Hardware Accelerator for Probabilistic Inference in 65-nm CMOS

Osama U. Khan, *Student Member, IEEE*, and David D. Wentzloff, *Member, IEEE*

***Abstract*— A hardware accelerator is presented to compute the probabilistic inference for a Bayesian network (BN) in distributed sensing applications. For energy efficiency, the accelerator is operated at a near-threshold voltage of 0.5 V, while achieving a maximum clock frequency of 33 MHz. Clique-tree message passing algorithm is leveraged to compute the probabilistic inference. The theoretical maximum size of a factor that the proposed hardware accelerator can handle is $2^{(8\times20)=160}$ entries, which is sufficient for handling massive BNs, such as PATHFINDER, MUNIN, and so on (>1000 nodes). A Logical Alarm Reduction Mechanism (ALARM) BN is used to benchmark the performance of the accelerator. The accelerator consumes 76 nJ to execute the ALARM network using a clique-tree message-passing algorithm, while the same algorithm executed on an ultralow-power microcontroller consumes 20 mJ.**

***Index Terms*— Bayesian network (BN), clique-tree, embedded machine learning, hardware accelerator, intelligent sensor node, message passing, probabilistic graphical model, probabilistic inference.**

I. Introduction

THE evolution of microelectronics will continue, fueled by Moore's Law for semiconductors [1], and is entering into a new era of ubiquitous connectivity enabled by distributed and embedded intelligent electronics [2]. It is envisioned that pervasive and smart sensors will surround our environment. The scale at which the smart sensors are expected to be deployed will generate massive amounts of data for information and knowledge extraction [3], [4]. These data can be processed on a cloud server, in a distributed smart sensors' network, or a hybrid approach can be taken where some data are preprocessed in a resource-limited sensor node, while other is off-loaded to a cloud server. Bayesian networks (BNs) are gaining popularity for use in making fast and intelligent decisions based on large sets of data, historical information, and current inputs. They have widespread cloud-based applications, including medicine, finance, robotics, fault diagnosis, structural health monitoring, and machine learning, and so on.

Manuscript received October 6, 2014; revised January 18, 2015 and March 24, 2015; accepted March 31, 2015. This work was supported by the National Science Foundation through the Signal Processing, Information Technology Program under Award CCF-0910765.

O. U. Khan is with the Department of Electrical Engineering, University of California at Berkeley, Berkeley, CA 94720 USA (e-mail: oukhan@berkeley.edu).

D. D. Wentzloff is with the Department of Electrical Engineering, University of Michigan, Ann Arbor, MI 48109 USA (e-mail: wentzlof@umich.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2015.2420663

With the advent of millimeter-scale computing [5], millimeter-scale robots [6], and cloud-based cognitive systems [7], it is just a matter of time before BNs are pushed from the cloud into the nodes, finding use in new emerging applications, e.g., security and biometric applications [8]. These miniature devices together with their cloud infrastructure will become far more capable and far more intelligent. The BNs are prohibitively complex for implementing on today's energy-constrained sensor nodes, and more efficient BN processing is desirable.

In this paper, we present a BN hardware accelerator to compute probabilistic inference for distributed embedded electronics targeting energy-constrained applications. Many signal processing algorithms can be represented as Bayesian probabilistic networks, such as forward–backward algorithms [9], the Viterbi algorithm, the iterative turbo-decoding algorithm, Pearl's belief propagation algorithm, and the Kalman filter and certain fast Fourier transform algorithms [10]. There is one previous attempt of implementing a BN based on probabilistic CMOS (PCMOS) [11]. Two basic probabilities are generated by PCMOS and a range of probabilities required for probabilistic inference is derived from it. The proposed architecture is not easily scalable for large networks and offers limited precision for probabilistic computation, which can affect the accuracy of the computed probabilistic inference severely. A high-throughput Bayesian computing machine [12] targeting high-performance designed on a field-programmable gate array (FPGA) is presented and its performance is extensively benchmarked against equivalent CPU and GPU implementations. Although they were able to demonstrate 15×–80× performance improvement, such a Bayesian machine is infeasible for energy constrained sensor nodes due to their limited power budget. There have been some other FPGA-based implementations of Bayesian classifier using neural networks [13], [14] but suffers from the same drawback. To the best of our knowledge, this is the first application specified integrated circuit (ASIC) implementation of computing a probabilistic inference using conventional CMOS.

The inference algorithms are mainly divided into two categories: 1) exact inference and 2) approximate inference. We propose a hardware accelerator to run the exact inference algorithms, namely, a clique-tree message passing algorithm (a variant of a sum-product inference algorithm) for BNs [15]. The clique-tree message-passing algorithm has been chosen, as it can be very easily adapted for distributed intelligent sensing across multiple nodes. The main advantage being the network has a tree structure, where leaves represent

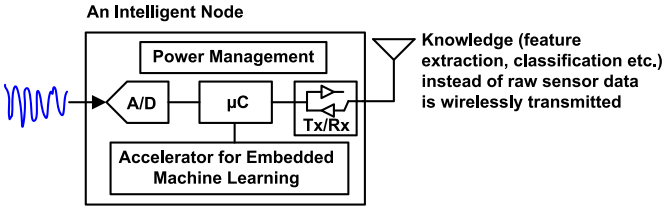

Fig. 1. Sensor node exploiting embedded machine learning.

the sensor inputs, greatly simplifying the interconnect in a large sensor network. Furthermore, only two messages are passed along any branch in the tree, minimizing the wireless traffic required between disjoint clusters of sensors that must operate together in a BN. Therefore, a clique or a group of cliques can be processed on a sensor node while message passing to cliques processed on other sensor nodes can be achieved over a wireless channel. Together the nodes implementing a BN are capable of making a decision without accessing the cloud.

It is known that the complexity of a BN grows exponentially with the number of nodes in the network [15]. This can be addressed by exploiting the hypothesis introduced by Pearl in his classical paper [16], that a large BN can be subdivided into smaller networks whose conditional probabilities are rather manageable, and thus can be implemented in a resource constrained sensor node in a distributed network.

