



Machine Learning on FPGA: The BondMachine Project

Mirko Mariotti ^{1,2} Giulio Bianchini ¹ Loriano Storchi ^{3,2} Giacomo Surace ²
Daniele Spiga ²

¹Dipartimento di Fisica e Geologia, Universitá degli Studi di Perugia

²INFN sezione di Perugia

³Dipartimento di Farmacia, Universitá degli Studi G. D'Annunzio

Outline

1 Introduction

- Challenges
- FPGA
- HDL workflow
- HLS Workflow
- Concepts
- Cloud

2 The BondMachine project

- Architectures handling
- Architectures molding
- Bondgo
- Basm
- API

3 Misc

- Project timeline

4 Machine Learning

- Train
- BondMachine creation
- Simulation
- Accelerator
- Benchmark

5 Optimizations

6 Conclusions and Future directions

- Conclusions
- Ongoing
- Future

Introduction

1 Introduction

Challenges

FPGA

HDL workflow

HLS Workflow

Concepts

Cloud

2 The BondMachine project

Architectures handling

Architectures molding

Bondgo

Basm

API

3 Misc

Project timeline

4 Machine Learning

Train

BondMachine creation

Simulation

Accelerator

Benchmark

5 Optimizations

6 Conclusions and Future directions

Conclusions

Ongoing

Future

Current challenges in computing

■ Von Neumann Bottleneck:

New computational problems show that current architectural models has to be improved or changed to address future payloads.

■ Energy Efficient computation:

Not wasting "resources" (silicon, time, energy, instructions).
Using the right resource for the specific case

■ Edge/Fog/Cloud Computing:

Making the computation where it make sense
Avoiding the transfer of unnecessary data
Creating consistent interfaces for distributed systems

Current challenges in computing

- Von Neumann Bottleneck:

New computational problems show that current architectural models has to be improved or changed to address future payloads.

- Energy Efficient computation:

Not wasting "resources" (silicon, time, energy, instructions).

Using the right resource for the specific case

- Edge/Fog/Cloud Computing:

Making the computation where it make sense

Avoiding the transfer of unnecessary data

Creating consistent interfaces for distributed systems

Current challenges in computing

- Von Neumann Bottleneck:

New computational problems show that current architectural models has to be improved or changed to address future payloads.

- Energy Efficient computation:

Not wasting "resources" (silicon, time, energy, instructions).

Using the right resource for the specific case

- Edge/Fog/Cloud Computing:

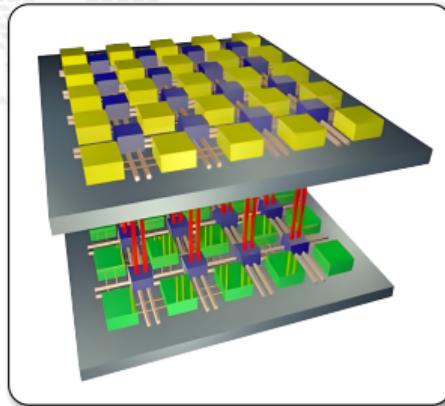
Making the computation where it make sense

Avoiding the transfer of unnecessary data

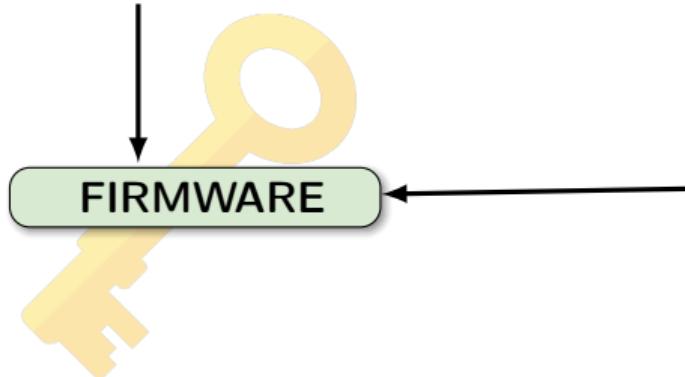
Creating consistent interfaces for distributed systems

FPGA

A field programmable gate array (FPGA) is an integrated circuit whose logic is re-programmable.



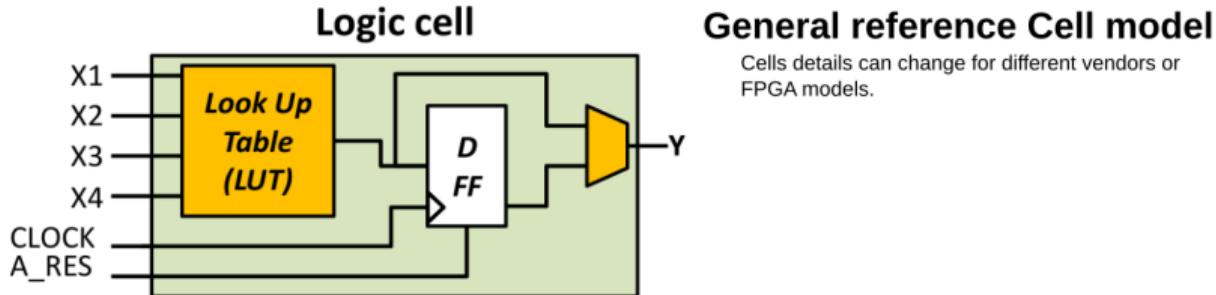
- Parallel computing
- Highly specialized
- Energy efficient



- Array of programmable logic blocks
- Logic blocks configurable to perform complex functions
- The configuration is specified with the hardware description language

FPGA Architecture

Logic Cell



Look-Up Table

X1	X2	X3	X4	Y
0	0	0	1	?
0	0	1	0	?
...	?
1	1	1	1	?



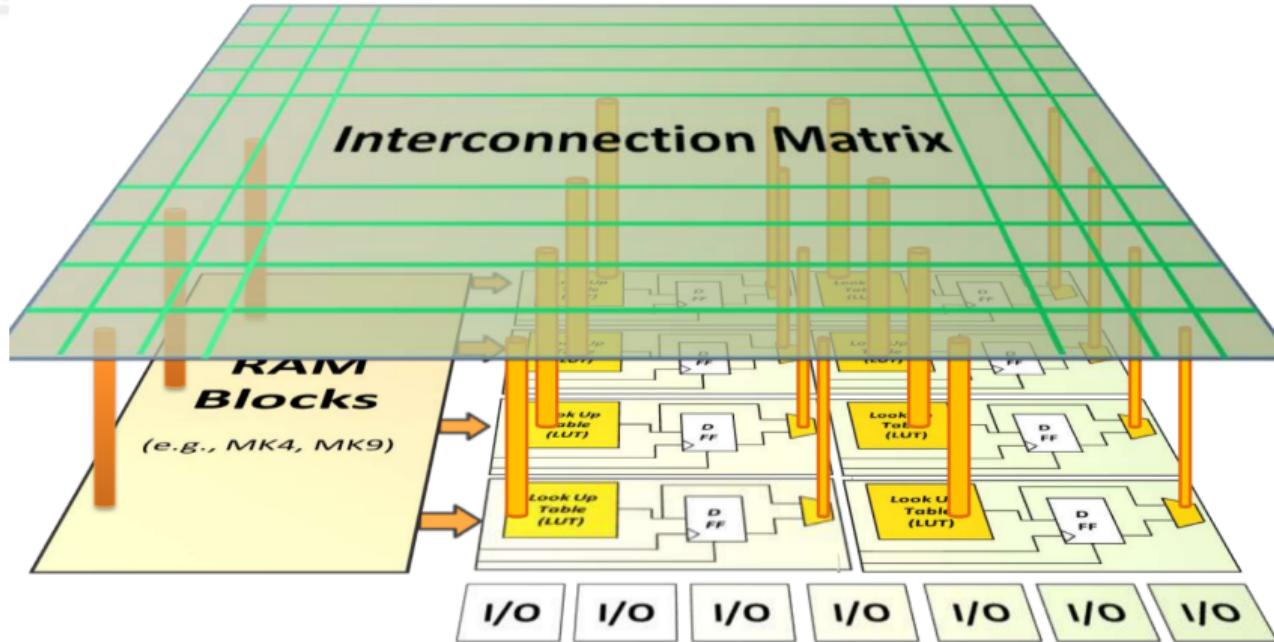
Programmable element

A Cell usually contains:

- A **look-up table**: allow to map any combinatorial function of 4 inputs and 1 output.
- a **FF of type D**: to store persistent data.
- A **mux 2 -> 1**: to eventually bypass the FF for purely combinatorial cells.

FPGA Architecture

Interconnection



FPGA

Use in computing

The use of FPGA in computing is growing due several reasons:

- can potentially deliver great performance via massive **parallelism**
- can address payloads which are not performing well on uniprocessors (Neural Networks, Deep Learning)
- can handle efficiently non-standard data types

FPGA

Use in computing

The use of FPGA in computing is growing due several reasons:

- can potentially deliver great performance via massive **parallelism**
- can address payloads which are not performing well on uniprocessors (Neural Networks, Deep Learning)
- can handle efficiently non-standard data types

FPGA

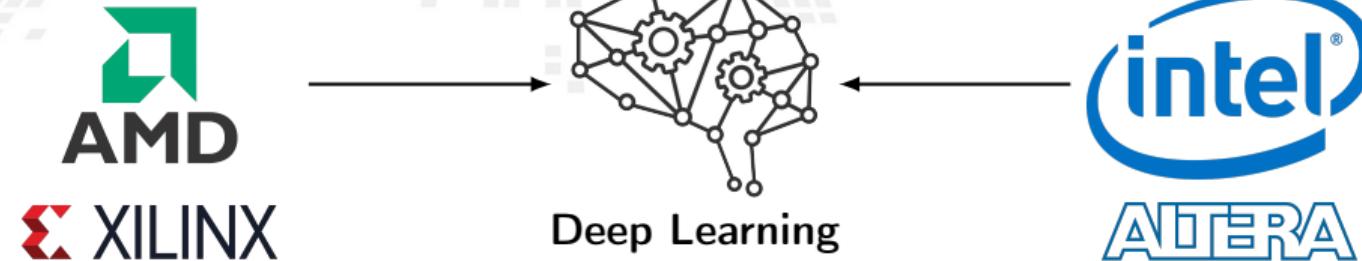
Use in computing

The use of FPGA in computing is growing due several reasons:

- can potentially deliver great performance via massive **parallelism**
- can address payloads which are not performing well on uniprocessors (Neural Networks, Deep Learning)
- can handle efficiently non-standard data types

Integration of neural networks on FPGA

FPGAs are playing an increasingly important role in the industry sampling and data processing.



In the industrial field

- Intelligent vision;
- Financial services;
- Scientific simulations;
- Life science and medical data analysis;

In the scientific field

- Real time deep learning in particle physics;
- Hardware trigger of LHC experiments;
- And many others ...

CPU vs GPU vs FPGA: Instructions, memory and parallelism

CPU	GPU	FPGA
Fixed architecture	Fixed architecture	Adaptable Architecture
Predefined instruction set	Predefined instruction set	No fixed instruction set
Fixed memory hierarchy	Fixed memory hierarchy	Customizable memory hierarchy
Thread-level parallelism	SIMD parallelism	Excels at all types of parallelism

FPGA Performance

- FPGA Performance is predictable
- There is no context switch, garbage collector or any background process
- The bitstream will be executed the same number of clock cycles every time
- The number of clock cycles needed can be computed easily

Slide credits: M.Barbone

Software point of view

Programmable Fixed

Interpreted program

Program
Interpreter
Machine Language
Microarchitecture
Circuits
Electrons

Compiled program

Program
Machine Language
Microarchitecture
Circuits
Electrons

FPGA implementation

Program
Microarchitecture
Circuits
Electrons

Performance



Slide credits: M.Barbone

FPGA

Drawbacks

On the other hand the adoption on FPGA has several drawbacks:

- Porting of legacy code is usually hard.
- Interoperability with standard applications is problematic.

FPGA

Drawbacks

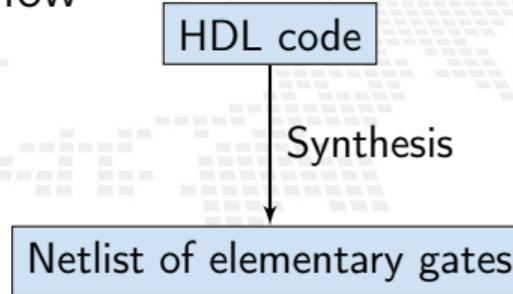
On the other hand the adoption on FPGA has several drawbacks:

- Porting of legacy code is usually hard.
- Interoperability with standard applications is problematic.

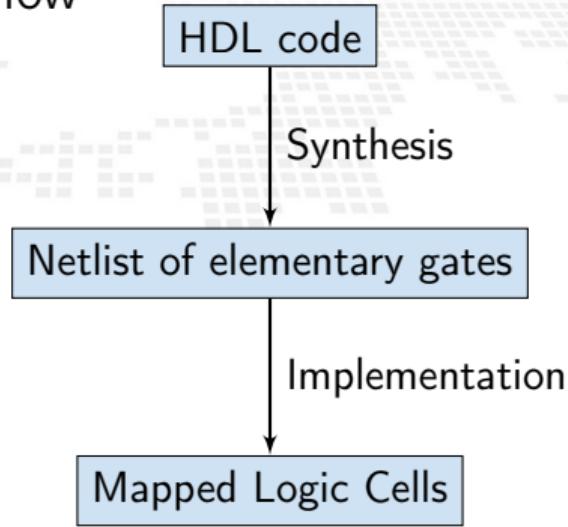
FPGA Programming workflow

HDL code

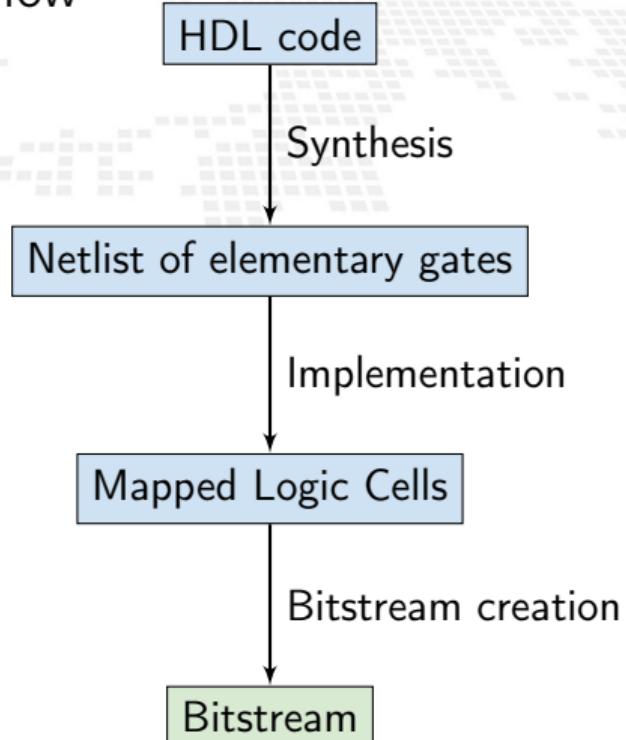
FPGA Programming workflow



FPGA Programming workflow



FPGA Programming workflow

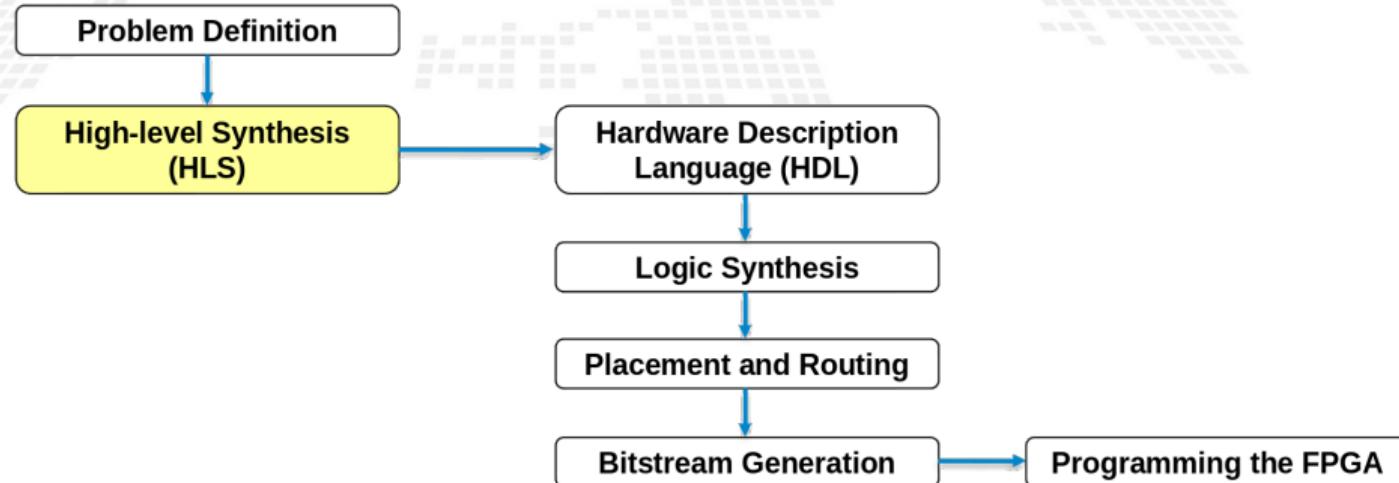


CPU vs GPU vs FPGA (HDL): Coding difficulty

	CPU	GPU	FPGA
Engineering time	short	medium	very long
Compilation time	short	short	very long
Debugging	easy	medium	hard
Maintainability	easy	medium	hard

Slide credits: M.Barbone

HLS Workflow



Slide credits: M.Barbone

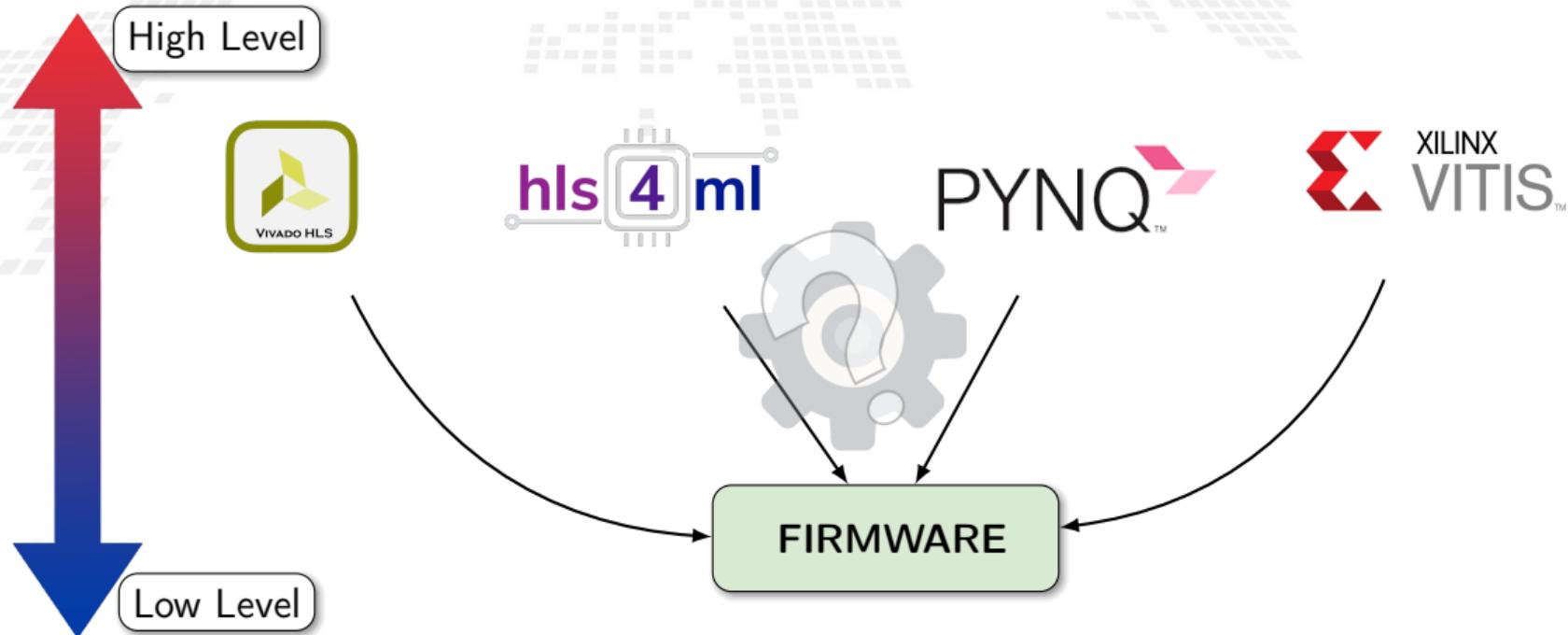
CPU vs GPU vs FPGA (HLS): Coding difficulty

	CPU	GPU	FPGA Sim	FPGA Hw
Engineering time	short	medium	medium	medium
Compilation time	short	short	short	Very long
Debugging	easy	medium	medium	medium
Maintainability	easy	medium	medium	medium

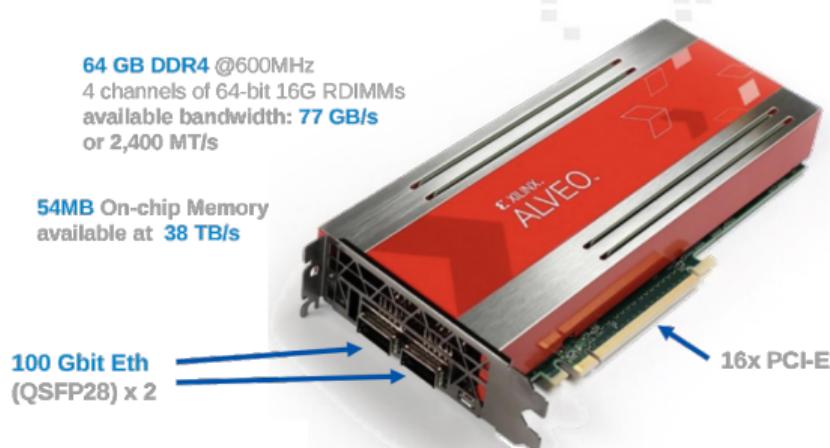
Slide credits: M.Barbone

Firmware generation

Many projects have the goal of abstracting the firmware generation and use process.



Connections



Latency and Throughput

- **Latency:** time passed from inputs to outputs
- **Throughput:** quantity of outputs in the unit of time

Latency in FPGA

FPGAs achieve lower latency than software:

- latency below the microsecond
- latency constant and predictable

CPUs are not suitable as thread overhead is order of 10 microseconds

GPUs are even worse since PCIe bus sys-call could require milliseconds

Examples:

- Ultra low latency trading
- CERN trigger system

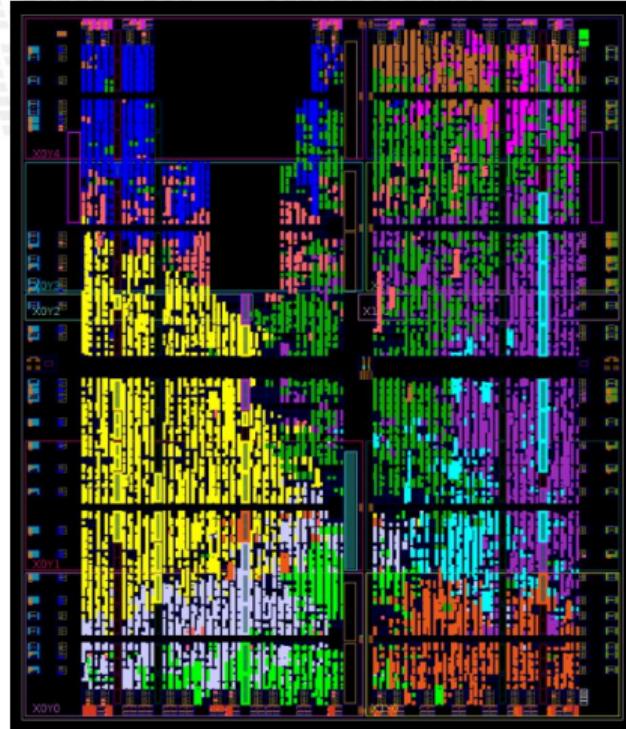
Slide credits: M.Barbone

Occupancy

Is the amount of the various resources used on and FPGA (LUTs, DSP, etc)

Problems:

- The algorithms has to create a circuit that can be contained in the available resources
- Tools to shrink or grow the occupancy
- The need for a runtime, platforms



Cloud resources

- [Amazon EC2 F1](#) use FPGAs to enable delivery of custom hardware accelerations. F1 instances are easy to program and come with everything you need to develop, simulate, debug, and compile hardware acceleration code.
- [Microsoft's Project Brainwave](#) is a deep learning platform for real-time AI inference in the cloud and on the edge. A soft Neural Processing Unit (NPU), based on a high-performance field-programmable gate array (FPGA).

