Miguel Mira Duarte Gomes

Using an FPGA Mini-Cluster to Implement Bayesian Application-Specific Integrated Circuits for Robotic Applications

Dissertação de Mestrado em Engenharia Eletrotécnica e de Computadores
09/2017

· U C ·

Universidade de Coimbra

FCTUC

UNIVERSITY OF COIMBRA
FACULTY OF SCIENCES AND TECHNOLOGY
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Using an FPGA Mini-Cluster to Implement Bayesian Application-Specific Integrated Circuits for Robotic Applications

Miguel Mira Duarte Gomes

Supervisor: Prof. Doctor Jorge Nuno de Almeida e Sousa Almada Lobo
Co-Supervisor: Prof. Doctor João Filipe de Castro Cardoso Ferreira

Jury:
Prof. Doctor Fernando Manuel dos Santos Perdigão
Prof. Doctor Gabriel Falcão Paiva Fernandes
Prof. Doctor Jorge Nuno de Almeida e Sousa Almada Lobo

Dissertation submitted to the Electrical and Computer Engineering Department of the Faculty of Science and Technology of the University of Coimbra in partial fulfilment of the requirements for the Degree of Master of Science.

Coimbra, September of 2017

Acknowledgements

I would like to begin by thanking both my thesis supervisor, Prof. Doctor Jorge Lobo, and co-supervisor, Prof. Doctor João Filipe Ferreira, at the University of Coimbra, for guiding and steering me in the right direction. I would also like to thank my colleagues at the Mobile Robotics Laboratory of the Institute of Systems and Robotics at the University of Coimbra for their companionship and help. Last, but not least I must also thank my family, my fiancée and friends for their continuous support throughout these academic years.

Thank you,
Miguel Mira Duarte Gomes

Abstract

Artificial perception systems need to be able to deal with uncertainty. A well tried solution for that problem is the use of probabilistic approaches, however these can easily overload standard architectures based on Central Processing Units (CPUs) and Graphics Processing Units (GPUs), which leads to slow computations. As opposed to this, the work presented in this dissertation uses machines developed around unconventional hardware based on stochastic signals, allowing the use of simple logic gates to perform computations and efficiently solve Bayesian inference.

These machines were developed during the European project "Bottom-up Approaches to Machines dedicated to Bayesian Inference" (BAMBI) from the Future and Emerging Technologies (FET) program. Since their architecture is not generic, meaning they would not be appropriate to implement with Application Specific Integrated Circuits (ASICs), a toolchain was developed that allows the spooling of these machines into Field-Programmable Gate Arrays (FPGAs).

By using FPGAs there are no off-the-shelf solutions to connect to a robot. To do this, the FPGAs were connected to a computer that is then able to use the Robot Operating System (ROS) to communicate with ROS-compliant robots/sensors/actuators/simulators. ROS is widely used for the development of robotic systems, and by using it we are able to design an interface that is easily integrated into many robotic projects.

In order for the host computer and the FPGAs to achieve high-speed data transfers, a PCI-Express (PCIe) connection was developed. The final objective is to provide an abstraction from all the communication layers of the interface so the user only needs to handle ROS topics to implement probabilistic models in the FPGA mini-cluster.

In Chapter 4 a case study is presented that successfully demonstrates the properties of the system.

Keywords: **Bayesian, Inference, Perception, FPGA Cluster, ROS, Rosbridge, PCIe**

"Probability theory is nothing but common sense reduced to calculation."
- Pierre-Simon Laplace

Resumo

Os sistemas de perceção artificial precisam de conseguir lidar com a incerteza. Uma solução robusta para esse problema consiste em usar métodos probabilísticos. No entanto, estes facilmente sobrecarregam arquiteturas padrão, baseadas em Unidades Centrais de Processamento (CPUs) ou Unidades de Processamento Gráfico (GPUs), o que leva a cálculos mais lentos. Por outro lado, o trabalho apresentado nesta dissertação usa máquinas desenvolvidas em redor de hardware não convencional baseado em sinais estocásticos, permitindo o uso de portas lógicas simples para realizar computações e lidar com inferência Bayesiana de forma eficiente.

Estas máquinas foram desenvolvidas durante o projecto Europeu "Bottom-up Approaches to Machines dedicated to Bayesian Inference" (BAMBI), no âmbito do programa Future and Emerging Technologies (FET). Como a sua arquitectura não é genérica, o que significa que não seriam apropriadas para implementar em Circuitos Integrados de Aplicação Especifica (ASICs), uma toolchain foi desenvolvida para permitir a implementação destas máquinas em Field-Programmable Gate Arrays (FPGAs).

Ao usar FPGAs não há soluções prontas a usar para estabelecer uma ligação a um robo. Para fazer isto as FPGAs necessitam de estar ligadas a um computador capaz de usar o Robot Operating System (ROS) para comunicar com robos/sensores/actuadores/simuladores compatíveis com ROS. O ROS é amplamente utilizado para o desenvolvimento de sistemas robóticos e ao usá-lo é-nos possível desenhar uma interface facilmente integrada em muitos projectos robóticos.

Também foi desenvolvida uma ligação PCI-Express (PCIe) para que o computador e as FPGAs alcancem transferências de dados de alta-velocidade. O objetivo final é fornecer uma abstração de todas as camadas de comunicação da interface de forma a que o utilizador apenas tenha que lidar com tópicos de ROS para implementar modelos probabilísticos num mini-grupo de FPGAs.

No Capítulo 4 apresenta-se um estudo de caso que ilustra as propriedades do sistema, bem como situações em que o mesmo possa ser aplicado com sucesso.

Palavras chave: **Bayesiana, Inferência, Perceção, Grupo de FPGAs, ROS, Rosbridge, PCIe**

List of acronyms

API	Application Programming Interface.
ASIC	Application Specific Integrated Circuits.
BAMBI	Bottom-up Approaches to Machines dedicated to Bayesian Inference.
(B)ASIC	Bayesian Application Specific Integrated Circuits.
BEAST	Bayesian Estimation And Stochastic Tracker.
BIND	Bayesian INference in DAG.
BM	Bayesian Machine.
CDE	Conditional Distribution Element.
CHREC	Center for High-Performance Reconfigurable Computing.
CPU	Central Processing Unit.
CRC	Cyclic Redundancy Check.
DAG	Directed Acyclic Graph.
FET	Future and Emerging Technologies.
FPGA	Field Programmable Gate Array.
GPU	Graphics Processing Unit.
GUI	Graphical User Interface.
IP	Internet Protocol.
JSON	JavaScript Object Notation.

JTAG Joint Test Action Group.

MCMC Markov Chain Monte Carlo.

NSF National Science Foundation.

OPS Odds Product Sampler.

OS Operating System.

OSRF Open-Source Robotics Foundation.

PCIe Peripheral Component Interconnect-Express.

RIFFA Reusable Integration Framework for FPGA Accelerators.

RNG Random Number Generator.

ROS Robot Operating System.

SR Samples Register.

URI Uniform Resource Identifier.

USB Universal Serial Bus.

VHDL Very-High-Speed Integrated Circuit Hardware Description Language.

Table of contents

List of figures

Chapter 1

Introduction

1.1 Motivation

Artificial perception systems must be able to efficiently deal with uncertainty [1, 2]. One robust solution for that problem is the use of probabilistic approaches [1, 2], however these can easily overload standard architectures based on Central Processing Units (CPUs) and Graphics Processing Units (GPUs) and quickly become intractable [3, 2].

This led to the conception of the European BAMBI project - Bottom-up Approaches to Machines dedicated to Bayesian Inference - from the Future and Emerging Technologies (FET) program. During this project new architectures that are able to efficiently solve Bayesian inference were developed. The original idea regarding a hardware implementation of these machines was to construct easy-to-deploy Bayesian Application-Specific Integrated Circuits ((B)ASICs), since ASICs are: a) energy efficient and b) have high performance [4]. However these (B)ASICs are application specific, meaning we would have to build new ones every time the probabilistic model changes [4]. Since ASICs have very high non-recurring engineering costs, a fixed function and are not reconfigurable, they may not be an appropriate platform for the deployment of these architectures [4].

FPGAs, on the other hand, posses the following characteristics:

- Reconfigurability $\implies$ no cost to produce extra ASICs for each function or update;
- A Low-Power architecture $\implies$ can be better tuned for low power and energy efficiency than CPUs and GPUs;
- Comparable performance to CPUs and GPUs, depending on the characteristics of the problem.

The reconfigurability of FPGAs allowed for the creation of a toolchain to build dedicated (B)ASIC machines when given an inference problem specified in ProBT, which is a proprietary python library from ProbaYes [3, 4].

After selecting FPGAs as the preferred means of implementing these Bayesian Machines (BMs), we still had to design an interface between them and robots, which leads us to this thesis' objective.

1.2 Objective

The main goal of the work described in this dissertation was to develop an interface, as generic as possible, between an FPGA mini-cluster and any ROS compliant robot/sensor/actuator, in order to implement probabilistic models for use on robotic applications. This comes in continuation of the work developed by Hugo Fernandes within the BAMBI project, under the supervision of Professor Doctor Jorge Lobo at the Institute of Systems and Robotics of the University of Coimbra, and aims to contribute to its development.

The chosen solution was to use a computer as the Host PC of the FPGA Mini-cluster with a Peripheral Component Interconnect-Express (PCIe) connection between them. The Host PC could then use a WebSocket connection to send/receive data to/from a robot. A high level view of such a system is presented in Figure 1.1.

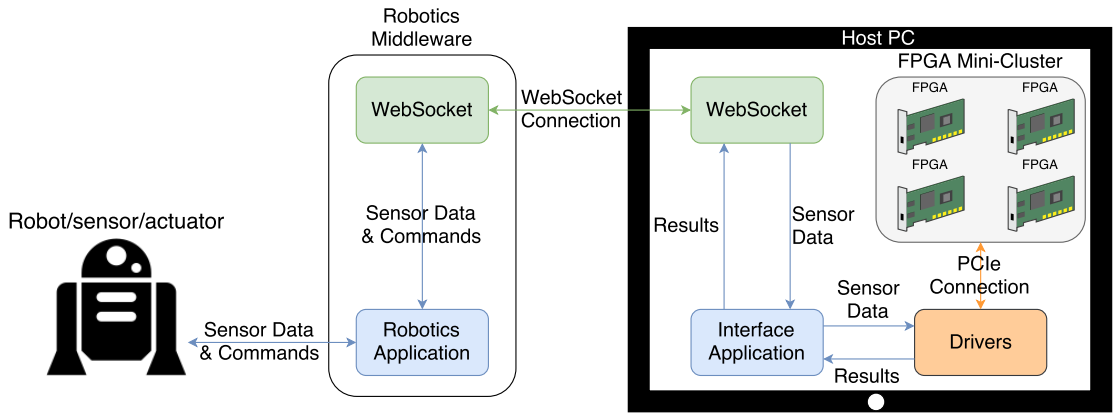


Fig. 1.1 High level view of the interface between the FPGA Mini-Cluster and a Robot.

1.3 Related Work

The BAMBI project was not the only project to develop unconventional architectures dedicated to Bayesian inference in recent years.

Two other projects, named SpinNaker and TrueNorth, present highly parallel machines that work like assemblies of neurons, however each processing unit is based on the traditional von Neumann architecture and performs either fixed or floating point operations [5].