A block diagram of a generic sensor node is shown in Fig. 1. The Analog-to-Digital Converter and microcontroller have energy efficiencies on the order of a picojoule per conversion and a picojoule per instruction, respectively, but the wireless communication, when active, is on the order of a nanojoule per bit (1000× higher) [17]. Therefore, in a wireless sensor network, it is often more energy-efficient to extract the useful information from the data on the node rather than streaming the raw sensor data to a cloud server for postprocessing. The proposed clique-tree hardware accelerator is operated at a near-threshold voltage for energy efficiency. A state-machine-based design and a simple dedicated hardware based on counters are used instead of relying on an Arithmetic Logic Unit (ALU) to perform intensive index computations required to execute the CT message-passing algorithm for computing the probabilistic inference on-node. This results in energy savings of $>10^6$ when compared with an ultralow-power microcontroller implementation for A Logical Alarm Reduction Mechanism (ALARM) network.

The rest of this paper is organized as follows. Section II gives a brief background of the BN. Section III discusses the factor operations required for inference computations. Section IV discusses the proposed hardware accelerator in detail. Section V discusses the measurement results. Finally, the conclusion is drawn in Section VI.

II. Bayesian Network

A BN is a probabilistic graphical model that represents a set of random variables (RVs) and their conditional dependencies via a directed (where the edges have a source and a target) acyclic graph (graph with no loops) [15]. Fig. 2 shows a toy BN example for a student [15]. The RVs in the network are

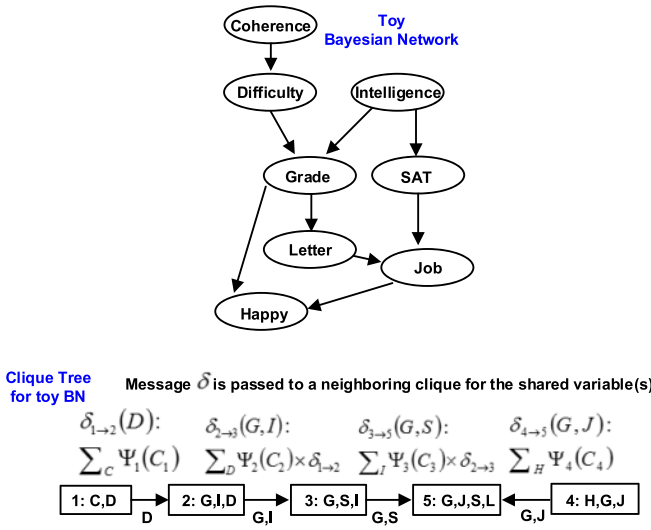

Fig. 2. Toy BN example and its corresponding clique-tree.

difficulty of the course that a student undertakes, intelligence of a student, the grade a student receives in a course, the letter of recommendation a student receives, and so on. The graphical model is a way to represent the prior knowledge about the system under consideration using probabilities that have been assigned to RVs in a network, which can be learned either from past data or with the help of a domain expert [15]. The network can be queried for inferences based on observing different RVs (sensor output). For example, given a student is intelligent and the course is easy, what is the probability of the student getting a job?

The corresponding clique-tree for the student BN example is shown in Fig. 2 with a sample run of message passing [15]. Suppose we were to compute the probability of getting a job, $P(J)$. We first select our root clique; which is any clique that contains variable J (e.g., Clique 5) and then execute message passing toward the root Clique 5. In the clique-tree, Clique 1 sends a message $\delta_{1\to 2}(D)$ to a neighboring Clique 2 updating it only for the shared RV D. Similarly Cliques 1–4 will send messages to Clique 5. Clique 5 will multiply all the incoming messages with its own factor and compute an updated belief for the RVs G, J, S, L. If we now want to obtain $P(J)$, we simply sum out G, L, and S.

Fig. 3 shows a hypothetical complex BN example monitoring the health of a person that can make intelligent decisions based on prior-knowledge of the subject and fusing it with the updated knowledge from the sensors. A clique-tree message-passing algorithm is exploited for distributed intelligent sensing.

The clique-tree message passing algorithm requires an upward message passing and a downward message passing in a clique-tree Fig. 4. After an upward and downward message passing, the posterior probabilities of all the variables in a graph can be computed [15].

III. Inference Operations

The basic data structure used in a probabilistic BN is a factor. A factor is a data structure used to represent probabilities [15]. For instance, a factor with two RVs A and B

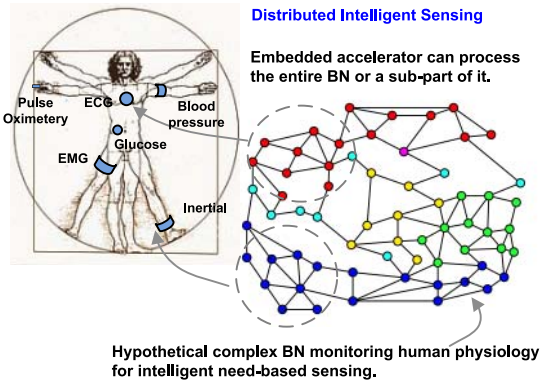

Fig. 3. Clique-tree algorithm for distributed intelligent sensing.

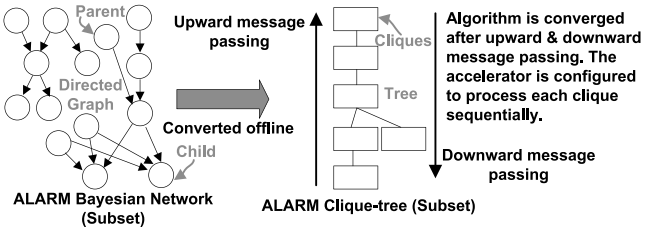

Fig. 4. Offline clique-tree generation and message passing for ALARM BN.

Φ1(A,B)

a1	b1	0.5
a1	b2	0.8
a2	b1	0.1
a2	b2	0
a3	b1	0.3
a3	b2	0.9

Fig. 5. Factor with RVs A and B.

and their corresponding probabilities is shown in Fig. 5 [15]. The RV A has cardinality 3 and RV B has cardinality 2. A factor or multidimensional table with an entry for each possible assignment to the RVs is stored as a flattened single array in the memory.

Various factor operations are required to execute the exact inference algorithms, namely, factor product, factor marginalization, factor reduction, and factor normalization. The clique-tree algorithm shown in Fig. 3 requires message computation. Each message computation at most requires factor product operation with incoming messages, factor marginalization operation on the resulting factor product, and factor normalization operation. We will now briefly illustrate these operations with simple examples.