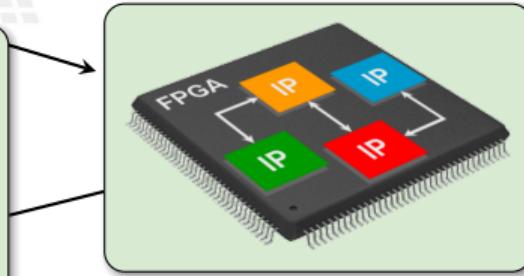
Firmware

Hardware Description Language

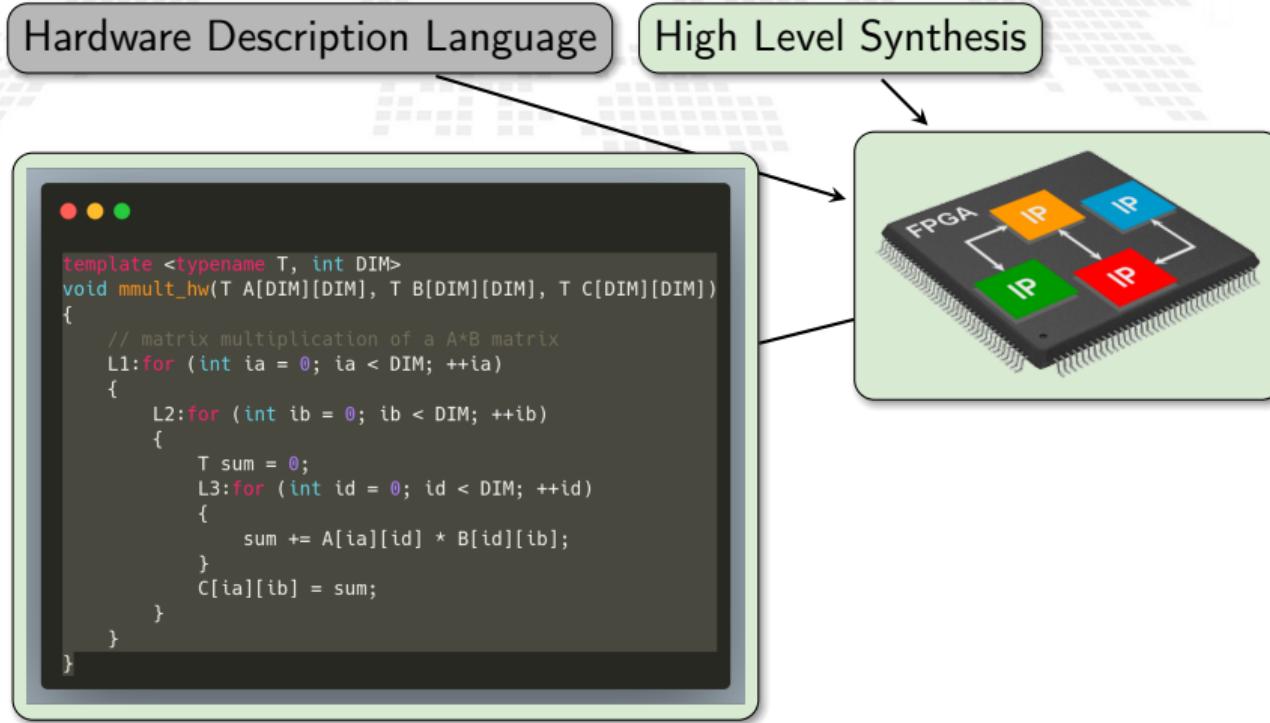
```
  ● ● ●

module Mat_mult(A,B,Res);
    input [31:0] A;
    input [31:0] B;
    output [31:0] Res;
    //internal variables
    reg [31:0] Res;
    reg [7:0] A1 [0:1][0:1];
    reg [7:0] B1 [0:1][0:1];
    reg [7:0] Res1 [0:1][0:1];
    integer i,j,k;

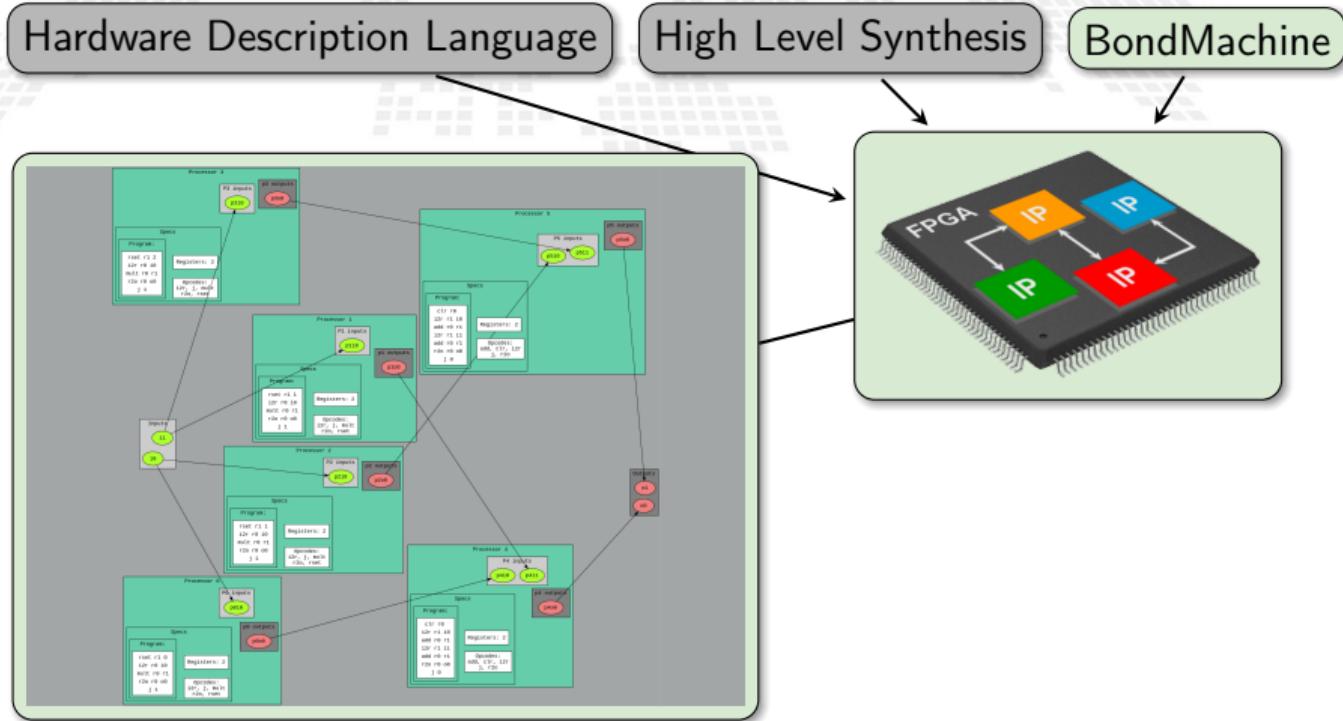
    always@ (A or B)
    begin
        {A1[0][0],A1[0][1],A1[1][0],A1[1][1]} = A;
        {B1[0][0],B1[0][1],B1[1][0],B1[1][1]} = B;
        i = 0;
        j = 0;
        k = 0;
        {Res1[0][0],Res1[0][1],Res1[1][0],Res1[1][1]} = 32'd0;
        for(i=0;i < 2;i=i+1)
            for(j=0;j < 2;j=j+1)
                for(k=0;k < 2;k=k+1)
                    Res1[i][j] = Res1[i][j] + (A1[i][k] * B1[k][j]);
        Res = {Res1[0][0],Res1[0][1],Res1[1][0],Res1[1][1]};
    end
endmodule
```



Firmware



Firmware



The BondMachine

High level sources: Go, TensorFlow, NN, ...

Building a new kind of computer architecture (multi-core and heterogeneous both in cores types and interconnections) which dynamically adapt to the specific computational problem rather than be static.



The BondMachine

High level sources: Go, TensorFlow, NN, ...

Building a new kind of computer architecture (multi-core and heterogeneous both in cores types and interconnections) which dynamically adapt to the specific computational problem rather than be static.

BM architecture Layer

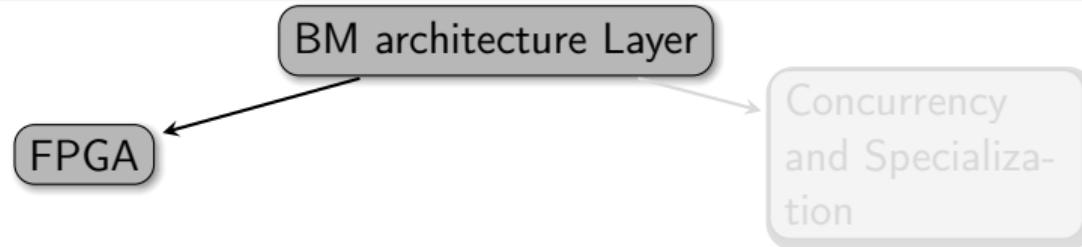
FPGA

Concurrency
and Specializa-
tion

The BondMachine

High level sources: Go, TensorFlow, NN, ...

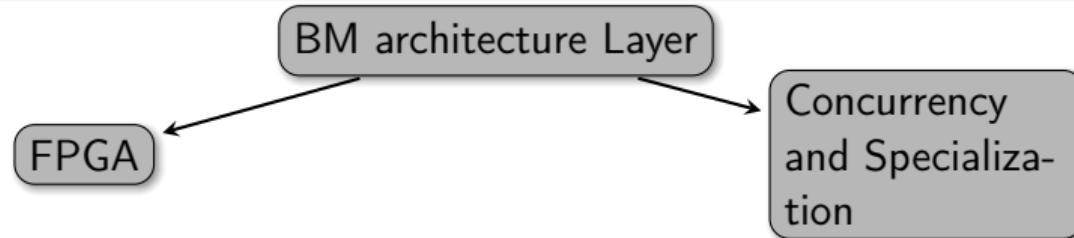
Building a new kind of computer architecture (multi-core and heterogeneous both in cores types and interconnections) which dynamically adapt to the specific computational problem rather than be static.



The BondMachine

High level sources: Go, TensorFlow, NN, ...

Building a new kind of computer architecture (multi-core and heterogeneous both in cores types and interconnections) which dynamically adapt to the specific computational problem rather than be static.

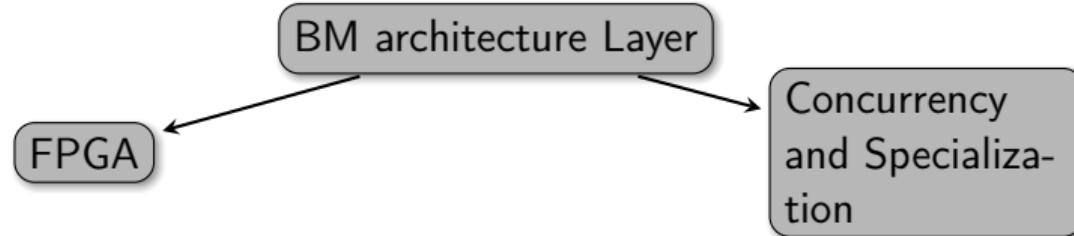


The BondMachine

High level sources: Go, TensorFlow, NN, ...



Building a new kind of computer architecture (multi-core and heterogeneous both in cores types and interconnections) which dynamically adapt to the specific computational problem rather than be static.



The BondMachine project

1 Introduction

Challenges

FPGA

HDL workflow

HLS Workflow

Concepts

Cloud

2 The BondMachine project

Architectures handling

Architectures molding

Bondgo

Basm

API

3 Misc

Project timeline

4 Machine Learning

Train

BondMachine creation

Simulation

Accelerator

Benchmark

5 Optimizations

6 Conclusions and Future directions

Conclusions

Ongoing

Future

Introducing the BondMachine (BM)

The **BondMachine** is a software ecosystem for the dynamic generation of computer architectures that:

- Are composed by many, possibly hundreds, computing cores.
- Have very small cores and not necessarily of the same type (different ISA and ABI).
- Have a not fixed way of interconnecting cores.
- May have some elements shared among cores (for example channels and shared memories).

Introducing the BondMachine (BM)

The **BondMachine** is a software ecosystem for the dynamic generation of computer architectures that:

- Are composed by many, possibly hundreds, computing cores.
- Have very small cores and not necessarily of the same type (different ISA and ABI).
- Have a not fixed way of interconnecting cores.
- May have some elements shared among cores (for example channels and shared memories).

Introducing the BondMachine (BM)

The **BondMachine** is a software ecosystem for the dynamic generation of computer architectures that:

- Are composed by many, possibly hundreds, computing cores.
- Have very small cores and not necessarily of the same type (different ISA and ABI).
- Have a not fixed way of interconnecting cores.
- May have some elements shared among cores (for example channels and shared memories).

Introducing the BondMachine (BM)

The **BondMachine** is a software ecosystem for the dynamic generation of computer architectures that:

- Are composed by many, possibly hundreds, computing cores.
- Have very small cores and not necessarily of the same type (different ISA and ABI).
- Have a not fixed way of interconnecting cores.
- May have some elements shared among cores (for example channels and shared memories).

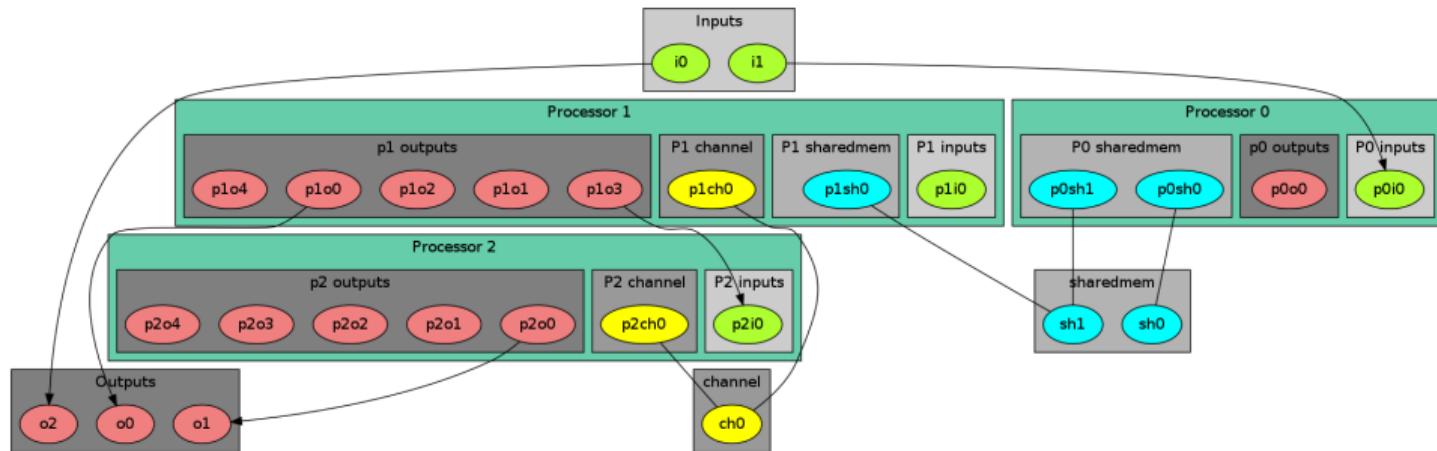
Introducing the BondMachine (BM)

The **BondMachine** is a software ecosystem for the dynamic generation of computer architectures that:

- Are composed by many, possibly hundreds, computing cores.
- Have very small cores and not necessarily of the same type (different ISA and ABI).
- Have a not fixed way of interconnecting cores.
- May have some elements shared among cores (for example channels and shared memories).

The BondMachine

An example



Connecting Processor (CP)

The computational unit of the BM

The atomic computational unit of a BM is the “connecting processor” (CP) and has:

- Some general purpose registers of size $Rsize$.

- Some I/O dedicated registers of size $Rsize$.

- A set of implemented opcodes chosen among many available.

- Dedicated ROM and RAM.

- Three possible operating modes.

Connecting Processor (CP)

The computational unit of the BM

The atomic computational unit of a BM is the “connecting processor” (CP) and has:

- Some general purpose registers of size **Rsize**.
 - Some I/O dedicated registers of size **Rsize**.
 - A set of implemented opcodes chosen among many available.
 - Dedicated ROM and RAM.
 - Three possible operating modes.

General purpose registers

2^R registers: r0,r1,r2,r3 ... r 2^R

Connecting Processor (CP)

The computational unit of the BM

The atomic computational unit of a BM is the “connecting processor” (CP) and has:

Some general purpose registers of size **Rsize**.

■ Some I/O dedicated registers of size **Rsize**.

A set of implemented opcodes chosen among many available.

■ Dedicated ROM and RAM.

■ Three possible operating modes.

I/O specialized registers

N input registers: i₀, i₁ ... i_N

M output registers: o₀, o₁ ... o_M

Connecting Processor (CP)

The computational unit of the BM

The atomic computational unit of a BM is the “connecting processor” (CP) and has:

- Some general purpose registers of size **Rsize**.

- Some I/O dedicated registers of size **Rsize**.

- A set of implemented opcodes chosen among many available.
- Dedicated ROM and RAM.
- Three possible operating modes.

Full set of possible opcodes

```
adc,add,addf,addi,and,chc,chw,cil,cilc,cir,cirn,clc,clr,cpy,cset,dec,div,divf,dpc,expf,hit  
hlt,i2r,i2rw,incc,inc,j,jc,je,jgt0f,jlt,jlte,jr,jz,lfsr82,lfsr162r,m2r,mod,mulc,mult,multf  
nand,nop,nor,not,or,r2m,r2o,r2owa,r2owaa,r2s,r2v,r2vri,ro2r,ro2rri,rsc,rset,sic,s2r,saj,sbc  
sub,wrd,wwr,xnor,xor
```

Connecting Processor (CP)

The computational unit of the BM

The atomic computational unit of a BM is the “connecting processor” (CP) and has:

- Some general purpose registers of size $Rsize$.

- Some I/O dedicated registers of size $Rsize$.

- A set of implemented opcodes chosen among many available.

- Dedicated ROM and RAM.

- Three possible operating modes.

RAM and ROM

- 2^L RAM memory cells.

- 2^O ROM memory cells.

Connecting Processor (CP)

The computational unit of the BM

The atomic computational unit of a BM is the “connecting processor” (CP) and has:

- Some general purpose registers of size $Rsize$.

- Some I/O dedicated registers of size $Rsize$.

- A set of implemented opcodes chosen among many available.

- Dedicated ROM and RAM.

- Three possible operating modes.

Operating modes

- Full Harvard mode.
- Full Von Neuman mode.
- Hybrid mode.

Shared Objects (SO)

The non-computational element of the BM

Alongside CPs, BondMachines include non-computing units called “Shared Objects” (SO).

Examples of their purposes are:

- Data storage (Memories).
- Message passing.
- CP synchronization.

A single SO can be shared among different CPs. To use it CPs have special instructions (opcodes) oriented to the specific SO.

Four kind of SO have been developed so far: the Channel, the Shared Memory, the Barrier and a Pseudo Random Numbers Generator.

Shared Objects (SO)

The non-computational element of the BM

Alongside CPs, BondMachines include non-computing units called “Shared Objects” (SO).

Examples of their purposes are:

- Data storage (Memories).
- Message passing.
- CP synchronization.

A single SO can be shared among different CPs. To use it CPs have special instructions (opcodes) oriented to the specific SO.

Four kind of SO have been developed so far: the Channel, the Shared Memory, the Barrier and a Pseudo Random Numbers Generator.

Shared Objects (SO)

The non-computational element of the BM

Alongside CPs, BondMachines include non-computing units called “Shared Objects” (SO).

Examples of their purposes are:

- Data storage (Memories).
- Message passing.
- CP synchronization.

A single SO can be shared among different CPs. To use it CPs have special instructions (opcodes) oriented to the specific SO.

Four kind of SO have been developed so far: the Channel, the Shared Memory, the Barrier and a Pseudo Random Numbers Generator.

Shared Objects (SO)

The non-computational element of the BM

Alongside CPs, BondMachines include non-computing units called “Shared Objects” (SO).

Examples of their purposes are:

- Data storage (Memories).
- Message passing.
- CP synchronization.

A single SO can be shared among different CPs. To use it CPs have special instructions (opcodes) oriented to the specific SO.

Four kind of SO have been developed so far: the Channel, the Shared Memory, the Barrier and a Pseudo Random Numbers Generator.

Shared Objects (SO)

The non-computational element of the BM

Alongside CPs, BondMachines include non-computing units called “Shared Objects” (SO).

Examples of their purposes are:

- Data storage (Memories).
- Message passing.
- CP synchronization.

A single SO can be shared among different CPs. To use it CPs have special instructions (opcodes) oriented to the specific SO.

Four kind of SO have been developed so far: the Channel, the Shared Memory, the Barrier and a Pseudo Random Numbers Generator.

Shared Objects (SO)

The non-computational element of the BM

Alongside CPs, BondMachines include non-computing units called “Shared Objects” (SO).

Examples of their purposes are:

- Data storage (Memories).
- Message passing.
- CP synchronization.

A single SO can be shared among different CPs. To use it CPs have special instructions (opcodes) oriented to the specific SO.

Four kind of SO have been developed so far: the [Channel](#), the [Shared Memory](#), the [Barrier](#) and a [Pseudo Random Numbers Generator](#).

Handle BM computer architectures

The BM computer architecture is managed by a set of tools to:

- build a specify architecture
- modify a pre-existing architecture
- simulate or emulate the behavior
- generate the Hardware Description Language Code (HDL)

Processor Builder

Selects the single processor, assembles and disassembles, saves on disk as JSON, creates the HDL code of a CP

BondMachine Builder

Connects CPs and SOs together in custom topologies, loads and saves on disk as JSON, create BM's HDL code

Simulation Framework

Simulates the behaviour, emulates a BM on a standard Linux workstation

Handle BM computer architectures

The BM computer architecture is managed by a set of tools to:

- build a specify architecture
- modify a pre-existing architecture
- simulate or emulate the behavior
- generate the Hardware Description Language Code (HDL)

Processor Builder

Selects the single processor, assembles and disassembles, saves on disk as JSON, creates the HDL code of a CP

BondMachine Builder

Connects CPs and SOs together in custom topologies, loads and saves on disk as JSON, create BM's HDL code

Simulation Framework

Simulates the behaviour, emulates a BM on a standard Linux workstation

Handle BM computer architectures

The BM computer architecture is managed by a set of tools to:

- build a specify architecture
- modify a pre-existing architecture
- simulate or emulate the behavior
- generate the Hardware Description Language Code (HDL)

Processor Builder

Selects the single processor, assembles and disassembles, saves on disk as JSON, creates the HDL code of a CP

BondMachine Builder

Connects CPs and SOs together in custom topologies, loads and saves on disk as JSON, create BM's HDL code

Simulation Framework

Simulates the behaviour, emulates a BM on a standard Linux workstation

Handle BM computer architectures

The BM computer architecture is managed by a set of tools to:

- build a specify architecture
- modify a pre-existing architecture
- simulate or emulate the behavior
- generate the Hardware Description Language Code (HDL)

Processor Builder

Selects the single processor, assembles and disassembles, saves on disk as JSON, creates the HDL code of a CP

BondMachine Builder

Connects CPs and SOs together in custom topologies, loads and saves on disk as JSON, create BM's HDL code

Simulation Framework

Simulates the behaviour, emulates a BM on a standard Linux workstation

Toolchains

A set of toolchains allow the build and the direct deploy to a target device of BondMachines

Bondgo Toolchain main targets

A file local.mk contains references to the source code as well all the build necessities
`make bondmachine` creates the JSON representation of the BM and assemble its code
`make hdl` creates the HDL files of the BM
`make show` displays a graphical representation of the BM
`make simulate [simbatch]` start a simulation [batch simulation]
`make bitstream [design_bitstream]` create the firwware [accelerator firmware]
`make program` flash the device into the destination target

Simulation

An important feature of the tools is the possibility of simulating BondMachine behavior.

