Bayesian Inference implemented on FPGA with Stochastic Bitstreams for an Autonomous Robot

Hugo Fernandes, M. Awais Aslam
ISR - Institute of Systems and Robotics
University of Coimbra, Portugal
{hfernandes, mawais}@isr.uc.pt

Jorge Lobo, João Filipe Ferreira
ISR - Institute of Systems and Robotics
DEEC, University of Coimbra, Portugal
{jlobo, jfilipe}@isr.uc.pt

Jorge Dias
ISR, DEEC, Univ. Coimbra, Portugal
Khalifa University, Abu Dhabi, UAE
jorge@isr.uc.pt

***Abstract*—This paper presents an FPGA implementation of a machine performing exact Bayesian inference using stochastic bitstreams. We revisited stochastic computing, not to perform better computations with unreliable hardware, but to perform approximate computations with less hardware. The underlying trade-off is between precision and computation time. An automatic design of probabilistic machines that compute soft inferences with an arithmetic based on stochastic bitstreams is presented. The computation tree provided by a Bayesian inference software is used to define the stochastic circuit. Tests were performed and results presented concerning accuracy and resource usage of the stochastic computing implementation of Bayesian machines performing exact inference. An application example is given of a Bayesian sensorimotor system that performs obstacle avoidance for an autonomous robot, fully implemented on an FPGA. Some conclusions were drawn on the followed approach, providing insights for future implementations.**

I. Introduction

A key challenge in robots that are required to autonomously operate in complex environments is the lack of cognitive systems able to efficiently deal with uncertainty when behaving in real world situations.

Biological neural systems excel in robustness and power-efficient operation, despite relying on low-precision, unreliable and massively parallel neural elements. To build devices with adequate computational power to dwell in uncertainty and decide with incomplete data, with limited resources and power, new approaches can learn from biology.

Probabilistic modelling approaches allow artificial systems to cope with the uncertainty and incompleteness. The Bayesian programming paradigm [1] allows the specification of Bayesian models, and questions can then be asked to the model about the phenomenon, generating specific Bayesian Machines implementing the computation corresponding to the desired probabilistic inference. However on standard architectures these translate to a heavy computational burden, limiting their application. Since we are operating on full probability distributions, we have a scalability issue even for a few variables with a low discretisation cardinality. However due to the nature of the computations, an approximate computation can be used.

We revisited stochastic computing, not to perform better computations with unreliable hardware, but to perform approximate computations with less hardware. The underlying trade-off is between precision and computation time. The flexibility of reconfigurable logic allowed us to have a working circuit performing Bayesian inference using stochastic bitstreams (not to be confused with FPGA configuration bitstream).

Bayesian theory and probabilistic inference is already used in robotics [2] [1] [3]. However, probabilistic computations easily overload standard von Neumann architecture computers due to the large dimensionality of the underlying entities, leading to slow computations. FPGAs have been used in several probabilistic applications providing very significant performance gains with respect to conventional computers. In [4] FPGAs were used to construct a high throughput Bayesian computing machine, suitable for directed graph probabilistic networks, addressing compilation and scheduling issues. Research on probabilistic gates has also resorted to reconfigurable logic to test proposed models. In [5] combinational stochastic logic is presented as an abstraction that generalizes deterministic digital circuit design (based on Boolean logic gates) to the probabilistic setting.

Stochastic Computing was proposed by von Neumann [6] and later by Gaines [7] as an alternative to perform better computations with unreliable hardware, using stochastic bit streams to encode probabilities and simple logic gates to perform arithmetic operations. In [8] and [9] we revisited stochastic computing to perform approximate computations with less hardware. A compilation toolchain was proposed, and simulation results presented. In this work we take it to the next step and have a full implementation on an FPGA for a target robotic application, presenting results on accuracy and resource usage.

II. The Bayesian Machine and compilation toolchain

A Bayesian Machine (BM) is a machine that solves an inference problem by taking probability distributions, or soft evidences, as inputs and outputs probability distributions over the searched variables [8]. Soft evidences represent uncertainty over known variables whereas hard evidences are deterministic observations.