In a factor product operation, two factors are multiplied to produce a third factor, mathematically described by (1) and shown in Fig. 6 [15]. Since factors $\emptyset 1$ and $\emptyset 2$ share a common RV B, the row 1 of factor $\emptyset 1$ needs to be multiplied with all the rows of factor $\emptyset 2$ containing the same value $b1$ of RV B, which in this example is rows 1 and 2. Similarly, other rows are being computed

$$\emptyset 3(A, B, C) = \emptyset 1(A, B) * \emptyset 2(B, C). \quad (1)$$

Factor marginalization is a process that produces a probability distribution by extracting a subset from a larger

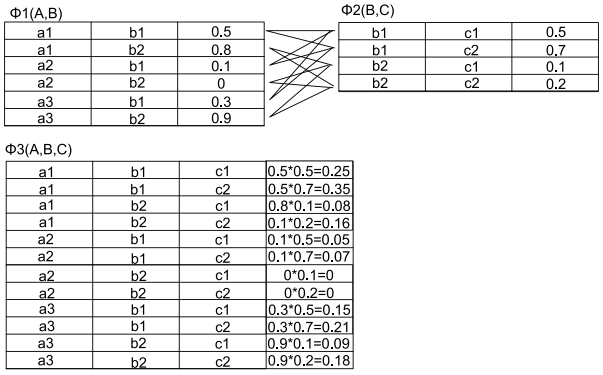

Fig. 6. Factor product.

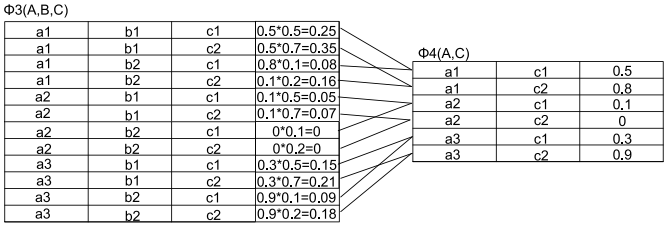

Fig. 7. Factor marginalization.

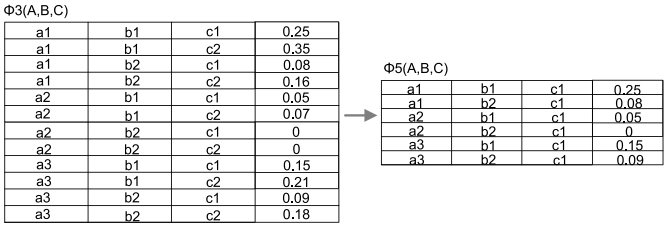

Fig. 8. Factor reduction.

probability distribution by eliminating some of the RVs, e.g., $\varphi(X) = \sum_Y \emptyset(X, Y)$. This operation is called summing out of Y in $\emptyset$ and is shown in Fig. 7 [15]. In Fig. 7, we are marginalizing RV B in factor $\emptyset 3$, so the resultant factor $\emptyset 4$ contains RVs A and C. The first row of $\emptyset 4$ is computed by summing all the rows of $\emptyset 3$ containing the same values for RVs A and C, i.e., $a1$ and $c1$, respectively, which are rows 1 and 3 in $\emptyset 3$. In a similar manner all other rows are being computed.

Factor reduction is a process, which makes the probability distribution consistent with current observations, i.e., with evidence or updated sensor values. Fig. 8 shows a simple factor reduction operation [15]. If for example value $c1$ has been observed for a RV C, all the rows in factor $\emptyset 3$ containg value $c2$ for RV C have been eliminated in factor $\emptyset 5$.

The factor normalization operation makes sure that the resulting factor, from factor operations described above, is a valid probability density function by scaling it with a non-negative value. This is accomplished by computing a running sum of each entry in a factor and dividing it with the total cumulative sum.

IV. Hardware Accelerator

A simplified block diagram of the hardware accelerator is shown in Fig. 9. The accelerator can be configured to perform

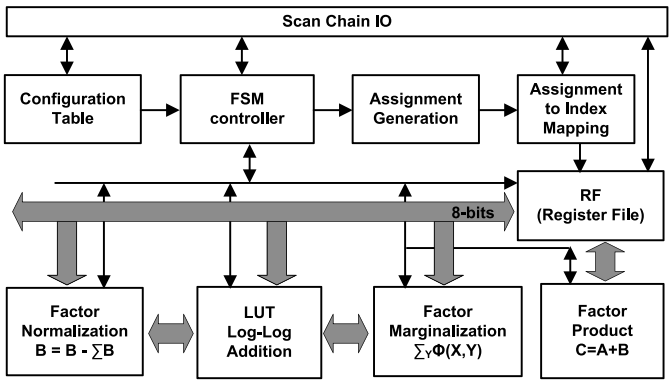

Fig. 9. Simplified block diagram for the hardware accelerator.

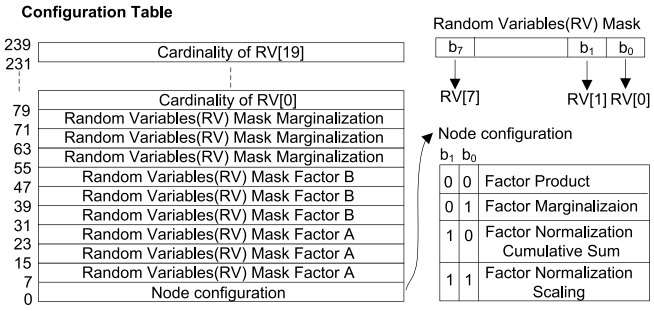

Fig. 10. Configuration table.

any of the factor operations described in Section III using a configuration table (CT) Fig. 10. The CT consists of 30 8-bit registers. The first two bits b_1, b_0 of the node configuration register are used to configure the hardware accelerator in any of the four factor operations mode, as shown in Fig. 10. The hardware accelerator can operate on two factors at a time. To avoid confusion, we call these factors Factor A and Factor B. There are three 8-bit registers that define a RV mask for Factor A, Factor B, and marginalization.

The hardware accelerator can support a factor with a maximum 20 RVs each having maximum cardinality of 256. The number of RVs present in a factor is defined using the RV mask register. To support 20 RVs a 20-bit mask register is used. For example, a factor $\emptyset 1$ shown in Fig. 6 has RVs A and B, while factor $\emptyset 2$ has variables B and C. The RVs are mapped to the mask register. The first three bits of the mask register RV[0], RV[1], and RV[2] can be mapped to RVs C, B, and A, respectively, (Fig. 10). Then, the RV Mask registers for $\emptyset 1$ (Factor A) and $\emptyset 2$ (Factor B) would have hex value 0×00006 and 0×00003, respectively. The cardinality of each RV is defined in 20 8-bit registers (since maximum of 20 RVs are supported in a factor) in the CT. In the above example, the cardinality of RV[0] register (corresponding to RV C) would have value 2, the cardinality of RV[1] and the cardinality of RV[2] registers (corresponding to RVs B and A) would have values 2 and 3, respectively. This information is used later on to generate the appropriate index for various factor operations.