An event input file describes how BondMachines elements has to change during the simulation timespan and which one has to be reported.

The simulator can produce results in the form of:

- Activity log of the BM internal.
- Graphical representation of the simulation.
- Report file with quantitative data. Useful to construct metrics

Graphical simulation in action

Simulation

An important feature of the tools is the possibility of simulating BondMachine behavior.

An event input file describes how BondMachines elements has to change during the simulation timespan and which one has to be reported.

The simulator can produce results in the form of:

- Activity log of the BM internal.
- Graphical representation of the simulation.
- Report file with quantitative data. Useful to construct metrics

Graphical simulation in action

Simulation

An important feature of the tools is the possibility of simulating BondMachine behavior.

An event input file describes how BondMachines elements has to change during the simulation timespan and which one has to be reported.

The simulator can produce results in the form of:

- Activity log of the BM internal.
- Graphical representation of the simulation.
- Report file with quantitative data. [Useful to construct metrics](#)

[Graphical simulation in action](#)

Molding the BondMachine

As stated before BondMachines are not general purpose architectures, and to be effective have to be shaped according the specific problem.

Several methods (apart from writing in assembly and building a BondMachine from scratch) have been developed to do that:

- *bondgo*: A new type of compiler that create not only the CPs assembly but also the architecture itself.
- *basm*: The BondMachine Assembler.
- A set of API to create BondMachine to fit a specific computational problems.
- An Evolutionary Computation framework to “grow” BondMachines according some fitness function via simulation.
- A set of tools to use BondMachine in Machine Learning.

Molding the BondMachine

As stated before BondMachines are not general purpose architectures, and to be effective have to be shaped according the specific problem.

Several methods (apart from writing in assembly and building a BondMachine from scratch) have been developed to do that:

- *bondgo*: A new type of compiler that create not only the CPs assembly but also the architecture itself.
- *basm*: The BondMachine Assembler.
- A set of API to create BondMachine to fit a specific computational problems.
- An Evolutionary Computation framework to “grow” BondMachines according some fitness function via simulation.
- A set of tools to use BondMachine in Machine Learning.

Molding the BondMachine

As stated before BondMachines are not general purpose architectures, and to be effective have to be shaped according the specific problem.

Several methods (apart from writing in assembly and building a BondMachine from scratch) have been developed to do that:

- *bondgo*: A new type of compiler that create not only the CPs assembly but also the architecture itself.
- *basm*: The BondMachine Assembler.
- A set of API to create BondMachine to fit a specific computational problems.
- An Evolutionary Computation framework to “grow” BondMachines according some fitness function via simulation.
- A set of tools to use BondMachine in Machine Learning.

Molding the BondMachine

As stated before BondMachines are not general purpose architectures, and to be effective have to be shaped according the specific problem.

Several methods (apart from writing in assembly and building a BondMachine from scratch) have been developed to do that:

- *bondgo*: A new type of compiler that create not only the CPs assembly but also the architecture itself.
- *basm*: The BondMachine Assembler.
- A set of API to create BondMachine to fit a specific computational problems.
- An Evolutionary Computation framework to “grow” BondMachines according some fitness function via simulation.
- A set of tools to use BondMachine in Machine Learning.

Molding the BondMachine

As stated before BondMachines are not general purpose architectures, and to be effective have to be shaped according the specific problem.

Several methods (apart from writing in assembly and building a BondMachine from scratch) have been developed to do that:

- *bondgo*: A new type of compiler that create not only the CPs assembly but also the architecture itself.
- *basm*: The BondMachine Assembler.
- A set of API to create BondMachine to fit a specific computational problems.
- An Evolutionary Computation framework to “grow” BondMachines according some fitness function via simulation.
- A set of tools to use BondMachine in Machine Learning.

Molding the BondMachine

As stated before BondMachines are not general purpose architectures, and to be effective have to be shaped according the specific problem.

Several methods (apart from writing in assembly and building a BondMachine from scratch) have been developed to do that:

- *bondgo*: A new type of compiler that create not only the CPs assembly but also the architecture itself.
- *basm*: The BondMachine Assembler.
- A set of API to create BondMachine to fit a specific computational problems.
- An Evolutionary Computation framework to “grow” BondMachines according some fitness function via simulation.
- A set of tools to use BondMachine in Machine Learning.

Molding the BondMachine

As stated before BondMachines are not general purpose architectures, and to be effective have to be shaped according the specific problem.

Several methods (apart from writing in assembly and building a BondMachine from scratch) have been developed to do that:

- *bondgo*: A new type of compiler that create not only the CPs assembly but also the architecture itself.
- *basm*: The BondMachine Assembler.
- A set of API to create BondMachine to fit a specific computational problems.
- An Evolutionary Computation framework to “grow” BondMachines according some fitness function via simulation.
- A set of tools to use BondMachine in Machine Learning.

Use the BM computer architecture

Mapping specific computational problems to BMs

Symbond

Map symbolic mathematical expressions to BM

Boolbond

Map boolean systems to BM

Matrixwork

Basic matrix computation

Basm

The BondMachine assembler

Bondgo

The architecture compiler

ML tools

Map computational graphs to BM

more about these tools

Use the BM computer architecture

Mapping specific computational problems to BMs

Symbond

Map symbolic mathematical expressions to BM

Boolbond

Map boolean systems to BM

Matrixwork

Basic matrix computation

Basm

The BondMachine assembler

Bondgo

The architecture compiler

ML tools

Map computational graphs to BM

more about these tools

Use the BM computer architecture

Mapping specific computational problems to BMs

Symbond

Map symbolic mathematical expressions to BM

Boolbond

Map boolean systems to BM

Matrixwork

Basic matrix computation

Basm

The BondMachine assembler

Bondgo

The architecture compiler

ML tools

Map computational graphs to BM

more about these tools

Use the BM computer architecture

Mapping specific computational problems to BMs

Symbond

Map symbolic mathematical expressions to BM

Boolbond

Map boolean systems to BM

Matrixwork

Basic matrix computation

Basm

The BondMachine assembler

Bondgo

The architecture compiler

ML tools

Map computational graphs to BM

more about these tools

Use the BM computer architecture

Mapping specific computational problems to BMs

Symbond

Map symbolic mathematical expressions to BM

Boolbond

Map boolean systems to BM

Matrixwork

Basic matrix computation

Basm

The BondMachine assembler

Bondgo

The architecture compiler

ML tools

Map computational graphs to BM

more about these tools

Use the BM computer architecture

Mapping specific computational problems to BMs

Symbond

Map symbolic mathematical expressions to BM

Boolbond

Map boolean systems to BM

Matrixwork

Basic matrix computation

Basm

The BondMachine assembler

Bondgo

The architecture compiler

ML tools

Map computational graphs to BM

more about these tools

Use the BM computer architecture

Mapping specific computational problems to BMs

Symbond

Map symbolic mathematical expressions to BM

Boolbond

Map boolean systems to BM

Matrixwork

Basic matrix computation

Basm

The BondMachine assembler

Bondgo

The architecture compiler

ML tools

Map computational graphs to BM

more about these tools

The major innovation of the BondMachine Project is its compiler.

[Bondgo](#) is the name chosen for the compiler developed for the BondMachine.

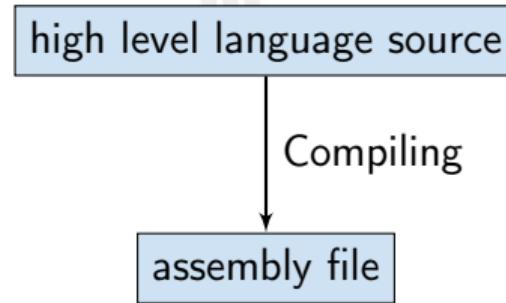
The compiler source language is Go as the name suggest.

This is the standard flow when building computer programs

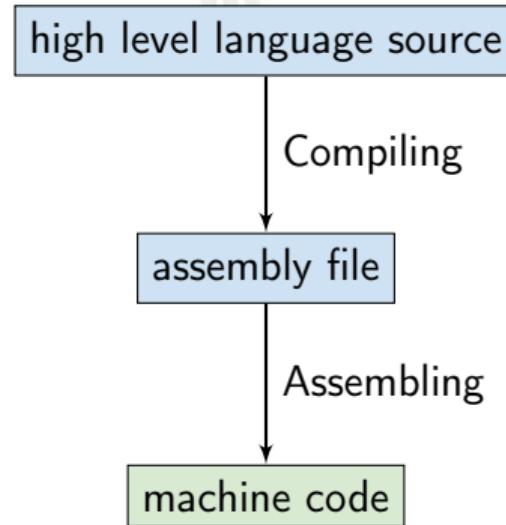
This is the standard flow when building computer programs

high level language source

This is the standard flow when building computer programs



This is the standard flow when building computer programs

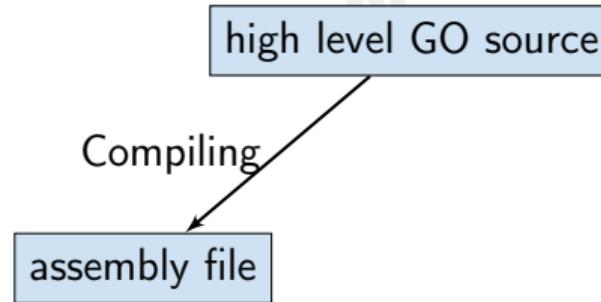


Bondgo does something different from standard compilers ...

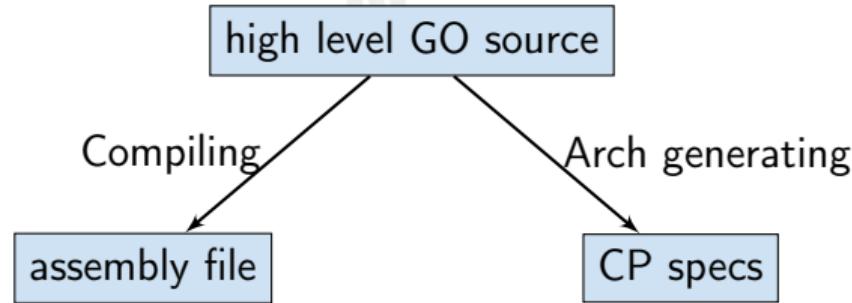
Bondgo does something different from standard compilers ...

high level GO source

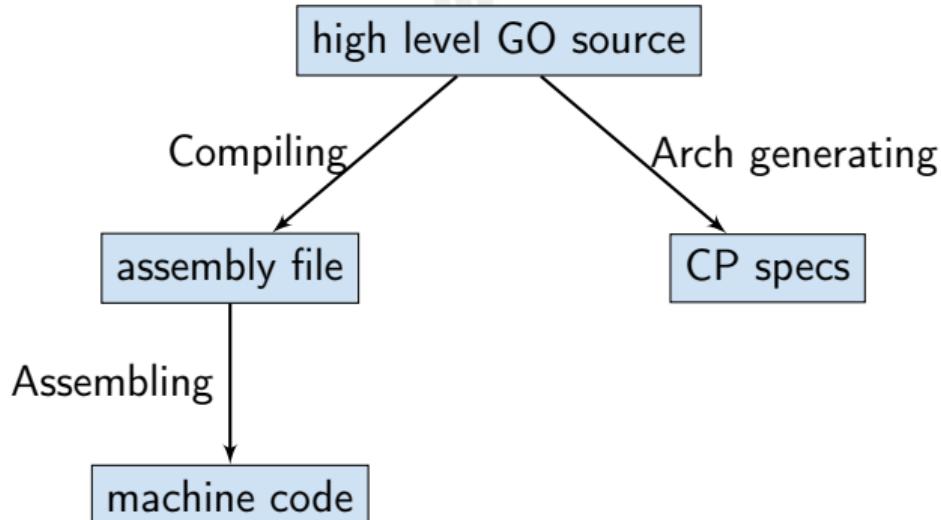
Bondgo does something different from standard compilers ...



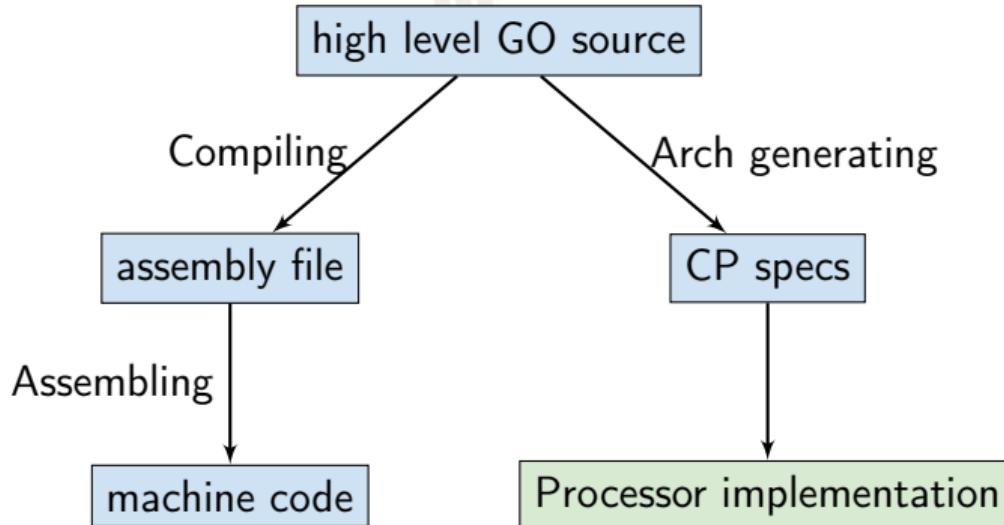
Bondgo does something different from standard compilers ...



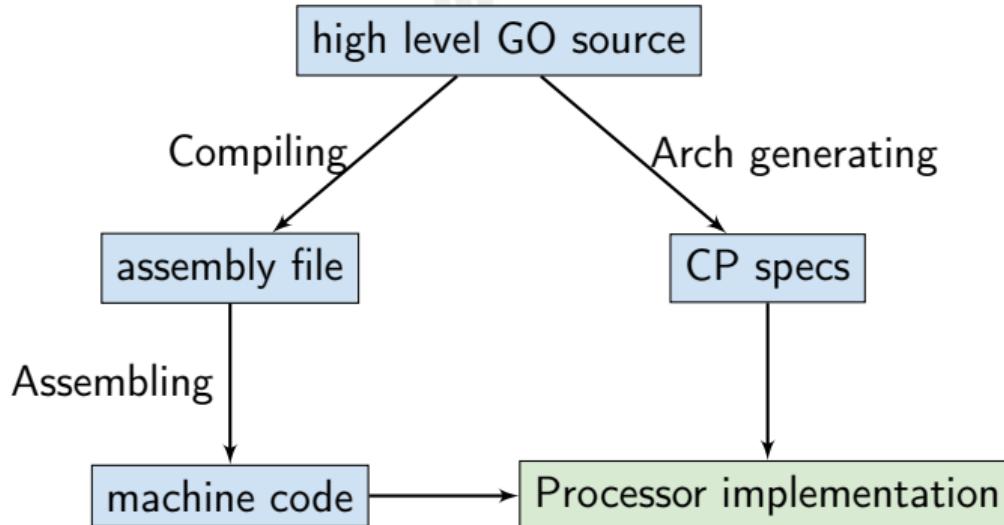
Bondgo does something different from standard compilers ...



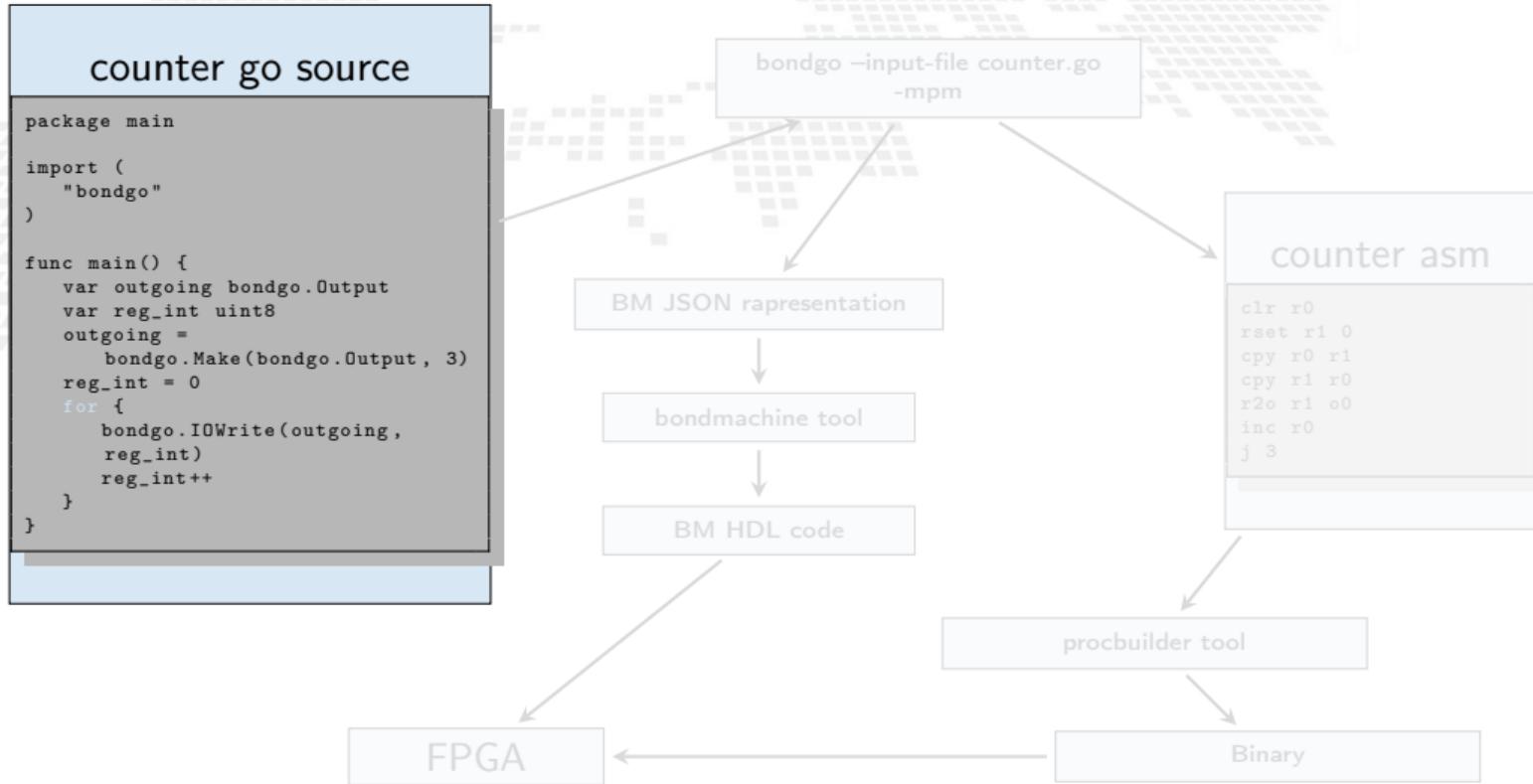
Bondgo does something different from standard compilers ...



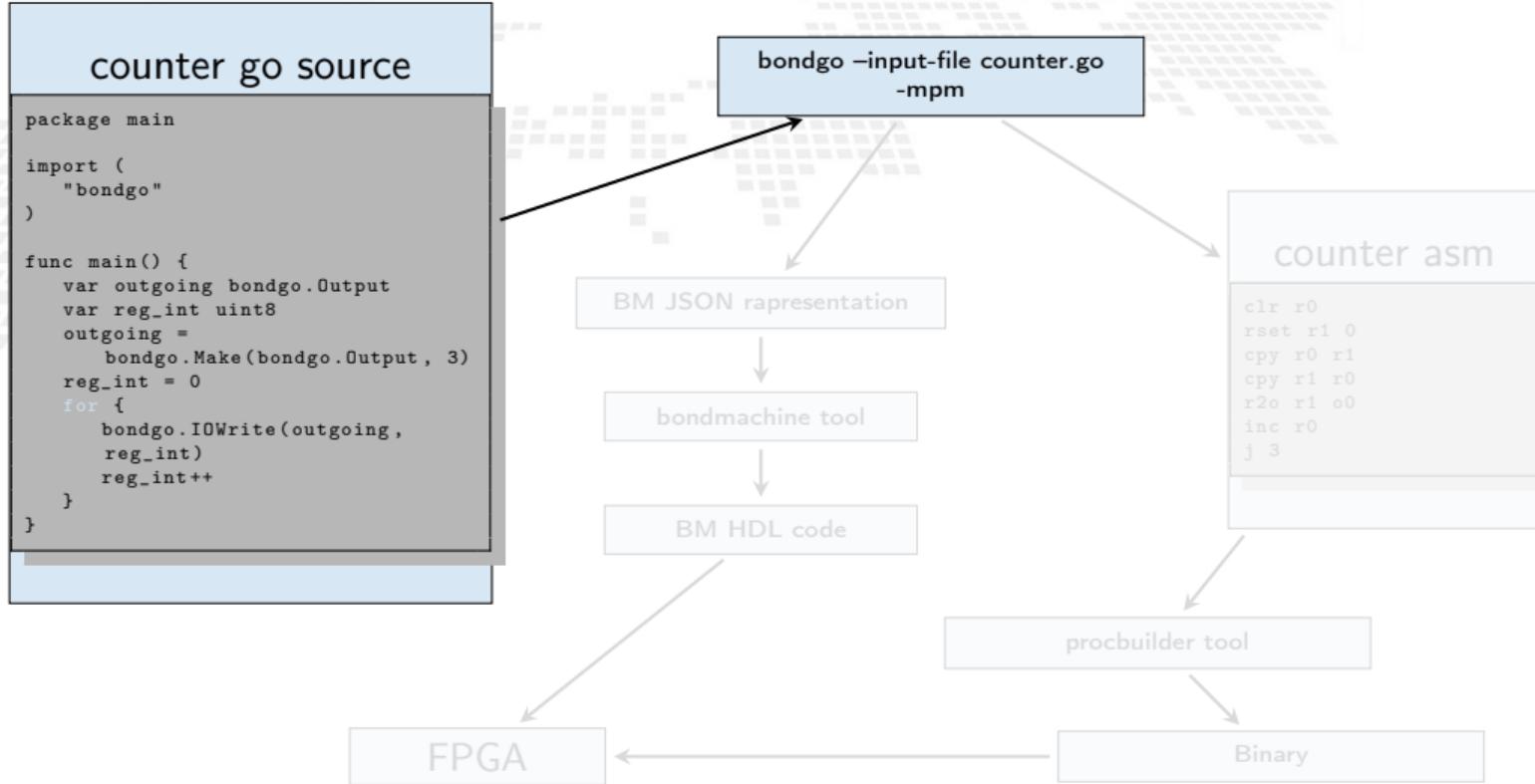
Bondgo does something different from standard compilers ...