V. Mansinghka and E. Jonas used sampling methods to compute approximate probabilistic inferences. The hardware implementation made use of hardware components performing standard operations, for example normalisation or exponentiation [5, 4].

The use of standard hardware to implement algorithms is referred to as a Top-Down approach. The BAMBI project, on the other hand, followed a Bottom-Up approach [4].

This involved designing unconventional hardware for probabilistic computation from new stochastic building blocks that allowed the parallelism to take place at the bit level. By doing so, operations like addition and multiplication can be done with very low-complexity components [5, 4].

Other teams have also developed unconventional hardware dedicated to Bayesian inference. For example, Vigoda presented architectures based on probabilities that made use of analogue signals, but these could not handle complex inference problems [5, 4].

The Nanoscale Computing Fabrics Laboratory at the University of Massachusetts Amherst [5] has also been pursuing research based in Directed Acyclic Graph (DAG) models. The Bayesian Estimation And Stochastic Tracker (BEAST) and Bayesian INference in DAG (BIND) frameworks have also been presented by Thakur [5]. It is worth mentioning that all of the projects mentioned above seem to be only theoretical and lack any robotic implementation.

The idea behind interfacing FPGAs with robots using ROS has been implemented before. In [6] a ROS-compliant FPGA component was developed which acts as the translator between the ROS topics and the hardware design. By having the hardware design and the ROS-compliant FPGA component on the same chip, high-speed data transfers between them were achieved, however the system becomes limited to only one board. To add more boards to the system would mean creating custom interfaces for those boards, meaning this system does not scale well to larger problems, a necessity for systems dealing with Bayesian inference. Furthermore FPGA resources are also required to implement the ROS-compliant FPGA component in hardware.

The system presented in this dissertation, on the other hand, can easily be scaled to bigger clusters. This is due to the ROS communication being performed in the host computer, thus freeing the FPGAs to serve their purpose of being co-processors for handling Bayesian inference computations. However this leads to the necessity of implementing a high-speed interface between the FPGAs and the host computer in order to avoid creating bottlenecks in the system. To achieve this a PCIe connection was implemented.

To establish a PCIe connection between an FPGA and a Host PC, one would usually have to design the interface in hardware, either using the FPGA logic, or a Hard PCIe block that is available in some FPGAs. These Hard PCIe blocks allow the user to create this connection without wasting precious resources on the FPGA chip. The user would then also have to design drivers for the PCIe interface or use an open-source framework like Reusable Integration Framework for FPGA Accelerators (RIFFA) [7].

In order to avoid this lengthy and troublesome process Gidel, the company that designed the boards that were used in this project, created an application that is able to instantiate both the hardware components for the PCIe interface and the drivers for the host computer.

1.4 Key Contributions

- Implementation of machines capable of efficiently computing Bayesian inference in an FPGA mini-cluster;
- A generic ROS interface for the FPGA mini-cluster that allows it to:

  Receive data from any ROS-compatible robot/sensor/actuator;

  Publish the results to a ROS topic.
- A PCIe connection between the host computer and the FPGA mini-cluster to allow for fast data transfers;

1.5 Structure of Dissertation

- **Chapter 1** contains the motivations, goal, related research and key contributions of this thesis;
- In **Chapter 2** the theoretical framework, the scientific background and methods, of this thesis are presented. To be more specific, it is in this chapter that we delve deeper into the concepts of Bayesian Machines, the Robot Operating System and PCI-Express;
- In **Chapter 3** we will go over the implementation details of both the Rosbridge connection and the PCI-Express links. An overview of the entire system is also displayed;
- In **Chapter 4** a case study is presented, along with results;
- Finally, in **Chapter 5** conclusions are drawn and future work is also presented.

Chapter 2

Theoretical Framework

2.1 Bayesian Machines

As mentioned in Section 1.1, a toolchain was devised that allows the user to build (B)ASIC machines for a given probabilistic model that is specified in ProBT. The user can further specify if the resulting machine should perform exact inference with approximate calculations, or if it should follow a sample-based approach to find an approximate solution. Depending on the choice, the toolchain will generate a BM1, for the first option, or a BM2 for the second [4].

Although their architecture and target applications may differ, they possess two things in common. First, they transform the input data into stochastic signals and perform all operations with those stochastic signals. Second, they possess a generic interface in the sense that they only receive as input the sensor data and output the full posterior probability distribution [4].

It is this generic interface of the BMs that makes them well suited to be used as co-processors for classical computers that need to run large Bayesian problems in an efficient fashion [4].

The architecture of these machines was already developed within the BAMBI project. However, the toolchain had to be adjusted, and a deep understanding of these architectures was required. In the following subsections, a summary description of the architecture and illustrative block diagrams of the modules of these machines are provided. In [4], a more elaborate description is available, however, it dwells in development details and variants that were not integrated in the final architectures of the BMs.

2.1.1 Stochastic Computing

Most of the computer performance gains of the last decades have been associated with a reduction in size of the processor's electronic components. This led to an increase in the amount of transistors one could fit on a chip. In fact Moore's Law - the observation that the number of transistors on a chip doubles approximately every two years - has been accurate for many decades, however as of recently it has been slowing down [8, 4].

In the search for a source of new performance gains old concepts have been revisited, one of them being stochastic computing. Instead of performing operations using fixed or floating-point representations, stochastic computing is based on a temporal coding of numbers by using streams of bits. A stream of n random bits codes the value $p = \frac{n_1}{n} = \frac{n_1}{n_1+n_0}$ where n_1 is the number of bits equal to one and n_0 is the number of bits equal to zero [4].

A stochastic bitstream that encodes the value p is generated by independently drawing n bits b_i following a Bernoulli law of parameter p, where: $P([b_i] = 1) = p$ and $P([b_i] = 0) = 1 - p$.

As such it can be demonstrated that if we have two stochastic bitstreams p_1 and p_2 we can obtain the bitstream that encodes the value $p_1 \times p_2$ with a simple AND gate. The operators for addition and subtraction are also very simple, therefore stochastic computing allows us to design small-area and low-power architectures [4].

Another important property of stochastic computing is its increased fault tolerance. Since no bit in a stochastic bitstream is more significant than the others, possible errors have less impact on the global results than on fixed or floating-point representations. By taking this into account we could greatly relax the accuracy requirements of the BMs and have highly energy-efficient circuits [4].

The main drawback of stochastic computing is the trade-off between precision and computation time. For a biststream obtained as n independent Bernoulli experiments of parameter p, the standard deviation of the estimated probability $\frac{n_1}{n}$ is:

$$\sigma(n_1/n) = \sqrt{p(1-p)/n} \tag{2.1}$$

According to equation 2.1 we could have an arbitrarily small error, however this would lead to a rapid increase in computation time [4]. One more problem associated with time-based representations of probabilities is related to coding very low probability values. By having multiple products between several stochastic bitstreams that encode values close to 0 we can obtain a bitstream with very few bits set to "1". This means that the number of bits necessary to get an accurate reconstruction of the distribution can become very high, which

causes the *time dilution problem*. This problem can lead to a necessity to run a machine for a very high number of cycles in order to collect the output data, which would be prohibitive for many applications [4, 5, 9].

In summary, stochastic computing provides us some advantages, such as: a) small-area and low-power architectures, and b) increased fault tolerance. However it also presents us with disadvantages: c) trade-off between precision and computation time, and d) time dilution problem.

2.1.2 BM1 - Exact Inference with Approximate Computations

The BM1 is a machine capable of performing naive Bayesian fusion in parallel where all the known variables are considered to be conditionally independent with one another for any given searched variable [4]. It computes the posterior probability distribution on a searched variable S, while knowing the prior distribution $P(S)$ and the conditional distributions $P(K_i|S)$ with $i \in [1,...,N]$ [5]. The resulting inference can be defined by equation 2.2, where Z is a normalization constant [4].

$$P(S|k_1,...,k_n) = \frac{1}{Z}P(S)\prod_{i=1}^{N}P(k_i|S) \propto P(S)\prod_{i=1}^{N}P(k_i|S) \tag{2.2}$$

If S has M possible values, we can consider $j \in [1,...,M]$, and say that the BM1 is capable of concurrently computing the M posterior probability values $P([S = s_j]|k_1,...,k_n)$ in parallel. To achieve this it uses a very simple architecture, a matrix of components with:

- Number of lines $= M$
- Number of columns $= N + 2$

Each line of the matrix is composed of a BM1 prior element, a number of BM1 tile elements equal to N and a counter where the final posterior probability calculated in that line is stored [4]. There is also a Random Number Generator (RNG) for every column of the BM1, except in the column with the counters. In Figures 2.1 and 2.2 you can see how the BM1 prior and tile elements work, in these figures the blue lines indicate binary numbers, whereas the red lines indicate stochastic bitstreams.

The prior element in Figure 2.1 receives a prior value that is used to select a probability from a table that is stored in local memory [4]. The probability values of this table are generated by the toolchain based on the implemented probabilistic model [4]. The resulting value is a binary number that represents the prior probability of the variable S: $P(S)$ [4]. The

random number and $P(S)$ are provided to the Stochastic Bitstream Generator, which then generates a stochastic signal encoding the value of $P(S)$ [4].

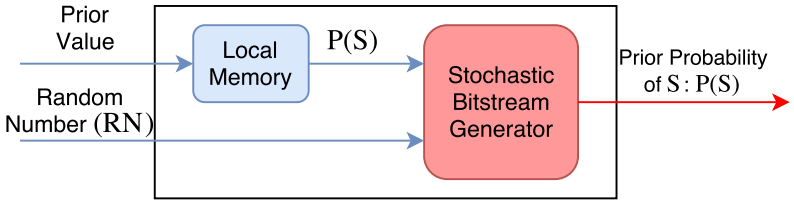


Fig. 2.1 BM1 prior element. Red lines indicate stochastic bitstreams, blue lines indicate binary numbers.

The tile element in Figure 2.2 is very similar to the prior element, the differences are that instead of receiving a prior value it receives a sensor value, K, and the prior probability, $P(S)$ [4]. The sensor value is used to select the conditional probability of $P(K|S)$. The resulting binary number is then converted into a stochastic bitstream and multiplied with the prior probability with an AND gate [4]. The resulting stochastic bitstream encodes the value $P(S)P(K|S)$ which is proportional to the posterior probability $P(S|K)$ [4].

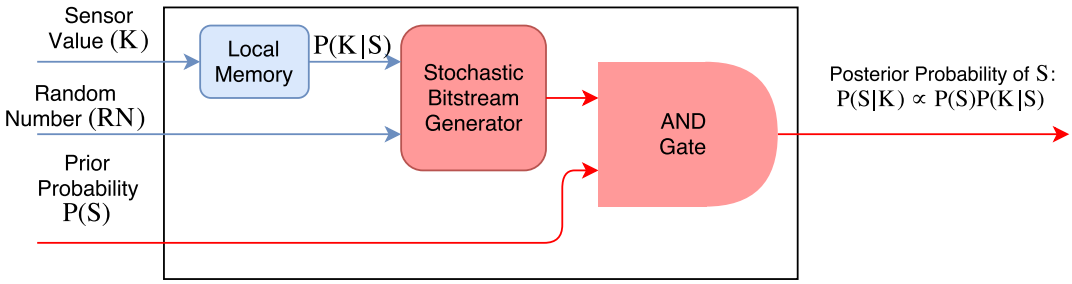


Fig. 2.2 BM1 tile element. Red lines indicate stochastic bitstreams, blue lines indicate binary numbers.