One of the most computationally intensive tasks is indexing the appropriate entries in the linear array in memory for each factor operation. Computing the memory index for a given

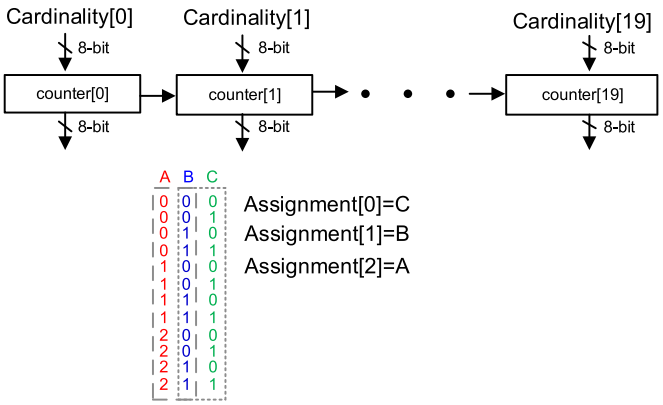

Fig. 11. On-the-fly assignment generation for factor operations.

factor operation can be computationally intensive [15] and for this reason index computation is done using a dedicated functional unit. Index computation for a given factor operation requires assignment generation. An assignment for a RV can be thought of as its state. For example, RV A has cardinality 3 so its possible assignments (state) are $a1$, $a2$, and $a3$.

The assignment generation module in Fig. 9 reads RV mask Factor A registers and the cardinality of RV registers from the CT to determine how many RVs are currently in use from the mask register and the cardinality of an each RV. This information is used to generate assignment for each RV. The assignments are generated in hardware on the fly. This is accomplished by 20 cascaded counters (for maximum 20 RVs), where each counter is counting up to the cardinality of the corresponding RV, as shown in Fig. 11. The RV mask register is used to enable the appropriate number of counters. For the factor product example in Fig. 6, a sample run of assignment generation is also shown in Fig. 11. Counter[0] is counting up to the cardinality of RV[0] (mapped to RV C) which is 2, counter[1] is counting up to the cardinality of RV[1] (mapped to RV B) which is 2 and counter[3] is counting up to the cardinality of RV[2] (mapped to RV A) which is 3. In this way the assignments for RVs in factor $\varphi 3$ are generated and the assignments of RVs in factor $\varphi 1$ and $\varphi 2$ are derived from it for index computation, which will be described in detail shortly.

The assignment to the index mapping block in Fig. 9 computes the stride (step size) of an RV in a factor on the fly. For example, as shown in Fig. 11, the factor $\varphi 3$ has variables A, B, and C with cardinalities $3, 2$, and 2 and their stride would be 4, 2, and 1, respectively. If we add another RV D then its stride would be 12 [15]. This is computed in hardware as is shown in Fig. 12 for only two RVs out of the supported 20 for simplicity. The assignment to the index mapping block reads in the RV mask Factor A registers and the cardinality of RV registers from the configuration table. The mask register bits drive the sel signal of the multiplexers in Fig. 12. The stride[0] for the first RV[0] is by definition 1. The stride[1] for the second RV[1] is equal to the cardinality of the first RV (cardinality[0]) if it is used in the current factor operation otherwise its cardinality is by-passed through a multiplexer and stride[1] is assigned a value 1. Similarly, stride[2] for the third RV[2] is equal to the product of the cardinality of the

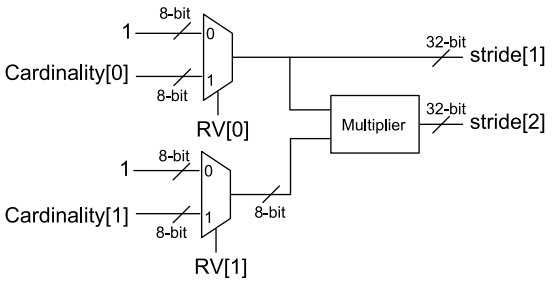

Fig. 12. Stride computation.

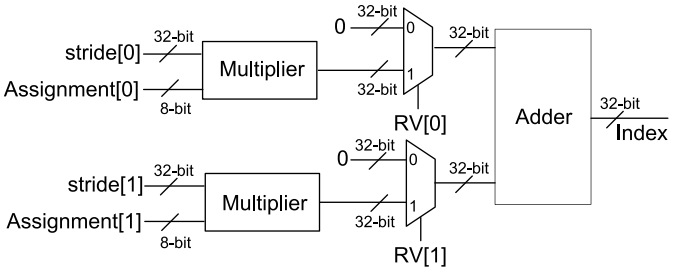

Fig. 13. Index computation.

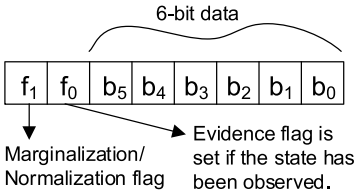

Fig. 14. 8-bit memory word structure.

second RV and the stride[1]. This way stride for RV C is computed as 1 (by definition), RV B is computed as 2 (cardinality of RV C), and RV A as 4 (product of stride of RV B and cardinality of B).

The assignment to index mapping block uses the assignment and stride information of each RV and converts it into an appropriate index using [15]

$$\text{index} = \sum_i \text{assignment}[i] * \text{stride}[i]. \tag{2}$$

Fig. 13 shows the hardware implementation illustrating only two RVs out of the supported 20 for simplicity. The mask register bits drive the sel signals of the multiplexers. If a RV is not used in a current factor operation its cardinality is not considered for the index computation. For example, to compute the index of row 10 of factor $\varphi 3$ in Fig. 6 with the assignment for RVs A, B, and C of $a3$, $b1$, $c2$ is computed using (2) as $(2*4)+(0*2)+(1*1)=9$.

The register file in Fig. 9 consists of 8-bit memory word whose structure is shown in Fig. 14. Six-bit of data stores the probability data value while two flags bits f_0 and f_1 are used for factor operations. The evidence flag f_0 is used for the factor reduction operation shown in Fig. 8.

