EM 083

Reconfigurable computing – The BondMachine Project

Team members：Mirko Mariotti, Loriano Storchi, Daniele Spiga / University of Perugia, Dept. Of Physics and Geology and INFN Perugia

1. High-level Project Description

1.1 Introduction

Nowadays most computational systems have been developed using a general purpose architectural approach, and all computational systems have achieved surely admirable targets, nevertheless we believe the mentioned approach is not suitable for at least two emerging scenarios. The first is the Big Data [1] scenario, which requires enormous amount of computing power. The second one is the scenario of systems consisting of many agents to be efficiently coordinated.

The key objectives for the upcoming future is then to enable everyone to build on demand a specialized HW tailored to solve a specific problem. To the best of our knowledge we faced this challenge in the easiest and more immediate way, creating a compiler, that is the heart of our system, which in addition to generating the SW, generates also the hardware, in the form of RLT code specific to the device that will run the program. In other words, starting from a source code both the SW and the HW are generated in a single step.

The described idea has clearly a huge impact on an efficient and time-critical processing of large amounts of data, such as for self-learning processes, machine learning, tensor processing and more. Indeed, the latest advances in these fields demonstrate that the concept of processor, intended as a static and general-purpose object, is not adequate to solve the computational problems required by all the new and emerging paradigms. Even GPU like approaches, although they make the situation better in linear algebra intensive problems, are still not sufficient. This is mainly because the development of optimized SW is more and more dependent on the knowledge of the underlying HW, which is often heterogeneous and with specialized components to perform different tasks.

1.2 Key features of the project

Taking advantage of the actual features of the today's reprogrammable HW technologies, our project plans to completely review the HW/SW stack and therefore to effectively rationalize the number of layers between the high-level description of a problem and the underlying hardware.

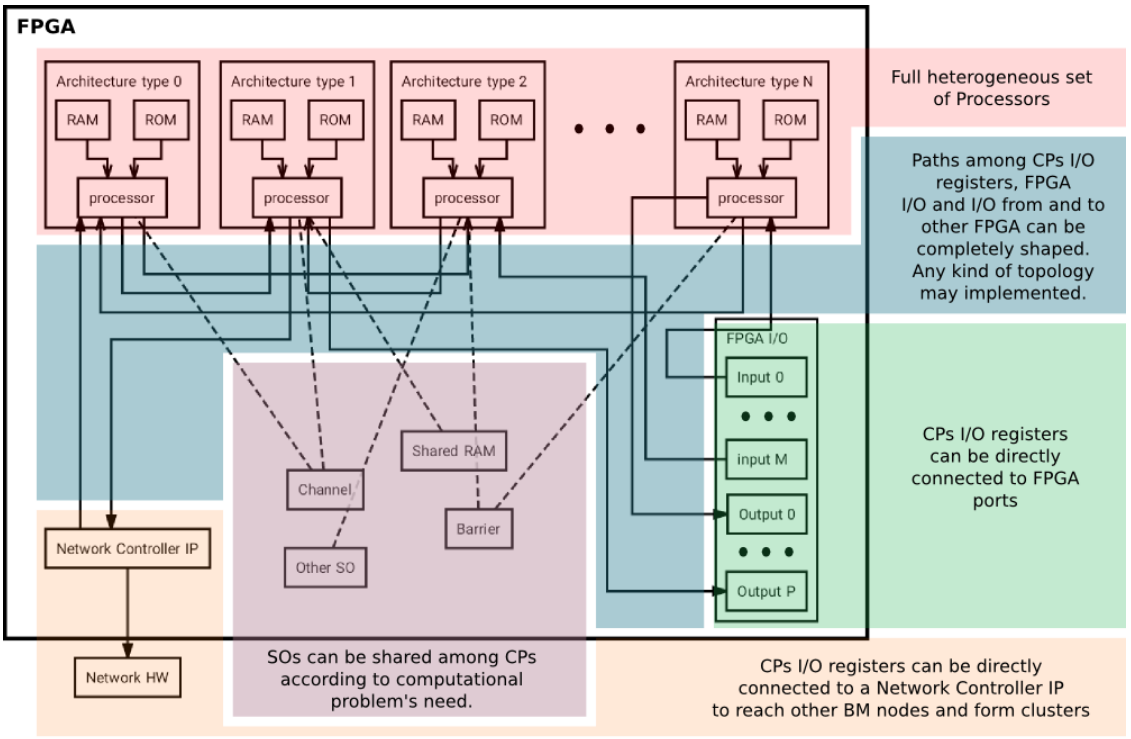
The main goal of the project BondMachine (BM) project [2] is to create a single layer of interconnected heterogeneous processors using one or many FPGAs [3]. Afterwards the project aims to extend this abstraction via the network through an ad-hoc protocols, and so to use this layer as an underlying support to build any kind of computational systems. The name we gave to this layer is BondMachine.

