Bayesian Sensor Fusion with Fast and Low Power Stochastic Circuits

Alexandre CONINX
Pierre BESSIÈRE
Emmanuel MAZER
and Jacques DROULEZ
CNRS: ISIR/UPMC and LIG/UGA
4, place Jussieu
75005 Paris, France
Email: alexandre.coninx@isir.upmc.fr

Raphaël LAURENT
ProbaYes S.A.S
82, Allée Galillée
38330 Montbonnot, France
Email: raphael.laurent@probayes.com

M. Awais ASLAM
Jorge LOBO
Institute of Systems and Robotics
Dept. of Electrical & Computer Eng.
University of Coimbra
3030-290 Coimbra, Portugal
Email: {mawais, jlobo}@isr.uc.pt

***Abstract*—As the physical limits of Moore's law are being reached, a research effort is launched to achieve further performance improvements by exploring computation paradigms departing from standard approaches. The BAMBI project (Bottom-up Approaches to Machines dedicated to Bayesian Inference) aims at developing hardware dedicated to probabilistic computation, which extends logic computation realised by boolean gates in current computer chips. Such probabilistic computing devices would allow to solve faster and at a lower energy cost a wide range of Artificial Intelligence applications, especially when decisions need to be taken from incomplete data in an uncertain environment. This paper describes an architecture where very simple operators compute on a time coding of probability values as stochastic signals. Simulation tests and a reconfigurable logic hardware implementation demonstrated the feasibility and performances of the proposed inference machine. Hardware results show this architecture can quickly solve Bayesian sensor fusion problems and is very efficient in terms of energy consumption.**

I. INTRODUCTION

The aim of the BAMBI [1] project (Bottom-up Approaches to Machines dedicated to Bayesian Inference) is to explore new computing methods inspired from cell signaling mechanisms. Within this project we search for models to explain how a unicellular alga (Chlamydomonas [2]) successfully evolves and reproduces in its microscopic but nevertheless challenging environment. One strong assumption made is that elementary cognitive capabilities are prerequisites for this alga to survive in situations where complex decisions have to be made and where bad ones often lead to destruction of the cell. Our line of thought is that the mechanisms leading to these efficient behaviors are implemented at the molecular level and are the result of stochastic processes which could be modeled by well established formalisms [3], [4]. Considering the limited sensing capabilities of Chlamydomonas, the decisions are made with a very incomplete picture of the state of the world.

We follow Laplace [5] and Jaynes [6] to state that probabilistic inference is a rational way to reason with incomplete knowledge and believe that Evolution has equipped some living organisms with mechanisms able to perform such inferences. Further, we assume time codes may be used by biological systems to represent probability distributions. To sustain all these hypotheses, we aim at synthesizing simple computing artifacts which could perform probabilistic inference using nano-scale devices while avoiding the von Neumann architecture and being energy efficient.

In [7] we proposed an architecture based on bitwise Gibbs sampling. A simulation of this machine on a standard computer was used to control a mobile robot. The same architecture was implemented on reconfigurable logic device (FPGA) and used to solve with approximate inference an intractable problem with a good precision. In [8] we presented a machine performing exact inference with approximate calculus based on arithmetics on stochastic bitstreams. In the present work we describe a different machine using similar basic principles to allow efficient computing of solutions for problems in the class of naive Bayesian inferences. Even if the machine is limited in scope, it could be used in a variety of sensor fusion applications and the results presented in this paper show truly remarkable performances in term of execution speed and power consumption.

In this paper, we first review related works on bio-inspired and probabilistic computing architectures. We then describe our stochastic machine and show how it can be used to compute naive Bayesian inference. This architecture is then tested on a simple sensor fusion problem, first with a simulated system and second with an FPGA implementation. The computation speed and power consumption demonstrated by those systems are presented, and we discuss the implications of those performances, as well as the potential applications and future developments of such systems.

II. RELATED WORKS

Neural networks are one of the major source of inspiration for bio-inspired designs with two blue riband projects: SpiNNaker [9] and TrueNorth [10]. In these projects, highly parallel machines are designed and organized to emulate or to work like assemblies of neurons. The core computation inside each processing unit is still based on the von Neumann architecture and makes use of fixed or floating point processing units. The design we propose is also massively parallel but differs in

978-1-5090-1370-8/16/$31.00

that the parallelism takes place at the bit level: an approach which was suggested thirty years ago by the designer of the Connection Machine [11].

When designing bio-inspired machines, the question of how they can be programmed is in our opinion a major issue. Many bio-inspired designs rely on learning as only programming mechanism and several chips have been developped (for instance by Google, Nvidia, Movidius) to speed up the recognition and classification phase. The main drawbacks of this approach are the huge amount of data required to train the networks, and the expertize needed to adjust meta-parameters used during learning.

As an alternative, the approach we follow consists in using the Bayesian probability theory to rethink the ways computers are programmed [12]. To create such a new probabilistic computing framework, new programming languages have been and are being developed, such as ProBT [13], [14], Church [15] or Anglican [16] to quote only a few. Accordingly, new programming methods are also suggested [14] to develop applications. As in standard languages, these methods encompass variables definition and scope, control and loops, except that they now become probabilistic.