When a certain value of a RV or a subset of RVs is observed, this is incorporated by setting the evidence flag for the corresponding states (assignments) in a factor. This makes sure that the factors are consistent with the observed/updated information. For example, in Fig. 8, if value $c1$ has been

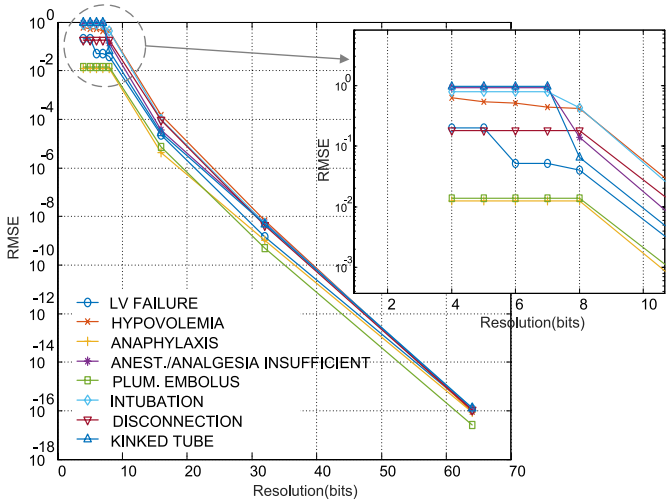

Fig. 15. MATLAB simulation model for accuracy versus bit-resolution tradeoff.

observed for the RV C then the rows corresponding to value $c1$ in factor $\varphi 3$, their evidence flag f_0 has been set (factor reduction), while the rows corresponding to value $c2$ for the RV C their evidence flag f_0 is cleared. The evidence flag f_0 is read during any factor computation and allows incorporating only values consistent with this new updated information about the states of RVs for inference computation. The flag f_1 is used during factor marginalization and factor normalization operation and its use will be explained later on. For now, it does suffice to say it is a data-flow control flag.

The accelerator can process a maximum of 1 K entries in a factor at a time with a maximum support of 20 RVs each taking 256 possible values. Thus, the theoretical maximum size of a factor that the proposed hardware accelerator can handle is $2^{(8\times 20)=160}$ entries, which is sufficient for handling massive BNs, such as PATHFINDER, MUNIN, and so on (>1000 nodes) [18].

For numerical considerations, operations such as the factor product involve multiplying many small numbers together, can lead to underflow problems due to finite precision arithmetic [15]. This is addressed by renormalizing the factor after each operation. However, if each entry in the factor is computed as the product of many terms, underflow can still occur [15]. This is avoided by performing the computation in log-space, replacing multiplications with additions. The factor marginalization operation requires summing entries. To avoid converting from log-space to linear-space and storing the result back into log-space, a lookup table (LUT) for addition is used instead, to speed up the operation.

A. Accuracy Versus Bit Resolution

A MATLAB simulation model is developed to evaluate the design tradeoff between accuracy and the bit-resolution for log-space computation Fig. 15 for a medium-size (20–60 nodes) ALARM BN, as shown in Fig. 16 [19]. The Root Mean Square Error (RMSE) is computed and averaged for 10 K Monte Carlo simulations and is plotted against the resolution (number of bits 4, 5, 6, 7, 8, 16, 32, and 64) being used for log-domain computation. The RMSE is computed

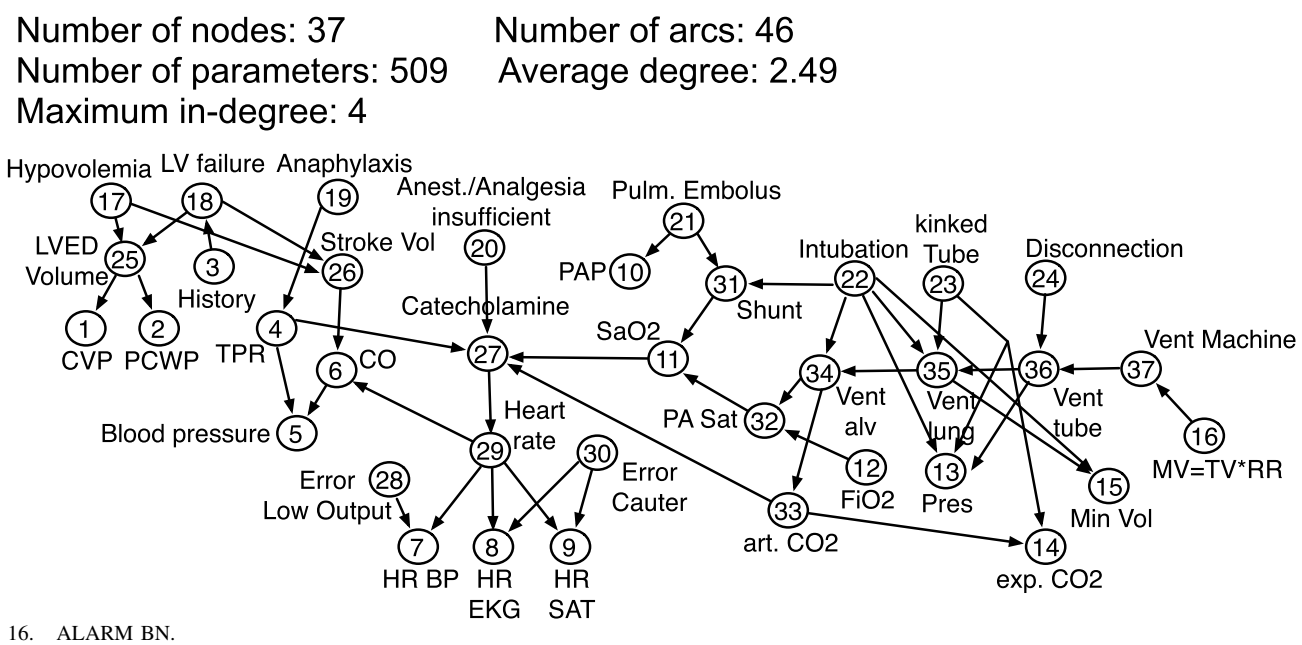

Fig. 16. ALARM BN.

by subtracting the 64-bit floating-point computations from the fixed-point finite precision integer computations. As expected, the RMSE decreases with increasing bit-resolution beyond 8 bits. Below 8-bits, the RMSE does not decrease for some variables with increasing bit resolution (4, 5, 6, 7, and 8). This is because of the very low resolution in the log-domain. The RMSE is different for each variable as the quantization error is data-dependent. In the plot, the RMSE is computed for the probabilistic inference of eight variables, which consist of both diagnostic as well as measurement variables. For example, Left Ventricular FAILURE correspond to left ventricular failure, HYPOVOLEMIA is measure of decrease in volume of blood plasma and ANEST./ANALGESIA INSUFFICIENT correspond to insufficient anesthetic and so on. The accuracy of the probabilistic inference required for a given application will determine the target RMSE, and hence the bit-resolution for the hardware accelerator. A 6-bit resolution was chosen to keep the silicon area to $\sim$1 mm^2 for the prototype chip for the mean square error of 10^{-1}.