By correctly arranging these blocks we are able to compute $P(S)\prod_{i=1}^{N} P(k_i|S)$ for every searched variable. The last step necessary to obtain the full posterior probability distribution is to normalize the final results. The counters are responsible for: a) recovering the full posterior probability as a numeric representation, and b) perform the normalization [4, 5].

In order to recover the probability value of each line as a numeric representation, each counter integrates the incoming stochastic signals by counting the number of '1' bits (n_1) in a given number of clock cycles n. This results in the fixed point representation of the value $\frac{n_1}{n}$, which is an estimation of the probability value encoded by the stochastic bitstream. The accuracy of this estimation increases with the size of the counters and the integration time (n)

due to the trade-off between accuracy and computation time when dealing with stochastic signals [4, 5].

In each counter there is a component that detects when its buffer overflows. The first counter to overflow is considered to correspond to the maximum probability value. When this happens all other counters need to stop. The maximum count value, at overflow, is then used to normalise all other counters to obtain the output distribution up to a scale factor λ.

$$P([S = s_j]|k_1,...,k_n) = \lambda \frac{count_j}{count_{max}} \tag{2.3}$$

By multiplying the values in every line of the matrix by the same factor, we are able to preserve the distribution profile since only the normalization constant, Z, from the inference equation 2.2 is modified [4]. The scale factor λ is determined by the fact that $P([S = s_j]|k_1,...,k_n)$ is a probability distribution, i.e.: $\sum_{j=1}^{M} P([S = s_j]|k_1,...,k_n) = 1$.

2.1.2.1 BM1 Example - Boat Localization Problem

In this problem we have to find the location of a boat in a 64×64 grid using only the values provided by the sensors of the boat.

There are three beacons on the shore and the boat has both distance and bearing sensors that are capable of detecting those beacons. An illustration of this problem can be seen in Figure 2.3. The distances and bearings from the boat to each beacon are stored in the following variables: D1, D2, D3, B1, B2 and B3. These variables become the known variables of the probabilistic model, whereas the searched variables are the grid cells.

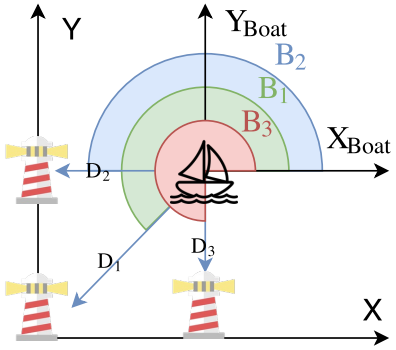


Fig. 2.3 Illustration of the Boat Localization Problem.

The hardware implementation of the boat localization problem can be seen in Figure 2.4. The number of lines, M, is equal to the number of grid cells, in this case $64 \times 64 = 4096$. The number of columns equals the number of sensors, 6, plus two.

You can see that the random numbers for each column are shared between all the elements of that column. This allows us to save hardware resources, however it is only possible because all of the searched variables (lines in the matrix) are considered to be independent [4]. Another important detail is that the number of searched variables has no influence on the amount of clock cycles the BM1 has to spend to compute the posterior probabilities. This number only affects the size of the machine and the amount of resources required. The determining factors in the amount of time the machine has to spend to obtain accurate results are: a) the number of sensors or known variables in the module (columns), and b) the size of the counters.

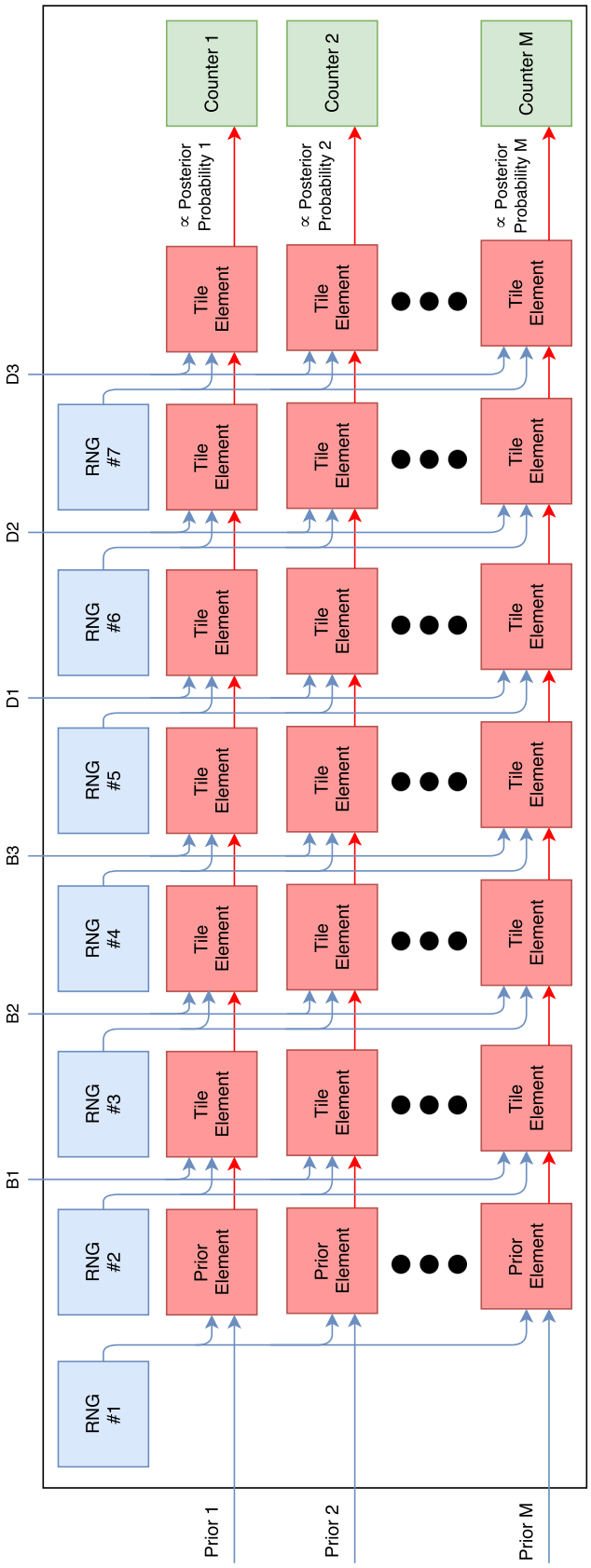


Fig. 2.4 BM1 architecture for the boat localization problem. Red lines indicate stochastic bitstreams, blue lines indicate binary numbers. Red blocks compute stochastic bitstreams, blue blocks output binary numbers and green blocks store the probability values as unsigned integers.

2.1.3 BM2 - Approximate Solution of Otherwise Intractable Problems

In the BM1 the stochastic circuits are based on an exhaustive inference paradigm where the computation trees, corresponding to the considered inferences, are fully represented in the circuits [4]. As the size of the probabilistic problems increases and the marginalization over high cardinality variables becomes necessary, the BM1 starts to present scalability problems in terms of size and power consumption [4]. This issue arises from the fact that probabilistic inference is an NP-hard problem, therefore computing the exact output distribution for some inference problems can quickly become intractable [4].

An alternative approach involves using approximate inference and output samples drawn from the target distribution by using Markov Chain Monte Carlo (MCMC) sampling techniques based on Metropolis-Hastings or Gibbs sampling [4]. The concept and design of the BM2 followed this alternative approach. This machine follows a sampling inference paradigm that allows us to find an approximate solution for probabilistic problems that would be intractable in exact inference [4, 10].

The BM2 uses a bit level Gibbs sampling algorithm: an approximate inference algorithm based on Gibbs sampling where, given a decomposition of the joint probability distribution over some discrete variables, it efficiently produces samples in the joint space bit-by-bit without excessive memory requirements [4]. By adapting the Gibbs sampling algorithm to operate at the bit level it becomes possible to implement approximate inference by sampling in stochastic circuits [4]. Considering a set of discrete and finite variables $V = \{V_i\}$ partitioned into two sets, the known variables $K = \{K_i\}$ and unknown variables $U = \{U_i\}$, and $P(K,U)$ as a joint probability distribution over these variables, we end up with Algorithm 1 [4].

Algorithm 1: Gibbs sampling

Initialization:

for $K_i \in K$ **do**
  Set K_i to the thrown values k_i
end
for $U_i \in U$ **do**
  Set variable U_i to a value u_i drawn randomly
end

Sampling:

while *TRUE* **do**
  for $U_i \in U$ **do**
    Draw a sample u_i of U_i according to $P(U_i|\{k_j\},\{u_{j\neq i}\})$
    Update U_i with the value u_i
    Output the current values of all variables U_i as a sample on the joint space U
  end
end

Fig. 2.5 Bit level Gibbs sampling algorithm.

This algorithm tells us that all the bits of the set of unknown variables need to be sampled individually in order for the BM2 to output a sample on the posterior distribution $P(U|\{k_i\})$, however the difficulty lies in properly drawing samples according to the conditional distributions [4]:

$$P(U_i|\{k_j\},\{u_{j\neq i}\}) \tag{2.4}$$

Because we are drawing one sample, u_i, at each iteration of the algorithm, equation 2.4 becomes reduced to making a binary choice between $u_i = 1$ or $u_i = 0$, which we can denote as the following Bernoulli distribution of parameter p [4]:

$$P([U_i = 1]|\{k_j\},\{u_{j\neq i}\}) = p \tag{2.5}$$
$$P([U_i = 0]|\{k_j\},\{u_{j\neq i}\}) = 1 - p \tag{2.6}$$

From this distribution it is possible to generate a stochastic bitstream of size n encoding the value of p, however it may take a long time to get accurate results. If we revisit equation 2.1 we find that the representation error of a stochastic bitstream decreases as the inverse of the square root of n [4]. Since we are using a sampling algorithm, we are able to produce a single sample with the right probability at each iteration by foregoing the need to explicitly compute p with equation 2.7. If we were to use this equation it would be necessary to compute the normalisation constant $P(\{k_j\},\{u_{j\neq i}\})$ [4].

$$p = P([U_i]|\{k_j\},\{u_{j\neq i}\}) = \frac{P([U_i],\{k_j\},\{u_{j\neq i}\})}{P(\{k_j\},\{u_{j\neq i}\})} \tag{2.7}$$

Instead we can use the *odds* representation to parametrise the stochastic bitstreams with the following relations: $o = \frac{p}{1-p}$, $p = \frac{o}{1+o}$. By using this representation we obtain equation 2.8 where we no longer need to compute the normalisation constant [4].

$$o = \frac{P([U_i = 1]|\{k_j\},\{u_{j\neq i}\})}{P([U_i = 0]|\{k_j\},\{u_{j\neq i}\})} = \frac{P([U_i = 1],\{k_j\},\{u_{j\neq i}\})}{P([U_i = 0],\{k_j\},\{u_{j\neq i}\})} \tag{2.8}$$

If we assume that the joint distribution over the $|V|$ variables of V is given as a product of positive functions, we can obtain equation 2.9, where $S_j \subseteq \{V_1,...,V_{|V|}\}$ denotes the set of all variables of the positive function $f_j : S_j = \{S_1^j,...,S_{|S_j|}^j\}$ [4]:

$$P(V_1,...,V_{|V|}) = \prod_j f_j(S_j) \tag{2.9}$$

If $M = \{j : U_i \in S_j\}$ is the set of indices j such that U_i is a variable of function f_j, then we can define equation 2.10 where the *odds o* can be seen as a product of *odds* o_m [4]:

$$o = \prod_{m \in M} \frac{f_m(s_1^m, ..., s_{l-1}^m, U_i = 1, s_{l+1}^m, ..., s_{S_m}^m)}{f_m(s_1^m, ..., s_{l-1}^m, U_i = 0, s_{l+1}^m, ..., s_{S_m}^m)} = \prod_{m \in M} o_m \tag{2.10}$$

At each iteration of the Gibbs algorithm a sample is drawn from the distribution of equation 2.4 thanks to a bitstream of *odds o*, which can be obtained as a product of *odds* o_m [4].