The idea of developing hardware dedicated to execute these programs has been pursued by several teams [17] [18] [19] with goals similar to the BAMBI project's: exploring different computation paradigms to perform probabilistic inference. Vigoda designed architectures [17] based on probabilities represented by analog signals, and used the message passing algorithm to compute exact inference. Mansinghka uses sampling methods for approximate inference [18]. In a similar way, Jonas designed Markov Chain Monte Carlo based algorithms to provide a representation of probability distributions as sets of samplers [19]. The Nanoscale Computing Fabrics Laboratory (University of Massachusetts Amherst) is conducting a research project closely related to inference based on DAG models [20]. This research group has designed an unconventional hardware architecture based on electro-magnetic computations to perform inference on Bayesian Network models. The approach taken by Thakur and all [21] is similar to ours: they use stochastic bitstreams and target special inference problems. They have proposed two frameworks, BEAST (Bayesian Estimation And Stochastic Tracker) and BIND (Bayesian INference in DAG), to perform inference using stochastic electronics on two types of Bayesian models, Hidden Markov Models and Direct Acyclic Graphs (DAG) respectively.

In the framework of the BAMBI project, another stochastic architecture has been proposed to perform naive Bayesian fusion using Muller C-Elements [22], which achieves exact inference with normalization for binary random variables, but create harmful correlations in the stochastic signals and can't be easily extended to non-binary discrete distributions.

III. The Bayesian Machine 1

A. Problem statement

Our computational architecture is focused on the goal of performing naive Bayesian fusion [14]: computing the posterior probability distribution on a searched variable S, knowing a prior distribution $P(S)$ and the conditional distributions $P(K_i|S)$ on some variables $K_1, \dots K_N$ with known values $k_1, \dots k_N$. In naive Bayesian fusion the K_i variables are conditionally independant with one another given S ($P(K_i|K_j, S) = P(K_i|S)\ \forall i, j \in \{1, \dots, N\}$). The inference is computed by:

$$P(S|k_1, \dots, k_n) = \frac{1}{Z} P(S) \prod_{i=1}^{N} P(k_i|S) \tag{1}$$

where Z is a normalization constant.

We will focus on the case where S is a discrete random variable with cardinality M, taking possible values $s_1, \dots, s_M$. The posterior probability distribution can therefore be expressed by the probability values $P([S = s_j]|k_1, \dots, k_n)$ with $j \in \{1, \dots, M\}$, and computed as a probability product:

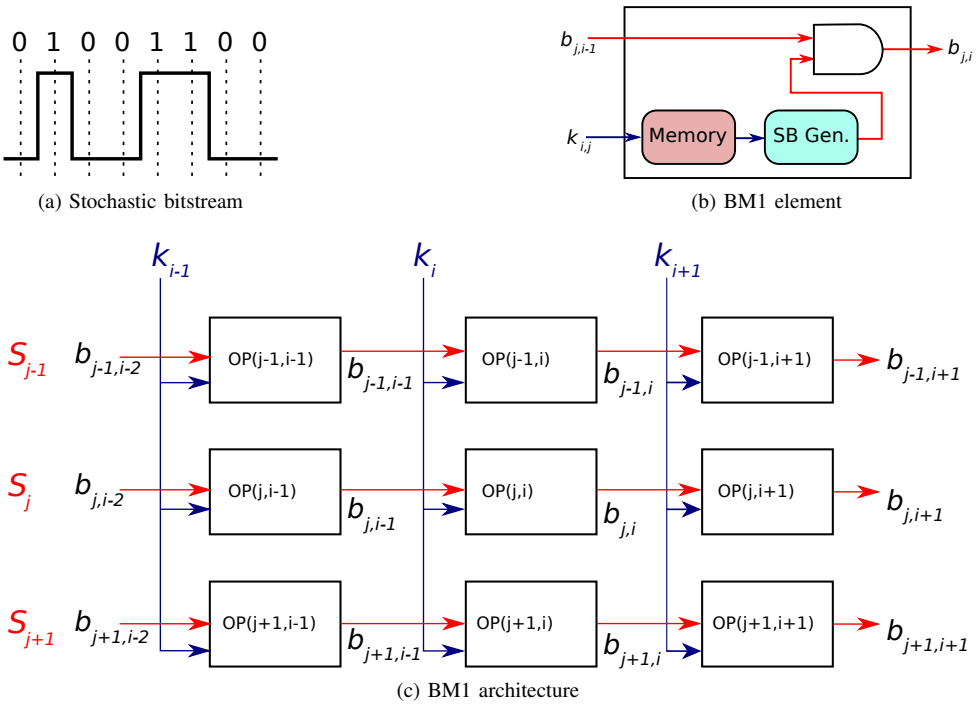
$$P([S = s_j]|k_1, \dots, k_n) = \frac{1}{Z} P([S = s_j]) \prod_{i=1}^{N} P(k_i|[S = s_j]) \tag{2}$$

where $P([S = s_j])$ is the prior probability of value s_j, and $P(k_i|[S = s_j])$ is the likelihood of the observed value k_i of variable K_i when the searched variable has value s_j.

B. Proposed architecture

Following some previous work on stochastic circuits[23], [24], we propose to represent those probability values with clocked stochastic digital signals ("stochastic bitstreams") using temporal coding: a probability value p is represented by a digital signal B such as at each timestep, $P([B = 1]) = p$ and $P([B = 0]) = 1 - p$ (Figure 1a). It is immediate that if two such independant signals B_1 and B_2 encoding probability vaues p_1 and p_2 are fed to an AND gate, the output signal encodes probability $p_1\ p_2$: the stochastic bitstream data representation allows to compute probability product with a single boolean logic operation.