1.3 Components

The present project is aimed to build a full reconfigurable computing ecosystem made of several components. In the following we will report and describe all these components:

- The Architecture Description (AD), that is a full specification of the computing and non-computing shared objects which constitute the BondMachine (BM).
- The BM handling tools: a set of standard computer programs to manage any aspect of the BMs starting from their creation up to the RTL code production.
- The BM front end tools: a set of programs to use the BM, that is a set of API, a compiler and other fundamental tools such as a web interface, translators from Boolean expressions, mathematical expressions and so on.

2. Block Diagram



The image shows a template of the block diagram for a BondMachine design on an FPGA. The BM architecture changes in order to satisfy the specific computational problem, so there is not a single FPGA design. Instead the tools that generate the architectures can be tought as firmware generators.

The BondMachine (BM) architecture consists of the full specification of the interconnections among Connecting Processors (CPs) and Shared Objects (SOs), being non-computing units, that can be shared among the various CPs. The main features of the BM architecture are the possibility to fully configure: i) the number and type of the processor cores, ii) the number of inputs and outputs, iii) the topology of the interconnections between processors and iv) the number and type of the Shared Objects used by each processor.

3. Intel FPGA Virtues in Your Project

3.1 Boost performance

Given that many processors (CPs) can be interconnected in custom topologies, any computational problems can be better optimized respect to what one can obtain using many CPUs or GPUs. CPU is manly oriented to be used as a general-purpose system and a GPU is expected to perform well mainly on linear algebra operations. However, in both cases the development of optimized code is strongly linked to the perfect knowledge of the hardware on which it will run and in particular the interconnection between the processors and the elements they share, such as the memories. In our case instead the hardware itself will adapt to the specific problem we want to solve.

A BM may be used as an hardware accelerator so that one can mix all together CPU and BM threads, that is one can off-load a task or a function using the BM (i.e. the FPGA)

3.2 Adapt to Changes

Because the user can deploy an entire HW/SW cluster starting from a code written in a High-Level language, this represent per se a guarantee that the system will better adapt to evolving standards and algorithmic methods.

Scalability and extensibility of a HW / SW system built as described in the present project is greatly improved. Indeed, a system with interacting agents, of whatever type they are, would be the expression of a single and coherent program written in high level language.

3.2 Expand I/O

A BondMachine can be used as standalone device for specific applications. Its I/O registers can be mapped to the DE10-Nano GPIO and it can be used to build peripheral tailored to specific applications, with the advantage of the FPGA in terms of computing power and with the user friendliness of and high-level programming language like Go [4].

A BondMachine can be used as a hardware accelerator directly linked with the HPS. Its I/O registers can be mapped to the DE10-Nano GPIO. The result will be a hybrid system where BM acquires and process external data signals and share information (results, control commands, etc.) with the HPS processor. This could enable the extension of HPS processor capabilities to handle not native interfaces.