B. Factor Operations

As discussed the hardware accelerator can be configured to perform any of the factor operations using a CT. Now, we will discuss how the hardware accelerator can perform these factor operations. For the factor product example in Fig. 6, the RVs involved in the factor operation (A, B, and C) and their cardinalities are defined in the RV mask registers and cardinality registers, respectively, in the CT. Then, the assignment generation block in Fig. 9 generates the assignment for each RV, as shown in Fig. 11. Now, since the factor $\emptyset 3$ involves all three RVs (A, B, and C), therefore, the assignment for each of these RVs is used to generate the index for the factor $\emptyset 3$, i.e., the assignment $a1$, $b1$, and $c1$ corresponding to row 1 in Fig. 11 generates index 0 for factor $\emptyset 3$ using (2). Similarly, since factor $\emptyset 1$ and factor $\emptyset 2$ uses RVs A and B and B and C, respectively, only assignments of these RVs are used to generate their index for the given factor operation, i.e., the assignment $a1$, $b1$, and $c1$ corresponding to row 1 in Fig. 11 generates index 0 (using $a1$, $b1$) for factor $\emptyset 1$ and (using $b1$, $c1$) factor $\emptyset 2$ using (2). This allows to add (log-domain) row 1 of factor $\emptyset 1$ to row 1 of factor $\emptyset 2$ and store the result in row 1 of factor $\emptyset 3$. The next value generated by the assignment generation block is $a1$, $b1$, and $c2$ corresponding to row 2 in Fig. 11 generates index 0 (using $a1$, $b1$) for factor $\emptyset 1$, index 1 (using $b1$, $c2$) for factor $\emptyset 2$ and index 1 (using $a1$, $b1$, and $c2$) for factor $\emptyset 3$. This way row 1 of factor $\emptyset 1$ is added to row 2 of factor $\emptyset 2$ and the result is stored in row 2 of factor $\emptyset 3$ as required by the factor product example in Fig. 6. The hardware accelerator serially processes the factors until the assignment generation block finishes its count or the maximum 1 K entries have been reached in a factor.

The factor marginalization example as shown in Fig. 7 requires summing out a RV or a group of RVs from a given probability distribution. The RVs involved in the factor operation and their cardinalities are defined in the CT as discussed before in the factor product operation, but with the addition of setting the RV mask marginalization registers for the RVs that need to be marginalized. In our example, the 20-bit mask register would have hex value 0×00002, since the RV B mapped to RV[1] needs to be marginalized from the factor $\emptyset 3$. The assignment generation block generates the assignment for factor $\emptyset 3$ as before. Now, since the factor $\emptyset 4$ involves RVs A and C, the assignment to index mapping block along with the marginalization mask registers generates its index using (2). For example, the assignments $a1$, $b1$, and $c1$ corresponding to row 1 in Fig. 11 generate index 0 for factor $\emptyset 3$ and index 0 (using $a1$, $c1$) for factor $\emptyset 4$. This way row 1 of factor $\emptyset 3$ is added using a LUT in Fig. 9 to row 1 of factor $\emptyset 4$ and the marginalization/normalization flag bit f_1 of factor $\emptyset 3$ row 1 is set indicating that the current value has

been processed to avoid duplicate counting. The next value generated by the assignment generation block is $a1$, $b1$, and $c2$ corresponding to row 2 in Fig. 11 generates index 1 (using $a1$, $b1$, and $c2$) for factor Ø3 and index 1 (using $a1$, $c2$) for factor Ø4. The row 2 of factor Ø3 is added using the LUT to row 2 of factor Ø4 and the f_1 flag is set for factor Ø3 row 2. The next value generated by the assignment generation block is $a1$, $b2$, and $c1$ corresponding to row 3 in Fig. 11 generates index 2 (using $a1$, $b2$, and $c1$) for factor Ø3 and index 0 (using $a1$, $c1$) for factor Ø4. The row 3 of factor Ø3 is added to row 1 of factor Ø4 and the f_1 flag is set. Therefore, row 1 and row 3 of factor Ø3 are added together to generate row 1 of factor Ø4 marginalizing RV B as required by the factor marginalization example in Fig. 7. The accelerator will process the entire factor serially until the assignment generation block finishes its count or the maximum 1 K entries have been reached in a factor.

The factor normalization operation scales the resulting factors to make them valid probability distributions. The RVs involved in the factor and their cardinalities are defined in the CT, the assignment for each RV is generated and the index is computed as before. The normalization operation is split into two states. In the first state factor normalization cumulative sum, a running sum of each entry in a factor is computed using the LUT. In the second factor normalization scaling state, the computed accumulated sum is subtracted (log domain) from each entry in the factor, hence, normalizing the probability value. The accelerator will process the entire factor serially until the assignment generation block finishes its count or the maximum 1 K entries have been reached in a factor.

V. Measured Results

A medium-size (20–60 nodes) ALARM BN is chosen for benchmarking the performance of the hardware accelerator as shown in Fig. 16 along with its graph properties [19]. There are a total of 37 nodes, 46 arcs, and 509 parameters in the network [18]. The ALARM network is an expert system designed for hospital patient monitoring to reduce the false alarm rate in a hospital setting. It calculates probabilities for differential diagnosis based on available evidence and measurements from the sensors [19]. There are nine observation variables in the network, e.g., patient history, heart rate, Electrocardiogram, and blood pressure and so on and eight diagnostic variables that predict a patient's condition. The network consists of 27 cliques and the maximum number of variables in a clique is 5, while the maximum clique size is 144. For the clique-tree message passing algorithm, the ALARM network is converted offline into its corresponding clique-tree using HUGIN Expert software [20] Fig. 4. This conversion is a one-time process unless the graphical model is updated incorporating new knowledge, which will require updating the clique-tree. The prototype chip is fabricated in a 65-nm CMOS process with an active area of 0.52 mm^2 shown in Fig. 17. The chip is fully synthesized using a high-voltage process and operated at a near-threshold voltage of 0.5 V to save power and improve energy efficiency while operating at maximum clock frequency of 33 MHz. The total