The probability product described in equation 2 can therefore be performed by a sequence of N AND gates operating on stochastic bitstreams. The sequence starts with a stochastic bitstream encoding the prior probability value $P([S = s_j])$ and at each step, it integrates the i-th data likelihood term by performing an AND operation between the signal output from the previous step and a bitstream encoding the probability values $P(k_i|[S = s_j])$, generated from the probability value stored in local memory (Figure 1b). A set of M such sequences of elementary circuits can be used to concurrently compute the M posterior probability values $P([S = s_j]|k_1, \dots, k_n)$ in parallel (Figure 1c). We name the resulting architecture the Bayesian Machine 1 (BM1).



(a) Stochastic bitstream

(b) BM1 element

(c) BM1 architecture

Fig. 1. Architecture of our BM1 stochastic machine. Probability values are represented as stochastic digital signals using temporal coding ("stochastic bitstreams") as shown in figure 1a (the depicted signal encodes a probability value $p = \frac{3}{8}$). The product between the probability value encoded by a bitstream $b_{j,i-1}$ and a likelihood value $k_{i,j}$ can be performed by the circuit shown in figure 1b, by storing the probability value for $k_{i,j}$ in a memory, using a stochastic bitstream generator to create a stochastic signal encoding the corresponding value, and using an AND gate to compute the product. A matrix of such circuits (figure 1c) can be used to compute naive Bayesian fusion on discrete variables, each line j of the architecture computing the posterior probability for $S = s_j$, and each column i integrating the likelihoods for the data term K_i.

The BM1 yields a set of stochastic bitstreams (a "stochastic bus") representing the full posterior probability distribution. In order to recover the resulting probability values as numeric representation, we can use M simple counter circuits, which integrate the stochastic signals by counting the number of "1" bits n_1 in a given number of clock cycles n. The final state of the counter constitutes a numeric fixed-point representation of the value $\frac{n_1}{n}$, which is an estimation of the probability value encoded by the bitstream, with the accuracy of that estimation increasing with the counter size and the integration time n.

C. Distribution normalization

The proposed BM1 architecture performs probability products between discrete probability distributions, but it does not perform any normalization operation: the signals on the output bus encode probability values equal to the product of the values encoded by the prior and the likelihood terms. Although working with such unnormalized distibutions is possible, issues specific to our data representation may arise from the very low probability values incurred by multiple probability products. For example, in the trivial case where the prior distribution and all data likelihood distributions are uniform, the probability values of all generated stochastic signals are $\frac{1}{N}$ and the probability values of all output signals are $p_{out} = \frac{1}{N^{M+1}}$. A stochastic bitstream encoding such a probability value would therefore have only one every N^{M+1} bit set to "1", which means the machine has to be run for an implausibly high number of cycles before any output data can be collected. We call this issue the *time dilution problem*.

We propose to address this issue by modifying the way the prior and data likelihood distributions are encoded as stochastic signals in order to maximize the output probability values. Instead of representing a probability distribution $P(X)$ on a discrete variable X by bitstreams encoding the values $P([X = x_i])$, we use bitstreams encoding probability values $\frac{P([X=x_i])}{P_{max}}$, where $P_{max} = \max_{i \in \{1,..,N\}} P([X = x_i])$. The value x_i corresponding to the maximum probability value is therefore represented by a probability 1, which is encoded as a constantly up signal, and other probabilities are scaled according to that maximum value.

With this method, the issue in the previous trivial example is resolved as all signals encode a probability value of 1. Similar problems arising with weakly peaked distributions are resolved satisfyingly in a similar manner. Since the probabilities for the different lines of the BM1 are all multiplied by the same factor $\frac{1}{P_{max}}$, only the normalization constant Z from equation 2 is modified and the distribution profile is preserved.

IV. Experimental results

A simple scenario was considered as a case study to measure the performance of the proposed architecture: a data fusion problem where the location of a boat is estimated from the

readings of six noisy sensors. The probem is to infer the (X, Y) coordinates of the boat on a 64×64 grid (see figure 2) given six sensor readings: the distances (D_0, D_1 and D_2) and bearings (B_1, B_2 and B_3) to three landmarks.

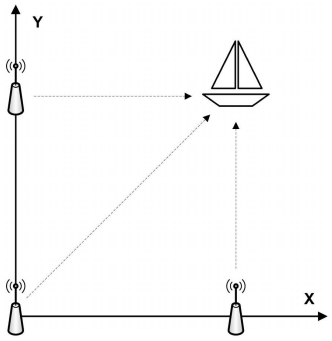


Fig. 2. The boat localization problem

A. Problem specifications

The values of the problem parameters used to test our architecture are the following. The (X, Y) coordinates of the landmarks are $(0, 0)$, $(0, 32)$ and $(32, 0)$, and the boat actual position is $(32, 32)$. For each landmark i, the distance sensor model $P(D_i \mid X\ Y)$ is a Gaussian probability distribution specified by a mean μ_{di} equal to the Euclidian distance between the landmark i and the possible location (X, Y), and by a standard deviation σ_{di} given by the function $\sigma_{di} = 5 + \mu_{di}/10$. Likewise, the angle sensor model $P(B_i \mid X\ Y)$ of each landmark is a Gaussian probability distribution specified by a mean μ_{ai} equal to the angle of view of the landmark from the boat, and by a standard deviation $\sigma_{bi} = 14.0625$ degrees.

Figure 3 illustrates those models by showing the shape of the likelihood functions defined by the sensor models for the landmark at position $(0, 32)$ when the values of the sensor readings are 32 for the distance d_1 and 0 for the angle b_1.