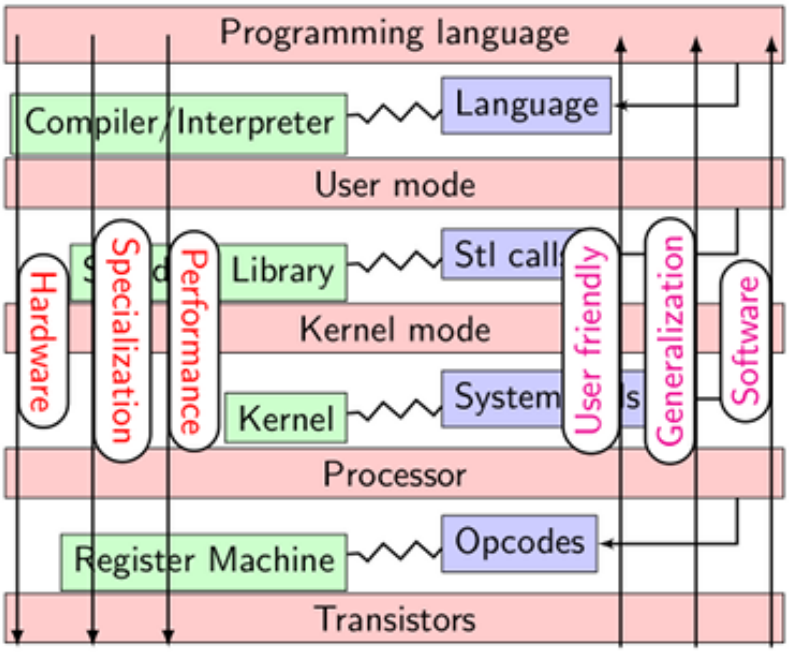
4. Design introduction

4.1 Design purpose and application scope

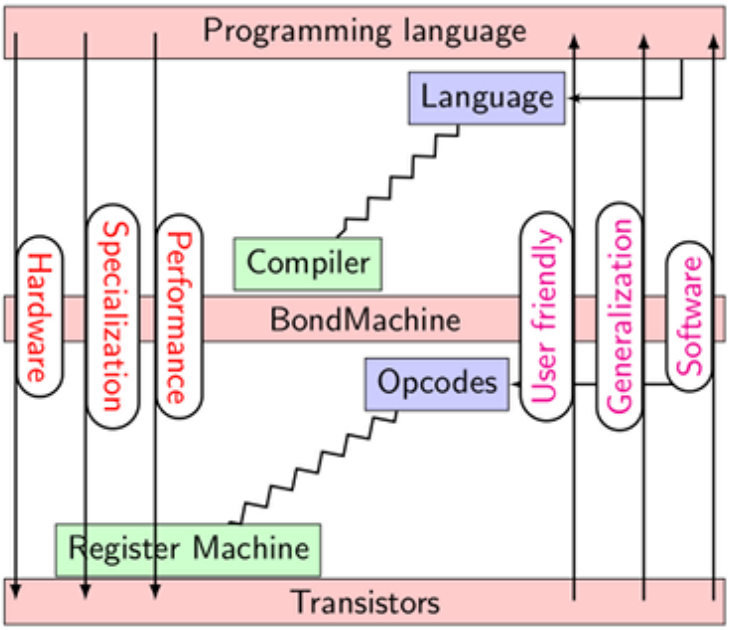
The main purpose of the design is to create a processors abstraction layer on top of some FPGA devices having the following characteristics:

- Each FPGA is a multi-core device with computing units simple as much as possible. they independently process their own code and they can communicate in a simple way through dedicate IO registers. They may share also non-computing elements.
- These units are eventually of different type, and the interconnection patterns among them can be changed following the needs of the computational problem. The device is thus heterogeneous.
- This layer of heterogeneous interconnected processors has to be extendable to a multi FPGA scenario.

We think that a similar layer could be the base for a wide range of applications. We also believe that making this kind of flexible abstraction and using a modern programming language as a source of both the architecture generation and its programs, surely will allow the reduction of the Hardware/Software gap. The Go programming language has been chosen for this purpose. Moreover, having a flexible architecture, we may bring the optimization process typical of the software directly to the hardware, simplifying and reducing the layers between the high-level code and the transistor, as shown in the following images:



A simplified example of layers in a standard system.