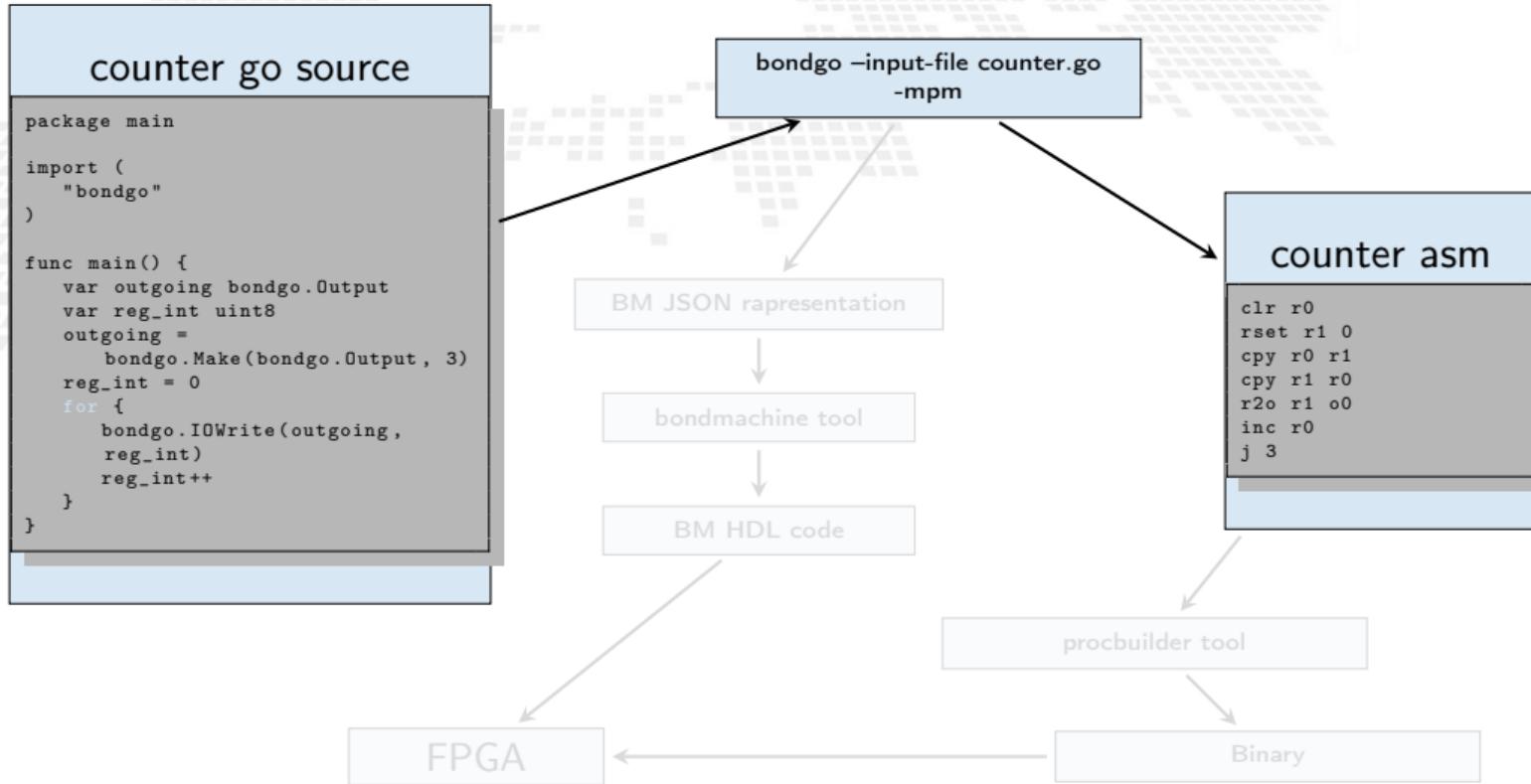
Bondgo workflow example



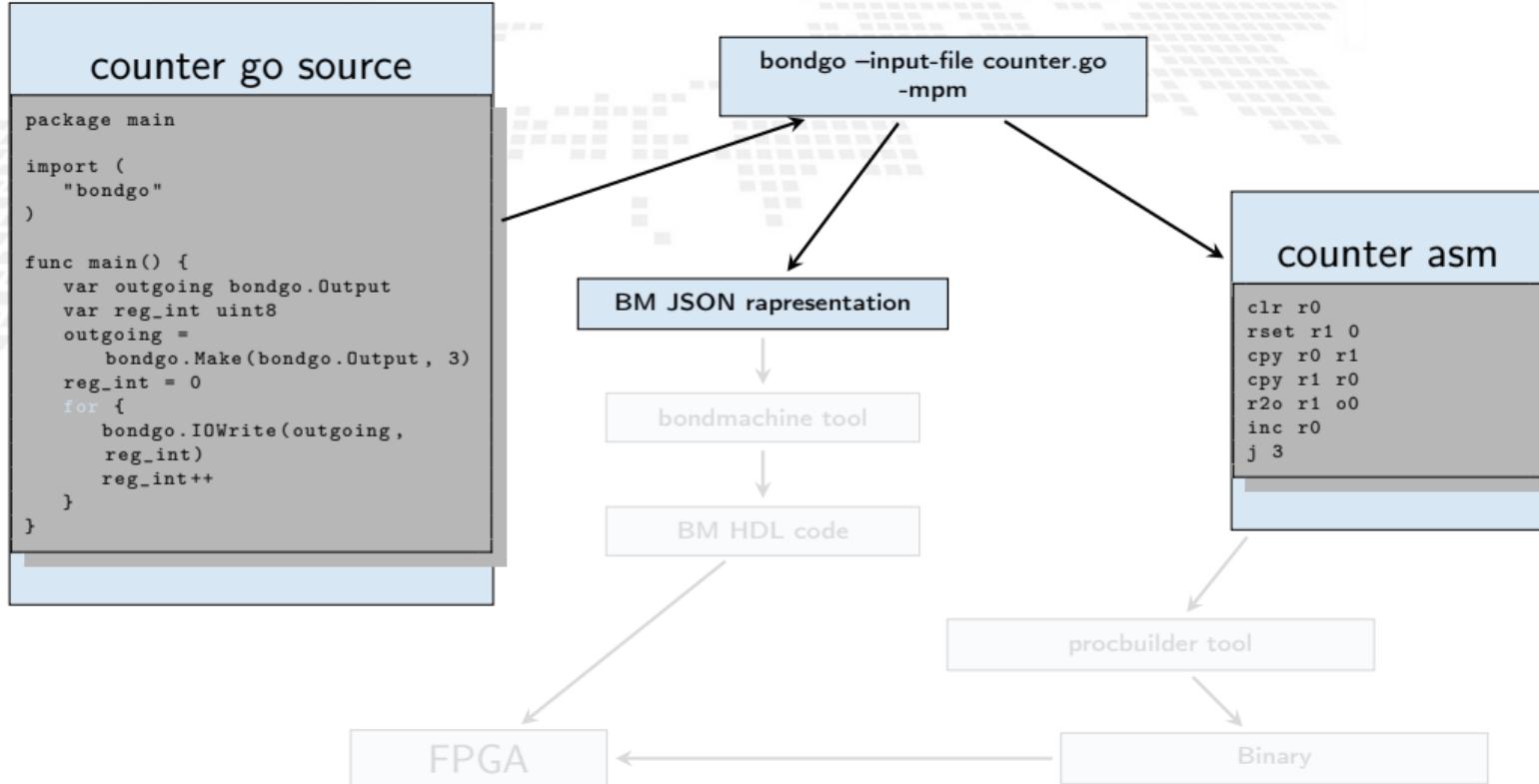
Bondgo workflow example



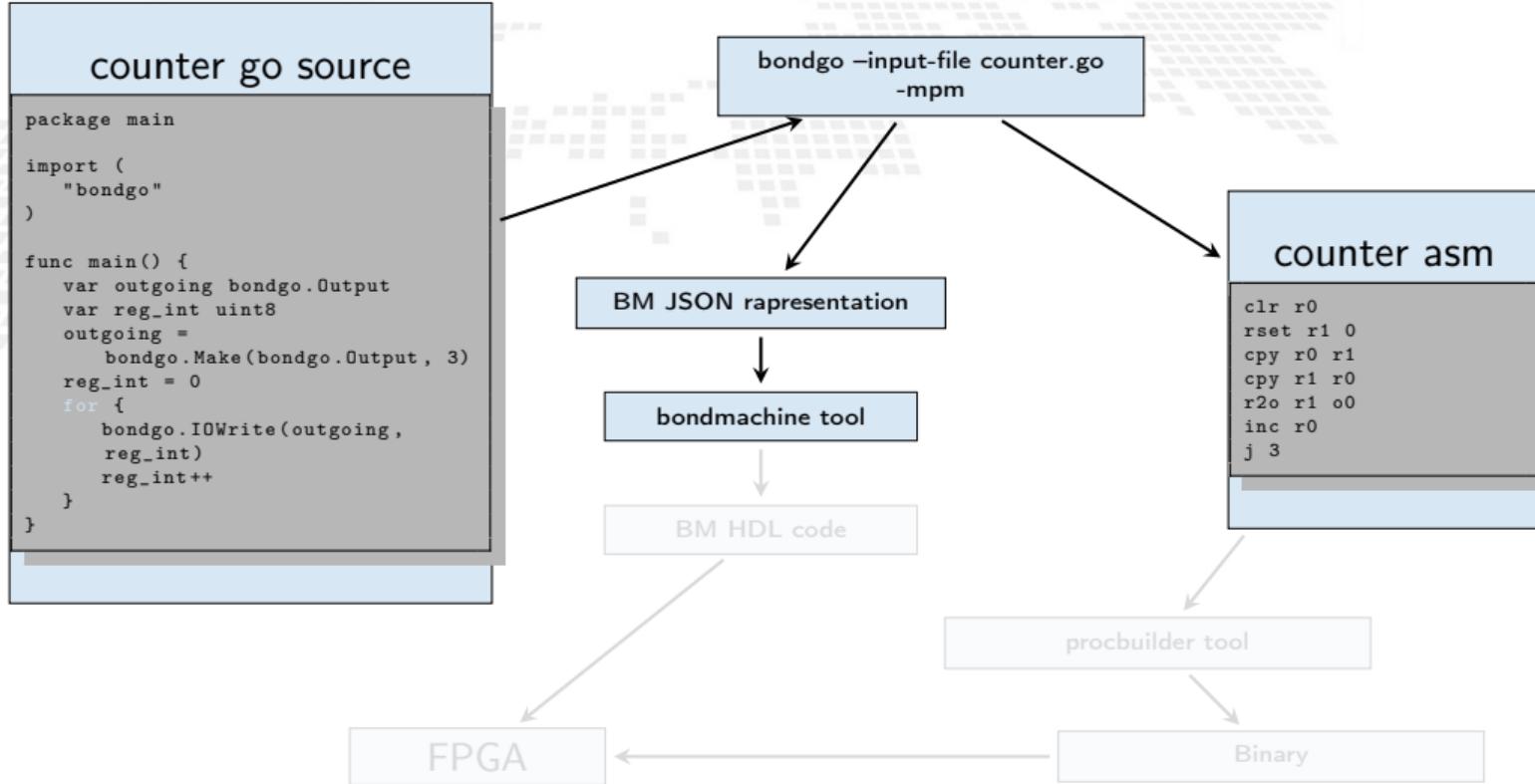
Bondgo workflow example



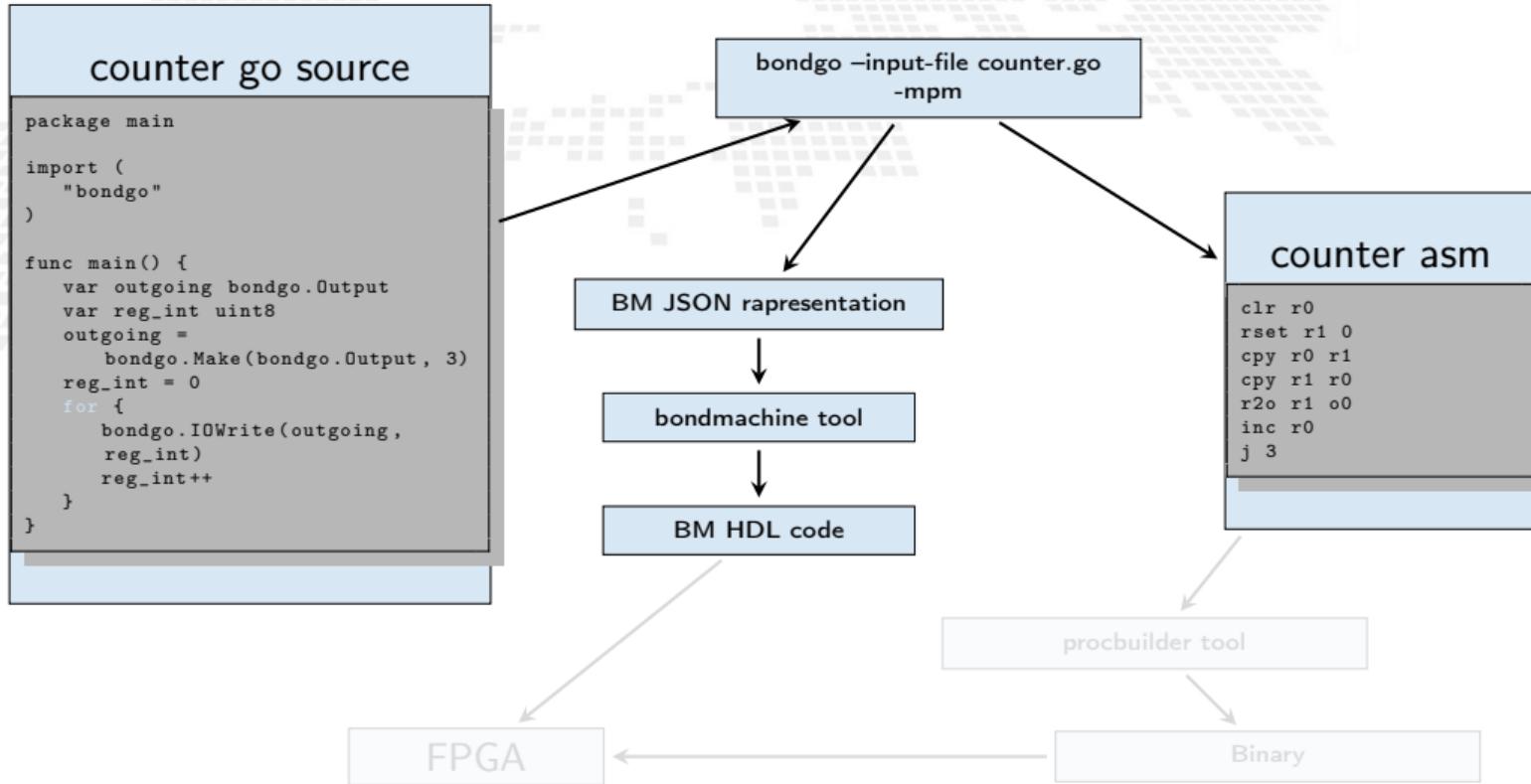
Bondgo workflow example



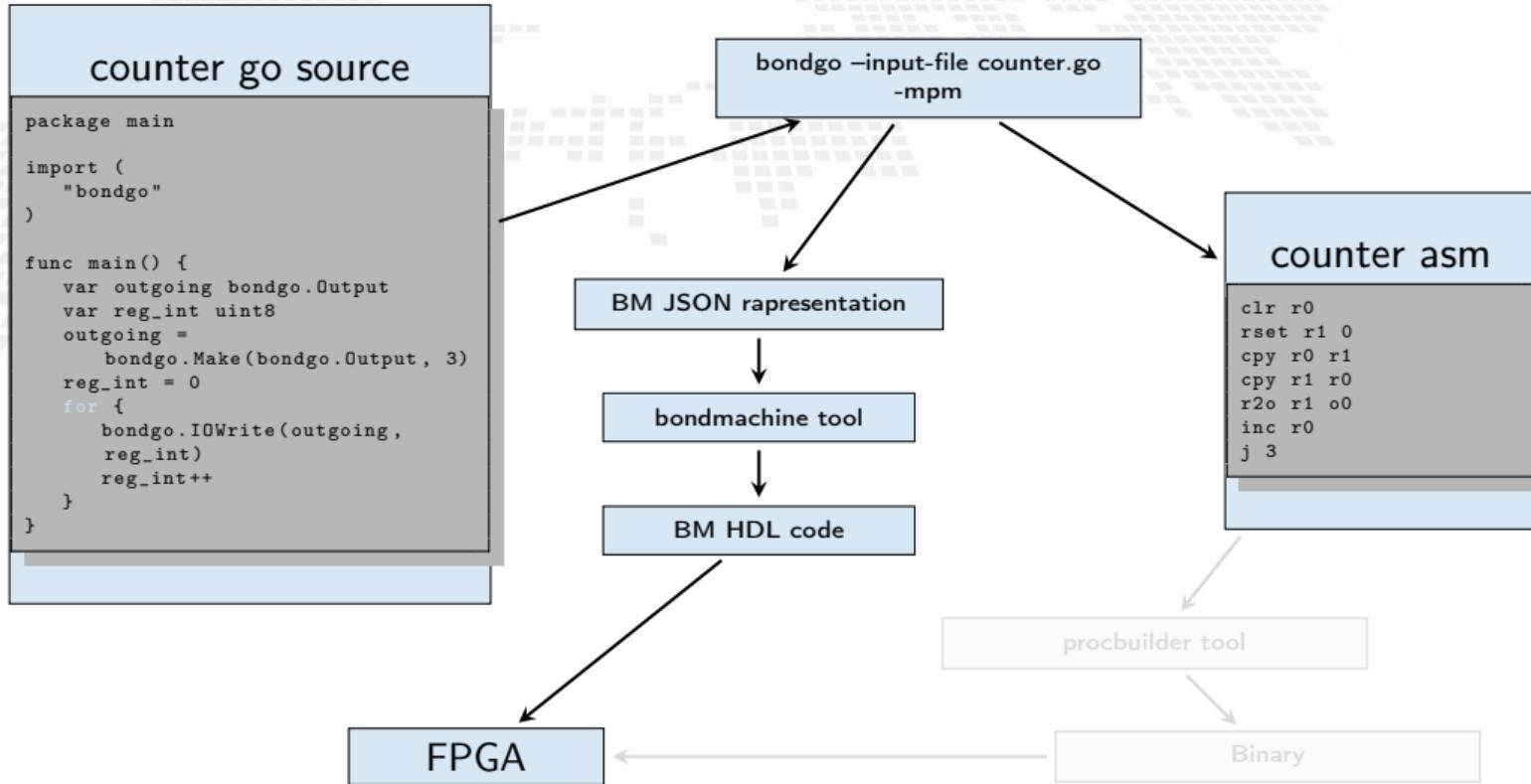
Bondgo workflow example



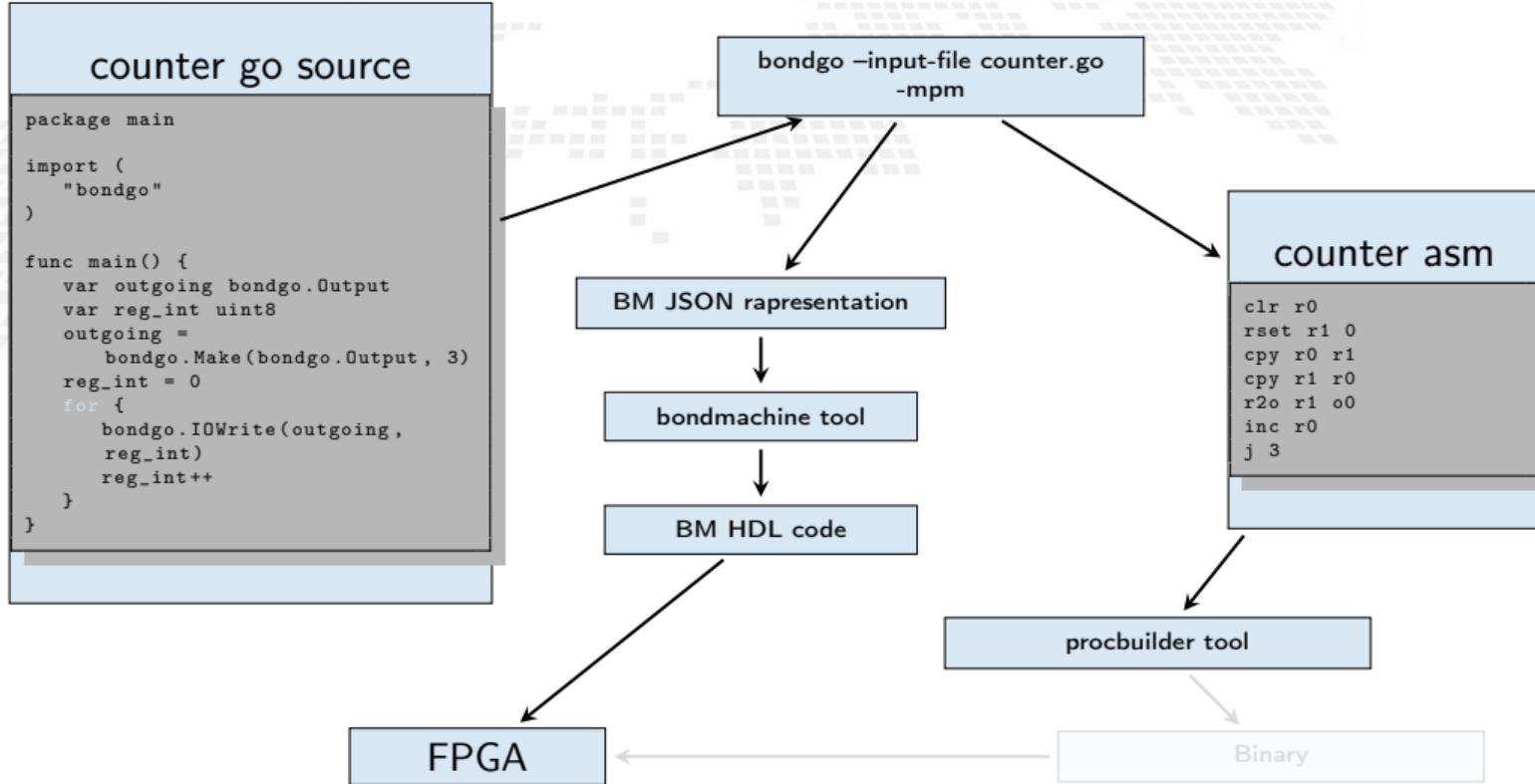
Bondgo workflow example



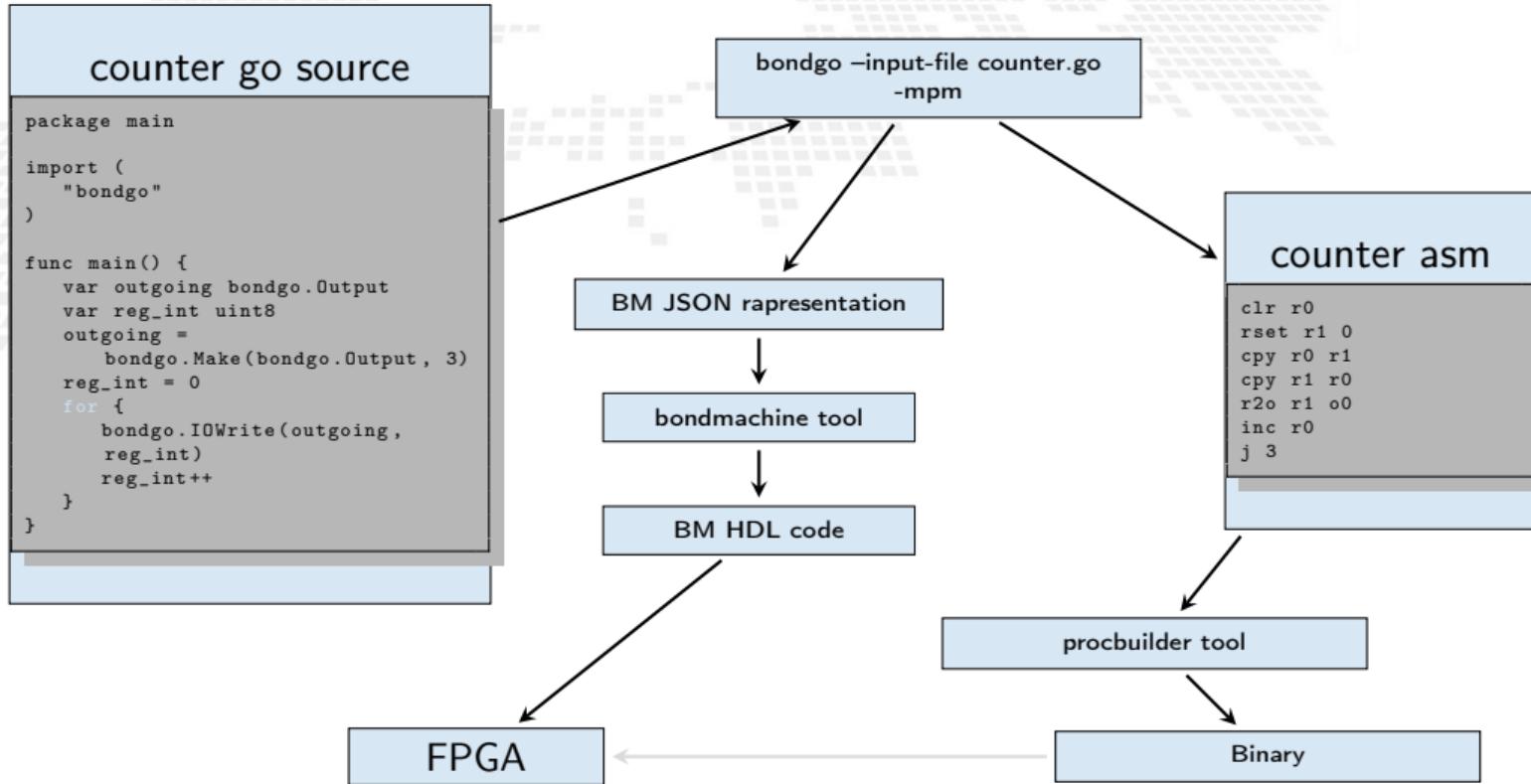
Bondgo workflow example



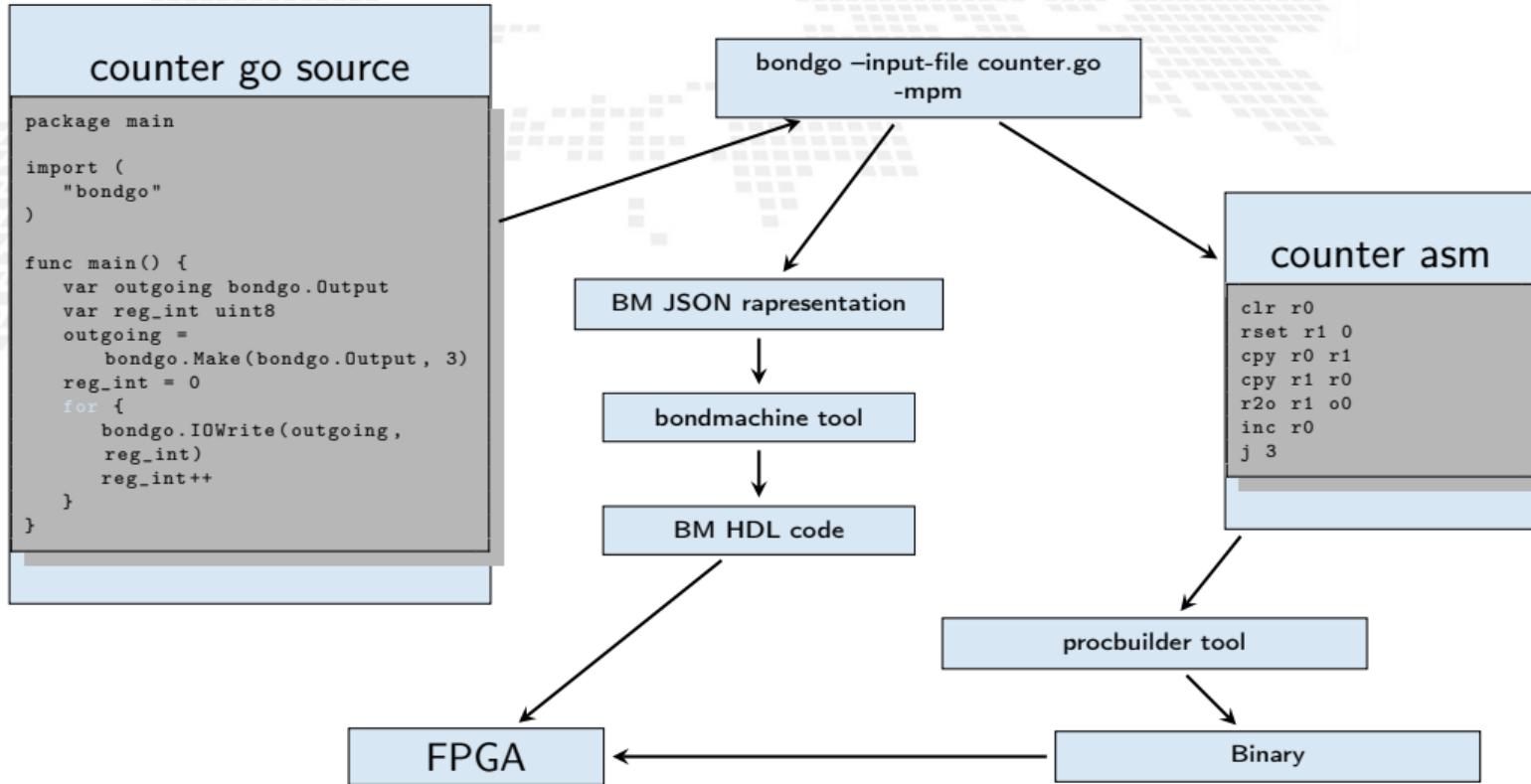
Bondgo workflow example



Bondgo workflow example



Bondgo workflow example



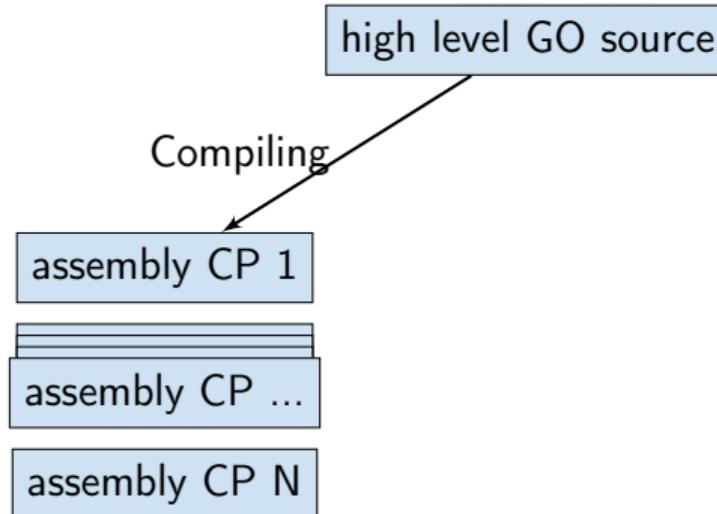
... *bondgo* may not only create the binaries, but also the CP architecture, and ...

... it can do even much more interesting things when compiling concurrent programs.

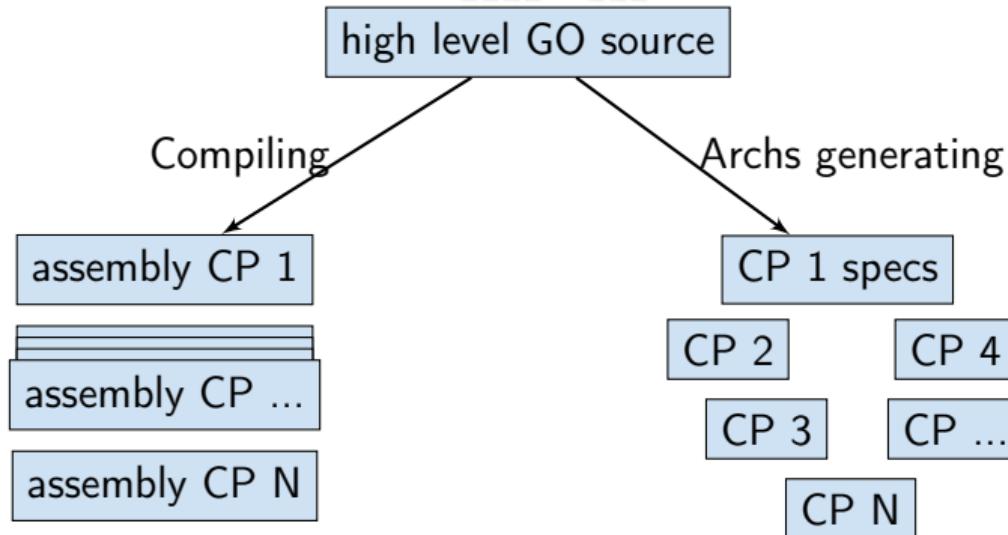
... it can do even much more interesting things when compiling concurrent programs.

high level GO source

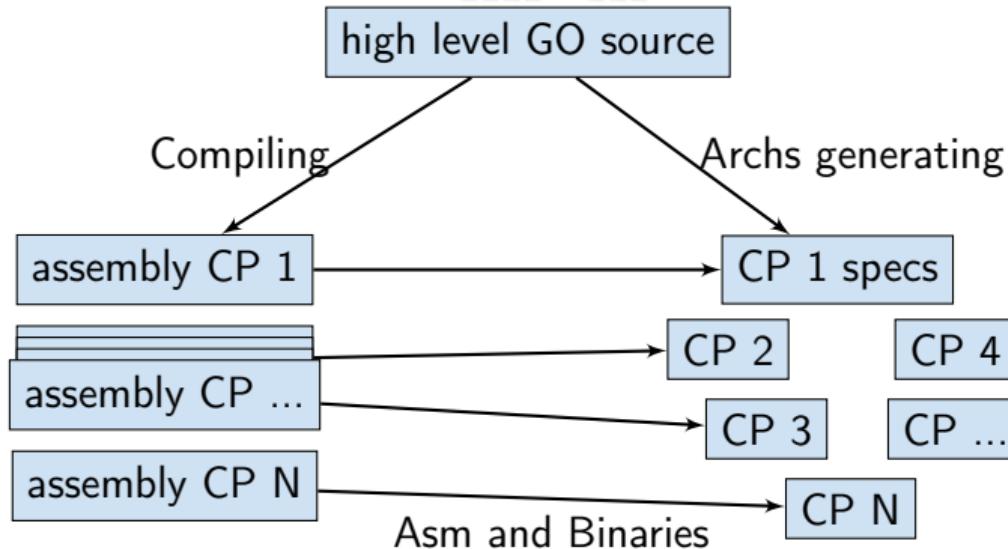
... it can do even much more interesting things when compiling concurrent programs.



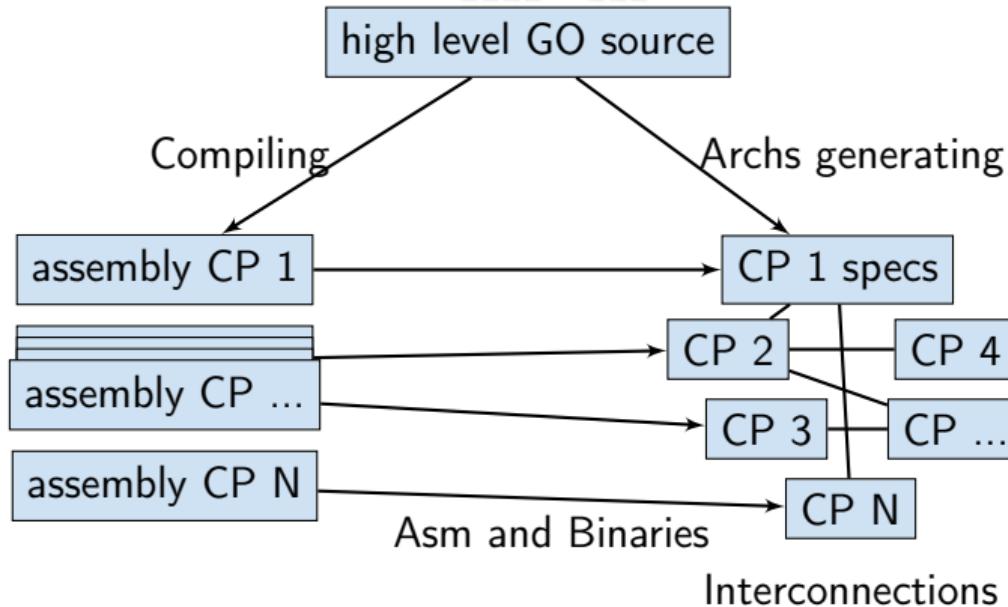
... it can do even much more interesting things when compiling concurrent programs.



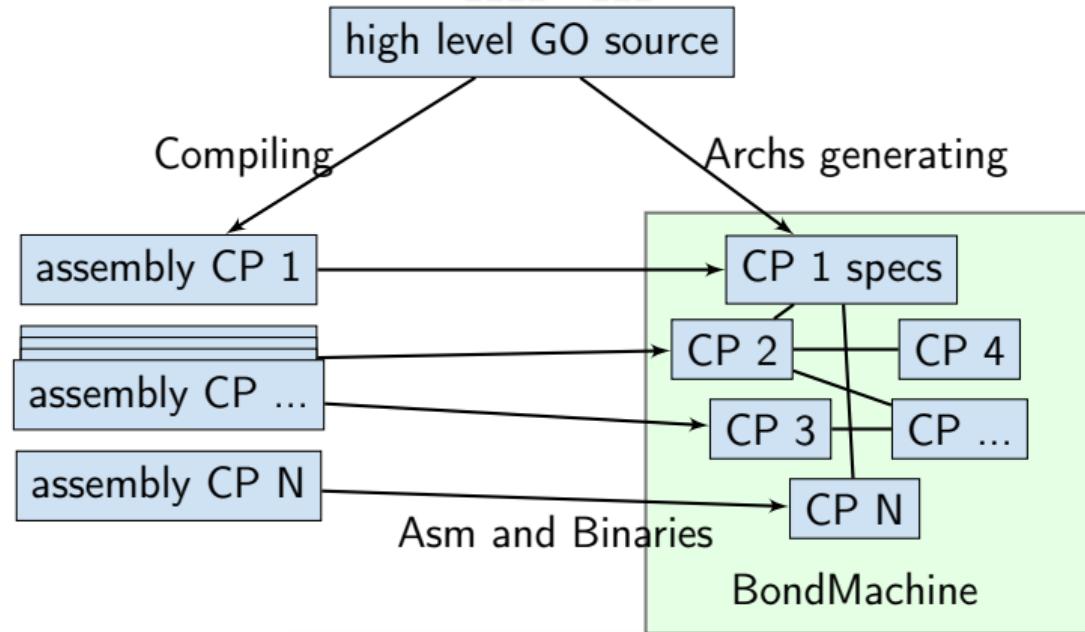
... it can do even much more interesting things when compiling concurrent programs.



... it can do even much more interesting things when compiling concurrent programs.



... it can do even much more interesting things when compiling concurrent programs.



Bondgo

A multi-core example

multi-core counter

```
package main

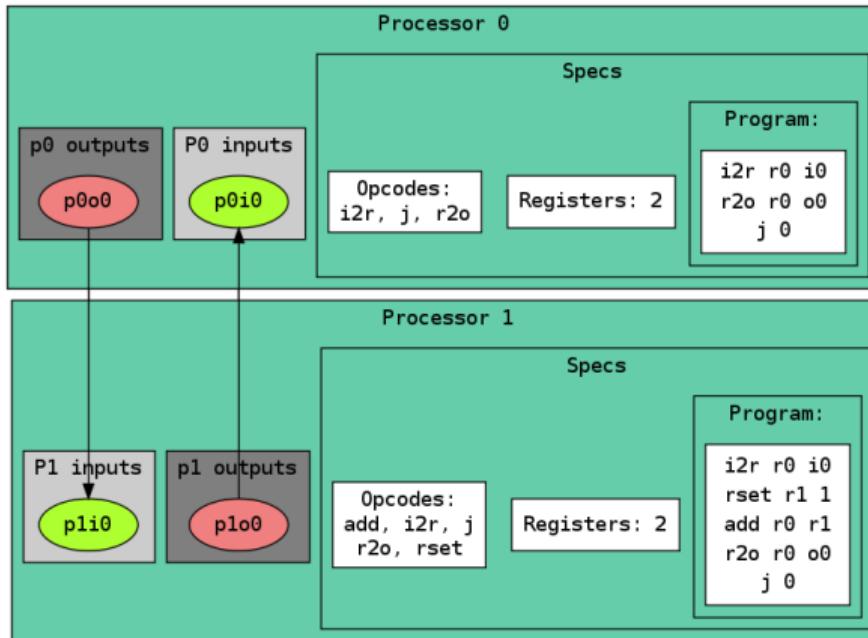
import (
    "bondgo"
)

func pong() {
    var in0 bondgo.Input
    var out0 bondgo.Output
    in0 = bondgo.Make(bondgo.Input, 3)
    out0 = bondgo.Make(bondgo.Output, 5)
    for {
        bondgo.IOWrite(out0, bondgo.IORRead(in0)+1)
    }
}

func main() {
    var in0 bondgo.Input
    var out0 bondgo.Output
    in0 = bondgo.Make(bondgo.Input, 5)
    out0 = bondgo.Make(bondgo.Output, 3)
device_0:
    go pong()
    for {
        bondgo.IOWrite(out0, bondgo.IORRead(in0))
    }
}
```

Bondgo

A multi-core example



Compiling Architectures

One of the most important result

The architecture creation is a part of the compilation process.

The BondMachine assembler *Basm* is the compiler complementary tools.

The BondMachine "fluid" nature gives the assembler some unique features:

- Support for template based assembly code
- Combining and rewriting fragments of assembly code
- Building hardware from assembly
- Software/Hardware rearrange capabilities

The BondMachine assembler *Basm* is the compiler complementary tools.