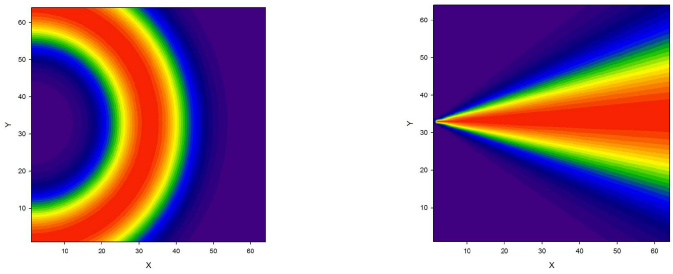
The complete inference will fuse the information provided by the sensor models to provide an estimate of the boat location, according to the Bayesian model:

$$P(X\ Y|D_1\ B_1\ D_2\ B_2\ D_3\ B_3) \propto P(X\ Y)\prod_{i=1}^{3} P(D_i \mid X\ Y) \cdot P(B_i \mid X\ Y) \quad (3)$$

We can perform that inference with the BM1 architecture described in section III, with parameters $N = 64^2$ (the cardinality of the location variable) and $M = 6$ (the number of known variables).

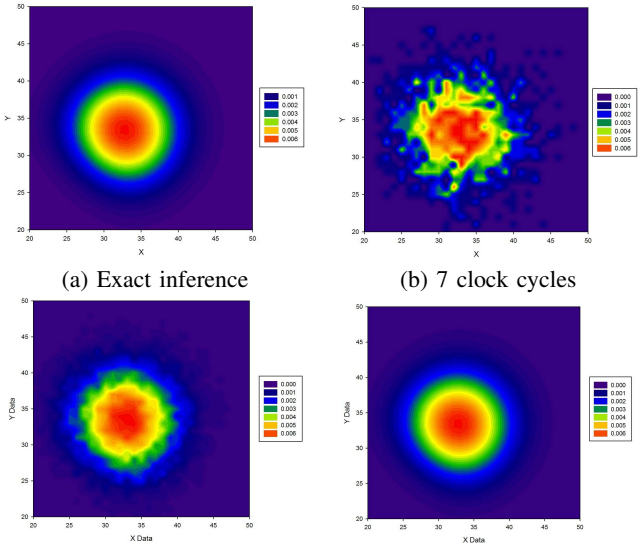
B. Algorithmic level performance assessment

Some of the properties of the proposed architecture can be assessed at the algorithmic level. Software simulations of the stochastic architecture were run to collect key performance indicators, and for comparison purposes the reference distribution was computed by the exact inference engine of the ProBT library [13], [14]. Figure 4 highlights a major feature of the proposed architecture: very few clock cycles are necessary to obtain a qualitatively good estimation of the boat location.



(a) (b)

Fig. 3. Likelihood of the boat location for the first landmark, (a) for the distance sensor model when the distance is 32, and (b) for the angle sensor model when the angle is 0.



(a) Exact inference (b) 7 clock cycles

(c) 100 clock cycles (d) 1M clock cycles

Fig. 4. Qualitative evaluation of the stochastic architecture on the boat problem using a 64×64 grid. Probability distributions on the boat position are infered (a) in exact inference, or with the stochastic architecture after a few clock cycles (b), (c) and (d). The plots are restrained to a sub-region of the $X \times Y$ grid since posterior probabilities are nearly null outside.

To quantify the precision of the stochastic inference, the variations of the Kullback-Leibler divergence (KL divergence) between the stochastic machine output distribution and the reference distribution computed by exact inference are shown (figure 5) as the number of clock cycles increases. This measure of distance between the two probability distributions provides a global assessment of the precision of the inference computed by the stochastic machine. The performances of the stochastic architecture are compared with what would be obtained by an hypothetical implementation which would be able to draw one sample at each clock cycle from the target probability distribution.

These results highlight two features of the stochastic architecture. First, as it is common for sampling-based approaches, by increasing the number of clock cycles the error can be made

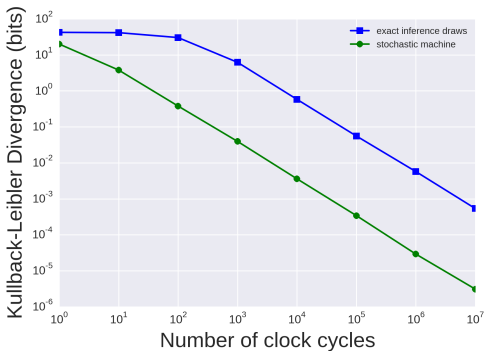


Fig. 5. Quantitative evaluation of the precision of the stochastic computation. The Kullback-Leibler Divergence between the probability distribution computed by the BM1 and the reference distribution computed by exact inference provides a global error measure.

arbitrarily small (but there is a direct impact on computation time and energy consumption). Second, the stochastic architecture is dramatically faster ($\times 153$ in average) than a solution that would draw one sample from the target distribution at each clock cycle.

C. Hardware implementation

To implement the Bayesian machine in hardware we used reconfigurable logic devices (Field Programmable Gate Arrays, or FPGAs). The topology and memory contents of the BM1 stochastic circuits are not fixed, but depend on the model and the inference problem. The flexibility and reconfigurability of FPGAs allow us to spool different machines onto hardware as needed.

FPGAs are devices built from elements with programmable logic functions and interconnects, allowing them to be reconfigured in circuit. Compared to custom built Application Specific Integrated Circuits (ASICs), they are less efficient since there is an overhead in terms of silicon resources and a lower working frequency. They are widely used for prototyping digital circuits, and they can also have an edge over CPUs and GPUs since the fine-grain parallelism provided by reconfigurable logic hardware provides a better fit to the geometry of the computation data flow and the required precision, but can be readily deployed without needing to fabricate a custom chip as happens with the ASICs.