An example of layers in a BondMachine system

4.2 Design Components

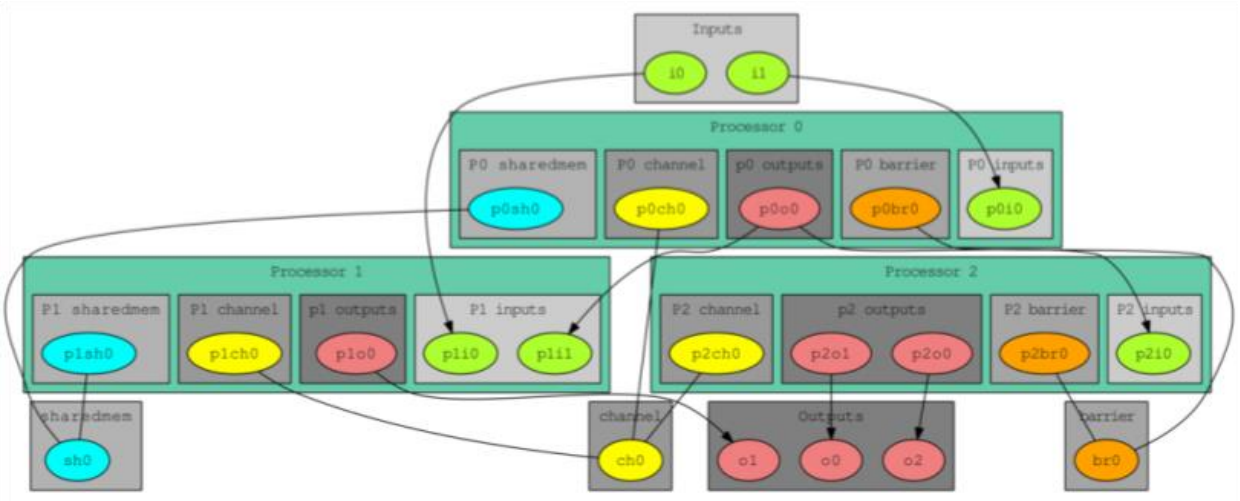
As cited before the BondMachine is made of several modules, in the following we will detail about some of the main basic components that can be defined and used within the BM.

4.2.1 Connecting Processors

The Connecting Processor is the computing core of the BondMachine. One of the main capability of a Connecting Processor, as the name suggest, is to be configured in such a way to be connected to other processors and to any Shared Objects. CPs are as simple as possible, specialized and optimized to perform a single task. In fact any CP can be created with a different number of registers, different number of I/O registers and different instruction set (i.e. opcodes) with respect to the other ones.

4.2.2 Shared Objects

These are non-computing objects shared between all or some of the CPs. Several kind of objects can be implemented to increase the processing capability and functionality of the BMs improving the high-speed synchronization and communication between tasks running on separate CPs. Examples of SOs are: Channels, Shared Memories and Barriers.



In the Figure we are reporting a scheme of a complete example of the BondMachine architecture. Specifically it consists of two inputs and tree outputs interconnected between the input/output registers of the processors. One can easily see as the shared objects, such as memory, channels and barriers, are connected among the processors.

4.2.3 Network Component

An interesting feature of the project is that several BondMachines can be connected via a custom protocol, that is a distributed cluster of heterogeneous multicore can be built in such a way.

To do so every BM joining a cluster has a network component within the FPGA that extends the same logic to other FPGAs within a cluster.

4.3 Advantages and outcomes of the project

The described approach has several advantages:

- Firstly, whenever high performances are required, we expect that the reduction of the number of layers will lead to an increase of the overall performances.
- The fact we will use a register machine (processor), and not for example directly the FPGA, means that will be relatively easy to import well known techniques used in computer science.
- Clearly the device resource utilization is a function of the problem one needs to solve

We would like finally to cite some main outcomes of our project:

- The optimization process can be extended up to the hardware. Indeed, if the program does not expect the processor to implement a particular operation, the resulting hardware will not have that instruction.
- Distributed systems can be managed as a multicore system having a distributed abstraction.
- It makes the implementation of hardware accelerators (hybrid computing) accessible (i.e. transparent). This feature makes the "Terasic DE10-Nano" board with its hybrid structure absolutely well suited for this type of development.