Since Algorithm 1 only requires one sample to be drawn with odds o it becomes unnecessary to evaluate with precision the value of $o = \prod_m o_m$ [4]. In fact, if we take into consideration a set of independent bitstreams $(S_1, ..., S_n)$ that are generated with odds $(o, ..., o_n)$ we can obtain a sample S with *odds o* as soon as all the bitstreams are equal [4].

To implement Algorithm 1 in hardware, the BM2 uses four types of components:

- The Sequencer;
- The Samples Register (SR);
- The Conditional Distribution Encoders (CDEs);
- And the Odds Product Samplers (OPS).

In order to select all the bits of the set of unknown variables, U, one by one, we use the Sequencer. This component updates the index of the pointed bit in a linear or random order [4].

The state of the system and the final sample of the joint space are stored in the Samples Register [4].

The state of the system, along with the index of the pointed bit are used by the n number of CDEs in the machine to select the correct *odds* o_m value from a table stored in local memory. This value is then compared with a random number to generate a stochastic bitstream encoding the value of o_m [4]. An illustration of a CDE can be seen in Figure 2.6.

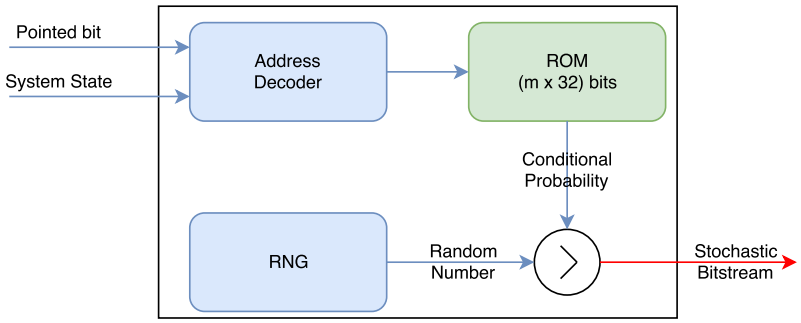


Fig. 2.6 The Conditional Distribution Encoder (CDE). Red lines indicate stochastic bitstreams, blue lines indicate binary numbers.

The resulting bitstreams of two CDEs become the inputs of an OPS. This component, shown in Figure 2.7, is able to detect the moment when the incoming bitstreams are equal and outputs a sample S with *odds* o and a validation flag V. The new sample is sent to the SR and is used to update the registers that store the state of the system. The validation flag informs the Sequencer it needs to iterate the pointed bit and is used by the SR to determine which registers are enabled [4].

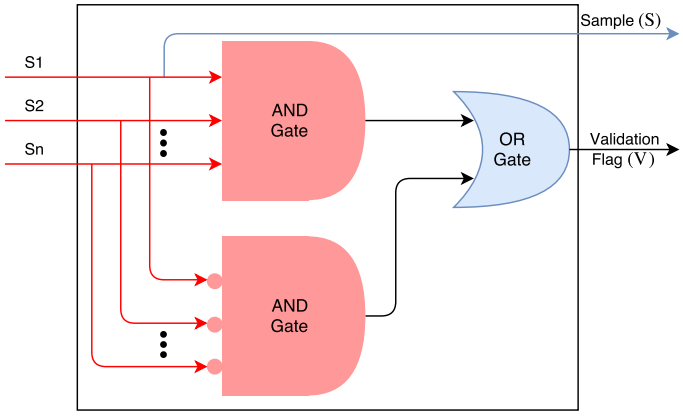


Fig. 2.7 The Odds Product Sampler (OPS). Red lines indicate stochastic bitstreams, blue lines indicate binary numbers and the black lines indicates validation flags.

In addition to the validation flag, V, and the new sample, S, the SR also receives the pointed bit for sampling from the Sequencer and the data of the known variables from the sensors. The pointed bit acts as an enable signal for the registers in conjunction with the validation flag. The data from the known variables is used to update the values of their respective registers of the state of the system. The output of this component is a binary number that describes the current state of the system.

2.1.3.1 BM2 Example - Sprinkler Bayes net

The classical "sprinkler" Bayes net example will now be used to display how all of the components of the BM2 are connected. This example can also be seen in [10]. Considering R, S and G as binary variables that correspond, respectively, to: “it has been Raining”, “the Sprinkler was turned on” and “the Grass is wet”. The joint probability distribution $P(R,S,G)$ is decomposed as $P(R)P(S|R)P(G|R,S)$. Our objective is to compute the probability of $P(R|G=g)$, meaning: the probability that it rained, knowing that the grass is wet.

From the model question we obtain the known, K, and unknown, U, variables. For this example they are $K=\{G\}$ and $U=\{R,S\}$. According to Algorithm 1 we need to sample, in odds, every bit of U. This is achieved with the formulas:

$$O(R|G=g,S=s)=\frac{P(R=1)}{P(R=0)}\cdot\frac{P(s|R=1)}{P(s|R=0)}\cdot\frac{P(g|R=1,s)}{P(g|R=0,s)} \tag{2.11}$$

$$O(S|G=g,R=r)=\frac{P(r)}{P(r)}\cdot\frac{P(S=1|r)}{P(S=0|r)}\cdot\frac{P(g|r,S=1)}{P(g|r,S=0)} \tag{2.12}$$

The ratios in these formulas are stored in memories in the CDEs. The number of CDEs of the machine are directly related to the decomposition of the joint probability distribution, in this case we will need three of these components. Since we have three CDEs we will also need two OPS.

The SR for this problem is shown in Figure 2.8. It uses the validation flag and the pointed bit for sampling to only update the register of the value that was sampled. Since the known variable is G, it is the input bit that determines whether the register G is updated or not.

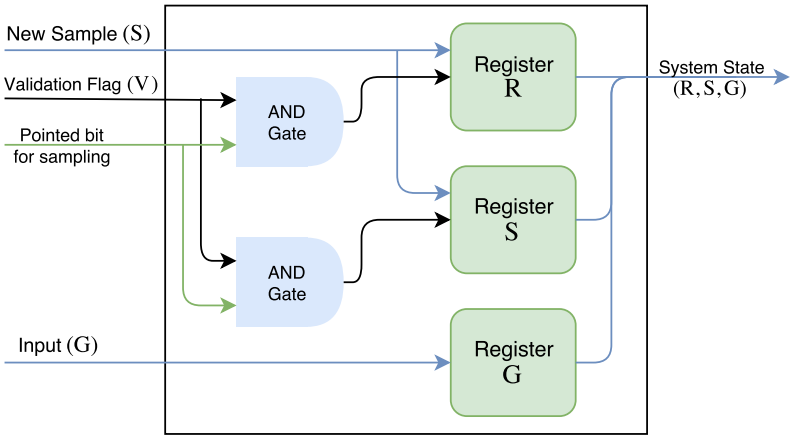


Fig. 2.8 The Sample Register (SR) of the BM2 for the Sprinkler Example. Blue lines indicate binary numbers, black lines indicate validation flags and green lines indicate the pointed bit for sampling.

By properly combining these components and the Sequencer we obtain the circuit in Figure 2.9. The input of the machine is the value of the observed variable G indicating if the grass is wet or not. The output is a sequence of joint space samples.

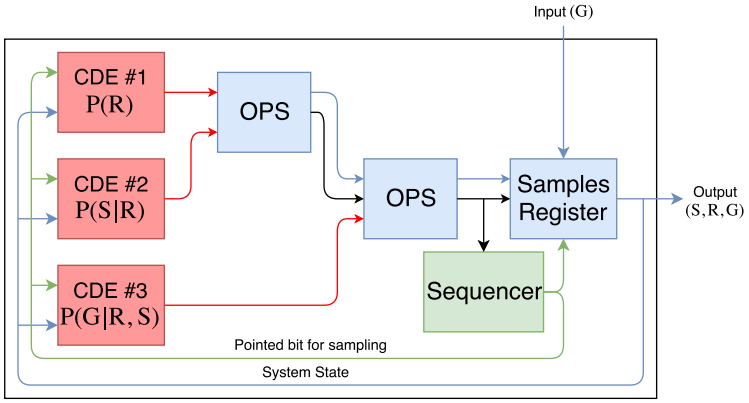


Fig. 2.9 BM2 architecture for the "sprinkler" Bayes net example. Red lines indicate stochastic bitstreams, blue lines indicate binary numbers, black lines indicate validation flags and green lines indicate the pointed bit for sampling. Red blocks compute stochastic bitstreams, blue blocks output binary numbers, the green block outputs the pointed bit for sampling.

2.2 The Robot Operating System (ROS)

Despite its name, ROS is not an Operating System in the traditional sense; it is however an open-source framework that provides hardware abstraction, low-level device control, message-passing between processes, and package management for robotic applications [11, 12]. ROS is built for UNIX platforms [11] and its aim is to build robotic systems with software components for robotic research and development with support for software reuse [6].

In this framework we design robotic systems using one or more components called nodes [6, 12]. These nodes communicate with each other by sending/receiving messages through topics or services in an assyncronous communication model called Publish-Subscribe messaging [6]. By using this model we can have a dynamic network where nodes can easily be added or removed [6].

Each ROS node can publish and/or subscribe to any given topic/service, as long as it has the name of the topic/service and the type of the message published on that topic [12]. A typical example of two nodes communicating in a ROS network is given in Figure 2.10. One node publishes a message on a given topic, making the message available for any node that has subscribed to that topic [6]. ROS topics act in a similar way to buses, not buffers, so if a node were to subscribe to that topic at a later time it would not receive the same message [6].

ROS nodes are unaware of other nodes in the network [6]. In order for them to communicate with each other there needs to be a ROS Master that provides naming and registration services, as well as tracking publishers and subscribers to any given topics [12]. This ROS Master can be any computer that has a ROS distribution installed.

Fig. 2.10 Communication model between ROS nodes.

2.2.1 Rosbridge

Rosbridge provides a JavaScript Object Notation (JSON) Application Programming Interface (API) that allows non-ROS programs to communicate with ROS nodes [13]. It's, in essence, a JSON interface to ROS that encapsulates topics' and nodes' information in JSON based commands and makes it available through a WebSocket [13].

JSON is a lightweight data-interchange format [12] and by using it the data can be read or written by any programming language that can handle that format, such as Java, JavaScript, Python or C++.

In Figure 2.11 you can see how the rosbridge protocol works:

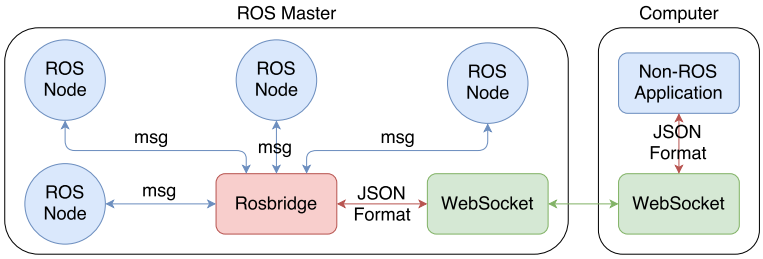


Fig. 2.11 Rosbridge communication model.