Computed energy profile for Alarm Network

	This Work	PIC MCU
Network Initialization	12.2nJ	1.9mJ
Upward message passing	21.8nJ	5.6mJ
Downward message passing	20.1nJ	6.4mJ
Beliefs computation	21.3nJ	5.2mJ
Inference for LV FAILURE, HYPOVOLEMIA, DISCONNECT ANEST./ANALGESIA INSUFFICIENT, ANAPHYLAXIS, KINKED TUBE, PLUM. EMBOLUS	86pJ	845µJ
Inference for INTUBATION	99pJ	
Execution Time	350µs	11.1s
Total	76.2nJ	19.9mJ

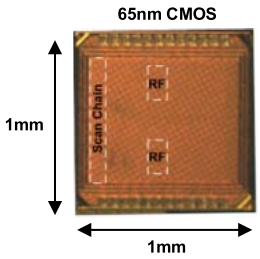

Fig. 17. Computed results for ALARM BN.

measured active power is 218 μW. Significant energy savings have been achieved through near-threshold operation [21].

The steps to compute a probabilistic inference using the proposed hardware accelerator are as follows. First, the BN is converted offline into its corresponding clique-tree. The hardware accelerator is then used to multiply the factors assigned to each clique in a clique-tree to compute initial potentials. This step is referred to as network initialization. Then, the clique-tree data structure is used to pass messages between neighboring cliques. This is accomplished manually through an FPGA in our measurement setup but will be handled by microcontroller software in an embedded application, as shown in Fig. 1. Second, the message passing start from the leaves of the clique-tree and proceeds up the tree toward the root clique, this step is referred to us upward message passing. Third, the message passing start from the top root clique and proceeds down the tree toward the leaf cliques, this step is referred to us downward message passing. Fourth, each clique in a clique-tree multiplies all the received messages with its own initial potential. The result is a factor called the beliefs. This step is referred to as beliefs computation. The hardware accelerator is used serially to perform the factor operations required in each step. For example to compute a message, a clique potential (factor) and an incoming message are written into the RF using the scan chain. In our measurement setup an FPGA is used to configure the hardware accelerator for factor operations, which for a message computation typically involves factor product, factor marginalization, and factor normalization operations. After the appropriate factor operation(s) the message (factor) is read from the RF using the scan chain.

For a continuous operation using the proposed hardware accelerator, Microcontroller Unit (MCU) software will write the configuration bits and read/write the factor data serially into the RF. The software will use the clique-tree data structure to control the computation and the flow of messages.

There are eight-diagnostic variables in the ALARM network; after the clique-tree algorithm is converged the network can, then, be queried for the probabilistic inference. This is accomplished by identifying the beliefs (factors) containing the RV of interest. The probabilistic inference is then computed by marginalizing all but the RV of interest in a factor (belief). The probabilistic inference computation involves factor marginalization and factor normalization operations.

The calculated total energy to evaluate the ALARM network is 76.2 nJ and is distributed for the clique-tree algorithm among its constituent components, as shown in Fig. 17.

This energy computation does not consider the energy required for Input Output operations, e.g., the energy spent moving the data in and out of the memory. The total energy for the ALARM network is computed by first measuring the energy required for each of the factor operations to compute a message on chip, and then calculating the number of messages and factor operations required to run the entire clique-tree message-passing algorithm. The energy required to compute the probabilistic inference is computed for each of the queried variables and is shown in Fig. 17. The higher energy required for INTUBATION inference is expected, since the node has higher out-degree (number of children) in the graph Fig. 16 and its corresponding clique in the clique-tree requires more factor operations relative to other inference variables.

To implement the probabilistic inference algorithm on a sensor node in the absence of any ASIC hardware accelerator, a low-power MCU will be a natural choice for an embedded designer. Many sensor node platforms are based on an ultralow-power MCU and, therefore, the proposed hardware accelerator is benchmarked against it. For total energy comparison if the same ALARM network were to be implemented on an ultralow-power MCU in a sensor node, we look at a Peripheral Interface Controller (PIC) MCU from Microchip [22], which consumes 30 μA/MHz at 1.8 V. This MCU is a Reduced Instruction Set Computer-based architecture with only 49 all single cycle instructions except branches. It consumes 1.8 mW of power at a 33-MHz clock frequency. The total number of ALU operations required to execute CT message passing algorithm for the ALARM network were computed and it would take 11.1 s for the CT algorithm to converge on a PIC MCU. This corresponds to total energy of 2 mJ, which is roughly six orders of magnitude higher than the proposed hardware accelerator. It should be noted that the PIC MCU has other functional blocks running; e.g., power management, on-chip clock, and timing generation but nevertheless 4–5 orders of magnitude energy savings is pragmatic.

VI. Conclusion

Embedded machine learning is likely to provide another layer of service in a distributed sensor network enabling intelligent sensing. Since the smart sensors are expected to be low-cost and have long battery-life it is necessary to use hardware accelerators to achieve high-energy efficiency required for an energy-constrained application. In this paper, we presented a hardware accelerator for computing a probabilistic inference using BNs. The hardware accelerator is operated at near-threshold voltage of 0.5 V and consumes 77 nJ of total energy for running a clique-tree message-passing algorithm for the ALARM network. The theoretical maximum size of a factor that the proposed hardware accelerator can handle is $2^{(8\times20)=160}$ entries, which is sufficient for handling massive BNs, such as PATHFINDER, MUNIN, and so on ($>$1000 nodes).

References

[1] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics*, vol. 38, no. 8, pp. 1–4, Apr. 1965.

[2] W. Weber, J. Rabaey, and E. H. L. Aarts, Eds., *Ambient Intelligence*. New York, NY, USA: Springer-Verlag, Mar. 2005.

[3] *Cisco Keynote Highlights From CES*, Cisco, San Jose, CA, USA, 2014.

[4] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Comput. Netw.*, vol. 54, no. 15, pp. 2787–2805, Oct. 2010.

[5] G. Chen *et al.*, "A cubic-millimeter energy-autonomous wireless intraocular pressure monitor," in *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers (ISSCC)*, San Francisco, CA, USA, Feb. 2011, pp. 310–312.