5. Functional Description

The described interconnected processors layer is the base of the project. Clearly, we developed all the tools needed to use it efficiently, to modify the layer and to map a computational problem on it.

5.1 Handling tools

The complexity of the BondMachine architecture can be managed using a set of software tools. These tools allow to build a specific architecture as a function of the task one want to perform, to easily modify the architecture, to simulate the behavior and finally to check the functionality with the aim to generate the Register Transfer Level (RTL) code for a programmable device, i.e. a FPGA device.

The full set of tools can be subdivided in two different categories: i) the CP builder: that manages the configuration parameters of the CPs (procbuilder), and ii) a BM builder: that manages the interconnection between the CPs and the SOs (bondmachine). Moreover, all the tools share the capability of using the generated BM architecture, so that the full architecture may be emulated directly on a workstation, using the so-called simbox framework, without the need of a FPGA.

5.2 Networked BondMachines

Several BMs may be connected together to form clusters or to interact with the external world. BMs may communicate using a native protocol called EtherBond. Its purpose is to replicate the electronic behavior of BMs registers and to extend it over the device boundaries. In other words clusters of BMs may be created and their behavior is driven by the same rules of separate BM devices. The main objective is then to handle devices and cluster in the same way. Interestingly the EtherBond protocol has been ported to Linux, so that BMs can now communicate also with a standard PC software.

5.3 Front-end applications

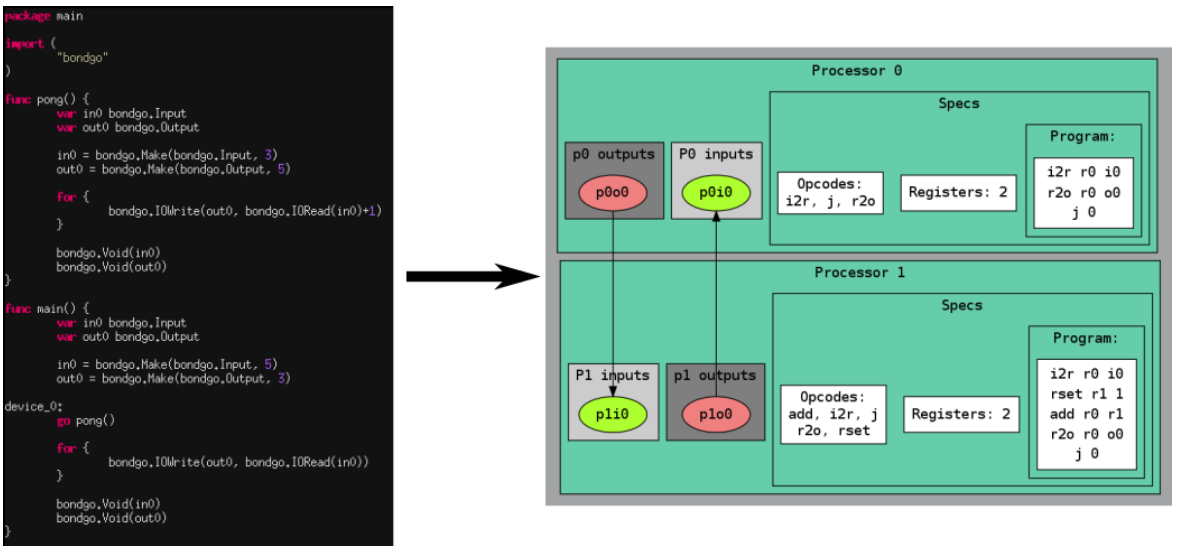
BondMachines can be created in several ways: i) manually with its building tools; ii) with a set of APIs that targets specific problems; ii) lastly with a dedicated compiler that creates the architectures as part of the conversion of a source code to machine code. The latter is the major innovation of the BondMachine Project.