2.3 Peripheral Component Interconnect-Express (PCIe)

At the beginning of this thesis two choices for the interface between the CPU and the FPGA mini-cluster were available to us: Universal Serial Bus/Joint Test Action Group (USB/JTAG) and PCIe.

According to Intel FPGA [14] the USB/JTAG connection has a maximum speed of 24 Mbps, on the other hand PCIe has displayed an average throughput of 20800 Mbps in read/write operations in our system. This means that the PCIe connection is on average around 800 times faster than a traditional USB/JTAG connection.

This number was obtained with the diagnostics application provided by Gidel - the vendor of the boards used in this thesis - with PCIe Gen 2 in a computer running CentOS 6.9.

PCIe is the new generation of I/O interconnect technology of the PCI bus. It is a high performance expansion bus standard for connecting a computer to one or more peripheral devices. It defines standards for the physical, data link and transaction layers, with the physical layer being replaceable [15]. A high level diagram of the layers of the PCIe protocol is shown in Figure 2.12. [16].

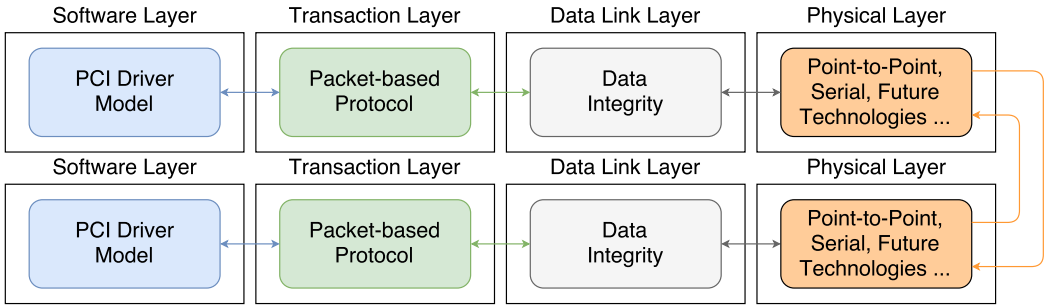


Fig. 2.12 High level view of the PCIe architecture.

The PCI drivers in the software layers generate read and write requests which are then transported by the transaction layers to the I/O devices through a packet-based protocol. The data link layer is responsible for adding sequence numbers and Cyclic Redundancy Checks (CRCs) to these packets. Lastly, the physical layer transmits and receives data through PCIe links like the one presented in Figure 2.13

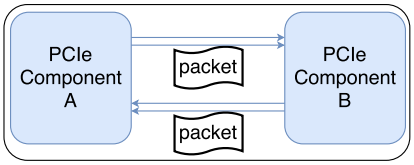


Fig. 2.13 PCIe link.

In these links data is transferred over two differential signal pairs, with each set being called a "lane". In each lane there are two wires for transmitting and two for receiving, that allow for the transmission and reception of eight-bit data packets simultaneously between the two endpoints of the link [17]. The number of lanes in a PCIe link can scale from one to

thirty two and the number used has a direct impact on the transmission throughput between the two endpoints [17].

Although PCIe preserves many features of PCI - like its usage model, load-store architecture, and software interfaces - there is one major difference. Instead of PCI's parallel architecture, it uses a highly scalable serial interface [7].

By using a serial interface, PCIe allows for a lower latency and higher data transfers between devices than the former PCI bus. This is due to devices not having to compete for bandwidth since they are connected to the motherboard with their own PCIe link and are not sharing the same bus [17].

A high level diagram of the interface between the FPGA mini-cluster and the CPU and memory of the host computer can be seen in Figure 2.14. In this system the Root Complex becomes the root of the I/O hierarchy that connects the CPU and memory to the FPGA mini-cluster. The PCIe endpoints implemented on the FPGAs can be the requesters or completers of a PCIe data transaction [7].

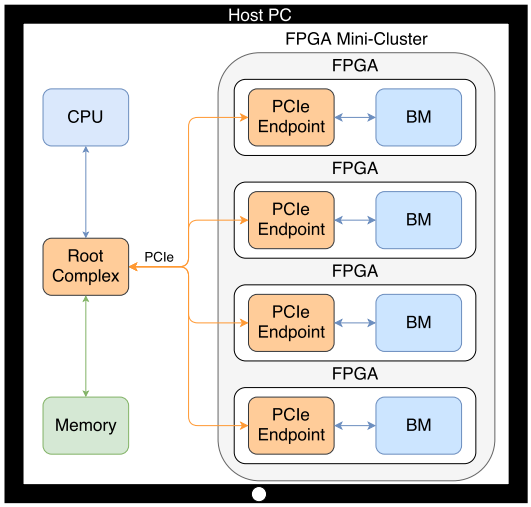


Fig. 2.14 PCIe Interface for the FPGA mini-cluster.

In the case that the FPGA has Hard PCIe controllers, these can be used to implement the PCIe endpoint and save between 8000 to 30000 logic elements, depending on their configuration. These Hard PCIe controllers embed the PCIe protocol stack into the FPGA, including the transceiver modules, physical layer, data link layer, and transaction layer [18]. They are present in many FPGAs developed by Intel FPGA, including the devices of the Stratix V family (Intel FPGA's flagship devices), the family of the devices used during this thesis [18].

Chapter 3

Implementation

As mentioned in Section 1.2, the objective of this thesis was to develop an interface between an FPGA mini-cluster and a ROS compliant robot, in order to implement probabilistic models for use on robotic applications. Also in this section, Figure 1.1 displayed the original idea for an implementation.

This idea has since matured into the system presented in Figure 3.1.

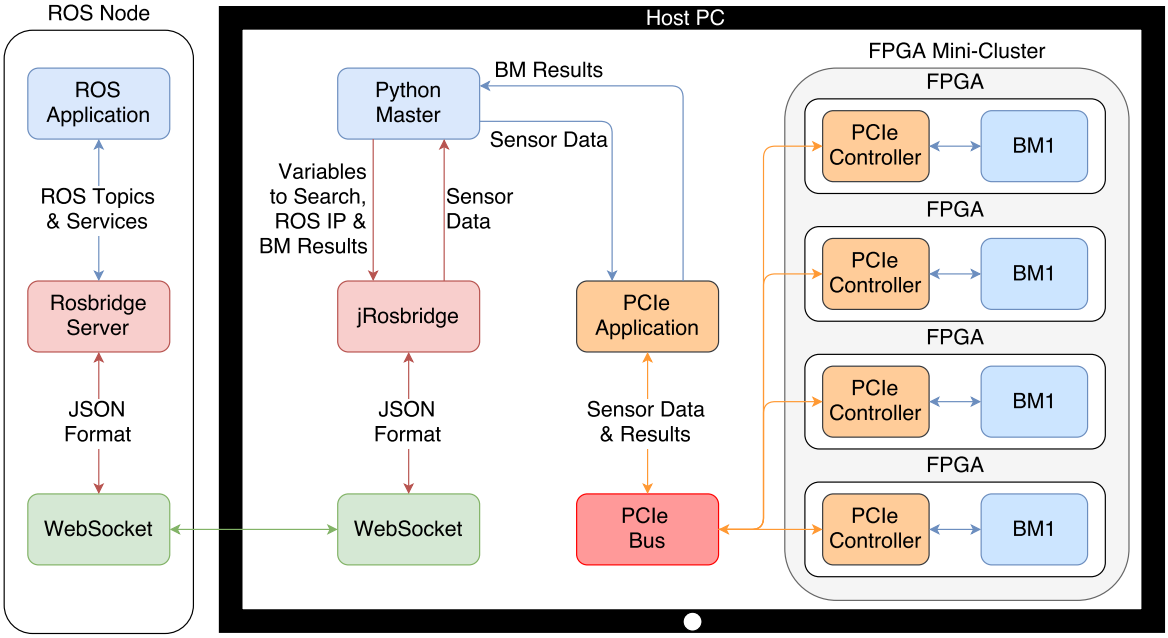


Fig. 3.1 High level view of the implemented system.

This system can be split into four parts:

1. The jRosbridge → implements the Rosbridge protocol and secures communication with any ROS node through a WebSocket;
2. The PCIe application → controls the data transfers between the host computer and the FPGA mini-cluster through the PCIe connection;
3. The Python master script → manages the system by controlling the flow of data between the jRosbridge and the PCIe application according to the implemented probabilistic model;
4. The FPGA mini-cluster → processes the incoming data with Bayesian Machines.

3.1 jRosbridge - Implementing a Rosbridge in a Computer Without ROS Compatibility

Creating a connection between a computer and a ROS compliant robot can be achieved by installing a ROS distribution with the Rosbridge packages in the computer, however ROS is not compatible with all Operating Systems. The Proce V boards used in our system are only compatible with Windows 7 or CentOS 6, which in turn do not have support for ROS.

To circumvent this problem a Java API called jRosbridge was used. The jRosbridge provides an abstraction from the Rosbridge handshake and allows us to communicate with other ROS nodes through a Rosbridge connection, even without installing ROS. Its only requirement is Java 8, which can be installed in most modern Operating Systems.

Two '.jar' files were created using this API: a) the *'Subscriber.jar'* and b) the *'Publisher.jar'*

The *'Subscriber.jar'* receives as inputs:

- The Rosbridge WebSocket Uniform Resource Identifier (URI) of the desired ROS node;
- The topic where the sensor data is published;
- The type of that topic;
- The name of the variables it needs to search for.

It then establishes a Rosbridge connection to the desired ROS node and searches the obtained JSON message recursively until it finds the variable(s) it is looking for. If it fails to find the variable(s), it outputs an error indicating which variable it failed to find.

The *'Publisher.jar'* receives as inputs:

- The Rosbridge WebSocket URI of the desired ROS master;
- The message type containing the number of results from the counters of the BM;
- The results of the BM.

This file is responsible for publishing the results of the BM so they become available to any ROS node in the same ROS network. The user can then create a ROS file that subscribes to the topic *'/BMresults'* and performs any desired operation with the results.

In Figure 3.2 an illustration of the implemented Rosbridge connection is presented.

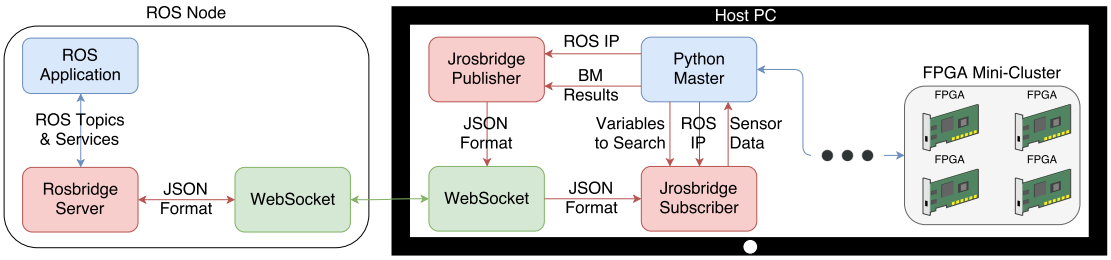


Fig. 3.2 Implementation of a Rosbridge between the host computer and a ROS node with the jRosbridge API.

