nodes at a given time-step are completely parallel. The total number of message

propagation steps required by the algorithm determines the total execution time. This is given by the maximum diameter of the network in trees [1]. If the number of levels is $n$ and execution time per level $T_l$, the total execution time is then given by

$$T_{PEAR} = (2n - 1) \times T_l + T_{comm}. \quad (53)$$

Here, $T_{comm}$ is the latency of communicating probability messages between nodes. In order to estimate the execution time per level in the binary tree, we look at the critical path within a node and consider the worst-case delay for Bayesian Cell (Table 4). Propagating the message to a parent/child node is a near-neighbor voltage communication event, and is calculated by the switch-box delay, as described Section 5.2. This determines the communication delay for one step, and total number of message propagation events multiplied by this number yields the total communication delay.

### 5.4 Benchmarking Results

Our evaluations (see Figure 34) indicate that PEAR can provide 4 orders of magnitude performance speedup over 100-core processors, in supporting BNs with large problem sizes involving random variables in the millions. This is considering the best-case performance scenario for CMOS multi-core processors, and the worst-case delays in proposed physically equivalent architecture for a computational resolution of 0.1. Furthermore, it is able to support real-time intelligence capabilities at ~20 mWs power consumption and very low die area cost of around a few tenths of a $mm^2$ for problem domains in the order of 100 variables. This latter is adequate for many real-world systems such as sensors and automation controllers. Our vision is that every embedded application could incorporate intelligence capability at this problem scale. Alternative implementation using this information representation, probabilistic circuit and

architecture framework may be possible with all-spin devices where the charge current's role is replaced with spin current in All-Spin Probabilistic Composers.

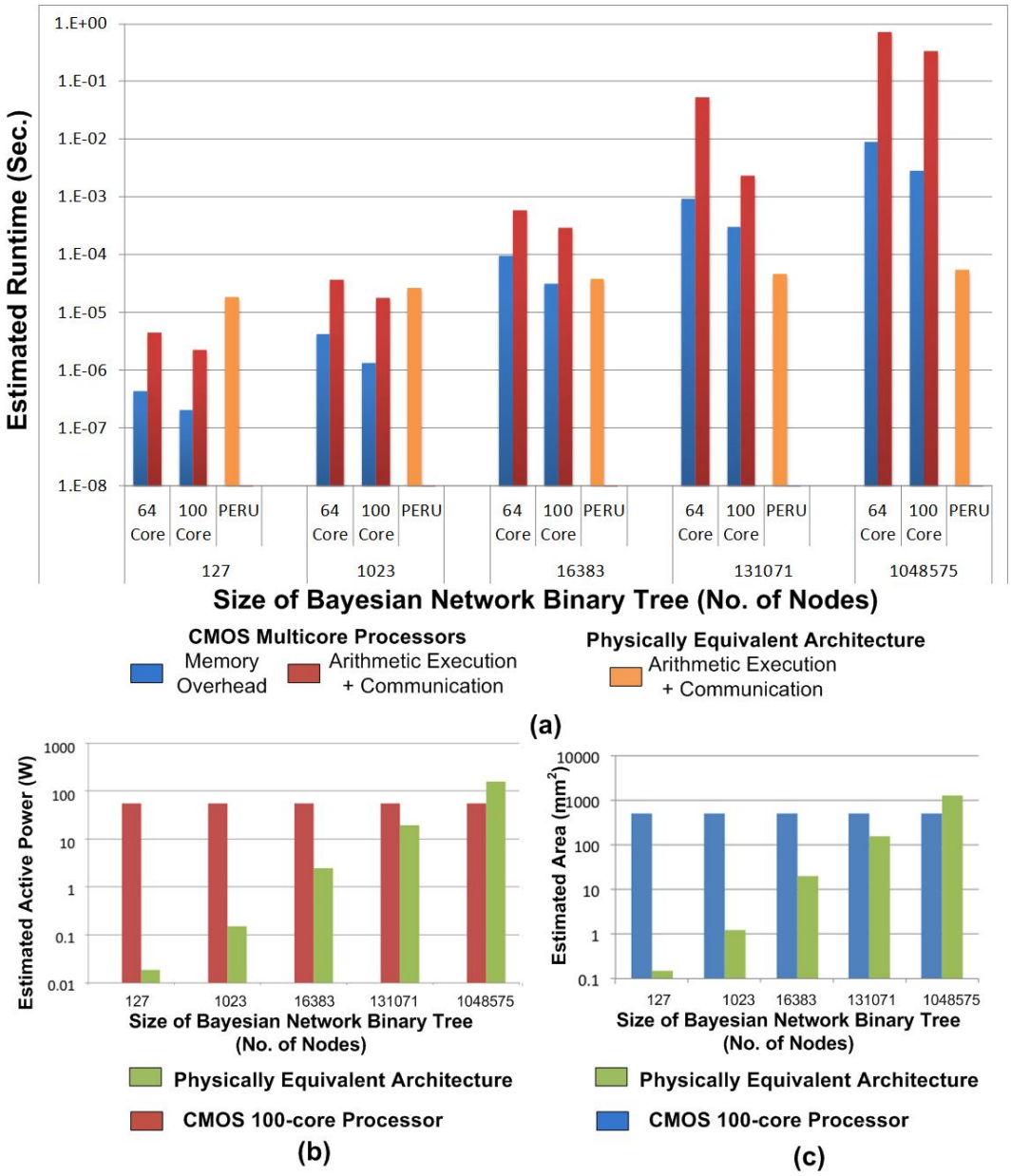



Figure 34. Comparison of BN implementation on CMOS multicore processors and PEAR for Bayesian Inference (Composers use resolution of 0.1). (a) Estimated runtime comparison; (b) Estimated worst-case active power dissipation; and (c) Estimated area for BN implementations of different network sizes.

### 5.5 Discussion on BN Accuracy

In the context of BNs, the definition of accuracy is application-specific. It is determined by several factors such as quality of BN structure (variables and relationships captured), quality of parameters in the model learned from available data or elicited from experts, etc. rather than arithmetic precision alone. It is widely believed in the BN community, with empirical support for some example applications, that BN inference is tolerant to imprecision in numerical parameters.