5.3.1 The Bondgo compiler

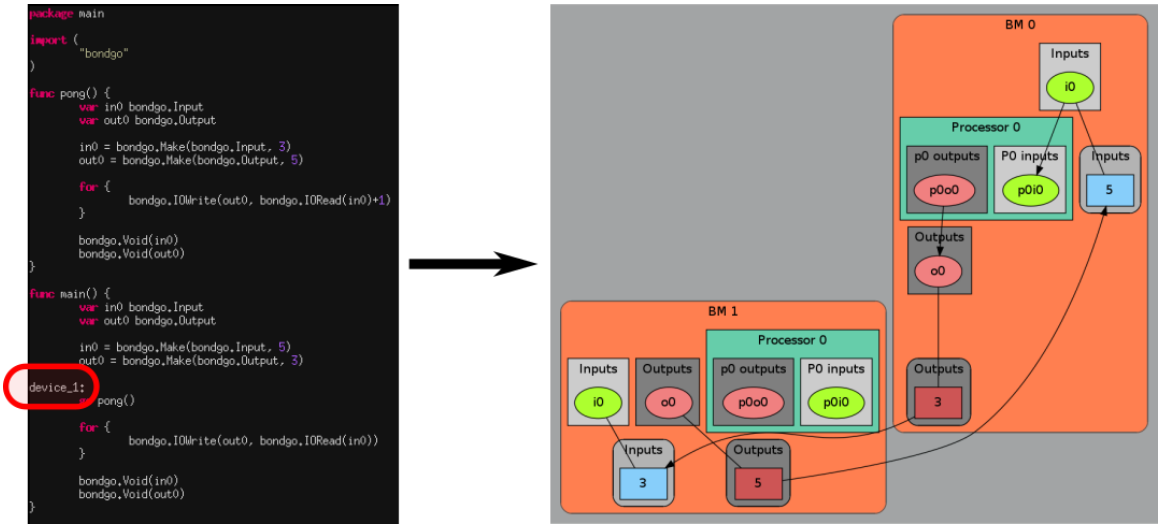
bondgo is the compiler, that starting from an high level language, in this case Go, produces the code needed to generate the assembly code of the architecture. The generated assembly code may be finally assembled with the procbuilder tool, this will generate the binary code for a CP. Unlike other compilers, Bondgo may create the assembly code so that a given processor can directly run it, or even creates a specific CP optimized to run the code. The resulting architecture will be generated with the minimal needed resources and will be highly specialized.

Moreover, Bondgo is able also to handle the concurrency of the source code by creating, if needed, a multicores BM, that is it can create a new CP in the BM every time a Goroutine is encountered (clearly it is optimized to run the code produced).

In the following we report an example of the BondMachine. This is a trivial example, yet it shows well the basic capabilities of the BondMachine architecture and ecosystem. Two Goroutines send an uint8 data value back and forth through I/O registers (created with bondgo.Make). The pong goroutine also increases the value by one before sending it back. Once the code has been compiled with Bondgo the result is a multicore BondMachine as shown in the following figure:



Finally, within Bondgo the developer can choose which routine (and thus which CP) runs on which FPGA, naturally allowing for the possibility of building autonomous clusters of multi-cores. Indeed, starting from the previous example and just changing the device_0 label with device_1, the compiler is instructed to put the two goroutines on different BondMachines. Numerically the result will be the same but now we will use a cluster of two BondMachines connected via the etherbond protocol. Interestingly, after a simple and quick change, a multicore system is transmuted in a distributed system as shown in the following figure:



5.3.2 API

Several libraries have been developed to map specific problems on BondMachines:

- **neuralbond** is an application that models a neural network over BondMachines. The final result of the process in a device composed by many CPs acting as neuron-like computing units.
- **symbond** permits the creation of BMs that computes a particular mathematical expression.
- **boolbond** handles BMs that manage boolean expressions.
- **matrixwork** to perform matrix calculations with the BondMachines.

6. Performance metrics / goals

Firstly, we would like to underline again that the aim of the BM project is to provide for a firmware generator. Thus, a set of tools that provides the user with the capability of modelling the underlying hardware as a function of the problem he needs to solve.

Thus, taking into account the inevitable physical limits related to the actual FPGA dimension, our final goal is to obtain better overall performances respect to the general-purpose CPU and GPU given a specific numerical problem.