In [8] and [9] a compilation toolchain which starts from a Bayesian model described in a Bayesian programming language, namely ProBT [10], automatically designs the probabilistic machine which implements the inference over the

This work was made possible thanks to the EU collaborative FET Project BAMBI FP7-ICT-2013-C, project number 618024 (www.bambi-fet.eu).

described model. This provides the computation tree that needs to be implemented in hardware. The ProBT software package, that runs on standard computers, also provides a ground truth reference for the expected outputs. Our aim here is to have a simpler circuit that can be used in embedded robotic devices.

The Bayesian machine is defined using properties, notation and formalism of Bayesian programming [10]. To design our machine we need to define the Bayesian model, the soft evidence inputs, the constant parameters of the model, and the inference to compute. A compilation toolchain starts from the Bayesian model in ProBT and generates a VHDL circuit using a library of generic components that we generated specifically to perform stochastic computing.

To illustrate this, let us assume a generic test-bed Bayesian model, consisting of a joint distribution on a conjunction of discrete variables $P(D \wedge M)$, in which D and M are themselves conjunctions of discrete variables, representing observed data and the variable for which we want to infer a posterior distribution, respectively [8]. More specifically, in the case of D we can write $D = D_1 \wedge D_2 \wedge D_3 \ldots \wedge D_n$. Let us now assume that $D_1 \ldots D_n$ are not directly observable, and that we only have access to probability distributions over possible observations, $\widetilde{P}(D_i)$. Since these distributions are defined over observed variables, they are called "soft evidence" over these variables, in opposition to "hard evidence", which would refer to a situation where these variables would be completely observable and hence instantiated directly as a specific value. The joint distribution defining this particular Bayesian Machine would be given by

$$P(M \wedge D_1 \wedge \ldots \wedge D_n) = P(M)P(D_1 \mid M) \ldots P(D_n \mid M) \quad (1)$$

By applying Bayes' rule and factoring in the soft evidence on observations, we know that the output of the corresponding Bayesian Machine would be given by the following equation:

$$P\Big(M \mid P'(D)\Big) =$$
$$= \frac{1}{Z} \sum_{D_1} \widetilde{P}(D_1) \ldots \sum_{D_i} \widetilde{P}(D_i) \ldots \sum_{D_n} \widetilde{P}(D_n) P(M \wedge D) \quad (2)$$

in which Z is a normalisation constant that can in turn be expressed as

$$Z = \sum_{M} \Big(\sum_{D} \widetilde{P}(D) P(M \wedge D) \Big). \quad (3)$$

Let us denote the resulting Bayesian machine as BM. A BM with two inputs and one output is shown in Figure 1. These two inputs are the probability distributions representing soft evidence on two finite and discrete variables, $D_1 = [1...n_1]$ and $D_2 = [1...n_2]$, and can be written as $\widetilde{P}(D_1)$ and $\widetilde{P}(D_2)$. The resulting output $P(M) \equiv P(M|\widetilde{P}(D))$ is also a probability distribution over a finite and discrete variable $M = [1...k]$.

The ProBT software, developed by ProBAYES [11], enables an almost direct specification of Bayesian Programs such as the one defined above using C++ or Python bindings. This has allowed for the development of a compilation toolchain add-on to ProBT that transforms these bindings into VHDL that is then instantiated into a Bayesian Machine supported by any computational approach we wish to implement, in our case we will use stochastic computing.

Fig. 1. Simple Bayesian Machine with two inputs and one output, and no free variables. The compilation toolchain takes a Bayesian program specification and uses ProBt to have a symbolic description of the computation used to synthesise the machine.

III. Bayesian Machine Implementation with Stochastic Computing

In the above specification of Bayesian machines discrete variables are being used, and given that the required computation expressed in equation 2 relies on a regular set of sums and multiplications, this can be efficiently implemented by exploiting the parallelism offered by the FPGA. Using stochastic computing, we can perform multiplication and addition with very simple circuits. The toolchain instantiates the set of stochastic arithmetic units required by the problem under consideration. They are then used in the Bayesian Machine design.

A. Stochastic Computing

Stochastic Computing (SC) is an alternative to conventional binary computing in which digitalised probabilities are used to represent and process information. Stochastic signals are generated by continuous time stochastic processes which produce either '0' or '1'. A *stochastic stream* [12], [6] is defined as a sequence of stochastic signals over time, and its value is defined as the number of '1s' over the total number of bits for a specific time window. Due to its low implementation cost and robustness to errors, SC has recently re-gained the attention of the scientific community [12]. However a linear increase in the precision of stochastic computations requires an exponential increase in the length of the bitstream, and the dynamic range of the representation in SC is also limited.