We quote two studies that were performed to analyze the impact of reducing numerical precision on BN accuracy. The first study was aimed at analyzing the impact of reduced numerical precision of parameters on medical diagnostic systems, conducted by Onisko et al. [31] to study BN sensitivity to numerical precision. The authors use a BN model of HAPAR II diagnostic system for liver disease diagnosis, and systematically reduce the numerical precision to different resolutions by rounding to coarser scales. For each resolution, they determined the percentage of cases from real patient data that were correctly diagnosed (most likely disorder among various modeled disorders given patient data). They concluded that as long as rounding to zero was avoided (by introducing a small error factor), the numerical precision of parameters did not impact the accuracy of diagnoses (see Figure 35). They also repeated this process for several other diagnostic systems [31] and found the same results in all cases. This anecdote indicates that numerical precision alone does not determine the accuracy of BN inference for some of the medical diagnostic applications.

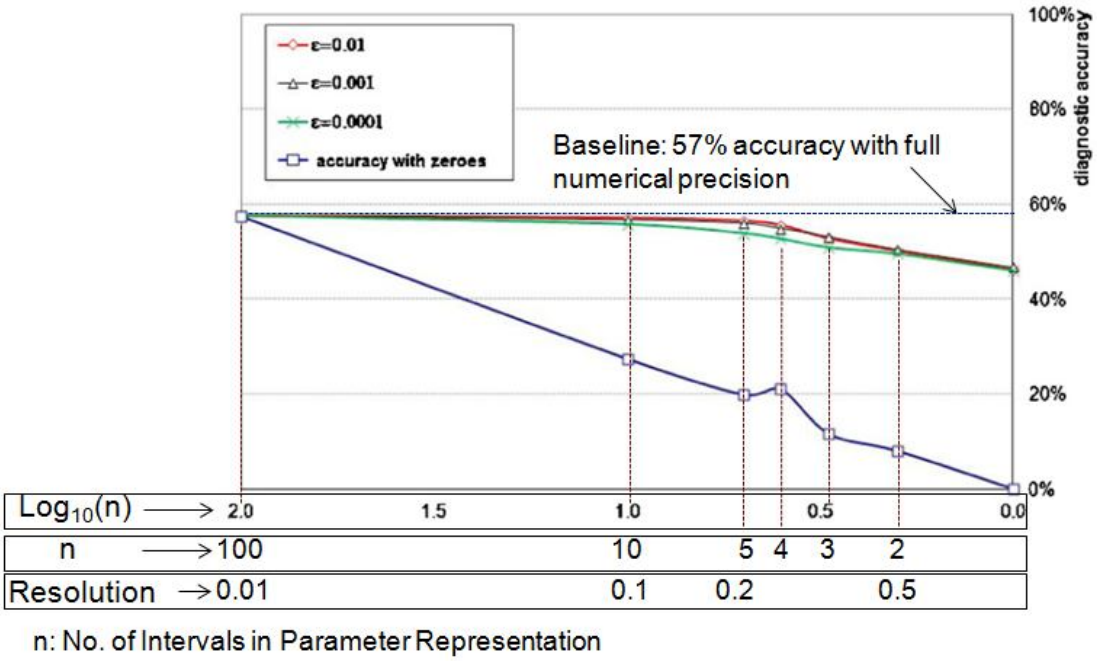



Figure 35. Diagnostic Accuracy vs. Numerical Precision in HEPAR II Bayesian Network for diagnosis of liver diseases. Here, ε represents an error factor added to prevent rounding to zeroes. This figure is adapted from ref. [31].

The second study analyzed the impact of reduced BN numerical precision in image classification applications, motivated by the possibility of using reduced precision BN implementations in hardware for embedded/low-power applications [32]. The authors reduced the bit-width of parameters and studied the impact on classification rate (percentage of correct classifications) on real-world datasets. Again, here they observed that even when using reduced bit-width of 3 to 4 bits for parameters, the classification rate was close to optimal (i.e. with full precision) (see Figure 36).

These empirical studies indicate that there are applications that are not sensitive to numerical precision of BNs for accuracy. In fact, capturing as many variables and relationships as possible for a given problem domain may be more important than supporting full numerical precision. With our proposed approach, we believe that we can

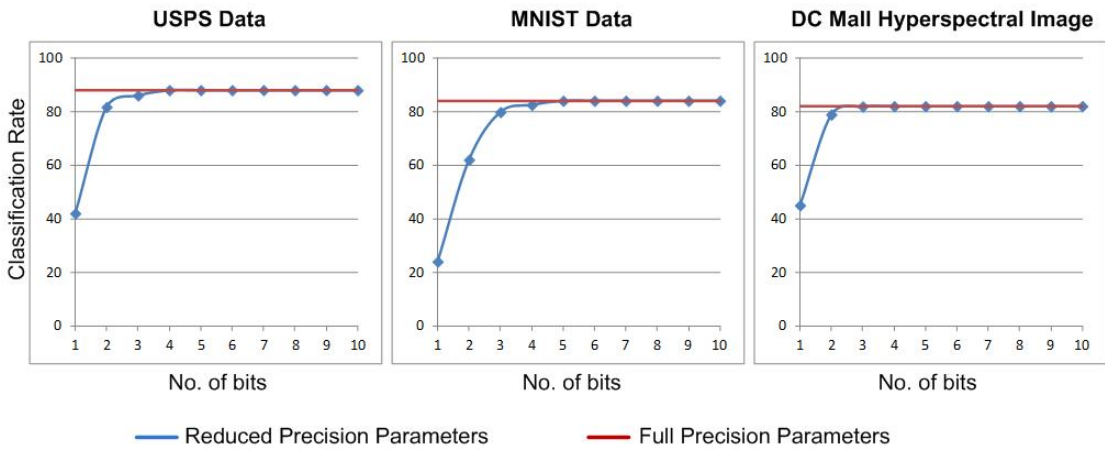



Figure 36. Classification rates for Bayesian Network Classifiers with reduced precision vs. number of bits used to represent parameters, adapted from ref. [32]. Different dataset samples were used in these experiments, as described in ref. [32]. *USPS Data*: This dataset contains 11000 uniformly distributed handwritten digit images from zip codes of mail envelopes. Each digit is represented as a 16x16 grayscale image, where each pixel is considered as feature. *MNIST Data*: This dataset contains 70000 samples of handwritten digits, i.e. 7000 samples of each digit. *DC-Mall Data*: This dataset contains a hyperspectral remote sensing image of the Washington D.C. Mall area. In total, there are 1280x307 hyper-spectral pixels, each containing 191 spectral bands. From these spectral bands, individual pixels are to be classified to one of 7 classes (roof, road, grass, trees, trail, water, or shadow).

do just that by incorporating variables in the order of a million, and still achieve orders of magnitude performance benefits over conventional microprocessors. This could enable learning more complex networks from data than what is possible today, and allow reasoning in real-time.