[6] P. Chirarattananon, K. Y. Ma, and R. J. Wood, "Adaptive control of a millimeter-scale flapping-wing robot," *Bioinspiration Biomimetics*, vol. 9, no. 2, p. 025004, 2014.

[7] IBM. *The Watson Ecosystem*. [Online]. Available: http://www.ibm.com/smarterplanet/us/en/ibmwatson/ecosystem.html, accessed Jan. 23, 2014.

[8] S. N. Yanushkevich, M. L. Gavrilova, V. P. Shmerko, S. E. Lyshevski, A. Stoica, and R. R. Yager, "Belief trees and networks for biometric applications," *Soft Comput.*, vol. 15, no. 1, pp. 3–11, Jan. 2011.

[9] L. E. Baum, T. Petrie, G. Soules, and N. Weiss, "A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains," *Ann. Math. Statist.*, vol. 41, no. 1, pp. 164–171, 1970.

[10] C. Bishop, *Pattern Recognition and Machine Learning*. New York, NY, USA: Springer-Verlag, 2006.

[11] Z. H. Kong, J.-J. Tan, B. E. S. Akgul, K.-S. Yeo, K. V. Palem, and W.-L. Goh, "Hardware realization of a medical diagnostic system based on probabilistic CMOS (PCMOS) technology," in *Proc. IEEE Int. Symp. VLSI Design, Autom. Test*, Apr. 2008, pp. 275–278.

[12] M. Lin, I. Lebedev, and J. Wawrzynek, "High-throughput Bayesian computing machine with reconfigurable hardware," in *Proc. 18th Annu. ACM/SIGDA Int. Symp. Field Program. Gate Arrays*, 2010, pp. 73–82.

[13] J.-F. Wang, A.-N. Suen, J.-R. Lee, and C.-H. Wu, "A Bayesian neural network chip design for speech recognition system," in *Proc. IEEE Int. Conf. Neural Netw.*, vol. 4. Nov./Dec. 1995, pp. 2027–2030.

[14] S. Marra, F. C. Morabito, P. Corsonello, and M. Versaci, "FPGA implementation of Bayesian neural networks for a stand-alone predictor of pollutants concentration in the air," in *Proc. IEEE Int. Joint Conf. Neural Netw.*, vol. 4. Jul. 2004, pp. 2613–2618.

[15] D. Koller and N. Friedman, *Probabilistic Graphical Models—Principles and Techniques* (Adaptive Computation and Machine Learning). Cambridge, MA, USA: MIT Press, 2009.

[16] J. Pearl, "Fusion, propagation, and structuring in belief networks," *Artif. Intell.*, vol. 29, no. 3, pp. 241–288, Sep. 1986.

[17] B. H. Calhoun *et al.*, "Body sensor networks: A holistic approach from silicon to users," *Proc. IEEE*, vol. 100, no. 1, pp. 91–106, Jan. 2012.

[18] M. Scutari. *Bayesian Network Repository*. [Online]. Available: http://www.bnlearn.com/bnrepository/, accessed Aug. 1, 2013.

[19] I. A. Beinlich, H. J. Suermondt, R. M. Chavez, and G. F. Cooper, "The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks," in *Proc. 2nd Eur. Conf. Artif. Intell. Med.*, 1989, pp. 247–256.

[20] *HUGIN EXPERT*. [Online]. Available: http://www.hugin.com, accessed Sep. 2, 2013.

[21] R. G. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge, "Near-threshold computing: Reclaiming Moore's law through energy efficient integrated circuits," *Proc. IEEE*, vol. 98, no. 2, pp. 253–266, Feb. 2010.

[22] Microchip. *PIC12(L)F1822 Data Sheet*. [Online]. Available: http://ww1.microchip.com/downloads/jp/DeviceDoc/jp547368.pdf, accessed Sep. 20, 2014.

[23] Analytic bridge. (Sep. 2011). *Bayesian Network Newsletter*. [Online]. Available: http://www.analyticbridge.com/profiles/blogs/bayesiannetwork-newsletter-september-2011

Osama U. Khan (S'08) received the B.E. (*summa cum laude*) degree in electronics engineering from the NED University of Engineering and Technology, Karachi, Pakistan, in 2007, and the M.S. and Ph.D. degrees in electrical engineering from the University of Michigan, Ann Arbor, MI, USA, in 2010 and 2013, respectively.

He was with the RF Division, Qualcomm, San Diego, CA, USA, in 2009. In 2014, he was with Psikick Inc., Charlottesville, VA, USA, an ultralow-power wireless startup, where he developed low-power crystal oscillator and short-range radio front-end. He is currently a Research Scholar with the University of California at Berkeley, Berkeley, CA, USA. His current research interests include battery independent, robust, adaptive microsystems, and their applications.

David D. Wentzloff (S'02–M'07) received the B.S.E. degree in electrical engineering from the University of Michigan, Ann Arbor, MI, USA, in 1999, and the M.S. and Ph.D. degrees from the Massachusetts Institute of Technology, Cambridge, MA, USA, in 2002 and 2007, respectively.

He co-founded Psikick Inc., Charlottesville, VA, USA, a fabless semiconductor company, developing ultralow-power wireless systems-on-chip in 2012. He has been with the University of Michigan since 2007, where he is currently an Associate Professor of Electrical Engineering and Computer Science. His current research interests include RF integrated circuits, with an emphasis on ultralow-power design.

Dr. Wentzloff is a member of the IEEE Circuits and Systems Society, the IEEE Microwave Theory and Techniques Society, the IEEE Solid-State Circuits Society, and Tau Beta Pi. He was a recipient of the DARPA Young Faculty Award in 2009, the Eta Kappa Nu Professor of the Year Award from 2009 to 2010, the DAC/ISSCC Student Design Contest Award in 2011, the IEEE Subthreshold Microelectronics Conference Best Paper Award in 2012, the NSF CAREER Award in 2012, the ISSCC Outstanding Forum Presenter Award in 2014, and the EECS Outstanding Achievement Award from 2014 to 2015. He has served on the Technical Program Committee of the International Symposium on Low Power Electronics and Design since 2011 and the IEEE Radio Frequency Integrated Circuits Symposium since 2013. He has served as a Guest Editor of the IEEE TRANSACTIONS ON MICROWAVE THEORY AND TECHNIQUES, the *IEEE Communications Magazine*, and the *Journal of Signal Processing: Image Communication* (Elsevier). He served on the Technical Program Committee of the IEEE International Conference on Ultrawideband from 2008 to 2010, and S3S from 2013 to 2014.