Because starting from a high-level description of the problem, for example starting from a Go source code, we are able to automatically generate the Verilog code and then synthesized it. Obviously, we will be able to measure and compare directly the time per operation needed to complete the task by a CPU or a GPU and a BM.

We may start using a simple architecture made of several CPs interconnected as a pipeline, that is the output of CP(i) is interconnected to the input of CP(i+1). While we expect the the time per operation measured in the CPU will increase almost linearly as the number of CPs grows. On the contrary in the FPGA, due to the intrinsic parallelism, we expect the time per operation to be constant. In the latter case we will reach a clear upper limit due to the filling of the available logic resources.

7. Design Architecture

As partially already stated in the previous sections the BondMachine (BM) architecture consists mainly of three components that, interestingly, all of them can be implemented on a single FPGA:

- The Connecting Processor (CP) is the computational core of the BM and it implements dedicated opcodes that build the functionality of the entire system. A single FPGA may contain several CPs. This is a guarantee of a high device utilization.
- The Shared Objects (SO) consist of any kind of non-computational component shared among processors, or only some of them. Several shared objects can be implemented at the present stage of the project: Channels, Shared Memory and the Barrier. Obviously, a single FPGA may contain several SO.
- If the FPGA is part of a cluster there will be a connection component (CC) with the specific purpose to distribute the layer on another FPGA. Evidently only a single CC is present on each FPGA.

Clearly all the characteristics of a BM can be configured in order to build the best architecture for a specific problem then it means that the BM is not a single machine, but

instead a class of machines. Inevitably this brings to the conclusion that there is not single FPGA design.

7.1 Connecting Processor (CP)

Is the computing core of the BondMachine. The name Connecting Processor well describes the main capability of the processor core that is to be configured in such a way to be connected to other processors and to a set of Shared Objects. CPs are as simple as possible, specialized and optimized to perform a single task. In fact, the CPs inside the BM architecture may have different number of registers, number of input/output registers and different instruction sets (i.e. opcodes) with respect to the other ones. The register size is the only parameter that needs to be equally sized among all components of a BM.

Registers within a CP are all general purpose and are named r0 ... rR, where R is not a constant, but it can change between the various CPs. In addition, a CP has two types of specialized registers: the input registers, named i0 ... iN, they can only be read by the CP, while the output registers, o0 ... oM , can only be written. The registers are used to connect different CPs and an input register of a CP can only be connected to the output register of another CP. Moreover, an input register can also be used as BM input, and similarly an output register may be used as the BM output.

Every CP has two kinds of memories. The ROM, that contains the instruction program for the processor, and the RAM, that is the local storage for the processor. Clearly the RAM will contain the instruction program and application data.

The instructions set are another configurable aspect of a CP. As already mentioned for every processor the implemented opcodes are chosen among a rich set of possibilities, not all the possible opcodes are implemented on every CP. The implemented opcodes consist of the classical ones such as: set register rset, clear register clr, conditional jump j, je, jz, increment/decrement register add, dec, inc, copy register cpy, read input register i2r, write output register r2o. In addition, we added some dedicated opcodes for controlling the Shared Objects such as operation related to the Internal and Shared Memory (r2m, m2r, s2r, r2s) as well as to Channel and Barrier management (wrd, wwr, chc, chw).

7.2 Shared Object (SO)

Several kind of objects can be implemented to increase the processing capability and the functionality of the BM improving the high-speed synchronization and communication between tasks running on separate CPs. In this project three kinds of objects have been currently implemented and described: Channels, Shared Memories and Barriers. Clearly other kind of components can be easily added in the future.

7.2.1 Channels

Channels are hardware message queues acting as conduits between two CPs. Following the concurrency model in Communicating Sequential Process (CSP) each processor has a dedicated interface to the Channel and can send or receive data to and from.

7.2.2 Shared Memory

Shared Memory consists of RAM shared between one or more CPs. In a Shared Memory

system, every processor has direct access with its dedicated interface (data input/output, address, write and enable signals), that is it can directly load or store data to any memory address. The system only synchronizes the read/write processors' requests without controlling the specified address and the conflict between the processors. The depth of the RAM is calculated considering the implemented task of each CP and the value of allocated memory address. Clearly the depth of each Shared Memory object can be individually configured.