B. Bayesian Machine Stochastic Circuit

To implement a Bayesian machine with a stochastic circuit we need to look at equation 2 and map it to a circuit. While sums and multiplications can be performed using a simple AND gate and a simple multiplexer [12], division is not so trivial and leads to complicated circuits. We can however avoid or delay the division. Since the output is a probability distribution, we can delay the normalisation.

One can use a set of n stochastic bitstreams as an alternative coding scheme for the n values of the probability distribution on a discrete variable. A signal bus carrying a stochastic bitstream is called a "stochastic bus" or "probabilistic bus". Because the encoded probability values in a probabilistic bus are not normalised, the sum is not equal to '1'. In fact values will tend to be low, and that allows us to perform stochastic addition with an "OR" gate and a memory (OR+)

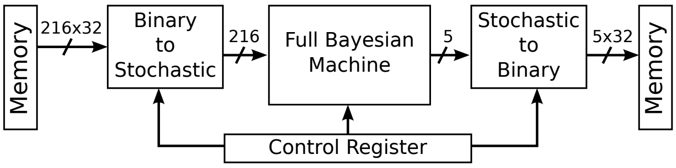

Fig. 2. Block diagram of the BM implementation.

[13], avoiding scaling that occurs with the multiplexer used as an adder. The actual p-value in a probabilistic bus can be obtained by counting the number of '1's over the total number of bits in the i^{th} bitstream of the stochastic bus.

Figure 2 presents a block diagram view of the BM implementation using a robot application example from [9]. At the core of the machine, the computations are made using cascades of stochastic operators in parallel. The additional blocks are required to have the system running on the FPGA. Binary values are loaded from memory, converted into stochastic bitstreams, and fed into the Bayesian Machine to perform the stochastic arithmetic operation. The output of the BM is then converted back to binary, with accumulator counters, and stored in memory.

IV. Results for Autonomous Robot Sensorimotor System

A classic robotic application is used to show our BM FPGA implementation for a realistic problem, with a significant number of variables and components. In [9] we presented the Bayesian sensorimotor system that performs obstacle avoidance for an autonomous robot. The robot adjusts its trajectory to avoid obstacles by controlling its rotation velocity while moving forward, given a distance estimate provided by 3 infrared and 3 ultrasonic sensors. Three levels of distance are defined: close, medium and far, and both sensor modalities used in the inference. The result of this inference is the probability distribution over the rotation velocity, defined over 5 discrete values: speed on the left, half speed on the left, null, half speed on the right, speed on the right. After describing the problem in ProBT, the toolchain is able to translate it into a VHDL circuit.

The robot example was synthesised and implemented in an Altera Cyclone IV FPGA. The generated circuit involves about 2500 components and several thousands of signals. The circuit is duplicated five times in parallel, since the searched variable has five possible values. All the binary inputs are converted into stochastic bitstreams using 32 bit linear feedback shift registers as the source of entropy (LFSRs [12] are compact and effective), and the 5 outputs are converted from stochastic bitstreams back to binary with accumulator counters.

To perform stochastic operations we used the logic AND gate for multiplication and the OR+, from [13], for the stochastic addition. Tests were run at a conservative clock frequency of 25 MHz, but optimising for speed this could easily be doubled with the current FPGA.

We performed extensive tests with the robot example in order to evaluate the accuracy of the BM output, the resource utilisation and the energy consumption on the FPGA. The output probability distributions were compared with the ground truth results acquired from the ProBT software running on a PC. Two sets of probability values need to be specified in order to test the circuit output. The first set includes internal parameters and the second set includes the soft evidence. Internal parameters of the model are the hard coded values that represent joint probability distributions. These describe the model knowledge associated to the inference computed by the circuit. The second set is the input of the circuit and represents probability distributions that depend on observations which can change over time.

A. Accuracy

Results were gathered over 20 test runs with different initialisation seeds for the LFSRs. Each test ran for 40 seconds at 25 MHz to generate stochastic bitstreams with lengths up to 10^9 bits. The output of the BM is displayed in Table I where it is compared with the ground truth value computed in ProBT.