We started by creating the required components in VHDL (a hardware description language used for hardware synthesis) specifically for the boat example inference model. Above the boat example is given for a 64×64 grid and 6 sensors, this leads to a matrix of $(6+1) \times 4096$ BM1 elements (including priors). We scaled down to 4×4 to have a manageable size for initial tests. After that, we made each component generic so that parameters could be modified to fit the needs of other models, or different spatial resolution for the same model. With generic components we can use an automatic toolchain to compile a VHDL circuit from the Bayesian program written with the ProBT probabilistic programming language. The development was made in Altera's Quartus II IDE to synthesise the circuits to the FPGA on the reconfigurable logic boards used in the experiments.

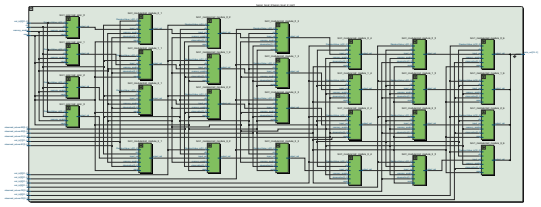


Fig. 6. RTL view of BM1 for the boat example with a 2×2 spatial grid, showing a regular matrix structure of $(1+6) \times 4$ modules.

As an illustrative example, the RTL (Register Transfer Level) circuit view of a BM1 for the boat localization problem with a 2×2 grid can be seen in figure 6. In this case the circuit is a matrix of 4×6 BM1 elements or modules, and 4 prior modules. For a grid size 32×32 this becomes $(1+6) \times 1024$. The likelihoods are stored in memory inside the BM1 modules, but each module only has the slice required to have the correct values for a given input. In this way we can have new observations at any given time and have the likelihood values readily available.

Stochastic bitstreams are generated by comparing the stored value with a random number at each clock cycle [24]. For the random number generators (RNGs) we tested three pseudo-random generators implementable on the FPGA. The Fibonacci Linear Feedback Shift Register (LFSR) [24] is very compact but did not give good results as it introduced a bias. The Mersenne Twister (MT32) [25], based on the Mersenne prime $2^{199937} - 1$, gave much better results, but required a big memory and more logic elements. The XSADD generator, a variant of XORshift RNG [26], has a repeat sequence of $2^{128} - 1$, requires more logic elements than the others, but no memory, and also has good statistical properties. Since memory is used elsewhere in the design, the XSADD is a better compromise and was used in the hardware implementation.

The BM1 module consists of a memory, a bitstream generator, and a simple AND logic gate that is the stochastic multiplier. Each bitstream generator requires a random number generator, however if we look at the topology of the machine, the lines in the matrix are independent, and instead of having a random number generator in each BM1 module, the random number can be shared down the same column. This leads to a more efficient implementation of the BM1 module than the direct implementation of the operator in figure 1. This is a major improvement on resource usage that improves scalability: increasing the size of the search space (*i.e.* the resolution on the inferred location) increases the number of BM1 modules, but not the number of RNGs, nor the computation time. Figure 7 shows the RTL view of the BM1 Module.

The bitstream generator compares the value looked-up in memory with the shared random number to generate a stochastic bitstream representing the probability value from the memory. The memory component *lpm_memory_inst* stores

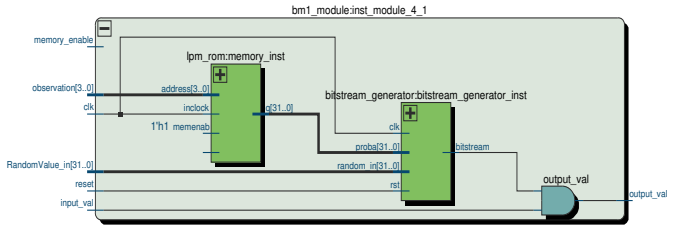


Fig. 7. RTL view of the BM1 module operator, with a local memory, a bitstream generator (that is only a comparator since the random number is an input), and a stochastic multiplier.

the likelihood associated with the corresponding searched variable. For the priors it is similar but with a single value memory and without the multiplier AND gate.

Since a custom circuit is synthesised the priors and likelihood memories are partitioned and distributed onto the instances of the above components exactly as needed.

For the hardware tests we used development boards with Altera FPGAs Cyclone IV, Stratix IV and Stratix V. VHDL was used for development of the design using Altera Quartus II IDE and to synthesise the circuits to the FPGA. ModelSim was used for offline simulations and Signal Tap II for in-circuit logic analysis on the FPGA. Power analysis was done using Altera PowerPlay Power analyser. A nominal clock frequency of 50 MHz was used for all boards in the tests, but higher frequencies are possible depending on the specific device and design optimisations.

D. Hardware performance assessment

Figure 8 shows the boat position inferred by a hardware implementation of the Bayesian machine using a 32×32 grid space. This was done using XSADD RNGs, on the larger FPGA, but occupied only a fifth of the device resources using the revised BM1 modules to save on the use of RNGs. After 10^3 clock cycles we already have a good output, and with more it quickly approaches the reference. With the conservative speed of 50 MHz, this means the machine has takes 0.02 ms to perform a very good approximation of the exact inference.

To better visualise the accuracy of the stochastic computation, as for the simulated implementation, we compute the KL divergence between the probability distribution computed by the BM1 and the reference distribution computed by exact inference. This global error measure is plotted in figure 9 for different grid sizes.