### 5.5.1 Study on Error Propagation due to Rounding in Binary Tree

In addition to the above studies, we investigate the propagation of errors in an example BN (binary tree that we used to project benefits of our approach) due to rounding of calculated results. Our aim is to identify the degradation in belief values (which is the result of inference) with respect to number of propagation levels in the tree. Given that the resolution of the machine we evaluated in the previous section is 0.1, we

assume that an error of $\pm$0.1 can be tolerated. We built a BN behavioral simulator using C++ for Pearl's belief propagation algorithm based on our proposed implementation. It performs inference calculations for each level in the tree by modeling Composer circuit behavior with resolution of 0.1, and compares against full numerical resolution to compute the error due to rounding at every propagation step.

We take an example BN, which is a binary tree with each variable having two states. Starting from the leaf nodes, we apply all possible combinations of observed states and propagate the evidence in the tree as per Pearl's Belief Propagation algorithm. The inputs at the leaf nodes (evidence) are assumed to be error-free, and any error observed in successive calculations is purely due to numerical rounding of results to a resolution of 0.1. Output statistics are collected at every level, which includes the output combinations with their statistical frequencies of occurrence, and the errors in belief values. We bin the errors into multiple intervals (see Figure 37) to get the distribution for errors in belief at each level.

We perform full simulation of all possible input combinations for levels 0-2 in Figure 38. Due to the explosion in the number of combinations as we go higher up the tree, it becomes infeasible to continue full simulation of all input combinations. After level 3 from the bottom (Figure 38) we use a million different input combinations (limited by the computing resources), selected randomly from the list of input combinations, for every succesive level. Each combination is weighted as per the statistical distribution obtained from the preceeding level, and multiple trials (in this case 12, limited by computing resources) are performed.

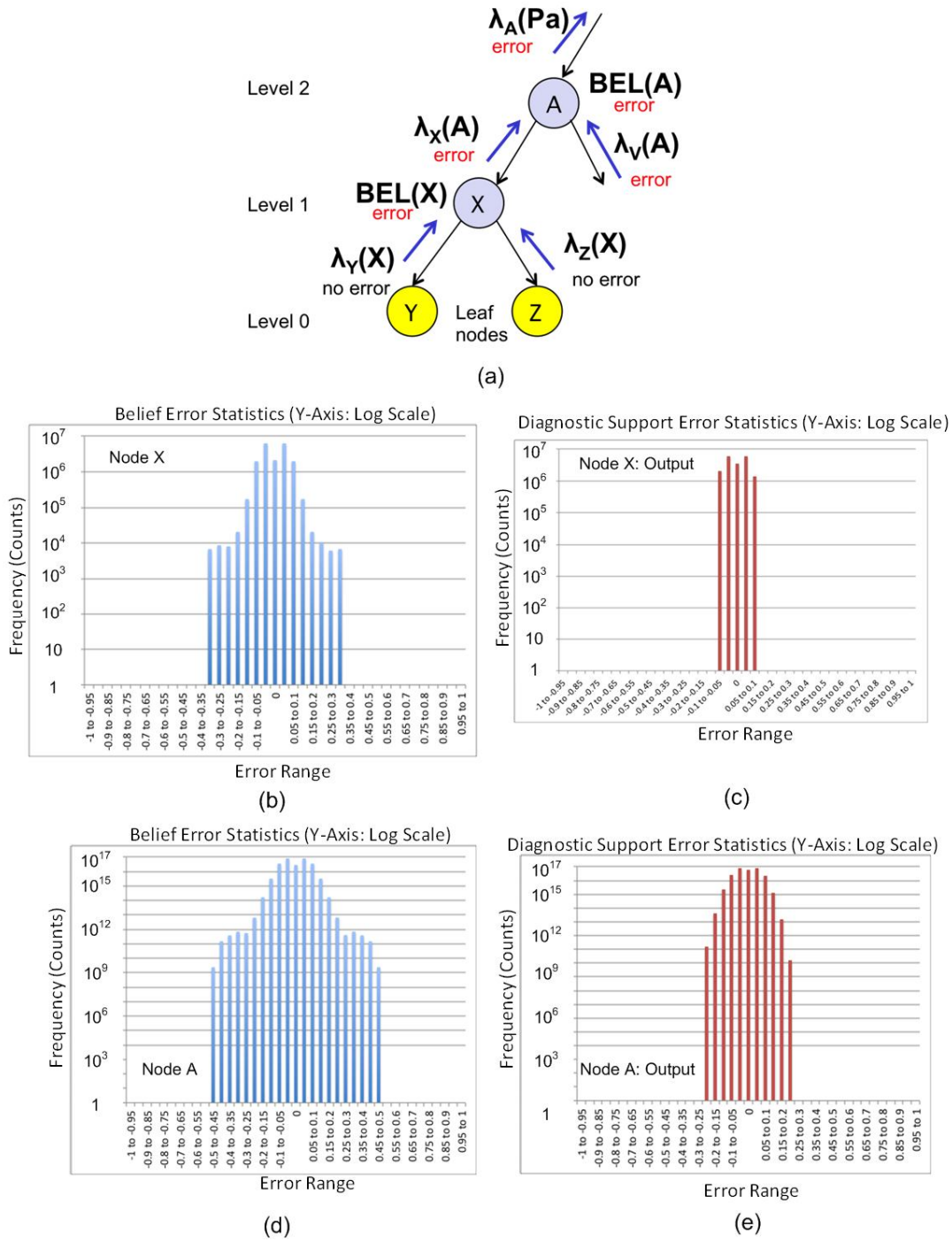



Figure 37. Methodology for study of propagation of errors due to rounding: (a) Figure showing a part of the binary tree BN. Leaf nodes are evidence variables and are assumed to have no errors in observations. Rounding errors start occurring from level 1 in belief calculations for each node and diagnostic support messages at the output of each node. (b) Rounding error statistics for belief at node X in (a). (c) Error statistics for diagnostic support message from node X to node A. (d), (e) Error statistics for belief and diagnostic support respectively at node A.