TABLE I
Comparison between theoretical result and FPGA output for different bitstream lengths in the Robot example

Bitstream Size	V_{ROT_0}	V_{ROT_1}	V_{ROT_2}	V_{ROT_3}	V_{ROT_4}
Ref.	**0.0048**	**0.0119**	**0.0682**	**0.2929**	**0.6220**
10^3	0	0.0062	0.0958	0.2166	0.6312
10^4	0.0074	0.0078	0.0601	0.2780	0.6466
10^5	0.0034	0.0121	0.0677	0.3024	0.6142
10^6	0.0048	0.0122	0.0698	0.2909	0.6221
10^7	0.0050	0.0120	0.0683	0.2927	0.6218
10^8	0.0047	0.0118	0.0678	0.2931	0.6223
10^9	0.0048	0.0119	0.0680	0.2929	0.6222

To check the accuracy, KL divergence, a statistical measure that quantifies how close a probability distribution is to a reference model distribution [3], was also computed between the output distribution values and the expected ground truth computed in ProBT (Fig. 3). From the KL divergence plot, we can conclude that with a bitstream length of 10^6 bits we can achieve a KL divergence lower than 0.0001. At 25 MHz, this would require just 40 ms of computation time.

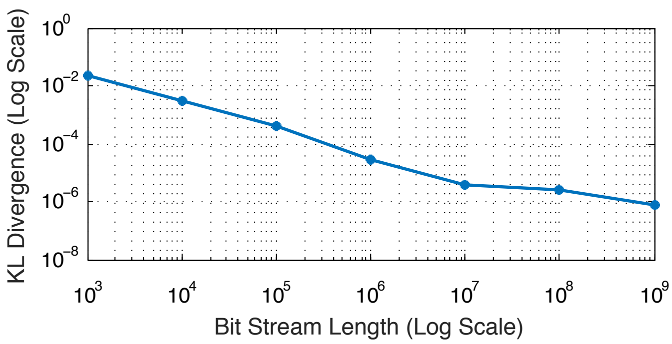

Fig. 3. KL divergence as a function of bitstream length.

B. Resource Utilisation

In Table II we can see that the robot problem used 60% of the total FPGA Logic Elements (LE). Although the BM itself uses a small amount of resources, in this case the main contribution is the bin-to-sto circuits which require 216 LFSRs. Compared to a stochastic multiplier, which is only an AND gate, LFSRs require a significant amount of resources.

TABLE II
RESOURCE UTILISATION ON ALTERA CYCLONE IV (EP4CE115F29C7)

Entity	Logic Elements	LC Registers	Memory Bits
BM	2820	1328	0
Bin-to-Sto circuits	47520	36936	0
System Total	68,832 (60%)	43,840 (38%)	257 (< 1%)

C. Power Consumption

The estimation of power consumption was based on simulation tools provided by the FPGA vendor. Results show a low instantaneous power consumption but from the energy point of view, we have to consider the trade-off between total energy and bitstream length required. In Table III power is shown for 25 and 50 MHz and we also included the energy estimation for two given lengths of bitstreams.

TABLE III
ESTIMATED POWER AND ENERGY CONSUMPTION.

Frequency	**25 MHz**	**50 MHz**
Core Dynamic	211.09 mW	421.07 mW
Core Static and I/O	133.78 mW	134.68 mW
Total Power Dissipation	344.87 mW	555.75 mW
Energy (10^3 **bits)**	13.79 uJ	11.1 uJ
Energy (10^5 **bits)**	1379.48 uJ	1111.5 uJ

If we were to use a current energy efficient computer, as listed in the Green500 (http://www.green500.org/), with an average performance per watt of 223.50 MFPLOS/W (Intel Core family), we get an estimate of 4.47 nJ for a floating point operation. For our robot example that requires 2665 operations we need about 11.9 uJ. This is the same order of magnitude as the BM for low precision, so our approach is limited. For a robot application the above numbers can be interesting when the onboard computing power is very limited, and problems with big cardinality need to be tackled. The required accuracy is also a key factor, since a shorter bitstream length has less accuracy, but gets a usable result with less energy. If direct stochastic bitstreams can be used by both sensors and actuators, this approach does have a significant edge, since the conversion circuits dominate resource usage.