The BondMachine "fluid" nature gives the assembler some unique features:

- Support for template based assembly code
- Combining and rewriting fragments of assembly code
- Building hardware from assembly
- Software/Hardware rearrange capabilities

The BondMachine assembler *Basm* is the compiler complementary tools.

The BondMachine "fluid" nature gives the assembler some unique features:

- Support for template based assembly code
- Combining and rewriting fragments of assembly code
- Building hardware from assembly
- Software/Hardware rearrange capabilities

The BondMachine assembler *Basm* is the compiler complementary tools.

The BondMachine "fluid" nature gives the assembler some unique features:

- Support for template based assembly code
- Combining and rewriting fragments of assembly code
- Building hardware from assembly
- Software/Hardware rearrange capabilities

The BondMachine assembler *Basm* is the compiler complementary tools.

The BondMachine "fluid" nature gives the assembler some unique features:

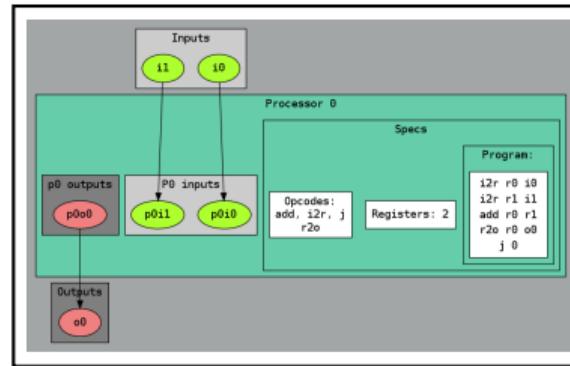
- Support for template based assembly code
- Combining and rewriting fragments of assembly code
- Building hardware from assembly
- Software/Hardware rearrange capabilities

Abstract Assembly

The Assembly language for the BM has been kept as independent as possible from the particular CP.

Given a specific piece of assembly code Bondgo has the ability to compute the “minimum CP” that can execute that code.

```
i2r r0 i0  
i2r r1 i1  
add r0 r1  
r2o r0 o0  
j 0
```



These are Building Blocks for complex BondMachines.

With these Building Blocks

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

- [Symbond](#), to handle mathematical expression.
- [Boolbond](#), to map boolean expression.
- [Matrixwork](#), to perform matrices operations.

[more about these tools](#)

With these Building Blocks

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

- **Symbond**, to handle mathematical expression.
- **Boolbond**, to map boolean expression.
- **Matrixwork**, to perform matrices operations.

[more about these tools](#)

With these Building Blocks

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

- **Symbond**, to handle mathematical expression.
- **Boolbond**, to map boolean expression.
- **Matrixwork**, to perform matrices operations.

[more about these tools](#)

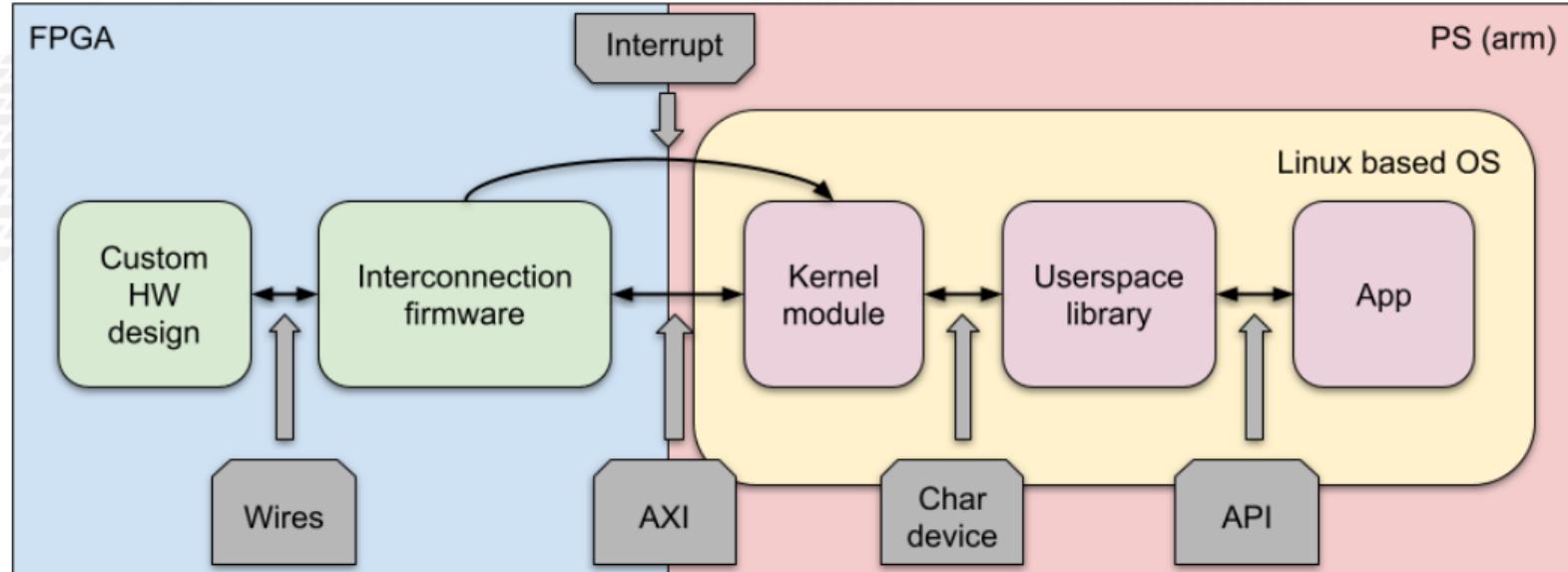
With these Building Blocks

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

- **Symbond**, to handle mathematical expression.
- **Boolbond**, to map boolean expression.
- **Matrixwork**, to perform matrices operations.