At each level, we calculate the percentage of belief values that lie within an error of ±0.1. For the levels where inputs were randomly sampled, we calculate the average value from all trials and measure the sample standard deviation. From Figure 38, we see that even at level 6 (which is up to 127 nodes in BN), the % of cases with errors in belief values within ±0.1 is 94.8%. Further levels show increasing standard deviation, indicating that the samples used are not sufficient. This study shows that atleast for a BN with about 100 nodes, the error propagation in belief values due to rounding is not severe. As discussed earlier, applications exist where a resolution of 0.1 is tolerable and provides an accuracy that is close to optimal. However, for larger networks with more than 10000 nodes, the accuracy rate falls to about 60%. Due to increasing standard deviation, it is difficult to conclusively state the accuracy rate for these large networks. Further study guided by application context is required to understand the impact on BNs with larger

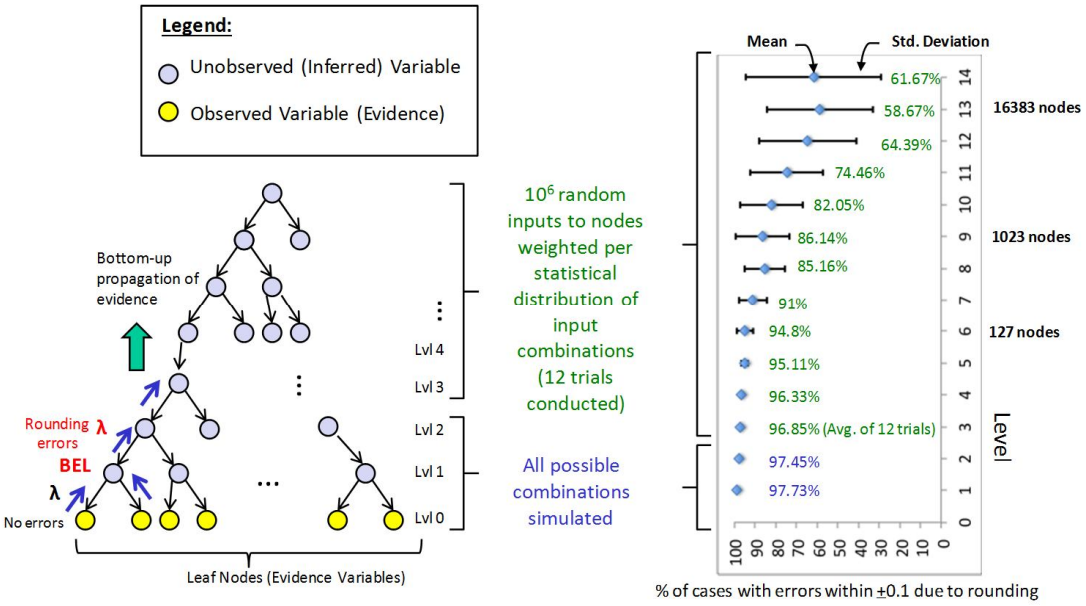



Figure 38. Results of error propagation study: (left) Binary tree BN considered; and (right) Error statistics for each level showing % of cases with errors in belief values within ±0.1 (resolution of the example circuits used in this work). From level 7 onwards, increasing standard deviation indicates more samples are required.

number of variables.

It should be noted that the benefits of the proposed approach highlighted in the Section 5.4 are valid as long as applications can tolerate a resolution of 0.1 for Bayesian inference. While some applications have been shown to be amenable to the proposed implementation, problem domains with large number of variables (several tens of thousands to million) would need to exercise caution when using reduced numerical resolution. Depending on the application requirements, different hybrid schemes may be implemented where critical variables or inference paths use higher computational resolution than others (discussed in a subsequent section). These schemes would incur a trade-off between computational resolution and area/power impact on each BC, and thus may limit the maximum number of variables that can be supported using the proposed approach. Alternatively, each BC may implement all computations in the analog domain with decomposers only at the output interface between BCs to maintain computational resolution. This would, however, depend on the noise sources for magneto-electric devices and available noise margins to be identified as research on these unconventional devices progresses.

### 5.5.2 Effect of Errors due to Probabilistic Switching of S-MTJs

When an input is applied to a S-MTJ for switching, there is a finite probability of switching failure associated with it due to random thermal fluctuations [14]. Ths S-MTJ can be designed to minimize this switching probability, and it has been shown that switching error probability is as low as $10^{-6}$ [14]. We analyze the impact of switching failures on BN accuracy in this section.

We follow a similar mindset as before, used for evaluating impact of reduced numerical precision on example BN binary tree. Here, we extend the simulation framework to include errors due to switching failure for every S-MTJ in Probability Arithmetic Composers associated with BN inference. We model an S-MTJ switching event as a binary random variable with two states (switching is *true* or *false*) using a Bernoulli distribution, which takes the probability of correct switching as a parameter. For each Composer, the number of S-MTJs that need to switch is a function of the applied input probability value. For every S-MTJ that is required to switch, we sample the switching event from the parameterized Bernoulli distribution. In case of switching failure, we add an error to the computation result that is proportional to the number of S-MTJs that failed to switch (for a computational resolution of 0.1, every S-MTJ switching failure results in an error of 0.1).

We analyze the impact of error propagation due to both rounding and S-MTJ switching failures in a BN binary tree, for a range of S-MTJ switching error probabilities

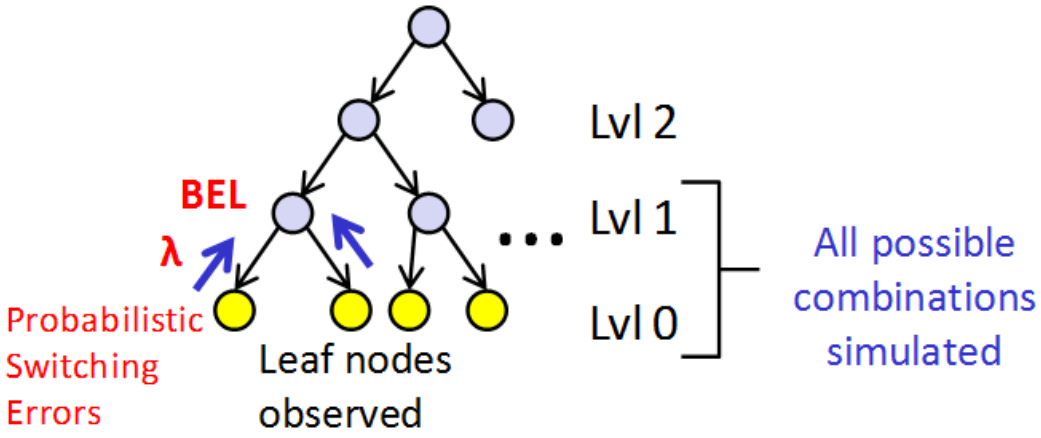



Figure 39. Methodology for analyzing error propagation due to conjunction of rounding errors and S-MTJ switching failures. The impact of switching failures starts to appear in the leaf nodes, even with the assumption that input observations are error-free.