V. CONCLUSIONS AND FUTURE WORK

We present a Bayesian machine FPGA implementation to perform exact inference using stochastic bitstreams. The main benefits of the proposed framework are the formal specification and design automation, and the ability to scale to any design size, depending on the resources available on the FPGA. The framework enables the automatic implementation of Bayesian Machines to perform computations using stochastic bitstreams. The approach allows us to have massively parallel circuits performing the desired computations. The trade-off between computation time and accuracy can be exploited in an adaptive way by the end use.

We present performance and resource usage of an FPGA implementation of a Bayesian sensorimotor system that performs obstacle avoidance for an autonomous robot.

Concerning accuracy, results showed an average KL divergence of less than 0.001 for bitstream lengths above 10^5 and less than 10^{-6} for bitstream lengths above 10^9.

The resource consumption for the stochastic circuits alone is very low, the generation of the bitstreams that has the LFRSs is what uses up most resources. The robot example uses up 60% of a low end FPGA, with the random number generation requiring most resources. Larger problems can still be implemented on higher end FPGAs.

Concerning power, for a robot application our implementation can be interesting when the onboard computing power is very limited, and problems with big cardinality need to be tackled. The required accuracy is also a key factor, since a shorter bitstream length has less accuracy, but gets a usable result with less energy.

Future work will address more target applications and higher dimension problems. As we saw, a limiting factor is the amount of resources used up to generate the random numbers, and we will look into more efficient solutions for this to be implemented in future custom reconfigurable devices. Nevertheless, for applications where the inputs and outputs can be direct bitstreams, FPGA implementations of the proposed Bayesian machine can be a viable solution.

REFERENCES

[1] Pierre Bessière, Christian Laugier, and Roland Siegwart. *Probabilistic reasoning and decision making in sensory-motor systems*, volume 46. Springer Science & Business Media, 2008.

[2] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT Press, 2005.

[3] João Filipe Ferreira and Lobo Dias. *Probabilistic approaches to robotic perception*. Springer, 2014.

[4] Mingjie Lin, Ilia Lebedev, and John Wawrzynek. High-throughput bayesian computing machine with reconfigurable hardware. In *Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '10, pages 73–82, New York, NY, USA, 2010. ACM.

[5] Joshua B. Tenenbaum, Eric M. Jonas, and Vikash K. Mansinghka. Stochastic digital circuits for probabilistic inference. Technical report, Massachusetts Institute of Technology, November 2008.

[6] John Von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. *Automata studies*, 34:43–98, 1956.

[7] B. R. Gaines. Stochastic computing. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), pages 149–156, New York, NY, USA, 1967. ACM.

[8] M. Faix, E. Mazer, R. Laurent, M. Othman Abdallah, R. Le Hy, and J. Lobo. Cognitive computation: A bayesian machine case study. In *Cognitive Informatics Cognitive Computing (ICCI*CC), 2015 IEEE 14th International Conference on*, pages 67–75, July 2015.

[9] M. Faix, J. Lobo, R. Laurent, D. Vaufreydaz, and E. Mazer. Stochastic bayesian computation for autonomous robot sensorimotor systems. In *Workshop on Unconventional Computing for Bayesian Inference, IEEE/RSJ International Conference on Intelligent Robot and Systems (IROS)*, 2015.

[10] Pierre Bessière, Emmanuel Mazer, Juan Manuel Ahuactzin, and Kamel Mekhnacha. *Bayesian programming*. CRC Press, 2013.

[11] Kamel Mekhnacha, Juan-Manuel Ahuactzin, Pierre Bessière, Emmanuel Mazer, and Linda Smail. Exact and approximate inference in ProBT. *Revue d'intelligence artificielle*, 21(3):295–331, 2007.

[12] Armin Alaghi and John P. Hayes. Survey of stochastic computing. *ACM Trans. Embed. Comput. Syst.*, 12(2s):92:1–92:19, May 2013.

[13] Marvin Faix, Emmanuel Mazer, Raphael Laurent, and Jorge Lobo. Cognitive computation: a bayesian machine case study. In *Cognitive Informatics and Cognitive Computing*. IEEE, 2015.