7.2.3 Barrier

A Barrier is an object used to synchronize Concurrent Processors on a shared architecture. Upon reaching a barrier, a processor must wait until all the other processors reach the barrier too. The processors are stalled waiting for the others and during this time they are in a idle state. The implementation of the barrier mechanism uses a dedicated opcode hit and a timeout counter to define a limit for the task completion.

7.3 Connection Component (CC)

The Connection Component is the part of the design that enable each specific FPGA to join a cluster. It is a hardware dependent component ad so far it has been tested with the ESP8266 for the wireless connection, and with the ENC28J60 for the wired one. This component translates the connections among CPs into network messages, thus It allows the layer to be spawn across different FPGAs.

8. Conclusion

The BondMachine is a new kind of computing device made possible in practice only by the emerging of re-programmable hardware technologies such as FPGA. The main goal of our project is the construction of a computer architecture where the hardware is shaped by the problem one aims to solve. Clearly this approach brings to an increased computing power and flexibility yet keeping a standard way of programming it. Following these reasons, the compiler is not anymore a software that translates a high level source code to a general purpose machine binary code, it becomes a software that creates the architecture, the binary code and the RTL code to run on FPGA devices.

The BondMachine may be used in several range of application:

- Go in hardware. The establishment of the relationship between Goroutines and CPs, Go channels as well other Go's characteristics brought in hardware. This creates a Go language directly mapped into devices without the need of an operating system or a runtime environment.
- Internet of Things (IoT) [5] and Cyber-Physical Systems (CPS). A network of standalone devices can be created, and all interconnected with a common communication layer. This clearly fulfill almost all the IoT and CPSs systems requirements.
- High performance. FPGA devices allow to achieve very high computational performances. With the proposed solution: i) algorithms have no more to be adapted to the architecture in use, but the architecture is instead shaped by the problem and then optimized, ii) all the operations are described via a high level Go source code that is easy to write and debug.

8.1 Future work

We aim to apply the BM to tackle some real physical problems as:

- Real time pulse shape analysis in neutron detectors: we are currently planning a BM based system may be applied to the real-time detection of some parameters of the neutrons produced in the new sources (e.g. ESS).
- A space experiments test beam: We are currently working on a system to be implemented on a FPGA, through small interconnected processors, that can be used on typical operations of a test beam related to space physics experiments. Nowadays this system is instead based on dedicated hardware,

On addition in the following we are quickly mention some future works we are currently planning:

- Quite clearly, the modularity of the proposed architecture naturally suggests that some of its elements may be created as standards pieces and subsequently used as building block for more complicated machines. This may for example be used for didactic purposes, where students can learn architectures easily assembling the various elements.
- BondMachine inputs and outputs may be connected directly to external hardware, thus a possible extension is to interface them with a standard Linux system running in chips. This will allow the use of BondMachines with any standard Linux applications.
- The structure of the BondMachine may be used to explore computer systems radically different from the usual architecture as for example "The Connection Machine" (CM) [6]. Following the CM example, it would be interesting build a Lisp like language as the one built for the CM for the BM.

9. References

[1] Xiaolong Jin | Benjamin W. Wah, Xueqi Cheng, Yuanzhuo Wang, Significance and Challenges of Big Data Research, Volume 2, Issue 2, June 2015, Pages 59-64

[2] Project website: http://bondmachine.fisica.unipg.it
Github: https://github.com/mmirko/bondmachine
Accessed: July 2018

[3] Farooq U., Marrakchi Z., Mehrez H. (2012) FPGA Architectures: An Overview. In: Tree-based Heterogeneous FPGA Architectures. Springer, New York, NY

[4] The Go Programming Language, https://golang.org/. Accessed: July 2018

[5] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. 2013. Internet of Things (IoT): A vision, architectural elements, and future directions. Future Gener. Comput. Syst. 29, 7 (September 2013), 1645-1660.

[6] Hillis, W Daniel, “The connection machine”, MIT press 1989