Table 7. Impact of S-MTJ switching errors and rounding errors on belief values at Level 1 in the binary tree BN (Figure 39)

| S-MTJ Switching Error Probability | Belief Error Statistics at Level 1 (% of cases with error within $\pm$0.1) |
|---|---|
| $10^{-3}$ | 96.75 |
| $10^{-4}$ | 97.63 |
| $10^{-5}$ | 97.72 |
| $10^{-6}$ | 97.733 |
| 0 | 97.734 |

from $10^{-3}$ to $10^{-6}$. For the leaf nodes, the inputs are considered to be error-free assuming that observations on the state of these variables is error-free. However S-MTJ switching failures can introduce errors even in the leaf nodes, which is taken into account (Figure 39). Then every successive computation takes S-MTJ switching errors into account in addition to rounding and propagates the result. The results for levels 0 to 1 are shown in Table 7. We see that for the target error probability of $10^{-6}$, the impact due to S-MTJ switching failures is minial when compared to impact due to rounding errors. As the switching error rate increases, it introduces a more significant degradation in error accumulation.

## 5.6 Improving Computational Resolution for Probability Composers and Decomposers

In this work, we used Composers with a resolution of 0.1 as an example, since applications exist as mentioned in earlier sections that can tolerate this resolution. However, the resolution for computation can be improved further by increasing the number of S-MTJs used in each Composer and Decomposer. There is an inverse relationship between the computational resolution and the nuber of devices in a

Composer (Figure 40a-b shows the relationship for a 2-state S-MTJ based Composer). The highest resolution achievable is limited by the number of input voltage levels that the Decomposer can succesfully distinguish.

Theoretically speaking, this limit can be estimated based on the switching characteristics of non-volatile S-MTJs. As shown in Figure 6, there is a window of about 4mV during which the S-MTJ is in the process of switching. Thus when a Decomposer is switching a non-volatile S-MTJ in its successive stage, it can theoretically distinguish a voltage difference of 4mV at its input assuming ideal conditions and no external/thermal noise. This would allow a theoretical computational resolution of up to 0.005 (about 200 voltage intervals between 0-1V). However in practice, the presence of other factors such as noise would limit the resolution. Noise sources for magneto-electric circuits are still being researched and more information will become available to be used for this analysis as research in this field progresses.

In our work, we estimate the impact of improving the computational resolution on a Bayesian Cell and Switch Box (one of each for every variable), for a resolution of up to 0.01 (see Figure 40). As the number of S-MTJs used in each Probability Composer/Decomposer increases, it has a linear impact on the area, power and delay for each Bayesian Composer. Using this, we project the impact for various computational resolutions on area and the power-delay product (energy efficiency) per variable (Bayesian Cell and Switch Box) using our proposed approach. The estimated impact is shown in Figure 40c-d. As we can see, there is a significant improvement in resolution for a modest cost for down to 0.02. After that, the area and energy costs rapidly increase for miminal resolution gains.

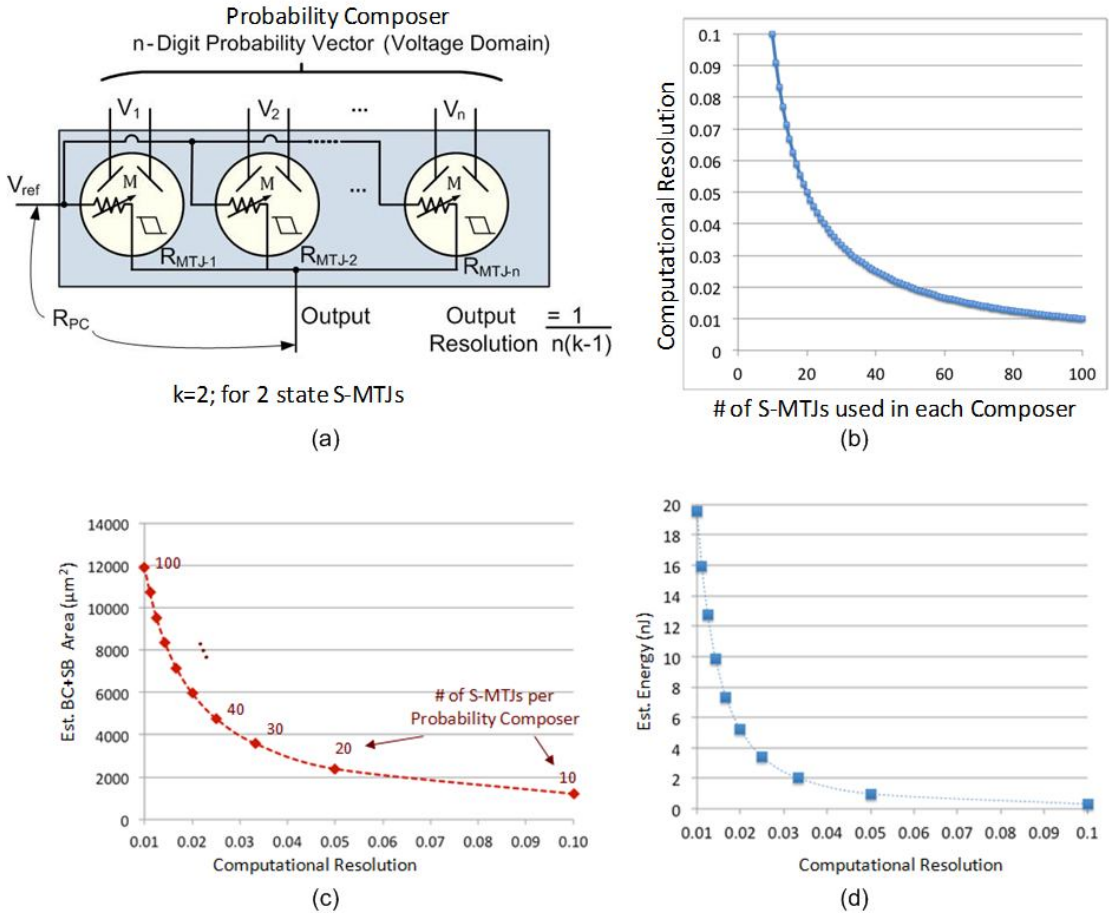



Figure 40. Improving computational resolution in Probability Composers. (a) Probability Composer schematic, and (b) Graph showing relationship between computational resolution and number of S-MTJ devices used in Composer circuits; (c) Estimated area, and (d) Estimated energy per Bayesian Cell for various computational resolutions.