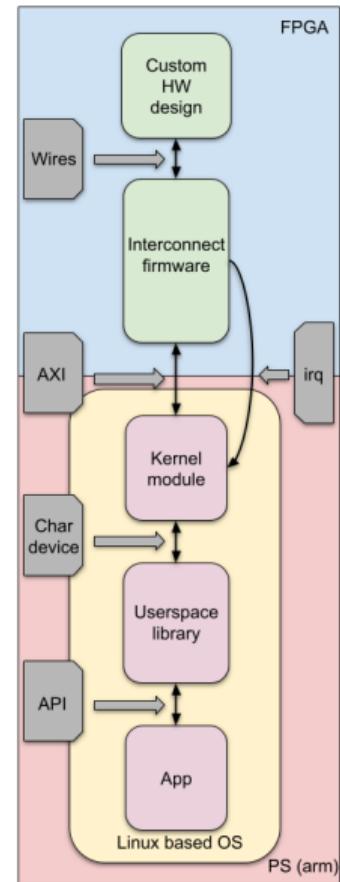
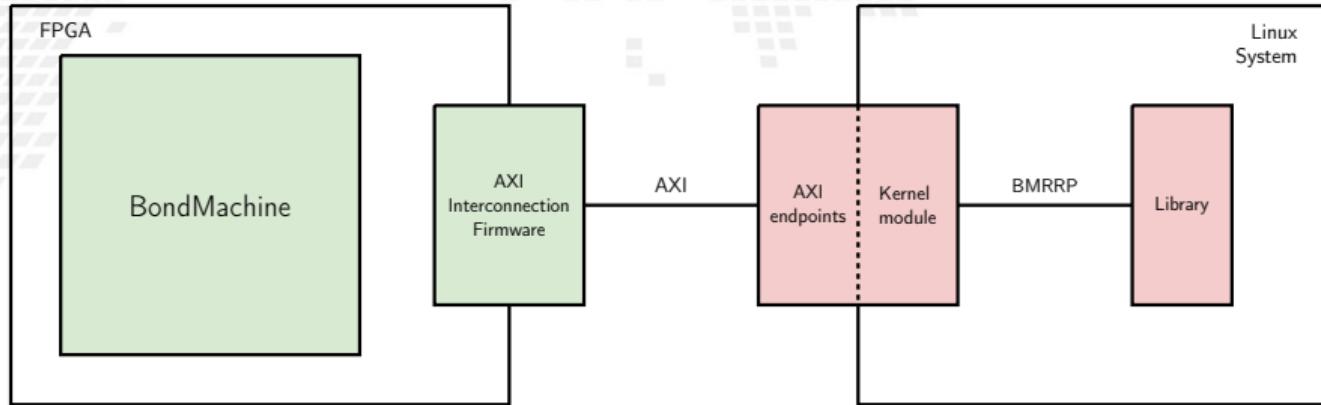
[more about these tools](#)

BondMachine as accelerators



Talk with details about how the accelerator is build

Accelerated Application



Misc

1 Introduction

- Challenges
- FPGA
- HDL workflow
- HLS Workflow
- Concepts
- Cloud

2 The BondMachine project

- Architectures handling
- Architectures molding
- Bondgo
- Basm
- API

3 Misc Project timeline

4 Machine Learning

- Train
- BondMachine creation
- Simulation
- Accelerator
- Benchmark

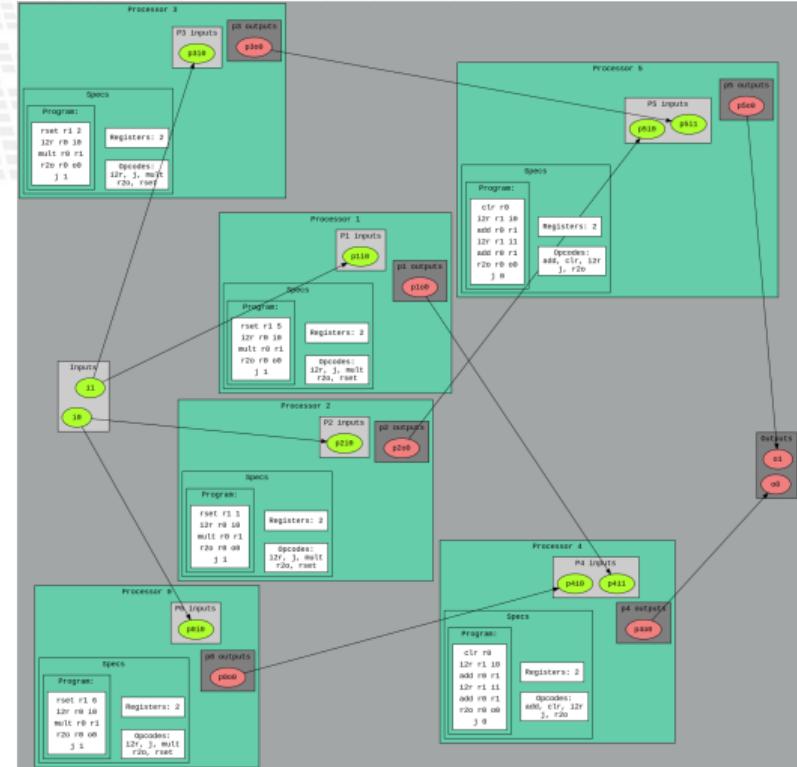
5 Optimizations

6 Conclusions and Future directions

- Conclusions
- Ongoing
- Future

BondMachine recap

- The BondMachine is a software ecosystem for the dynamical generation (from several HL types of origin) of computer architectures that can be synthesized of FPGA and
 - used as standalone devices,
 - as clustered devices,
 - and as firmware for computing accelerators.



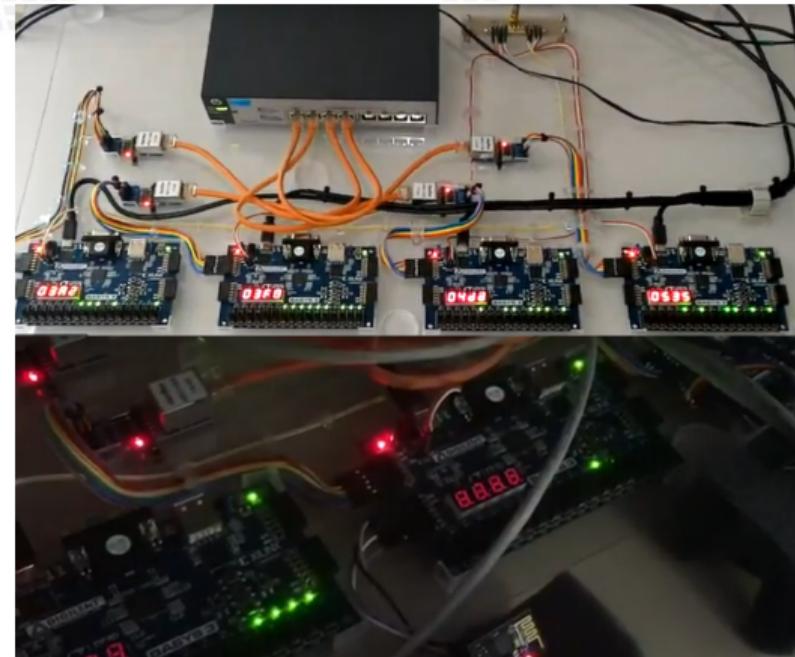
BondMachine recap

- The BondMachine is a software ecosystem for the dynamical generation (from several HL types of origin) of computer architectures that can be synthesized of FPGA and
- used as standalone devices,
- as clustered devices,
- and as firmware for computing accelerators.



BondMachine recap

- The BondMachine is a software ecosystem for the dynamical generation (from several HL types of origin) of computer architectures that can be synthesized of FPGA and
- used as standalone devices,
- as clustered devices,
- and as firmware for computing accelerators.



BondMachine recap

- The BondMachine is a software ecosystem for the dynamical generation (from several HL types of origin) of computer architectures that can be synthesized of FPGA and
- used as standalone devices,
- as clustered devices,
- and as firmware for computing accelerators.

Project timeline

- CCR 2015 First ideas, 2016 Poster, 2017 Talk

InnovateFPGA 2018 Iron Award Grand Final at Intel Campus (CA) USA

Invited lectures at: "Advanced Workshop on Modern FPGA Based Technology for Scientific Computing", ICTP 2019

- Invited lectures at: "NiPS Summer School 2019 – Architectures and Algorithms for Energy-Efficient IoT and HPC Applications"

- Golab 2018 talk and ISGC 2019 PoS

- Article published on Parallel Computing, Elsevier 2022

- PON PHD program

The BondMachine BM is a new computer architecture where many Connecting Processors (CPs) with different instruction set architecture (ISA) are connected together and share resources to form a heterogeneous interconnection fabric in a specific computational problem. These cores are implemented with the characteristic to be as minimal as possible and as simple as possible, and the capacity of solving problems rely mainly in how they are interconnected.

The BondMachine is designed to be a general purpose computer architecture, but it can also be used as a high specialized device. In order to satisfy and improve the power processing, the BondMachine is implemented by using the Field Programmable Gate Array (FPGA) chips, which are today's most powerful implementations of programmable hardware. However, the regular memory abstraction has been kept in order to let many well known tools and languages work on the architecture. This architecture can be used as general purpose computer architecture or as high specialized device perfectly suited to specific problems and flexible enough to be used in scenarios like Internet of Things, Intel® Cyber Physical Systems (CPS) and High Performance Computing (HPC).

Introduction

The BondMachine architecture

The BondMachine architecture consists of two groups of Connecting Processors and Shared Objects (SOs) build to implement a dedicated tasks. The main feature of this kind of architecture are the possibility to configure:

- the number of processor cores and their types;
- the number of shared objects and their types;
- the interconnection between processors;
- the number of SOs used by each processor.

Connecting Processor

The CP is the computing core of the BondMachine. Several CPs can be configured in parallel to implement different tasks within the BondMachine. They have a local memory and a number, instruction set, registers with respect to the other areas.

Shared Object

Any kind of component that can be shared among CPs. Shared Objects increase the processing power of the architecture and the BM improving the high-speed communication between tasks running on separate CPs.

Software Tools

The complexity of programming the BondMachine architecture is managed by the BondBuilder tool. It is a graphical interface that allows to build a specific application as function of the task, modify the created architecture, and generate assembly code to check the functionality with the aim to generate the Register Transfer Level (RTL) code for FPGA devices.

Processor Builder selects the CP specifics, assembles and disassembles, saves on disk or JTAG, emulates and creates the RTL code.

CodeBuilder takes the CP and SOs together to generate bootlegges, saves them on disk or JTAG, and recompiles the RTL code.

Arch-compiler compiles the C language to generate the CP assembly code and to create the optimized architecture to run that code.

Hardware implementation

The RTL code automatically generated by the builder is synthesized for the target FPGA. The BondBuilder generates a testbench to evaluate the performances of the architecture, logic resources, power consumption and maximum clock frequency.

The architecture consists of a shared memory shared by two CPs. This basic element has been replicated by varying the number of shared objects and the logic resources used by each architecture increase linearly with the number of shared objects.

The FPGA can run up to 128 CPs with a clock frequency of 250 MHz and a power consumption of 0.33 W.

The BondMachine architecture has been compared with the Go chip. A Go chip has been used to test the architecture to compare the two.

Case study

This example is a simple scenario with two CPs that have to exchange data. One of the CPs is a BondProcessor and sends the data through the Channel, the second one is a BondSharedObject and receives the data from the first one. When the Go source code is compiled the BondArch-compiler produces the assembly code for the two CPs. In this case, only the needed codes are produced, different ISA for it and the assembly code to run on it.

package main
import "fmt"
type BondProcessor struct {
 bus int
 mem int
 reg int
 busWait int
 busWaitCount int
 busWaitIndex int
}
type BondSharedObject struct {
 bus int
 mem int
 reg int
 busWait int
 busWaitCount int
 busWaitIndex int
}
func main() {
 cp := &BondProcessor{bus: 100, mem: 100, reg: 100, busWait: 100, busWaitCount: 100, busWaitIndex: 100}
 so := &BondSharedObject{bus: 100, mem: 100, reg: 100, busWait: 100, busWaitCount: 100, busWaitIndex: 100}
 cp.mem = 100
 cp.reg = 100
 cp.busWait = 100
 cp.busWaitCount = 100
 cp.busWaitIndex = 100
 so.mem = 100
 so.reg = 100
 so.busWait = 100
 so.busWaitCount = 100
 so.busWaitIndex = 100
}

Evolutionary BondMachine

Some particular problem may need a complex network of CPs and Shared Objects to be solved especially regarding the internal interconnections and the feature to have processor of different types.

For this reason, the BondMachine is a good candidate to be used in the field of Evolutionary Computing frameworks to explore the possibility of **evolving the architectures** to solve a specific problem.

Conclusion

The BondMachine is a new kind of computing device made possible in practice only by the emerging of new re-programmable hardware technologies such as FPGAs. The BondMachine is a general purpose computer architecture, but it can also be used as a high specialized device. Moreover the BondMachine architecture is high specialized device perfectly suited to specific problems and flexible enough to be used in many scenarios finding the better topology of interconnection's processors.

Workshop di CCR - La Meloria, 18-20 Maggio 2018 - Contact person: Mirko.Mariotti@unipg.it

Project timeline

- CCR 2015 First ideas, 2016 Poster, 2017 Talk
- InnovateFPGA 2018 Iron Award, Grand Final at Intel Campus (CA) USA
 - Invited lectures at: "Advanced Workshop on Modern FPGA Based Technology for Scientific Computing", ICTP 2019
 - Invited lectures at: "NiPS Summer School 2019 – Architectures and Algorithms for Energy-Efficient IoT and HPC Applications"
- Golab 2018 talk and ISGC 2019 PoS
- Article published on Parallel Computing, Elsevier 2022
- PON PHD program



Project timeline

- CCR 2015 First ideas, 2016 Poster, 2017 Talk
- InnovateFPGA 2018 Iron Award, Grand Final at Intel Campus (CA) USA
- Invited lectures at: "Advanced Workshop on Modern FPGA Based Technology for Scientific Computing", ICTP 2019
- Invited lectures at: "NiPS Summer School 2019 – Architectures and Algorithms for Energy-Efficient IoT and HPC Applications"
- Golab 2018 talk and ISGC 2019 PoS
- Article published on Parallel Computing, Elsevier 2022
- PON PHD program



Project timeline

- CCR 2015 First ideas, 2016 Poster, 2017 Talk
- InnovateFPGA 2018 Iron Award, Grand Final at Intel Campus (CA) USA
- Invited lectures at: "Advanced Workshop on Modern FPGA Based Technology for Scientific Computing", ICTP 2019
- Invited lectures at: "NiPS Summer School 2019 – Architectures and Algorithms for Energy-Efficient IoT and HPC Applications"
- Golab 2018 talk and ISGC 2019 PoS
- Article published on Parallel Computing, Elsevier 2022
- PON PHD program

The BondMachine Toolkit
Enabling Machine Learning on FPGA

Mirko Mariotti

Department of Physics and Geology - University of Perugia
INFN Perugia

NiPS Summer School 2019
Architectures and Algorithms for Energy-Efficient IoT and HPC
Applications
3-6 September 2019 - Perugia



Project timeline

- CCR 2015 First ideas, 2016 Poster, 2017 Talk
- InnovateFPGA 2018 Iron Award, Grand Final at Intel Campus (CA) USA
- Invited lectures at: "Advanced Workshop on Modern FPGA Based Technology for Scientific Computing", ICTP 2019
- Invited lectures at: "NiPS Summer School 2019 – Architectures and Algorithms for Energy-Efficient IoT and HPC Applications"
- Golab 2018 talk and ISGC 2019 PoS
- Article published on Parallel Computing, Elsevier 2022
- PON PHD program



Project timeline

- CCR 2015 First ideas, 2016 Poster, 2017 Talk
- InnovateFPGA 2018 Iron Award, Grand Final at Intel Campus (CA) USA
- Invited lectures at: "Advanced Workshop on Modern FPGA Based Technology for Scientific Computing", ICTP 2019
- Invited lectures at: "NiPS Summer School 2019 – Architectures and Algorithms for Energy-Efficient IoT and HPC Applications"
- Golab 2018 talk and ISGC 2019 PoS
- Article published on Parallel Computing, Elsevier 2022
- PON PHD program



Parallel Computing
Volume 109, March 2022, 102873



The BondMachine, a moldable computer architecture

Mirko Mariotti ^{a, b} , Daniel Magalotti ^b, Daniele Spiga ^b, Loriano Storchi ^c

Show more ▾

+ Add to Mendeley Share Cite

<https://doi.org/10.1016/j.parco.2021.102873>

[Get rights and content](#)

Highlights

- Co-design HW/SW of domain specific architectures via the modern GO language.
- Design of essential processors where only needed components are implemented.
- Creation of heterogeneous processor systems distributed over multiple fabrics.

Project timeline

- CCR 2015 First ideas, 2016 Poster, 2017 Talk
- InnovateFPGA 2018 Iron Award, Grand Final at Intel Campus (CA) USA
- Invited lectures at: "Advanced Workshop on Modern FPGA Based Technology for Scientific Computing", ICTP 2019
- Invited lectures at: "NiPS Summer School 2019 – Architectures and Algorithms for Energy-Efficient IoT and HPC Applications"
- Golab 2018 talk and ISGC 2019 PoS
- Article published on Parallel Computing, Elsevier 2022
- PON PHD program

Machine Learning

1 Introduction

- Challenges
- FPGA
- HDL workflow
- HLS Workflow
- Concepts
- Cloud

2 The BondMachine project

- Architectures handling
- Architectures molding
- Bondgo
- Basm
- API

3 Misc

Project timeline

4 Machine Learning

- Train
- BondMachine creation**
- Simulation
- Accelerator
- Benchmark

5 Optimizations

6 Conclusions and Future directions

- Conclusions
- Ongoing
- Future

Machine Learning with BondMachine

Architectures with multiple interconnected processors like the ones produced by the BondMachine Toolkit are a perfect fit for Neural Networks and Computational Graphs.

Several ways to map these structures to BondMachine have been developed:

- A native Neural Network library
- A Tensorflow to BondMachine translator
- An NNEF based BondMachine composer

Machine Learning with BondMachine

Architectures with multiple interconnected processors like the ones produced by the BondMachine Toolkit are a perfect fit for Neural Networks and Computational Graphs.

Several ways to map these structures to BondMachine have been developed:

- A native Neural Network library
- A Tensorflow to BondMachine translator
- An NNEF based BondMachine composer

Machine Learning with BondMachine

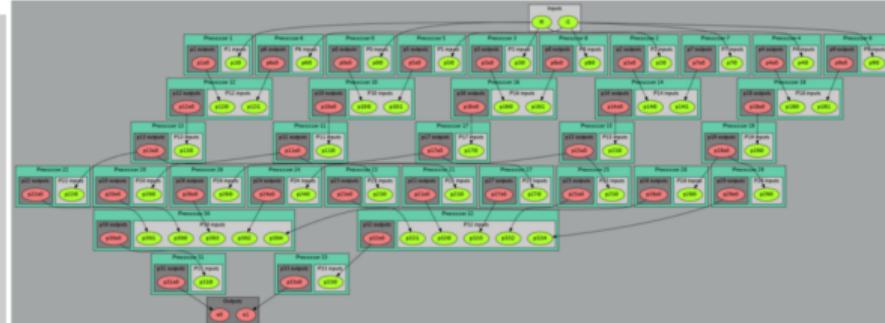
Native Neural Network library

The tool *neuralbond* allow the creation of BM-based neural chips from an API go interface.

- Neurons are converted to BondMachine connecting processors.
- Tensors are mapped to CP connections.

```
layers := []int{2, 5, 2}
weights := make([][]neuralbond.Weight, 0)

if *save_bondmachine != "" {
    if mymachine, ok :=
        neuralbond.Build_MLP(layers, weights); ok
    == nil {
        if _, err := os.Stat(*save_bondmachine);
        os.IsNotExist(err) {
            f, err := os.Create(*save_bondmachine)
            check(err)
            defer f.Close()
        }
    }
}
```

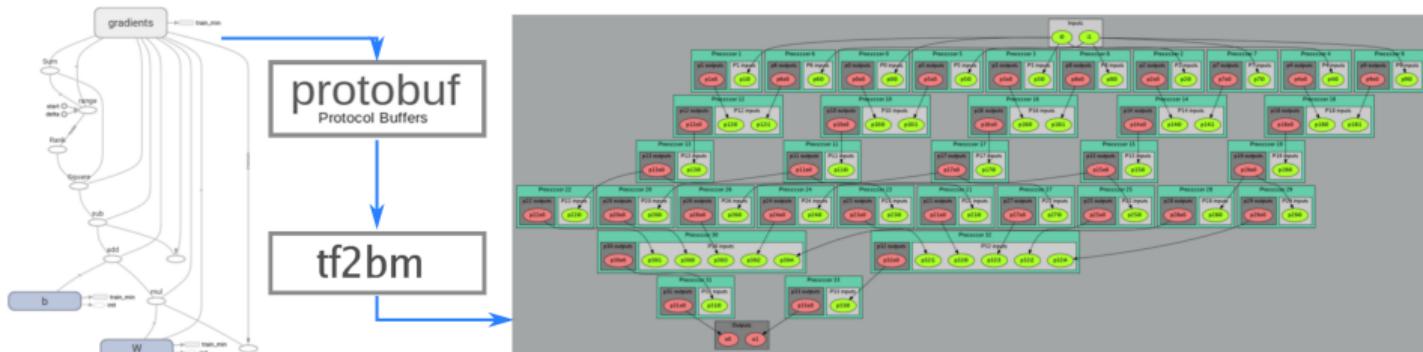


TensorFlow™ to Bondmachine

tf2bm

TensorFlow™ is an open source software library for numerical computation using data flow graphs.

Graphs can be converted to BondMachines with the **tf2bm** tool.



Machine Learning with BondMachine

NNEF Composer

Neural Network Exchange Format (NNEF) is a standard from Khronos Group to enable the easy transfer of trained networks among frameworks, inference engines and devices

The NNEF BM tool approach is to descent NNEF models and build BondMachine multi-core accordingly

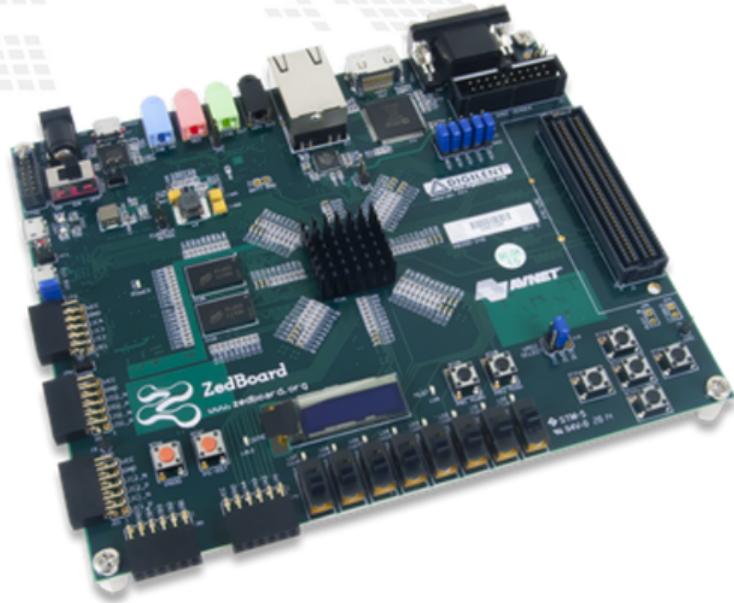
This approach has several advantages over the previous:

- It is not limited to a single framework
- NNEF is a textual file, so no complex operations are needed to read models

Specs

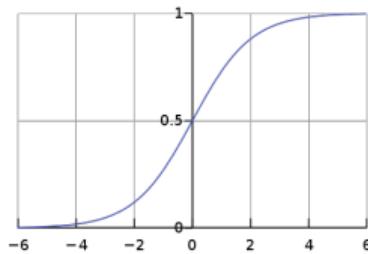
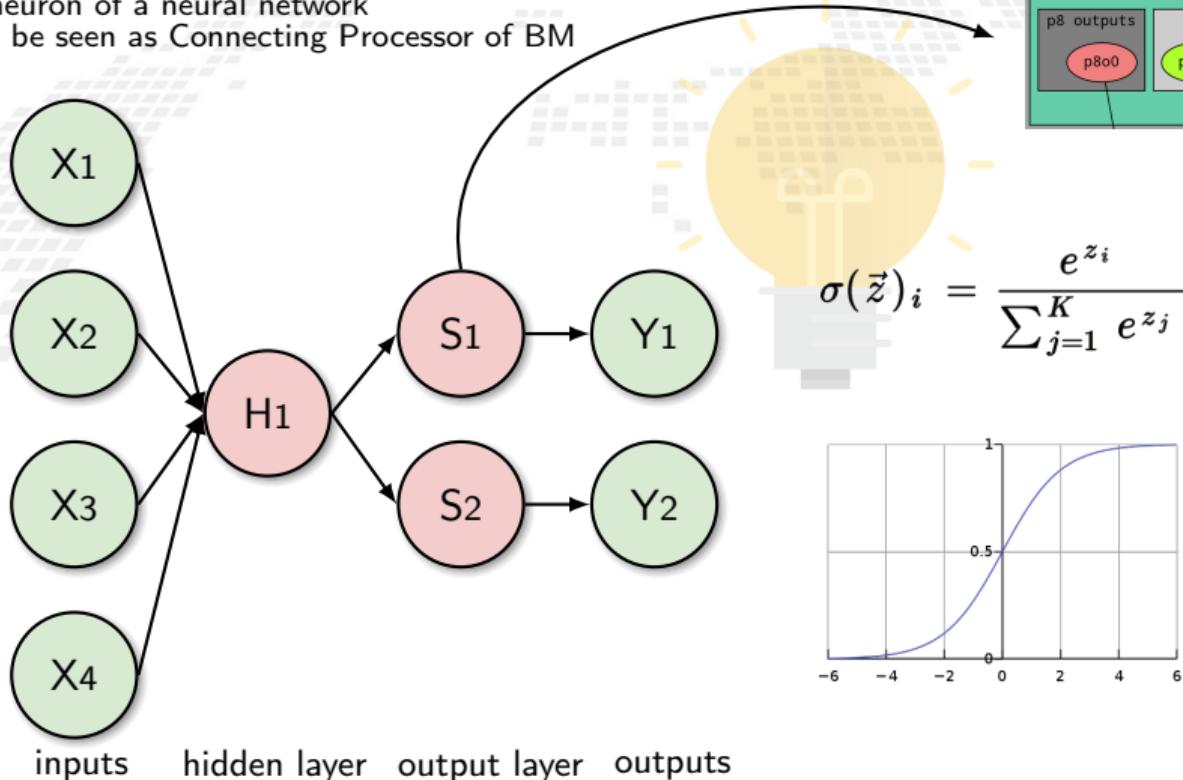
FPGA