In Table I, we can see how the BM1 gets mapped to different FPGAs. The resource utilisation provides a measure of the circuit *area* or fraction of resources in the chip. The implementation is using the XSADD RNGs, and the random values are shared down the columns to limit the number of RNGs used in the design. We can see that for increasing grid sizes the circuit becomes quite large. For a spatial grid of 32×32 the full BM1 takes up 20% of a large FPGA.

Tests were performed on the BM1 in various configurations using Modelsim and the Power Analyzer Tool from Altera Quartus II FPGA development IDE to estimate the power

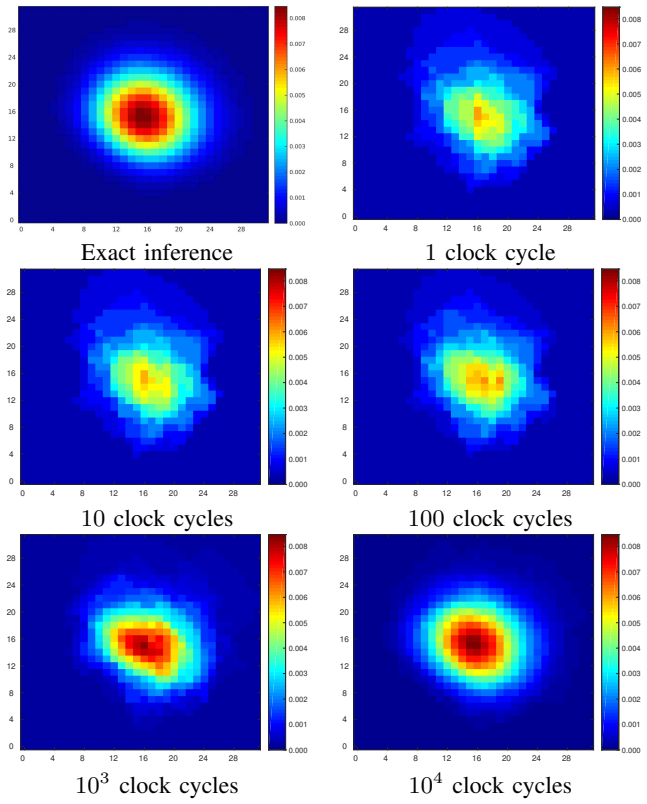


Fig. 8. Hardware results for 32×32 grid, showing a colour map of the posterior distribution $P(X, Y \mid k)$ resulting from the sensor fusion for exact inference reference and for BM1 with bitstream lengths of 1, 10, 100, 10^3 and 10^4.

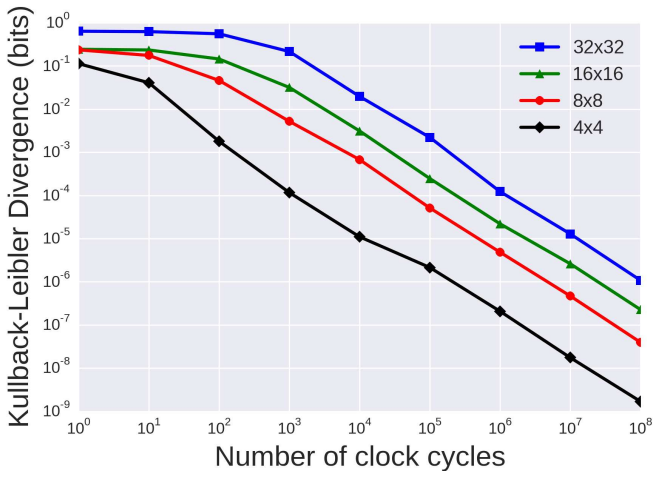


Fig. 9. KL divergence results of the boat example for different grid sizes and increasing bitstream lengths.

consumption. The power consumption of the boat example for a 4×4 grid size, using the XSADD RNGs, and sharing RNGs across the independent bus lines was estimated to be 213,29 mW. A single RNG has a power consumption of 4.76 mW. The analysis is done with the Cyclone IV and for the full test circuit that includes the output counters that convert the stochastic

TABLE I
RESOURCE UTILISATION OF BM1 FOR BOAT EXAMPLE

	Grid Size	Logic Utilization	Tot. Reg.	Mem. Bits
Cyclone IV	4×4	3,006(3%)	1,828(1%)	545(<1%)
	8×8	6,879(6%)	3,414(3%)	2081(<1%)
Stratix V	4×4	1,370(<1%)	1,892	545(<1%)
	8×8	3,469(1%)	3,472	2081(<1%)
	16×16	10,920(4%)	9,834	8225(<1%)
	32×32	52,298(20%)	35350	32801(1%)

bus to numeric values and some wrap I/O control logic of the machine. The estimated value includes the fraction of the core static power dissipation of the FPGA device corresponding to the resource usage. From this thorough simulation, and knowing the topology of BM1, we can estimate the power for bigger grid sizes, even if for 32×32 this would map to more than one Cyclone IV chip.

From this we can compute energy consumption, and figure 10 shows the increased accuracy obtained when more energy is used, i.e., when longer bitstream lengths are used. It follows the trend shown in figure 9, but since larger circuits draw more power it takes more energy for larger grid sizes to reduce the KL divergence. For comparison purposes, figure 10 also shows an estimation on the energy consumption required by running software performing the exact inference on a typical laptop computer processor (Intel(R) Core(TM) i7-2640M CPU @ 2.80GHz). This was done thanks to a software library [27] allowing to monitor the energy consumed at the process level [28].