3.2 PCIe Application - Communicating with FPGAs Through a PCIe Connection

At the beginning of this thesis it became apparent that in order to develop this interface, changes would have to be made to the toolchain and the architecture of the BMs. Due to time constraints it was only possible to implement the changes to one of them: the BM1.

Since the number of inputs and outputs used by the BM1 can vary wildly with the probabilistic model that is implemented, it was not possible at the time of writing to design a completely generic PCIe connection. Instead we followed the hardware requirements of the next generation of BMs that are currently being developed. These requirements stipulated that a future ASIC containing a BM would need to be able to receive the data from around

twenty different sensors and compute the data corresponding to around a thousand and twenty four different search variables.

With this in mind a PCIe connection was designed with *Gidel's ProcWizard* software tool that has the ability to transfer data from twenty different sources to the FPGA and a thousand and two hundred different sources from the FPGA. After specifying this interface, the *ProcWizard* generated:

- The required files for our hardware project;
- A header file containing the necessary libraries for a software application in *C++* to interact with an FPGA containing that interface.

These new files meant that slight alterations to the architecture and finite state machine of the BM1 had to be performed to accommodate for the PCIe connection. The finite state machine of the BM1 now sends a signal through the PCIe connection when a counter overflows, indicating the end of the computation of the full posterior probability distribution. It then waits for a signal from the PCIe connection indicating it should restart its operations.

In terms of the architecture, connections had to be made to registers in order for the BM1 to be able to receive the sensor data and output the respective results through a PCIe connection.

A new hardware project of the BM1, that already has a PCIe connection compliant with the hardware requirements mentioned above, was created. The only files that need to be updated when the probabilistic model is changed are:

1. The *'bm1_parameters.vhd'* in the *'bm_arch'* folder of the project;
2. The files contained in the *'Memory_MIFs'* folder that is also inside the *'bm_arch'* folder of the project.

Number 1) is used to define the number of lines, columns and which RNG will be used in the BM1. It also specifies: a) if the RNGs will be shared throughout the columns, b) the resolution of the probability tables, RNGs, sensor data and counters. The files referenced in point 2) define the probability tables of the BM1 prior and tile elements, and also the tables of the RNGs. When the user generates a new project, all of these files will be automatically updated by the toolchain based on the given probabilistic model.

By having a single file that stipulates the parameters that the BM1 needs to follow, we could also use it to adjust the PCIe connection automatically to the implemented probabilistic model, thus using only the necessary resources in the FPGA(s). The PCIe interface generated

by the *ProcWizard* also uses the Hard PCIe controllers present in the Proce V D8 boards. These controllers allow us to save resources on the FPGA(s) and provide a more energy efficient solution than implementing a controller with Logic Elements.

By taking into account the new finite state machine of the BM1, an application was designed in *C++* that implements the data flow illustrated in Figure 3.3. First the application writes in a register the number of target samples the BM1 needs to count before it outputs its results. Then it resets all of the components of the BM1 and writes the first string of sensor data. After this the application waits for the done signal from the BM1, indicating that it has computed the full posterior probability distribution. When the signal is received, the registers of the BM1 containing the results are read and the data is transferred to variables of the PCIe application. At this point a new string of sensor data is written to the BM1, the counters are reset and the finite state machine returns to the computing state and starts processing the new data. The loop is then restarted and its execution is only terminated when the Python Master script closes the PCIe application.

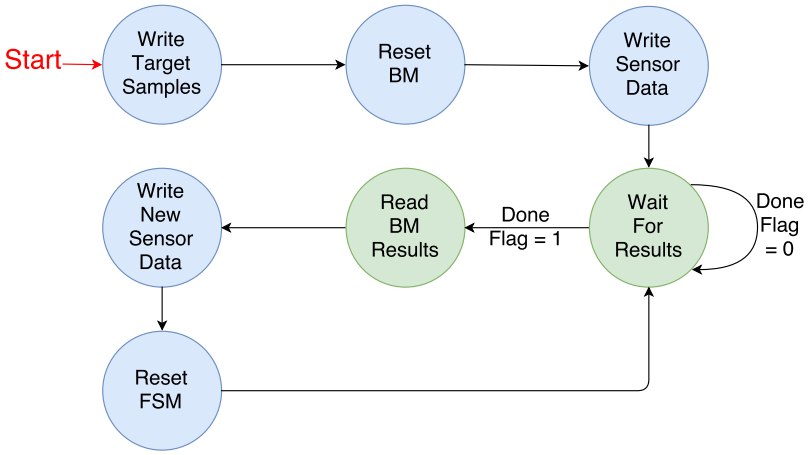


Fig. 3.3 Data flow of the PCIe connection from the Host PC's side. Blue nodes indicate write operations performed by the PCIe application, green nodes indicate read operations.

3.3 Python Master Script - Controlling the System

The Python Master script is responsible for executing the jRosbridge files and the PCIe *C++* application and transferring data between them. It also handles any errors returned by these files.

The script receives the sensor data from the *'Subscriber.jar'*, sends it to the PCIe application, waits for a *'done'* signal from it, reads the results of the BM(s) and sends them to the *'Publisher.jar'* file so they can be published on the topic *'/BMresults'*. It can do this once or continuously. When the script terminates its execution it also terminates the jRosbridge files and the PCIe application. An illustration of the data flow can be seen in Figure 3.4.

The Python Master script will be generated by the toolchain so that its parameters are adapted to the desired probabilistic model.

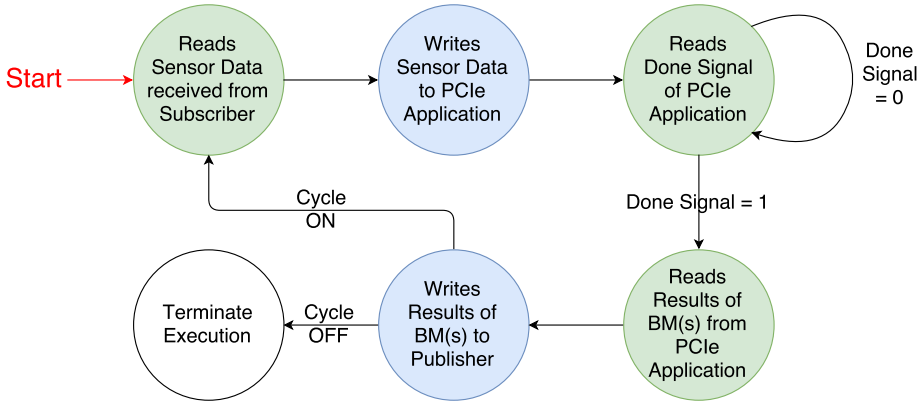


Fig. 3.4 Data flow of the Python Master script. Blue nodes indicate write operations performed by the script, green nodes indicate read operations.

3.4 Implementing Bayesian Machines in the FPGA Mini-Cluster

The use of this interface is relatively simple from a user's point of view. The user is first required to define a probabilistic model with the ProBT library.

He can then use the toolchain to generate: a) the memory files for the probability tables, which will be stored in the *'Memory_MIFs'* folder, b) the *'bm1_parameters.vhd'* file, which will be stored in the *'bm_arch'* folder, and c) the Python Master script.

The parameters the user has to give the toolchain so it can generate those files are:

- The probabilistic model;
- A number of options for the implementation of the BM1, including the type of RNG to be used;
- The number of partitions of the probabilistic model (only one by default). Useful if the model is too big to implement in a single FPGA. Can split up to a maximum of 4. If the user chooses to do this he will have to program more than one board;
- The ROS URI of the intended ROS master;
- The topic and type of topic where the sensor data will be published;
- The name of the variables of the sensors. This must be provided in the correct order, otherwise the data sent to the BM1 will be sent to the wrong columns.

All of the other files will then be automatically created, including the binary files for programming the FPGAs.

In case any alterations have to be made to the project, the toolchain provides the user options to:

- (Re)Generate the memory and parameters files;
- (Re)Compile the BM1 hardware project;
- (Re)Write the Python Master script;

This can be useful in many cases, such as not providing the correct order of the variables to the toolchain.

The *'SofLoader'* application, provided by Gidel, allows the user to program the FPGA(s) in the mini-cluster through the USB/JTAG port without having to restart the host computer. A restart would usually be necessary so the host computer could detect the new PCIe board(s).

After programming the FPGA(s) in the mini-cluster, the user can publish the sensor data in the ROS topic he provided to the toolchain, run the Python Master script, and subscribe to the ROS topic *'/BMresults'* to obtain the output of the BM1(s). After subscribing to the ROS topic *'/BMresults'*, the user can choose how to handle the data.

In Chapter 4 a case study is given of the operation of the toolchain for a probabilistic model.

Chapter 4

Case Study

The case study presented in this chapter consists of an implementation of the probabilistic model of the boat localization problem, previously mentioned in Section 2.1.2.1. This model is implemented in a BM1 with the system presented in Chapter 3.

Although this is a very simple problem of Bayesian inference, it has been used in many instances of the BAMBI project as a benchmark and a demonstration of the potential of the BM1. Therefore, it is an adequate case study to demonstrate the added value of the work presented in this dissertation.

Due to the unavailability of the required material for a robotic implementation, a simulator was used. The chosen simulator is named Gazebo and was selected due to its ability to use plug-ins to publish simulation data to ROS topics. It was installed, along with ROS and the Rosbridge packages, in a separate computer. A ROS node was also created in that computer that normalizes the simulation data and publishes it to another ROS topic. A Rosbridge is then used to make this ROS topic available to other computers through a WebSocket.

The system discussed in Chapter 3 can then acquire the sensor data through a WebSocket and process it in the FPGA mini-cluster. As mentioned in Section 3.1, the results are then published in the ROS topic *'/BMresults'*.

In the computer where Gazebo and ROS were installed, a new ROS application was created that subscribes to the ROS topic *'/BMresults'* and generates a graphical representation of the position of the boat in the grid according to the obtained results.

A block diagram of this experiment is shown in Figure 4.1.