This feature of improving resolution through number of S-MTJs provides opportunities per application requirements, where heterogeneous schemes may be explored with critical variables using higher resolution than other variables. There may be other avenues for improving computational resolution as well. One possible approach would be to use multi-state S-MTJs as they become available in future. Other approaches could look at using weighted number representations for probability data representation rather than flat non-weighted spatial vectors used in this work. However, such schemes

may be impacted significantly by S-MTJ switching failures depending on the position at which a failure occurs. Further improvements in S-MTJ switching reliability may enable such weighted number schemes to be used in future. Alternative emerging devices may present other avenues for implementing physically equivalent systems for machine intelligence at nanoscale.

# CHAPTER 6

# CONCLUSION

In this dissertation, we introduced the concept of physical equivalence for hardware implementation of unconventional computing frameworks for enabling machine intelligence. We illustrated the approach for Bayesian Networks (BNs), which is a highly successful and widely used probabilistic formalism for reasoning under uncertainty. We used physical equivalence at all layers starting from data representation, to non-volatile Probability Composer circuits that operate on probabilities directly using mixed-signal arithmetic, and finally a non von Neumann reconfigurable architecture that is capable of directly mapping BNs to hardware using Bayesian Cells for implementing the nodes and reconfigurable switch box based routing for implementing the links. We presented details on implementation of the Bayesian Inference Engine that peforms computations involved during Inference operation using Pearl's Belief Propagation algorithm.

We showed that due to computation-in-memory capability enabled by emerging straintronic MTJ devices and the proposed mangeto-electric mixed signal circuit framework, we can implement large-scale distributed reasoning system using Physically Equivalent Architecture. The projected benefits in terms of runtime was up to 4 orders of magnitude when compared to state-of-the-art CMOS 100 core processors, for a BN with up to a million variables when Composers used a resolution of 0.1. We also evaluated the propagation of errors in an example binary tree BN due to rounding to 0.1 computational resolution and probabilistic S-MTJ switching failures. The impact of S-MTJ switching failures was overshadowed by rounding errors when the S-MTJ switching error rate was

1 in $10^6$. We also evaluated the impact of increasing computational resolution on area and energy efficiency of a Bayesian Cell.

Future directions could explore specific applications using the proposed framework. Heterogeneous schemes may be explored with critical variables using higher resolution than other variables through some of the concepts discussed in this thesis. The proposed architecture may be extended with an implementation for a Learning Engine in a Bayesian Cell. Bayesian Network learning consists of parameter learning and structure learning. Learning tasks are performed on available data sets for a given problem and involve repeated inference tasks. This is where the tremendous performance benefit is expected to be leveraged.

The complexity of learning depends on the characteristics of available data sets. The simplest scenario for learning is when the structure of a BN is known, but parameters need to be learnt from data sets that are complete (all variables are observed and every data set has values assigned to all variables). The parameter learning task then reduces to a statistical estimation of joing probabilities of a parent-child state combination over the entire data set. A second scenario is when given a BN structure, the parameters need to be learnt from an incomplete data set. Here, incomplete data set means that some variables are missing assignments in the observations. Assuming that they are missing at random [16] (which is a typical assumption made to make the parameter estimation task tractable), the parameters of the BN can be estimated using iterative algorithms, such as the Estimation-Maximization (EM) algorithm [16]. In this scenario, an inference operation is performed to estimate the missing probabilitied for every data set so as to

complete the missing assignments. Then once the data set is completed, the probabilities for parent-child state combinations are estimated statistically.

The final scenario is when both structure and parameters of BN are unknown and need to be estimated from data. This has the highest complexity since the number of candidate structures is super exponential in the number of variables of a BN. Several heuritic techniques are used to narrow the search space for candidate graphs, such as using search-and-score methods with Hill-Climbing algorithm [16]. Then for every candidate graph, the parameters are estimated using EM algorithm. This approach however can easily get intractable for BNs of large sizes. A different approach was suggested that altenated between structure search and parameter estimation, called the Alternating Model Selection EM (AMS-EM) algorithm [21]. The learning operations are further complicated when the data is incomplete.

Practical situations typically have incomplete data sets from which either parameter or structure or both need to be estimated. BN learning then involves performing several inference runs to estimate the missing values in the data sets. As shown earlier in this thesis, the physically equivalent implementation for BNs shows up to 4 orders of magnitude performance improvement for BN inference over convetional software implementation using multi-core processors. Thus when learning is incorporated, it is expected to lead to tremendous performance benefits due to iterative inference operations involved, enabling critical real-world applications that may be infeasible today. Alternative emerging devices may present other avenues for implementing physically equivalent systems for machine intelligence at nanoscale.

## APPENDIX

## LIST OF PUBLICATIONS

1. **S. Khasanvis**, M. Li, M. Rahman, M. S. Fashami, A. Biswas, J. Atulasimha, S. Bandhyopadhay, and C. A. Moritz, "Self-similar Magneto-electric Nanocircuit Technology for Probabilistic Inference Engines," *IEEE Transactions on Nanotechnology, Special Issue on Cognitive and Natural Computing with Nanotechnology*, in press, 2015.
2. **S. Khasanvis**, M. Rahman, and C. A. Moritz. "Low-Power Heterogeneous Graphene Nanoribbon-CMOS Multistate Volatile Memory Circuit", *ACM Journal on Emerging Technologies in Computing, Special Issue on Advances in Design of Ultra-Low Power Circuits and Systems in Emerging Technologies*, in press, 2015.
3. **S. Khasanvis**, M. Li, M. Rahman, M. S. Fashami, A. K. Biswas, J. Atulasimha, S. Bandhyopadhay, and C. A. Moritz, "Physically Equivalent Magneto-electric Nanoarchitecture for Reasoning under Uncertainty," *in proceedings of the 11th IEEE/ACM International Symposium on Nanoscale Architectures*, in press, 2015.
4. **S. Khasanvis**, M. Rahman, M. Li, J, Shi, and C. A. Moritz, "Architecting Connectivity for Fine-grained 3-D Vertically Integrated Circuits," *in proceedings of the 11th IEEE/ACM International Symposium on Nanoscale Architectures*, in press, 2015.
5. M. Rahman, **S. Khasanvis**, and C. A. Moritz, "Nanowire Volatile RAM as an Alternative to SRAM", *ACM Journal on Emerging Technologies in Computing, Special Issue on Advances in Design of Ultra-Low Power Circuits and Systems in Emerging Technologies*, in press, 2015. Preprint available at- http://arxiv.org/abs/1404.0615.
6. M. Rahman, J. Shi, M. Li, **S. Khasanvis**, and C. Andras Moritz, "Manufacturing Pathway and Experimental Demonstration for Nanoscale Fine-Grained 3-D Integrated Circuit Fabric," *in proceedings of the 15th IEEE International Conference on Nanotechnology*, in press, 2015.
7. M. Rahman, **S. Khasanvis**, J. Shi, M. Li, and C. A. Moritz, "Architecting 3-D Integrated Circuit Fabric with Intrinsic Thermal Management Features," *in proceedings of the 11th IEEE/ACM International Symposium on Nanoscale Architectures*, in press, 2015.
8. J. Shi, M. Li, M. Rahman, **S. Khasanvis**, and C. A. Moritz, "Architecting NP-Dynamic Skybridge," *in proceedings of the 11th IEEE/ACM International Symposium on Nanoscale Architectures*, in press, 2015.
9. M. Rahman, **S. Khasanvis**, J. Shi, and C. A. Moritz, "Wave Interference Functions for Neuromorphic Computing," *IEEE Transactions on Nanotechnology, Special Issue on Cognitive and Natural Computing with Nanotechnology*, in press, 2015.