- Digilent Zedboard
- Soc: Zynq XC7Z020-CLG484-1
- 512 MB DDR3
- Vivado 2020.2
- 100MHz
- PYNQ 2.6 (custom build)



BM inference: A first tentative idea

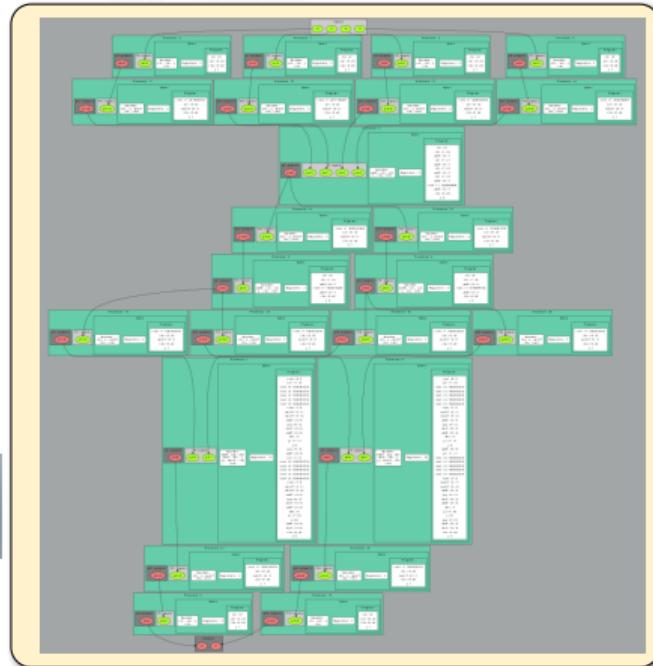
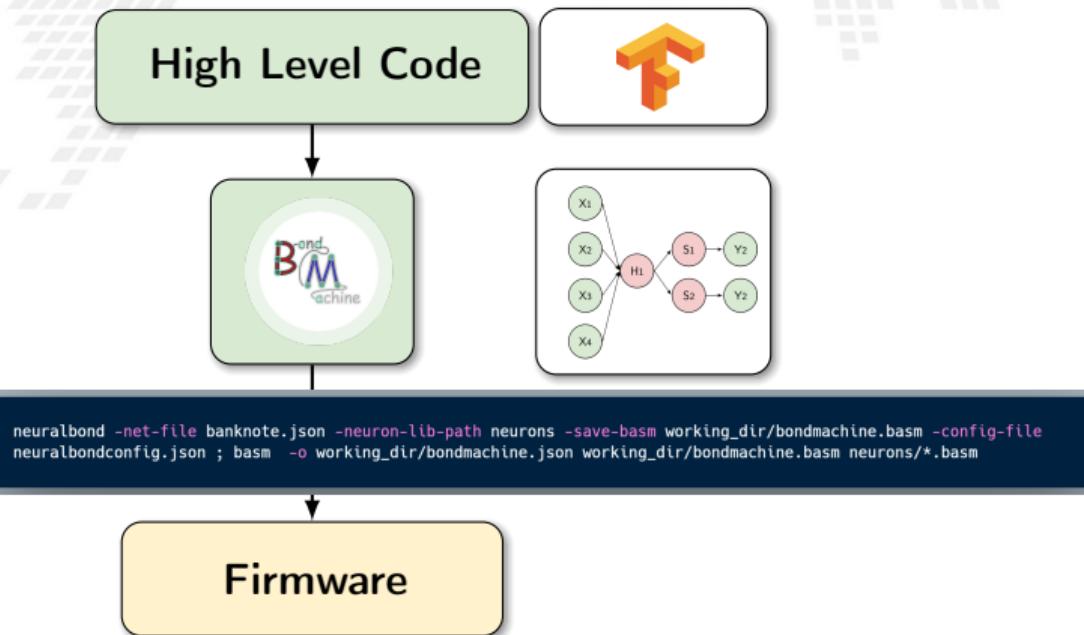
A neuron of a neural network
can be seen as Connecting Processor of BM



```
%section softmax .romtext iomode:sync
entry _start      ; Entry point
_start:
    mov r8, 0f1.0
    {{range $y := intRange "0" $.Params.inputs}}
    {{printf "%d r1,%d\n" $y}}
        mov r0, 0f1.0
        mov r2, 0f1.0
        mov r3, 0f1.0
        mov r4, 0f1.0
        mov r5, 0f1.0
        mov r7, {{$Params.expprec}}
        multf r2, r1
        multf r3, r4
        addf r4, r5
        mov r6, r2
        divf r6, r3
        addf r0, r6
        dec r7
        jz r7,exit{{printf "%d" $y}}
        j loop{{printf "%d" $y}}
    exit{{printf "%d" $y}}:
        {$z := atoi $.Params.pos}
        {{if eq $y $z}}
            mov r9, r0
    %endsection
```

From idea to implementation

Starting from High Level Code, a NN model trained with **TensorFlow** and exported in a standard interpreted by **neuralbond** that converts nodes and weights of the network into a set of heterogeneous processors.



A first test

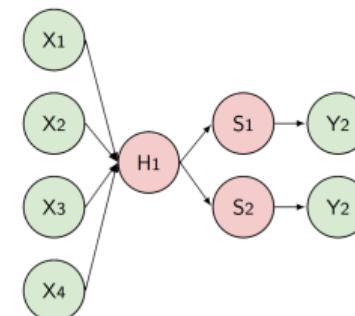
Dataset info:

- **Dataset name:** Banknote Authentication
- **Description:** Dataset on the distinction between genuine and counterfeit banknotes. The data was extracted from images taken from genuine and fake banknote-like samples.
- **N. features:** 4
- **Classification:** binary
- **Samples:** 1097

Neural network info:

- **Class:** Multilayer perceptron fully connected
- **Layers:**
 - ① An hidden layer with 1 **linear** neuron
 - ② One output layer with 2 **softmax** neurons

Graphic representation:



BM demo

NN demo N.1: Train

Goals are:

- To train the simple model and prepare it from the BM

Reference notebook: banknote-train.ipynb

BM demo

NN demo N.1 outcome

The outcome of this first part of the demo are three files:

- `sample.csv` is a test dataset that will be used to feed the inferences of both: the BM hardware and the BM simulation
- `sw.csv` is the software predictions over that dataset and will be used to check the BM inference probabilities and predictions
- `modelBM.json` is the trained network that will use as BM source in the next demo

BM demo

NN demo N.2: BondMachine creation

Goals are:

- Use modelBM created on the previous step to create a BondMachine

Reference notebook: proj_zedboard_ml_creation /notebook.ipynb

BM demo

NN demo N.2 outcome

The outcome of this second part of the demo are:

- `bondmachine.json`, a representation of the generated abstract machine
- All the HDL files needed to build the firmware for the given board

BM demo

NN demo N.3: BondMachine simulation

Goals are:

- Simulate the test dataset with the BondMachine simulator
- Compare the results with the keras prediction

Reference notebook: proj_zedboard_ml_simbatch /notebook.ipynb

Simulation

NN demo N.3 outcome

The outcome of this third part of the demo is:

- `simbatchoutput.csv`, a simulated CSV files containing the output probabilities and the prediction

BM demo

NN demo N.4: Accelerator creation

Goals are:

- Create the accelerator firmware

Reference notebook: proj_zedboard_ml_accelerator /notebook.ipynb

Accelerator creation

NN demo N.4 outcome



Notebook on the board - predictions and correctness

```
In [138]: from pynq import Overlay
from pynq import MMIO
import os
import numpy as np
import struct
import time

In [149]: # SETTING
firmware_name = "proj_dsa2405_saxalibone_expanded"
firmware_name = project_name+".bit"
n_input = 8
n_output = 4
benchcore = True

In [153]: # LOAD OVERLAY
overlay = Overlay(os.getcwd()+'/*'+firmware_name)

In [154]: # GET MEMORY ADDRESS OF IP
bm_starting_address = overlay.ip_dict["bondmachinip_0"]["phys_addr"]
print("Starting memory address of BondMachine IP is (in dec): ", bm_starting_address)
print("Starting memory address of BondMachine IP is (in hex): ", hex(bm_starting_address))

Starting memory address of BondMachine IP is (in dec): 1196656184
Starting memory address of BondMachine IP is (in hex): 0x45C039000

In [155]: # GET THE OBJECT NECESSARY TO INTERACT WITH AN IP
spio = MMIO(bm_starting_address, 128)

In [156]: # LOAD BENCHMARK TESTSET
X_test = np.load('benchmark-authentication_X_test.npy')
Y_test = np.load('benchmark-authentication_Y_test.npy')
# print(X_test[0])
# print(Y_test[0])
X_test = X_test[:10]
Y_test = Y_test[:10]
print("Example of first two input: ", X_test[:2])
print("Example of first two output: ", Y_test[:2])

Example of first two input: [[ 0.398803742  0.76609796 -0.39003127 -0.58781728]]
Example of first two output: [[ 1.  0.]]

In [157]: # IN THIS CASE I WANT TO SEND ONLY THE FIRST X INPUT SAMPLE
idx = 0
results_to_dump = []
for sample in X_test:
    offset = idx * 4
    for feature in list(sample):
        binToFloat = get_binary_from_float(feature)
        decValue = binToFloat[0]
        spio.write_mioffset(decValue) # WRITE THE FEATURE TO THE CORRESPONDING INPUT
        offset += offset + 4 # 4 BYTE = 32 BIT
    time.sleep(0.001)
    out = np.asarray(read_output())
    print("out", idx, " --> classification: ", np.argmax(out[0:2]))
    classification = np.argmax(out[0:2])
    if (classification == Y_test[idx]):
        results_to_dump.append([out[0], out[1], classification, out[2]])
    else:
        results_to_dump.append([out[0], out[1], classification])
    idx += idx + 1
    break
print(results_to_dump)

[[0.4350703135708326, 0.310423088153067913, 0, 0.662241, [0.3034991510201108, 0.42510313377793835, 0, 0.662241], [0.4350703135708326, 0.310423088153067913, 0, 0.662241, [0.3034991510201108, 0.42510313377793835, 0, 0.662241], 0.30418446201381496, 0, 0.728101, [0.3789190751220703, 0.21908080238410699, 0, 0.568041], 0.45921049355460713, 0.2078156043539461, 0.45921049355460713, 0.3577560564291, 0, 0.387156100357058, 0, 0.226701, [0.665764155023923, 0.31422338737962921, 0, 0.4591, [0.665764155023923, 0.31422338737962921, 0, 0.4591, [0.6377614832877582, 0, 0.36235718894179, 0, 0.142741, 0.473555481956449829, 0, 0.3344518804550171, 0, 0.195764]], 0.30418446201381496, 0, 0.728101, [0.3789190751220703, 0.21908080238410699, 0, 0.568041], 0.45921049355460713, 0.2078156043539461, 0.45921049355460713, 0.3577560564291, 0, 0.387156100357058, 0, 0.226701, [0.665764155023923, 0.31422338737962921, 0, 0.4591, [0.665764155023923, 0.31422338737962921, 0, 0.4591, [0.6377614832877582, 0, 0.36235718894179, 0, 0.142741, 0.473555481956449829, 0, 0.3344518804550171, 0, 0.195764]]]
```

Thanks to PYNQ we can easily load the bitstream and program the FPGA in real time.

With their APIs we interact with the memory addresses of the BM IP to send data into the inputs and read the outputs (not using BM kernel module)

Dump output results for future analysis

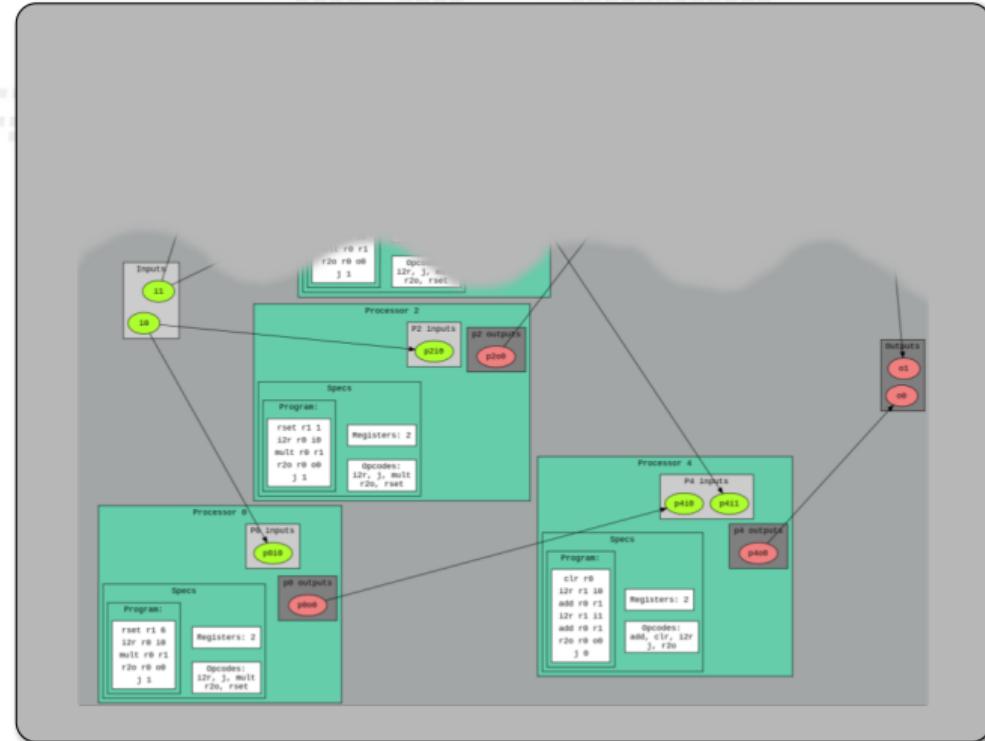
Open the notebook

Benchcore

Benchmark an IP is not an easy task.

Fortunately we have a custom design and an FPGA.

We can put the benchmarks tool inside the accelerator.

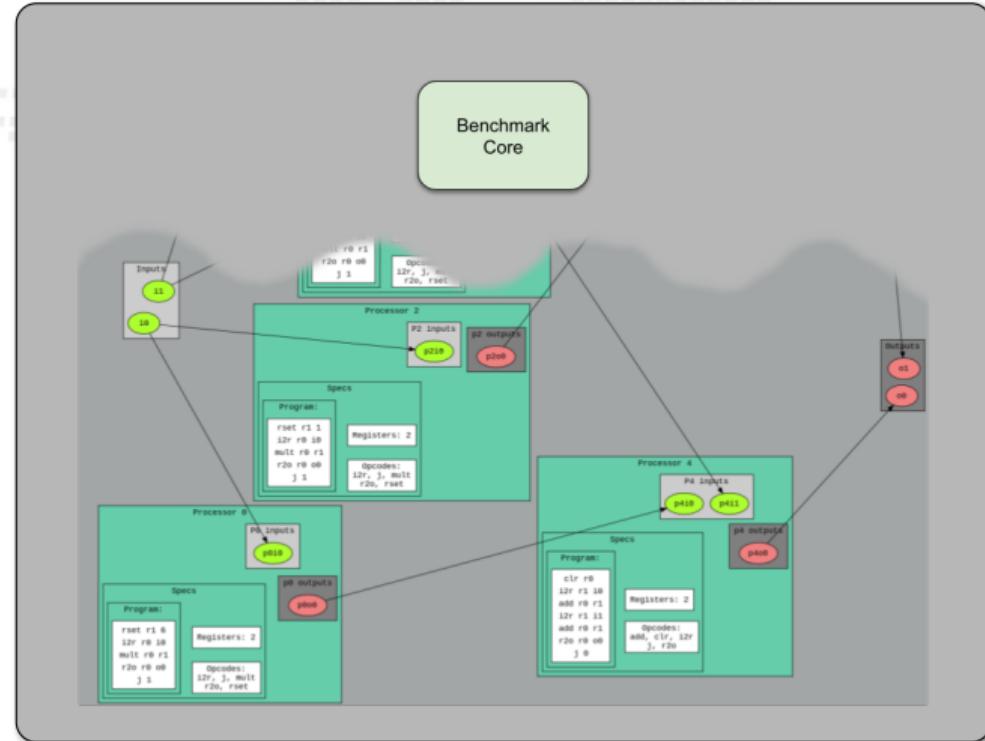


Benchcore

Benchmark an IP is not an easy task.

Fortunately we have a custom design and an FPGA.

We can put the benchmarks tool inside the accelerator.

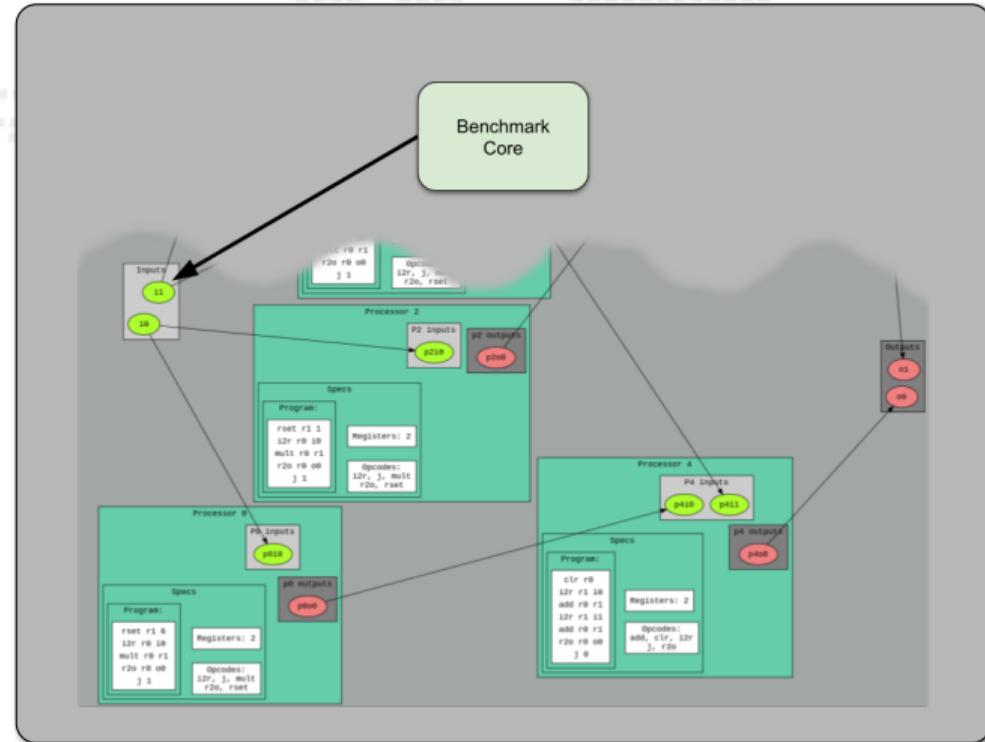


Benchcore

Benchmark an IP is not an easy task.

Fortunately we have a custom design and an FPGA.

We can put the benchmarks tool inside the accelerator.

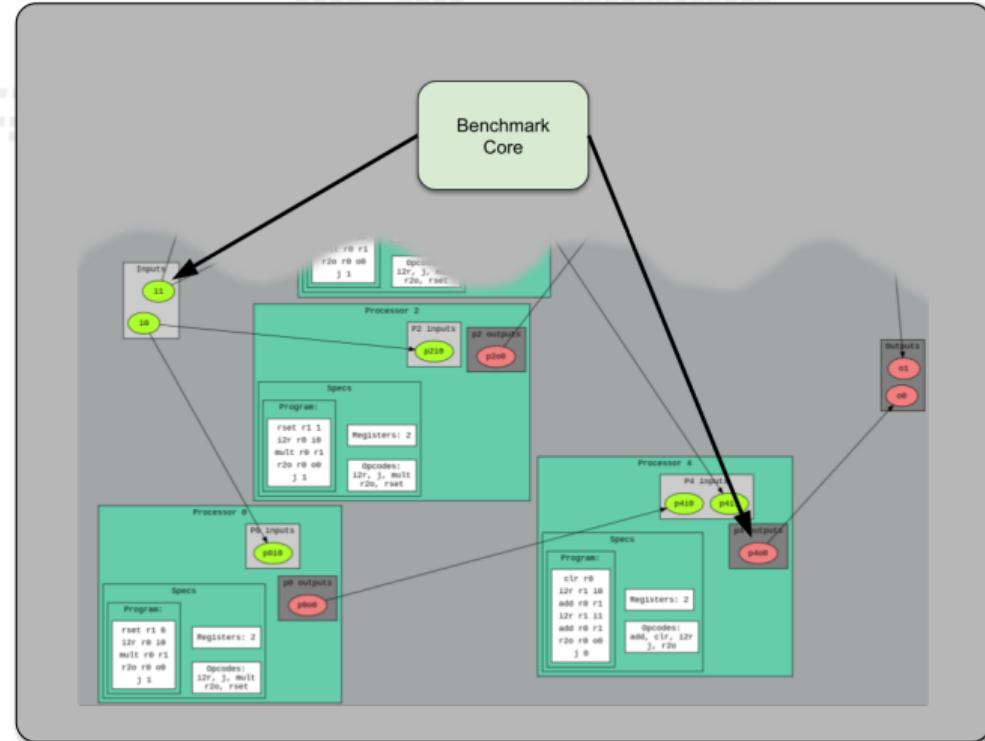


Benchcore

Benchmark an IP is not an easy task.

Fortunately we have a custom design and an FPGA.

We can put the benchmarks tool inside the accelerator.

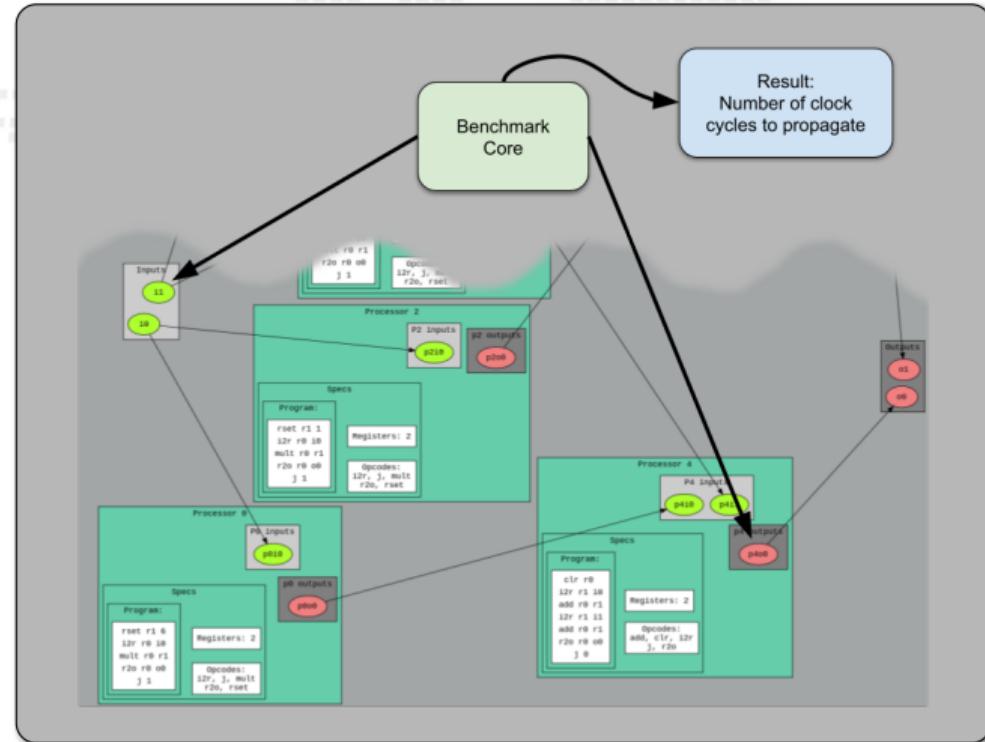


Benchcore

Benchmark an IP is not an easy task.

Fortunately we have a custom design and an FPGA.