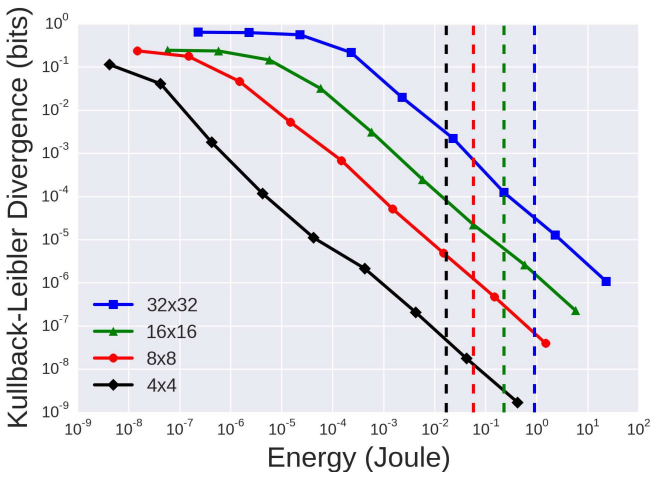


Fig. 10. KL divergence results of boat example for different grid sizes and increasing energy consumption of the BM1. Vertical lines indicate the energy used by a typical laptop computer to perform the corresponding exact inference.

Running the 32×32 grid size BM1 at 50 MHz, the machine takes 0.02 ms to run 1000 cycles and have a reasonable output as shown in figure 8. It does this using only 0.23 mJ, the software version on a typical laptop takes 919 mJ, this is a huge difference. If we run for 10^4 cycles, we have a KL divergence of 0.029 (Figure 9), and only use 2.3 mJ (Figure 10). The accuracy requirements of target applications are a key factor, since using short bitstream lengths decreases accuracy, but provides a usable result with much less energy.

The results show that the hardware implementation is feasible and efficient. To improve scalability to larger problems the memory structure can be improved. In the current implementation memory is used redundantly to be readily available in all modules, but a write loop can be used to have single registers and refresh when the known inputs change.

E. Discussion

One feature of the BM1 is the decoupling between the size of the sampling space (the number of lines of the BM1) and the sampling time which remains constant. Of course, increasing the resolution or the number of evidences (the number of columns) increases the number of BM1 elements and, as a consequence, the surface and the power consumption of the circuit.

The distributed memory used for the priors and likelihoods is a key feature of the design. The flexibility of the FPGA is used to store from the start the specific sets of values at each node, readily available at each clock cycle to generate the correct bitstreams. Since the lines in the stochastic bus are independent, the RNGs can be shared down the columns of the matrix of operators. But the circuit is still a large matrix of small memories, comparators and AND gates. This can be reduced by only having a local register and an external memory, but the register refresh updates for changing inputs would be a significant overhead.

The performance of our proposed architecture is highly dependant on the ability to easily generate high quality and independent stochastic bitstreams encoding chosen probability values. While such bitstreams can be generated by CMOS logic components using pseudo-random number generators, such generators have significant energy and memory requirements and are not suitable to large scale integration in stochastic circuits. Recent advances in new nanodevices based on spintronics, such as the superparamagnetic tunnel junction (SMTJ) [29], bear the promise that such generators could be available in the short or medium term. Experimental SMTJ devices have been shown to be able to generate high-quality stochastic bitstreams at a frequency of 500MHz with a very low power consumption of $5\mu W$. Those components can be built with CMOS technology using an area equivalent to 12 bytes of SRAM [30], making them suitable to large scale integration with the other components needed to build the stochastic machines described above. Since the proposed architecture could work without a central clock, SMTJ devices could directly be used as stochastic inputs generating asynchronous signals, further reducing the power consumption.

Another interesting property of this architecture is its robustness to noise. Early tests have shown the machine will continue to work well even when errors occur with a high probability while it is performing its simple logical operations. The error sources can be either external, such as Single Event Effects due to radiations in space environment, or internal such as

using low voltages to reduce the power consumption as in the PCMOS project [31].

V. Conclusion

We presented a general hardware architecture dedicated to solving sensor fusion problems using Bayesian inference. The solution proposed allows to swiftly produce good results thanks to a fine grain parallel sampling at the bit level. This architecture is now being used to compute disparity maps for stereo vision in mobile robotics applications.

Because of the time dilution problem the BM1 could not accommodate a large number of evidences and for the same reason it could not be used in a Bayesian filter. We are currently investigating BM1-like designs which address the time dilution problem by regenerating the stochastic bus after new evidences are fetched in the process (keeping a reasonable number of 1 in the stochastic bus). As it is, the BM1 is a good candidate to solve tractable inference problems and could successfully be used in real applications. Another fold of the Bambi project is to use Gibbs sampling at the bit level to tackle problems intractable in exact inference (for example problem with many latent variables). We follow two tracks, one is to design dedicated hardware to solve a given problem, allowing only the value of evidences to change, and the other is to build a general purpose sampling machine which could be implemented on an ASIC and which could accept any kind of Bayesian programs in the same way a standard processor is suited for any kind of programs after it has been properly compiled. Such an ASIC will lead to even better performances in terms of speed and energy consumption.

Acknowledgment

This work was performed within the EU Future and Emerging Technologies BAMBI project (FP7-ICT-2013-C, project number 618024).

References

[1] https://www.bambi-fet.eu/.

[2] E. H. Harris, D. B. Stern, and G. Witman, *The chlamydomonas sourcebook*. Cambridge Univ Press, 2009, vol. 1.