10. M. Rahman, **S. Khasanvis**, J. Shi, M. Li, and C. A. Moritz, "Skybridge: 3-D Integrated Circuit Technology Alternative to CMOS," 2014. Available Online- http://arxiv.org/abs/1404.0607.

11. **S. Khasanvis**, M. Rahman, and C. A. Moritz, "Heterogeneous Graphene-CMOS Ternary Content Addressable Memory," *Elsevier Journal of Parallel and Distributed Computing*, *Special Issue on Computing with Future Nanotechnology*, vol. 74, no. 6, pp. 2497-2503, 2014.

12. **S. Khasanvis**, M. Rahman, S. N. Rajapandian, and C. A. Moritz, "Wave-based Multi-valued Computation Framework", *in proceedings of the 10th IEEE/ACM International Symposium on Nanoscale Architectures*, pp.171-176, 2014. **(Best Paper Award)**

13. M. Rahman, M. Li, J. Shi, **S. Khasanvis** and C. A. Moritz, "A New Tunnel-FET based RAM Concept for Ultra-Low Power Applications", *in proceedings of the 10th IEEE/ACM International Symposium on Nanoscale Architectures*, pp. 57-58, 2014.

14. J. Zhang, M. Rahman, P. Narayanan, **S. Khasanvis**, and C. A. Moritz, "Parameter Variation Sensing and Estimation in Nanoscale Fabrics", *Elsevier Journal of Parallel and Distributed Computing*, *Special Issue on Computing with Future Nanotechnology*, vol. 74, no. 6, pp. 2504-2511, 2014.

15. **S. Khasanvis**, M. Rahman, P. Shabadi, P. Narayanan, H. S. Yu, C. O. Chui, and C. A. Moritz, "Nanowire Field-Programmable Computing Platform", *in proceedings of the 9th IEEE/ACM International Symposium on Nanoscale Architectures*, pp. 23-25, 2013.

16. M. Rahman, P. Narayanan, **S. Khasanvis**, J. Nicholson, and C. A. Moritz, "Experimental Prototyping of Beyond-CMOS Nanowire Computing Fabrics", *in proceedings of the 9th IEEE/ACM International Symposium on Nanoscale Architectures*, pp. 134-139, 2013. **(Best Paper Award)**

17. **S. Khasanvis**, S. N. Rajapandian, P. Shabadi, J. Shi, C. A. Moritz, "Embedded Processors based on Spin Wave Functions (SPWFs)", *in proceedings of the 9th IEEE/ACM International Symposium on Nanoscale Architectures*, pp. 32-33, 2013.

18. **S. Khasanvis**, K. M. M.Habib, M. Rahman, P. Narayanan, R. Lake and C. A. Moritz, "Ternary Volatile Random Access Memory based on Heterogeneous Graphene-CMOS Fabric", *in proceedings of the 8th IEEE/ACM International Symposium on Nanoscale Architectures*, pp. 69-76, 2012.

19. P. Shabadi, **S. Khasanvis**, S. N. Rajapandian, C. A. Moritz, et al., "Spin Wave Nanofabric Update", *in proceedings of the 8th IEEE/ACM International Symposium on Nanoscale Architectures*, pp. 196-202, 2012.

20. J. Zhang, P. Narayanan, **S. Khasanvis**, J. Kina, C. O. Chui, and C. A. Moritz, "On-Chip Variation Sensor for Systematic Variation Estimation in Nanoscale Fabrics", *in proceedings of the 12th IEEE International Conference on Nanotechnology*, pp. 1-6, 2012.

21. P. Shabadi, S. Rajapandian, **S. Khasanvis**, and C. A. Moritz, "Design of Spin Wave Functions Based Logic Circuits", *SPIN, Special Issue on Recent Progress in Spintronic Devices, World Scientific Publishing Company*, vol. 2, no. 3, pp. 1240006-1—1240006-9, 2012.

22. **S. Khasanvis**, K. M. M. Habib, M. Rahman, P. Narayanan, R. K. Lake and C. Andras Moritz, "Hybrid Graphene Nanoribbon-CMOS Tunneling Volatile Memory Fabric", *in proceedings of the 7th IEEE/ACM International Symposium on Nanoscale Architectures*, pp. 189-195, 2011.

## BIBLIOGRAPHY


[1] J. Pearl, *Probabilistic reasoning in intelligent systems: Networks of plausible inference*, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1988.

[2] A. Darwiche, *Modeling and reasoning with Bayesian networks*, Cambridge University Press, 2009.

[3] K. B. Korb, A. E. Nicholson, *Bayesian artificial intelligence*, 2nd ed., CRC Press, 2011.

[4] P. Sebastiani, M. F. Ramoni, V. Nolan, C. T. Baldwin, and M. H. Steinberg, "Genetic dissection and prognostic modeling of overt stroke in sickle cell anemia," *Nat. Genet.*, 37, pp. 435–440, 2005.

[5] C. Su, A. Andrew, M. R. Karagas, and M. E. Borsuk, "Using Bayesian networks to discover relations between genes, environment, and disease," *BioData Mining*, vol. 6, no. 6, pp. 6-1—6-21, 2013.

[6] P. Lucas, "Bayesian networks in medicine: a model-based approach to medical decision making," in proceedings of the EUNITE workshop on intelligent systems in patient care, pp. 1-6, 2001. Available Online [Retrieved on Nov. 28, 2014]: http://cs.ru.nl/~peterl/eunite.pdf.