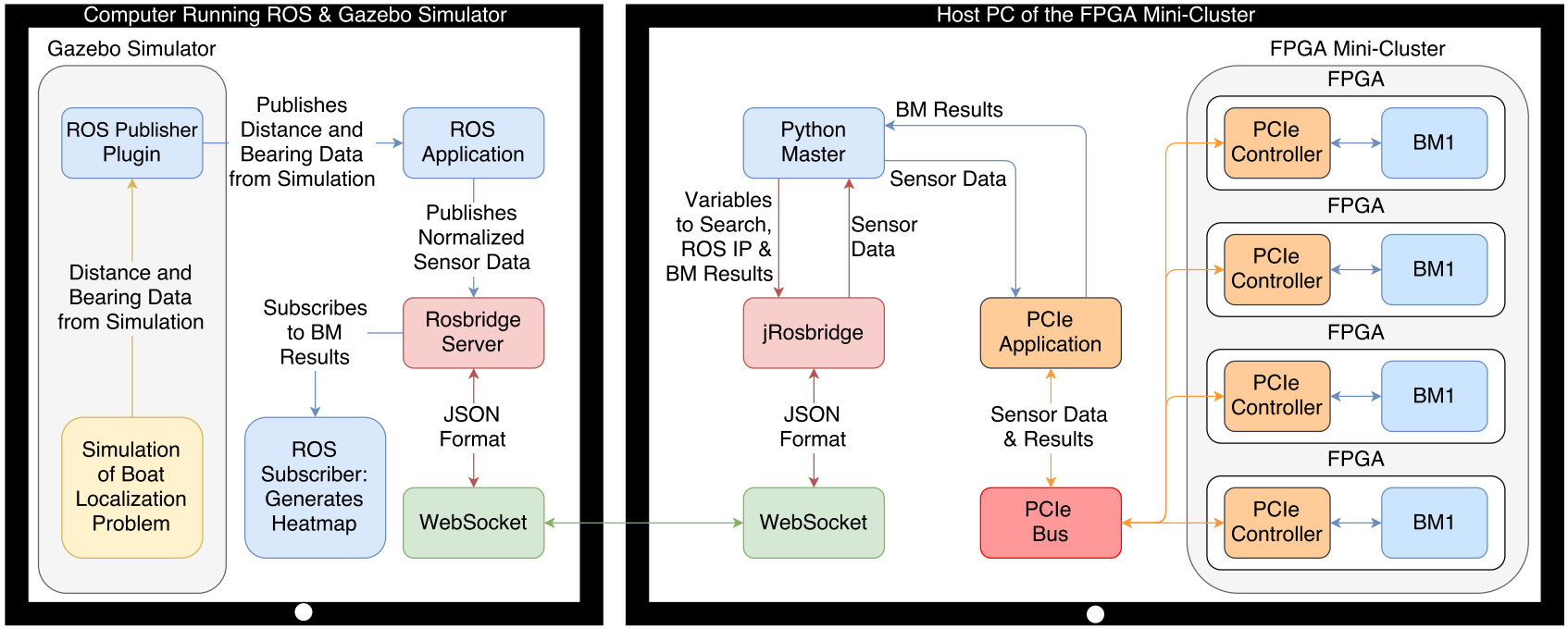


Fig. 4.1 Block diagram of the proposed experiment.

4.1 Gazebo - A ROS Compatible Simulator

Gazebo was developed by the Open-Source Robotics Foundation (OSRF) and provides the user with the ability to accurately simulate robots in complex environments. It also supports the use of plug-ins to implement new features, such as subscribing and publishing to ROS topics. This feature makes it suitable to simulate the operation of a robot that publishes its sensor data with a ROS node, and was therefore chosen as the tool to implement this case study.

The Operating System in the host computer of the FPGA mini-cluster does not support Gazebo or ROS, therefore another computer was used to install this software and implement the simulation. Although the correct functioning of this system could, in theory, be completely demonstrated in the same computer, this case study accurately portrays a typical data exchange between a ROS-compatible robot and the FPGA mini-cluster.

In Figure 4.2 is presented a diagram of the implementation of the Gazebo simulator for this case study, where a robot model was used as a surrogate for the boat. The world model contains the information of the simulation map and the declaration of each item in the world, including their starting positions. The robot model is where the plug-in is declared and associated with the robot. With this information the physics engine of Gazebo is able to send the information associated with the robot to a ROS plug-in that publishes that data to a ROS topic.

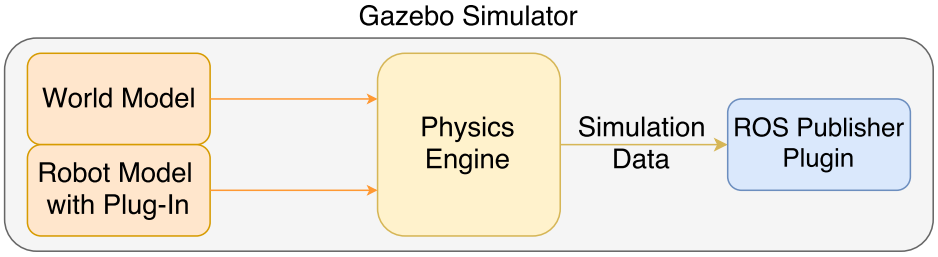


Fig. 4.2 Implementation of the Gazebo simulator.

In Figure 4.3 an image of the resulting map is displayed. The starting position of the robot is in the middle of the map. The beacons are placed, in order, in the following positions: 1) lower left corner; 2) middle point of the left line; 3) middle point of the bottom line.

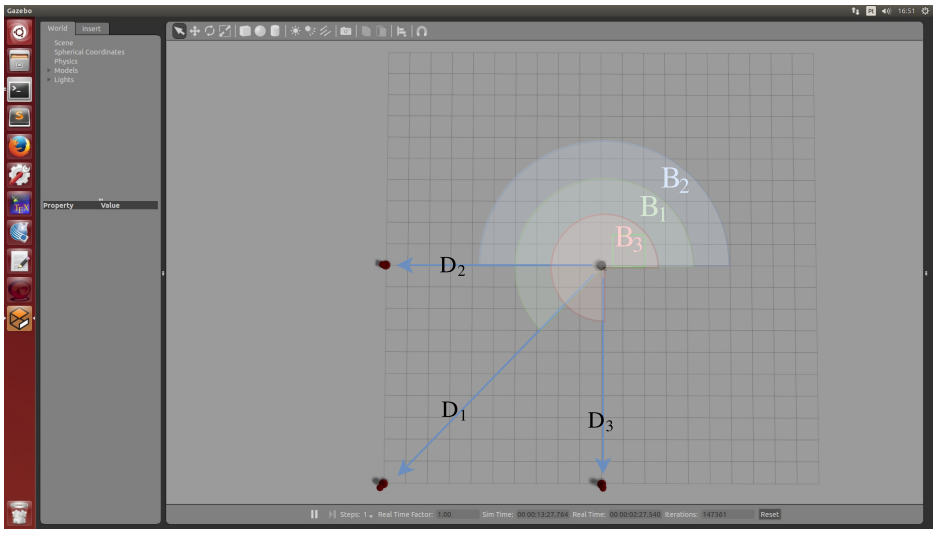


Fig. 4.3 Simulation in Gazebo. The overlay with the distances and bearings is not present in the simulator, it is shown here to better establish a connection to Figure 2.3.

4.2 Implementing the Probabilistic Model - Boat Localization Problem

The objective was to implement the boat localization problem in a thirty two by thirty two grid, however, this would lead to a hardware implementation similar to the one in Figure 2.4 with $32 \times 32 = 1024$ lines. After compiling the project, it became clear that the number of logic elements in a Proce V D8 board was not sufficient to implement a problem of this magnitude. As such, the toolchain was used to partition the search space of the problem into four pieces. Each of the FPGAs in the mini-cluster was then programmed with one of the pieces.

By using the toolchain to split the problem, the Python Master script was also generated with the ability to communicate with the four boards of the system. The only differences from Figure 3.4 are that the script now resets and loads the sensor data to four BMs and it waits until all of the BMs have finished executing before restarting the cycle.

4.3 Experimental Results - Finding the Position of the Boat

As mentioned before in this Chapter, a ROS application that handles the results of the BM1s was created in the computer that is running Gazebo and ROS. This application subscribes to the results of the FPGA mini-cluster and generates a heat map with the possible location of

the boat in the search space of the problem: a 32×32 grid. A user of this system would have to design a similar application to handle the results of the FPGA mini-cluster for his specific application.

Four instances of the position of the boat in the heat map, alongside the real position of the "boat" in Gazebo, are displayed in Figure 4.4.[1]

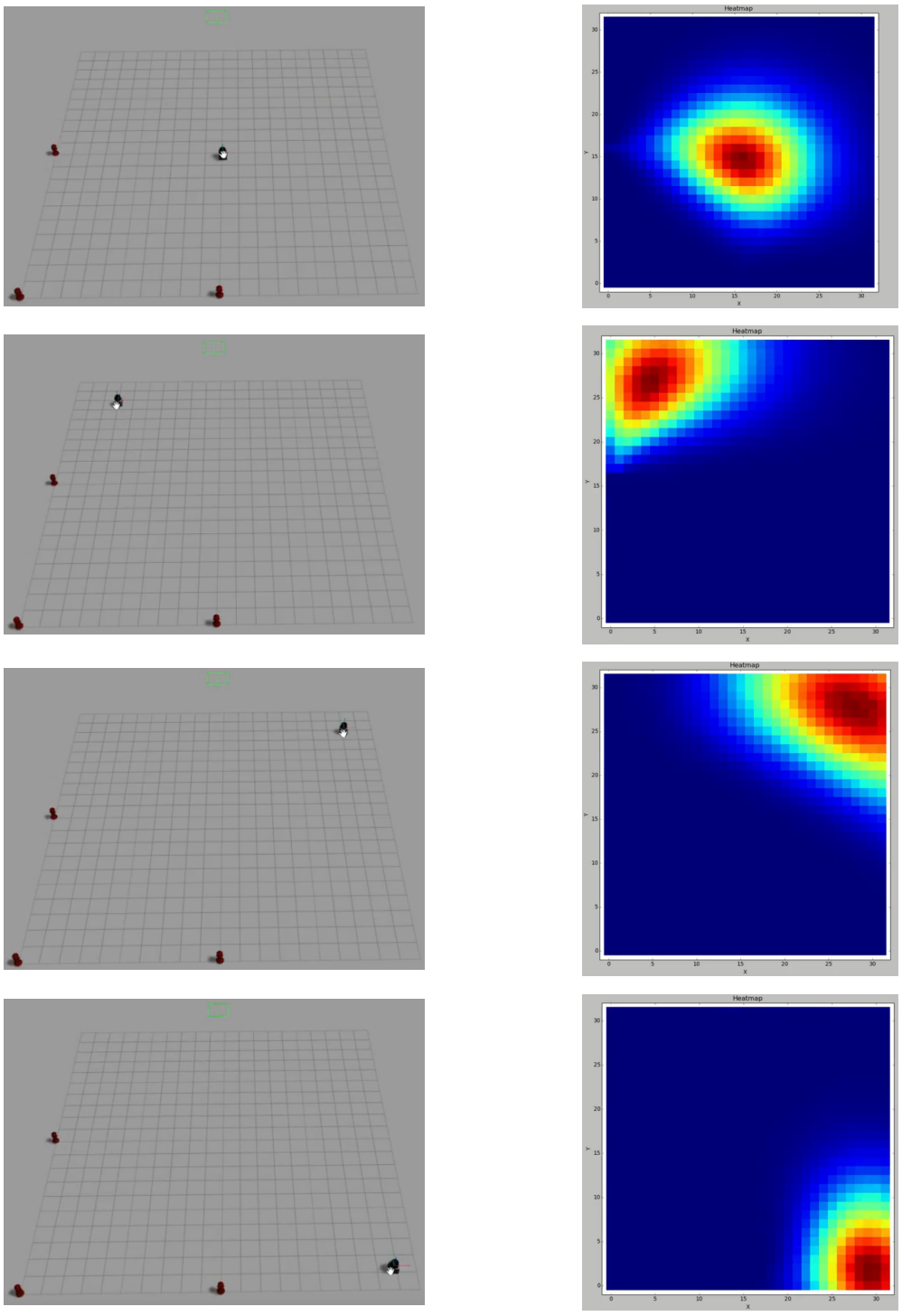
The results show that a robotic perception inference problem was successfully computed in the FPGA mini-cluster, with a seamless connection to the simulator under the ROS framework. The inference problem specified in ProBT was correctly computed, given the sensor data, and the boat was correctly localized in the grid.

This opens the door to a range of applications in robotics. Given an inference problem we only need the corresponding ProBT specification, the required ROS topics for the sensor inputs and subscribe to the ROS topic *'/BMresults'* to obtain the output distribution of the problem. In Section 4.4, we show that other characteristics of the problem need to be taken into consideration, besides the inference *per se*.