[3] A. Houillon, P. Bessière, and J. Droulez, "The probabilistic cell: implementation of a probabilistic inference by the biochemical mechanisms of phototransduction," *Acta biotheoretica*, vol. 58, no. 2-3, pp. 103–120, 2010.

[4] J. Droulez, D. Colliaux, A. Houillon, and P. Bessière, "Toward biochemical probabilistic computation," *arXiv preprint arXiv:1511.02623*, 2015.

[5] P. S. Laplace, "Mémoire sur la probabilité des causes par les évènements," *Mémoires de l'Academie des Sciences de Paris*, vol. 6, 1774.

[6] E. T. Jaynes, *Probability Theory: the Logic of Science*. Cambridge University Press, 2003.

[7] R. Canillas, R. Laurent, M. Faix, D. Vaufreydaz, and E. Mazer, "Autonomous robot controller using bitwise gibbs sampling," in *The 15th IEEE International Conference on Cognitive Informatics and Cognitive Computing*. IEEE, 2016.

[8] M. Faix, R. Laurent, J.Lobo, and E. Mazer, "Cognitive computation: a bayesian machine case study," in *The 14th IEEE International Conference on Cognitive Informatics and Cognitive Computing*. IEEE, 2015.

[9] http://apt.cs.manchester.ac.uk/projects/SpiNNaker/project/.

[10] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[11] W. D. Harris, *The Connection Machine*. MIT Press, 1989.

[12] J. D. Alves, J. F. Ferreira, J. Lobo, and J. Dias, "Brief Survey on Computational Solutions for Bayesian Inference," in *Workshop on Unconventional Computing for Bayesian Inference, 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2015)*, Hamburg, 2015.

[13] K. Mekhnacha, J.-M. Ahuactzin, P. Bessière, E. Mazer, and L. Smail, "Exact and approximate inference in ProBT," *Revue d'intelligence artificielle*, vol. 21, no. 3, pp. 295–331, 2007.

[14] P. Bessière, E. Mazer, J. M. Ahuactzin, and K. Mekhnacha, *Bayesian programming*. CRC Press, 2013.

[15] N. Goodman, V. Mansinghka, D. Roy, K. Bonawitz, and J. Tenenbaum, "Church: A language for generative models," in *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence (UAI)*, 2008, pp. 220–229.

[16] F. Wood, J. W. van de Meent, and V. Mansinghka, "A new approach to probabilistic programming inference," *arXiv preprint arXiv:1507.00996*, 2015.

[17] B. Vigoda, "Analog logic: Continuous-time analog circuits for statistical signal processing," Ph.D. dissertation, Massachusetts Institute of Technology, 2003.

[18] V. K. Mansinghka, "Natively probabilistic computation," Ph.D. dissertation, Massachusetts Institute of Technology, 2009.

[19] E. M. Jonas, "Stochastic architectures for probabilistic computation," Ph.D. dissertation, Massachusetts Institute of Technology, 2014.

[20] S. Khasanvis, M. Li, M. Rahman, M. Salehi-Fashami, A. K. Biswas, J. Atulasimha, S. Bandyopadhyay, and C. A. Moritz, "Self-similar magneto-electric nanocircuit technology for probabilistic inference engines," *Nanotechnology, IEEE Transactions on*, vol. 14, no. 6, pp. 980–991, 2015.

[21] C. S. Thakur, S. Afshar, R. M. Wang, T. J. Hamilton, J. Tapson, and A. van Schaik, "Bayesian estimation and inference using stochastic electronics," *Frontiers in neuroscience*, vol. 10, 2016.

[22] J. S. Friedman, L. E. Calvet, P. Bessiere, J. Droulez, and D. Querlioz, "Bayesian Inference With Muller C-Elements," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. PP, no. 99, pp. 1–10, 2016.

[23] B. Gaines, "Stochastic computing systems," *Advances in information systems science*, pp. 37–172, 1969.

[24] A. Alaghi and J. P. Hayes, "Survey of Stochastic Computing," *ACM Transactions on Embedded Computing Systems*, vol. 12, no. 2s, pp. 1–19, 2013.

[25] M. Matsumoto and T. Nishimura, "Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Trans. Model. Comput. Simul.*, vol. 8, no. 1, pp. 3–30, Jan. 1998.

[26] G. Marsaglia, "Xorshift rngs," *Journal of Statistical Software*, vol. 8, no. 1, pp. 1–6, 2003.

[27] https://github.com/Spirals-Team/powerapi.

[28] A. Bourdon, A. Noureddine, R. Rouvoy, and L. Seinturier, "PowerAPI: A Software Library to Monitor the Energy Consumed at the Process-Level," *ERCIM News*, vol. 92, pp. 43–44, Jan. 2013.

[29] N. Locatelli, A. F. Vincent, A. Mizrahi, J. S. Friedman, D. Vodenicarevic, J.-V. Kim, J.-O. Klein, W. Zhao, J. Grollier, and D. Querlioz, "Spintronic Devices as Key Elements for Energy-Efficient Neuroinspired Architectures," *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, vol. 1, pp. 994–999, 2015.

[30] D. Querlioz, "Review of IEF's work - Modelling of superparamagnetic MTJs," in *BAMBI-FET second year annual meeting*, Paris, 2016.

[31] L. N. Chakrapani, B. E. Akgul, S. Cheemalavagu, P. Korkmaz, K. V. Palem, and B. Seshasayee, "Ultra-efficient (embedded) SOC architectures based on probabilistic CMOS (PCMOS) technology," in *Proceedings of the conference on Design, automation and test in Europe*. European Design and Automation Association, 2006, pp. 1110–1115.