[7] Valdes and K. Skinner, "Adaptive, model-based monitoring for cyber attack detection," in *Recent Advances in Intrusion Detection (RAID)*, pp. 80–92, 2000.

[8] M. L. de Campos, and A. E. Romero, "Bayesian network models for hierarchical text classification from a thesaurus," *International Journal of Approximate Reasoning*, vol. 50, issue 7, pp. 932-944, July 2009.

[9] G. J. Brostow, R. Cipolla, "Unsupervised Bayesian detection of independent motion in crowds," in proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp.594-601, 17-22 June 2006.

[10] K. Roy, S. Bandyopadhyay and J. Atulasimha, "Energy dissipation and switching delay in stress-induced switching of multiferroic nanomagnets in the presence of thermal fluctuations," *Journal of Applied Physics*, vol. 112, p. 023914-1—023914-8, 2012.

[11] K. Roy, S. Bandyopadhyay and J. Atulasimha, "Hybrid spintronics and straintronics: A magnetic technology for ultralow energy computing and signal processing," *Applied Physics Letters*, vol. 99, pp. 063108-1—063108-3, 2011.

[12] J. Atulasimha and S. Bandyopadhyay, "Bennett clocking of nanomagnetic logic using multiferroic single domain nanomagnets," *Applied Physics Letters*, vol. 97, pp. 173105-1—173105-3, 2010.

[13] M. S. Fashami, J. Atulasimha and S. Bandyopadhyay, "Magnetization dynamics, throughput and energy dissipation in universal multiferroic nanomagnetic logic gate with fan-in and fan-out," *Nanotechnology*, vol. 23, no. 10, pp. 105201-1—105201-10, 2012.

[14] A. K. Biswas, S. Bandyopadhyay, and J. Atulasimha, "Complete magnetization reversal in a magnetostrictive nanomagnet with voltage-generated stress: A reliable energy-efficient non-volatile magneto-elastic memory," *Applied Physics Letters* vol. 105, pp. 072408-1—072408-5, 2014.

[15] C. Tomazou, F. J. Lidgey, and D. G. Haigh, editors, *Analogue IC design: The current-mode approach*, Peregrinus, Stevenage, Herts., UK, 1990.

[16] D. Koller and N. Friedman, *Probabilistic graphical models: Principles and techniques*, MIT Press, 2009.

[17] P. Junsangsri, F. Lombardi, and J. Han, "Macromodeling a phase change memory (PCM) cell by HSPICE," in proceedings of IEEE/ACM International Symposium Nanoscale Architectures (NANOARCH), pp.77-84, 4-6 July 2012.

[18] Synopsys *HSPICE® User guide: Simulation and analysis*, Version B-2008.09, September 2008. Available Online [Retrieved on Nov. 28, 2014]: http://cseweb.ucsd.edu/classes/wi10/cse241a/assign/hspice_sa.pdf.

[19] C. Ramey, "TILE-Gx100 manycore processor: Acceleration interfaces and architecture", Aug. 2011, Tilera Corporation. Available Online [Retrieved on Nov. 28, 2014]: http://www.hotchips.org/wpcontent/uploads/hc_archives/hc23/HC23.18.2-security/HC23.18.220-TILE-GX100-Ramey-Tilera-e.pdf

[20] N. Friedman, M. Linial, I. Nachman, and D. Pe'er, "Using Bayesian networks to analyze expression data," *Journal of Computational Biology*, vol. 7(3-4), pp. 601-620, 2000.

[21] N. Friedman, "Learning belief networks in the presence of missing values and hidden variables," in proceedings of the fourteenth International Conference on Machine Learning (ICML97), San Francisco: Morgan Kaufmann, pp. 125-133, 1997.

[22] J. Cui, J. L. Hockel, P. K. Nordeen, D. M. Pisani, C.-Y. Liang, G. P. Carman, and C. S. Lynch, "A method to control magnetism in individual strain-mediated magnetoelectric islands," *Applied Physics Letters*, 103, pp. 232905-1—232905-5, 2013.

[23] C.-Y. Liang, S. M. Keller, A. E. Sepulveda, W.-Y. Sun, J. Cui, C. S. Lynch, and G. P. Carman, "Electrical control of a single magnetoelastic domain structure on a clamped piezoelectric thin film – analysis", *Journal of Applied Physics*, vol. 116, , pp. 123909-1-123909-9, 2014.

[24] S. Bandyopadhyay and M. Cahay, *Introduction to Spintronics*, CRC Press, Boca Rato, 2008.

[25] G. Salis, R. Wang, X. Jiang, R. M. Shelby, S. S. P. Parkin, S. R. Bank, and J. S. Harris, "Temperature independence of the spin-injection efficiency of a MgO based tunnel spin injector," *Applied Physics Letters,* vol. 87, no. 26, pp. 262503 – 262503-3, 2005.

[26] K. Roy, S. Bandyopadhyay and J. Atulasimha, "Binary switching in a 'symmetric' potential landscape," *Scientific Reports*, vol. 3, pp. 3038-1—3038-8, 2013.

[27] M. Salehi-Fashami, J. Atulasimha, and S. Bandyopadhyay. "Energy dissipation and error probability in fault-tolerant binary switching," *Scientific Reports*, vol. 3, pp. 3204-1–3204-7, 2013.

[28] M. Salehi-Fashami , K. Munira, S. Bandyopadhyay, A. W. Ghosh, and J. Atulasimha, "Switching of dipole coupled multiferroicnanomagnets in the presence of thermal noise: reliability analysis of nanomagnetic logic," *IEEE Transactions on Nanotechnology*, vol. 12, no. 6, pp. 1206-1212, 2013.

[29] K. Roy, S. Bandyopadhyay and J. Atulasimha, "Energy dissipation and switching delay in stress-induced switching of multiferroic devices in the presence of thermal fluctuations," *Journal of Applied Physics*, vol. 112, pp. 023914-1—023914-8, 2012.

[30] A. K. Biswas, S. Bandyopadhyay and J. Atulasimha, "Energy-efficient magnetoelastic non-volatile memory," *Applied Physics Letters*, vol. 104, pp. 232403-1– 232403-5, 2014.

[31] A. Onisko, and M. J. Druzdzela, "Impact of precision of Bayesian network parameters on accuracy of medical diagnostic systems," *Artificial Intelligence in Medicine (Elsevier)*, vol. 57, no. 3, pp. 197 – 206, 2013.

[32] S. Tschiatschek, and F. Pernkopf, "On Bayesian network classifiers with reduced precision parameters," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol 37, no. 4, pp. 774-785, 2014.