Some inference problems may not be well suited for the BM1. If the cardinality of the matrix were to become very high, this machine would start to present scalability issues. In these cases the BM2, being a sample-based approach, may provide a more adequate solution. The work developed in this thesis also allows for the use of the BM2, with some minor adjustments in the boundary I/O of the Bayesian Machine.

During this experiment, the speed of communications and computations were measured in several steps of the system to better assess the presence of any bottlenecks. The experiment was repeated 25 times and the average results obtained can be observed in Figure 4.5.

[1] Two videos of the simulation can be seen in: https://youtu.be/Bn_hL0gBCK8 and https://youtu.be/ESrzyTnmLiI. In the first video you can see the simulation running in Gazebo where the user manipulated the position of the robot to generate different results. The new position of the robot is only updated by the simulator when the user drops the robot in the new position. In the second video you can see a graphical representation of the obtained results responding to the movements performed by the user in the simulator.



(a) (b)

Fig. 4.4 Experimental results of the boat localization problem. The first column (a) displays the position of the robot in the simulator. The second column (b) presents the obtained results for that same position.

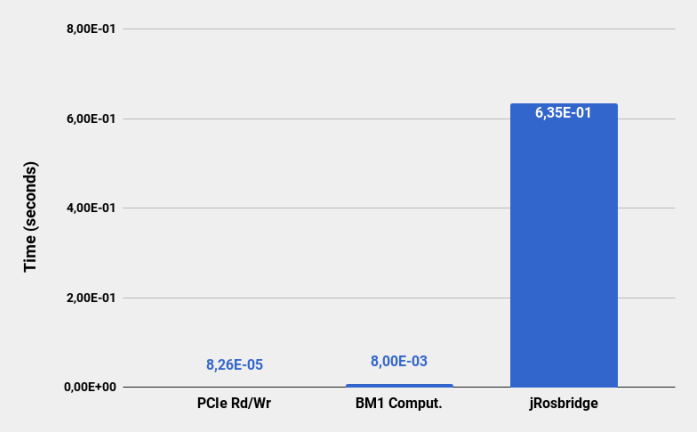


Fig. 4.5 Average execution times, in seconds, of: jRosbridge subscriptions/publications, total PCIe reads and writes, and BM1 computations for the boat localization problem, for a $32x32$ grid.

As can be seen from Figure 4.5 the slowest component of the system, by far, is the jRosbridge. It takes up about 98.74% of the execution time of the entire system, whereas the computations of the FPGA mini-cluster take up around 1.24%. The impact of the total amount of PCIe read and write operations between the host computer and the mini-cluster for one cycle of computations of the BM1(s) is barely noticeable, and is only responsible for around 0.01% of the execution time of the entire system.

The execution time of the PCIe read and write operations is directly dependent on the size of the data transfers. The BM1 computation time, on the other hand, is related to the amount of columns in the architecture for the implemented probabilistic model. This means that the more sensor inputs there are in the model, the longer it will take the BM1 to obtain its results.

Further tests need to be performed to evaluate the behaviour of the jRosbridge's execution time for different problems. While the Rosbridge protocol provides a convenient, generic connection to ROS-compliant components, it was not specified with speed in mind. However, the key characteristic we were aiming for with this component was ease of integration with any ROS robotic system.

4.4 Exploratory Research - Bayesian Computing of Binocular Disparity

Another probabilistic model was also considered for this case study, the model for Bayesian computation of binocular disparity presented in [9].

This model would use a BM1 to compute the most likely disparity value between two pixels from two different images. Since the proposed architecture was relatively small, we could adjust the spectrum of disparity values to be computed, within an acceptable range, in order to be able to program one FPGA with many BM1s. By doing so, an FPGA would be capable of computing, in parallel, the most probable disparity value between a number of pixels from two images. This number would be equal to the number of BM1s with which the FPGA was programmed.

Despite this, a big shortcoming of this model is its necessity to implement, in the host computer, three filters for each of the two images in an image pair. The computer would then also have to use a function to compute the matching cost between each pair of filters. Only then can the data can be sent to the FPGAs.

Although the filters were simple and fast to implement, the computations of the matching costs, in the host computer, for each pixel of the images would take a long time for a video application, approximately 0.6 seconds for each pair of images. This meant that there was a bigger computational burden on the host computer than on the FPGA mini-cluster. Since the gains obtained from using this system would be limited, we found that this would not be the best model to display the benefits of using the framework presented in this dissertation.

This, along with the necessity to alter the architecture of the BM1, caused us to abandon this model as a potential case study. The required alterations to the architecture to implement this model were:

- Implement the ability to program many BM1s in one FPGA according to their size;
- Allow for different sensor data information to be sent to every cell of the matrix, instead of the current architecture where the sensor data is shared through an entire column.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

This dissertation presented a system that can successfully interface an FPGA mini-cluster with ROS-compatible robots, in order to implement probabilistic models for robotic applications. The details of the implementation of this system were presented in Chapter 3.

Machines that are capable of efficiently performing Bayesian inference were implemented in an FPGA mini-cluster. Their architecture was altered to allow for the development of a high-bandwidth PCIe connection. This connection replaces the previous USB/JTAG connection to the FPGAs, thus allowing for fast data transfers between the mini-cluster and the host computer and eliminating one of the bottlenecks of the system.

A connection was established with a Rosbridge server in a different computer by using the developed jRosbridge. The jRosbridge files granted a system that was unable to have ROS installed, the ability to interact with ROS-compatible robots/sensors/actuators/simulators through WebSockets.

A Python script manages the data transfers between the jRosbridge and the PCIe connection, thus ensuring that the BM(s) in the FPGA mini-cluster are able to receive the needed sensor data and output its/their results.

The toolchain developed within the BAMBI project that allowed for the spooling of Bayesian Machines onto FPGAs, based on a given probabilistic model, was altered to account for the new hardware changes. This toolchain is now also able to customize the Python Master script so that it is automatically adjusted to any newly implemented probabilistic model.

In Chapter 4 a case study was presented using a probabilistic model for the boat localization problem. This case study was a success, displaying the previously mentioned properties

of the system. However one limitation was observed, namely the speed of the jRosbridge. This component is now the bottleneck of the system, but in the future, other, faster protocols can be used to replace it.

5.2 Future work

The framework developed during this thesis allows for many future developments, namely:

- Creation of a generic ROS subscriber that subscribes to the *'/BMresults'* ROS topic and has a function for the user to handle the incoming data. This will avoid the necessity for the user to constantly create a new ROS subscriber to obtain the results of the FPGA mini-cluster. The user will still have to write the necessary code in the given function to perform the desired operations with the results;
- Development of a connection to ROS topics using a different protocol in order to accelerate what is currently the slowest component of the system;
- Improvements to the PCIe connection, such as using Gidel's MultiFIFO hardware function, are possible. This would allow for the transfer of data in bursts and a more efficient use of the PCIe connection;
- Implementation of the Bonjour protocol with the jRosbridge to automatically resolve host-names. The user can then simply select his robot/sensor/actuator/simulator from a name list instead of having to insert an Internet Protocol (IP) address, thus providing ease-of-use;
- Implementation of the fibre optic interface that can be used between the FPGAs of the mini-cluster. This would require a new hardware project. The toolchain would also have to be adapted to implement this interface. In the case of the BM1s may not make much of a difference due to the nature of their architecture and the ability to partition a BM1 through any number of FPGAs.
- Design a PCIe interface for the BM2 to allow for probabilistic models to be implemented with both BMs. Research could then be made on the more suitable type of BM for any given robotic application, with experimental results;
- Make the interface more user-friendly by implementing a Graphical User Interface (GUI). Should be relatively easy to adapt inputs of the toolchain and the Python Master script execution to such an interface;

- Collaboration with the National Science Foundation (NSF) Center for High-Performance Reconfigurable Computing (CHREC) at the University of Florida to implement these architectures and interface on the Novo-G supercomputer. The boards used in this project were already selected with this in mind.

References

[1] J. F. Ferreira and J. M. Dias, *Probabilistic approaches to robotic perception*. Springer, 2014.

[2] H. Fernandes, M. A. Aslam, J. Lobo, J. F. Ferreira, and J. Dias, "Bayesian inference implemented on FPGA with stochastic bitstreams for an autonomous robot," in *Field Programmable Logic and Applications (FPL), 2016 26th International Conference on*, pp. 1–4, IEEE, 2016.

[3] J. Alves, J. Ferreira, J. Lobo, and J. Dias, "Brief survey on computational solutions for Bayesian inference," in *Workshop on Unconventional computing for Bayesian inference*, 2015.

[4] BAMBI, "D3.10: Emulation of a probabilistic computer on current reconfigurable logic," tech. rep., ISR, CNRS, ProBayes, 2017.

[5] A. Coninx, P. Bessière, E. Mazer, J. Droulez, R. Laurent, M. A. Aslam, and J. Lobo, "Bayesian sensor fusion with fast and low power stochastic circuits," in *Rebooting Computing (ICRC), IEEE International Conference on*, pp. 1–8, IEEE, 2016.

[6] T. Ohkawa, K. Yamashina, T. Matsumoto, K. Ootsu, and T. Yokota, "Architecture exploration of intelligent robot system using ROS-compliant FPGA component," in *Rapid System Prototyping (RSP), 2016 International Symposium on*, pp. 1–7, IEEE, 2016.

[7] "Interconnecting a Linux Host with a FPGA Board through PCI-Express," Master's thesis, National and Kapodistrian University of Athens, 2016.

[8] J. M. Shalf and R. Leland, "Computing beyond Moore's Law," *Computer*, vol. 48, pp. 14–23, Dec 2015.

[9] A. Coninx, P. Bessière, and J. Droulez, "Quick and energy-efficient Bayesian computing of binocular disparity using stochastic digital signals," *International Journal of Approximate Reasoning*, vol. 83, pp. 400–412, 2017.

[10] M. Faix, R. Laurent, P. Bessiere, E. Mazer, and J. Droulez, "Design of stochastic machines dedicated to approximate bayesian inferences," *IEEE Transactions on Emerging Topics in Computing*, vol. PP, no. 99, pp. 1–1, 2017.

[11] "ROS/introduction." http://wiki.ros.org/ROS/Introduction. Accessed: 2017-08-20.

[12] "Development of a Remote 3D Visualisation, Control and Simulation Framework for a Robotic Head," Master's thesis, University of Coimbra, 2016.

[13] “Rosbridge Suite.” http://wiki.ros.org/rosbridge_suite. Accessed: 2017-08-20.

[14] “How do I change the clock frequency of the USB-Blaster II download cable?.” https://www.altera.com/support/support-resources/knowledge-base/solutions/rd06242013_922.html. Accessed: 2017-08-20.

[15] “PCI Express’ speed to usher in a new server era.” http://searchdatacenter.techtarget.com/opinion/PCI-Express-speed-to-usher-in-a-new-server-era. Accessed: 2017-08-28.

[16] “PCI Express – An Overview of the PCI Express Standard.” http://www.ni.com/white-paper/3767/en/. Accessed: 2017-08-28.

[17] “Peripheral Component Interconnect Express (PCIe, PCI-E).” http://searchdatacenter.techtarget.com/definition/PCI-Express. Accessed: 2017-08-28.

[18] “PCI Express* Hard IP.” https://www.altera.com/solutions/technology/transceiver/protocols/pro-hard-ip.html. Accessed: 2017-08-28.