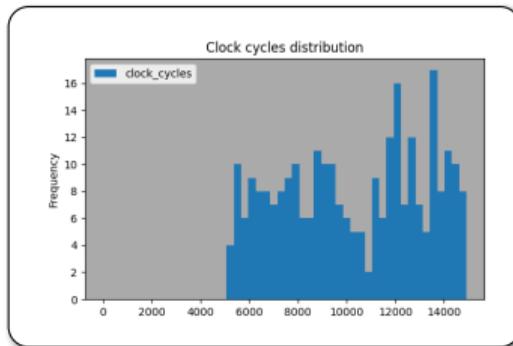
We can put the benchmarks tool inside the accelerator.



Inference evaluation

Evaluation metrics used:

- **Inference speed:** time taken to predict a sample i.e. time between the arrival of the input and the change of the output measured with the **benchcore**;
- **Resource usage:** luts and registers in use;
- **Accuracy:** as the average percentage of error on probabilities.



- σ : 2875.94
- Mean: 10268.45
- Latency: 102.68 μ s

Resource usage

resource	value	occupancy
regs	15122	28.42%
luts	11192	10.51%

Analysis notebook

Another notebook is used to compare runs from different accelerators.

Software

prob0	prob1	class
0.6895	0.3104	0
0.5748	0.4251	0
0.4009	0.5990	1

BondMachine

prob0	prob1	class
0.6895	0.3104	0
0.5748	0.4251	0
0.4009	0.5990	1

The output of the bm corresponds to the software output

Reference notebook: [analysis /zedboard_banknote /analysis.ipynb](#)

Optimizations

1 Introduction

- Challenges
- FPGA
- HDL workflow
- HLS Workflow
- Concepts
- Cloud

2 The BondMachine project

- Architectures handling
- Architectures molding
- Bondgo
- Basm
- API

3 Misc

Project timeline

4 Machine Learning

- Train
- BondMachine creation
- Simulation
- Accelerator
- Benchmark

5 Optimizations

6 Conclusions and Future directions

- Conclusions
- Ongoing
- Future

A first example of optimization

Remember the softmax function?

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^N e^{z_j}}$$

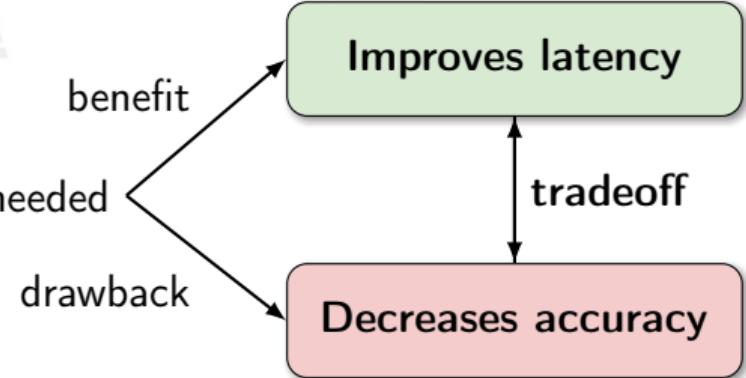
$$e^x = \sum_{l=0}^K \frac{x^l}{l!}$$

```
%section softmax .romtext iomode:sync
    entry _start      ; Entry point
_start:
    mov r8, 0f0.0
{{range $y := intRange "0" .Params.inputs}}
{{printf "i2r r1,i%b\n" $y}}
    mov     r0, 0f1.0
    mov     r2, 0f1.0
    mov     r3, 0f1.0
    mov     r4, 0f1.0
    mov     r5, 0f1.0
    mov     r7, {{$Params.expprec}}
loop{{printf "%d" $y}}:
    multf  r2, r1
    multf  r3, r4
    addf   r4, r5
    mov    r6, r2
    divf   r6, r3
    addf   r0, r6
    dec    r7
    jz     r7,exit{{printf "%d" $y}}
    j      loop{{printf "%d" $y}}
exit{{printf "%d" $y}}:
{{$z := atoi $.Params.pos}}
{{if eq $y $z}}
    mov r9, r0
%endsection
```

A first example of optimization

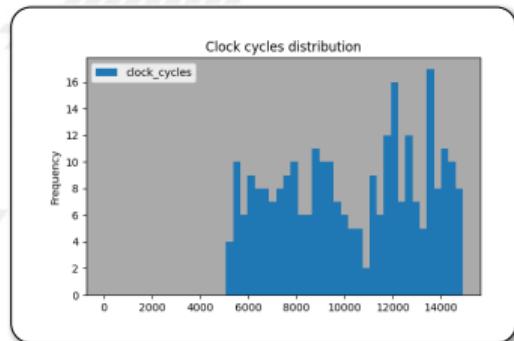
$$e^x = \sum_{l=0}^K \frac{x^l}{l!}$$

K can be customize as needed

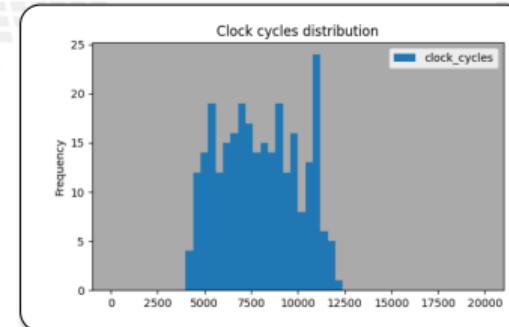


Results of optimization

Changing number of K of the exponential factors in the softmax function...



- $K: 20$
- $\sigma: 2875.94$
- Mean: 10268.45
- Latency: 102 μ s
- Prediction: 100%



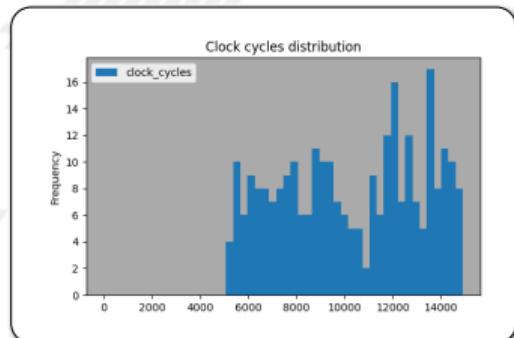
- $K: 16$
- $\sigma: 2106.32$
- Mean: 7946.16
- Latency: 79 μ s
- Prediction: 100%

	mean	σ
prob0	1.6470E-07	1.2332E-07
prob1	1.6623E-07	1.2142E-07

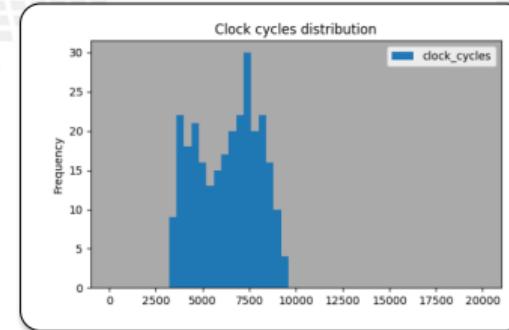
	mean	σ
prob0	1.6470E-07	1.2332E-07
prob1	1.6623E-07	1.2142E-07

Results of optimization

Changing number of K of the exponential factors in the softmax function...



- $K: 20$
- $\sigma: 2875.94$
- Mean: 10268.45
- Latency: 102 μ s
- Prediction: 100%



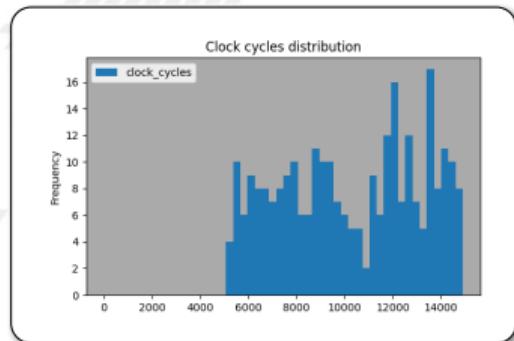
- $K: 13$
- $\sigma: 1669.88$
- Mean: 6312.26
- Latency: 63 μ s
- Prediction: 100%

	mean	σ
prob0	1.6470E-07	1.2332E-07
prob1	1.6623E-07	1.2142E-07

	mean	σ
prob0	1.6470E-07	1.2332E-07
prob1	1.6623E-07	1.2142E-07

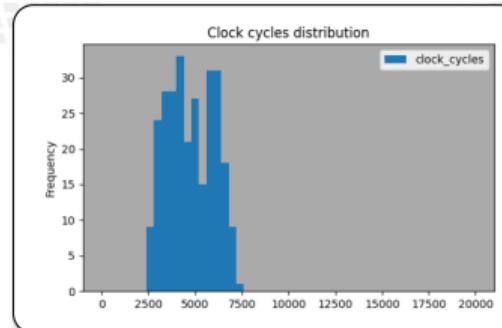
Results of optimization

Changing number of K of the exponential factors in the softmax function...



- $K: 20$
- $\sigma: 2875.94$
- Mean: 10268.45
- Latency: 102 μ s
- Prediction: 100%

	mean	σ
prob0	1.6470E-07	1.2332E-07
prob1	1.6623E-07	1.2142E-07

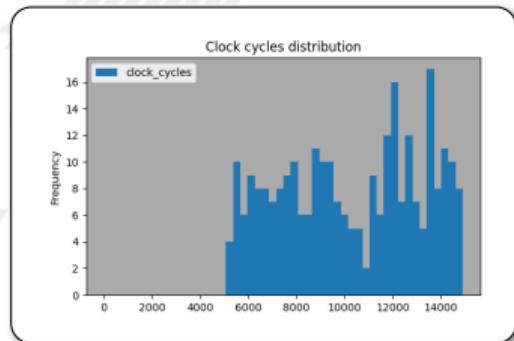


- $K: 10$
- $\sigma: 1232.47$
- Mean: 4766.75
- Latency: 47 μ s
- Prediction: 100%

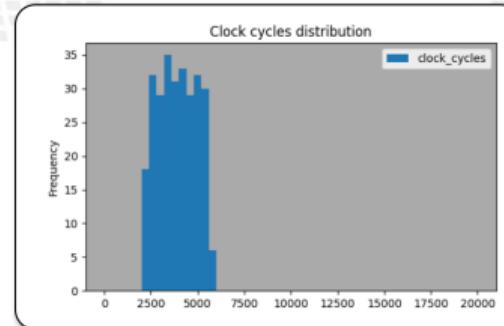
	mean	σ
prob0	1.6162E-07	1.1013E-07
prob1	1.6525E-07	1.1831E-07

Results of optimization

Changing number of K of the exponential factors in the softmax function...



- $K: 20$
- $\sigma: 2875.94$
- Mean: 10268.45
- Latency: 102 μ s
- Prediction: 100%



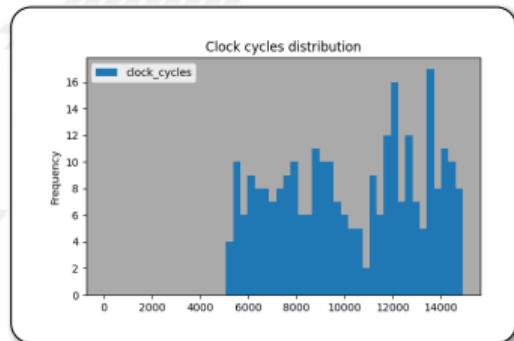
- $K: 8$
- $\sigma: 1015.50$
- Mean: 3913.66
- Latency: 39 μ s
- Prediction: 100%

	mean	σ
prob0	1.6470E-07	1.2332E-07
prob1	1.6623E-07	1.2142E-07

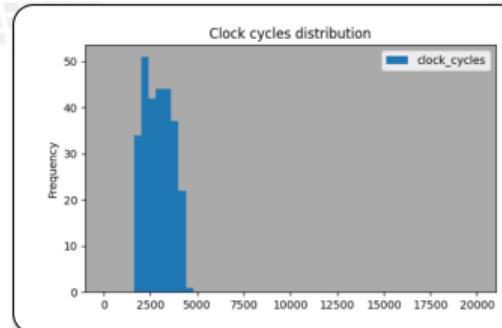
	mean	σ
prob0	6.5562E-05	1.7607E-05
prob1	6.6098E-05	1.7609E-05

Results of optimization

Changing number of K of the exponential factors in the softmax function...



- $K: 20$
- $\sigma: 2875.94$
- Mean: 10268.45
- Latency: 102 μ s
- Prediction: 100%



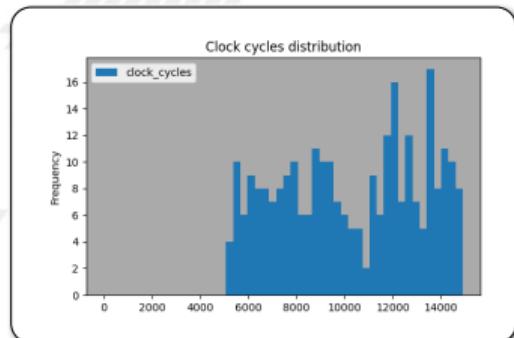
- $K: 5$
- $\sigma: 740$
- Mean: 2911
- Latency: 29 μ s
- Prediction: 100%

	mean	σ
prob0	1.6470E-07	1.2332E-07
prob1	1.6623E-07	1.2142E-07

	mean	σ
prob0	3.1070E-05	7.5290E-05
prob1	3.1070E-05	7.5290E-05

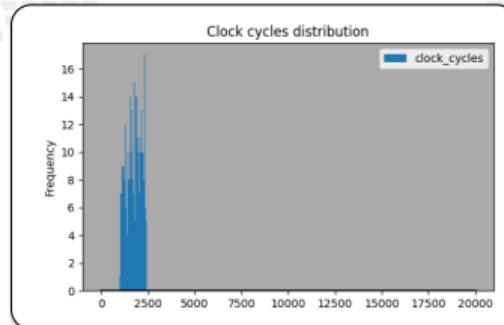
Results of optimization

Changing number of K of the exponential factors in the softmax function...



- $K: 20$
- $\sigma: 2875.94$
- Mean: 10268.45
- Latency: 102 μ s
- Prediction: 100%

	mean	σ
prob0	1.6470E-07	1.2332E-07
prob1	1.6623E-07	1.2142E-07

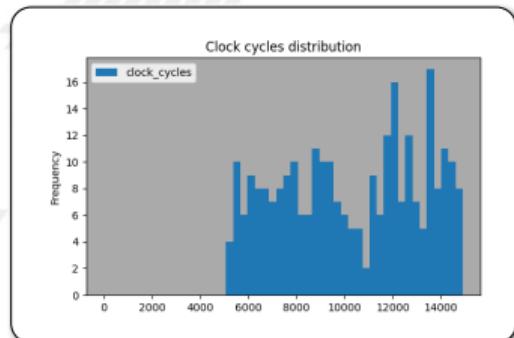


- $K: 3$
- $\sigma: 394.10$
- Mean: 1750.93
- Latency: 17 μ s
- Prediction: 100%

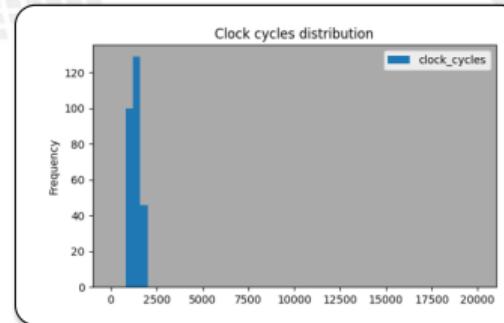
	mean	σ
prob0	0.0053	0.0090
prob1	0.0053	0.0090

Results of optimization

Changing number of K of the exponential factors in the softmax function...



- $K: 20$
- $\sigma: 2875.94$
- Mean: 10268.45
- Latency: 102 μ s
- Prediction: 100%



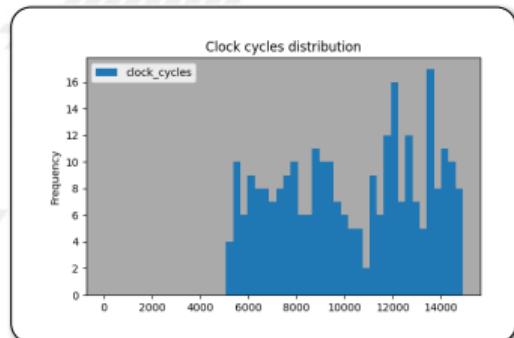
- $K: 2$
- $\sigma: 268.69$
- Mean: 1311.11
- Latency: 13.11 μ s
- Prediction: 100%

	mean	σ
prob0	1.6470E-07	1.2332E-07
prob1	1.6623E-07	1.2142E-07

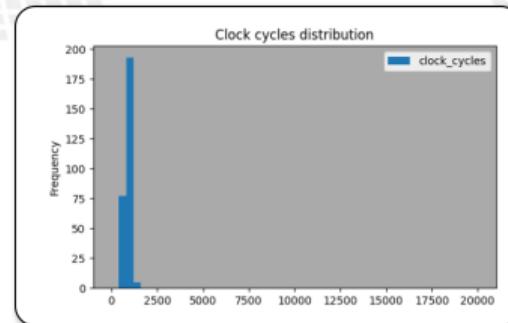
	mean	σ
prob0	0.0193	0.0232
prob1	0.0193	0.0232

Results of optimization

Changing number of K of the exponential factors in the softmax function...



- $K: 20$
- $\sigma: 2875.94$
- Mean: 10268.45
- Latency: 102 μ s
- Prediction: 100%

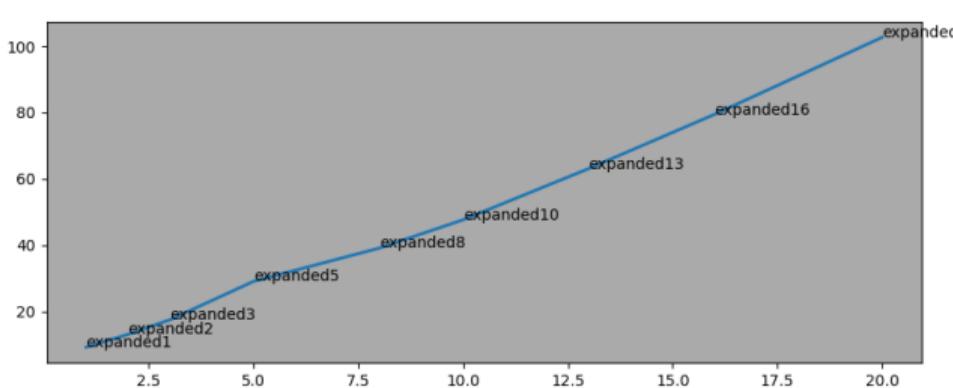


- $K: 1$
- $\sigma: 173.25$
- Mean: 923.71
- Latency: 9.23 μ s
- Prediction: 100%

	mean	σ
prob0	1.6470E-07	1.2332E-07
prob1	1.6623E-07	1.2142E-07

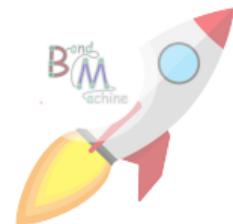
	mean	σ
prob0	0.0990	0.1641
prob1	0.0990	0.1641

Results of optimization



K	Inference time
1	9.23 µs
2	13.11 µs
3	17.50 µs
5	29.11 µs
8	39.13 µs
10	47.66 µs
13	63.12 µs
16	79.46 µs
20	102.68 µs

Reduced inference times by a factor of 10 ... only by decreasing the number of iterations.



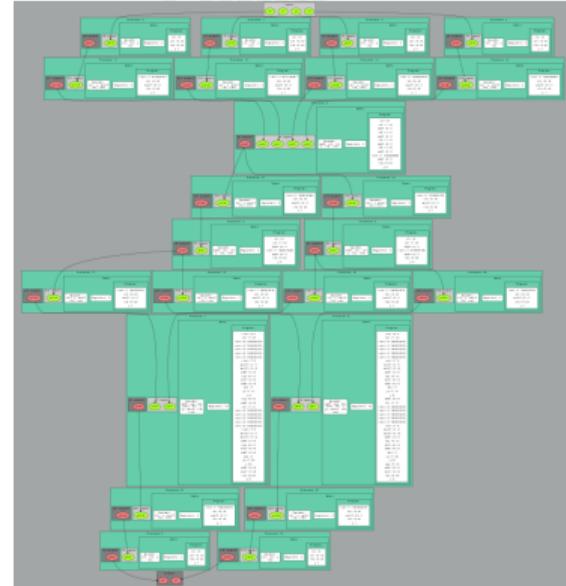
Fragments composition

- The tools (neuralbond+basm) create a graph of relations among fragments of assembly
 - Not necessarily a fragment has to be mapped to a single CP
- They can arbitrarily be rearranged into CPs
- The resulting firmwares are identical in term of the computing outcome, but differs in occupancy and latency.



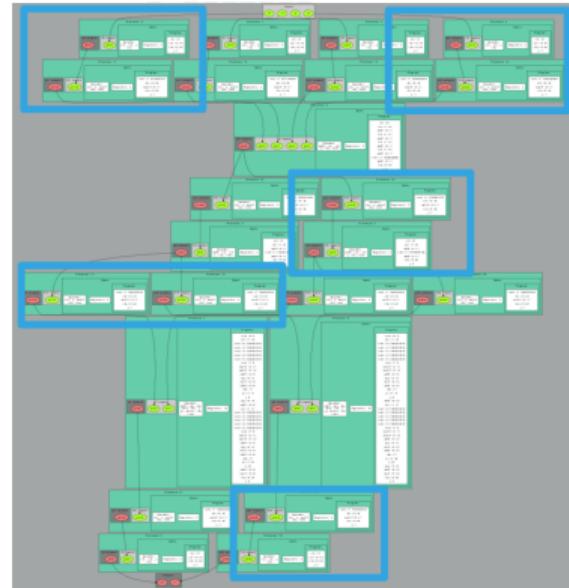
Fragments composition

- The tools (neuralbond+basm) create a graph of relations among fragments of assembly
- Not necessarily a fragment has to be mapped to a single CP
- They can arbitrarily be rearranged into CPs
- The resulting firmwares are identical in term of the computing outcome, but differs in occupancy and latency.



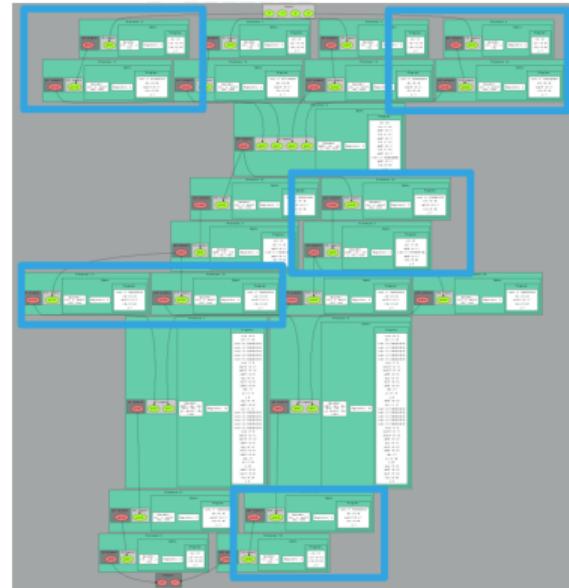
Fragments composition

- The tools (neuralbond+basm) create a graph of relations among fragments of assembly
- Not necessarily a fragment has to be mapped to a single CP
- They can arbitrarily be rearranged into CPs
- The resulting firmwares are identical in term of the computing outcome, but differs in occupancy and latency.



Fragments composition

- The tools (neuralbond+basm) create a graph of relations among fragments of assembly
- Not necessarily a fragment has to be mapped to a single CP
- They can arbitrarily be rearranged into CPs
- The resulting firmwares are identical in term of the computing outcome, but differs in occupancy and latency.



BM demo

NN demo N.5: CP pruning

Goals are:

- Prune a processor and find out the consequences

Reference notebook: proj_zedboard_ml_cp_pruning /notebook.ipynb

BM demo

NN demo N.6: CP collapsing

Goals are:

- Collapse processors and find out the consequences

Reference notebook: proj_zedboard_ml_cp_collapsing /notebook.ipynb

BM demo

Goals are:

- Copy a project directory and try pruning, collapsing, simulating and the assembly of the neurons

Several ways for customization and optimization

The great control over of the architectures generated by the BondMachine gives several possible optimizations.

Mixing hardware
and software
optimizations

CP Pruning
and/or collapsing

Fabric
independent

HW instructions
swapping

Fine control over
occupancy vs
latency

Fragment
composition

HW/SW
Templates

Software based
functions

Conclusions and Future directions

1 Introduction

- Challenges
- FPGA
- HDL workflow
- HLS Workflow
- Concepts
- Cloud

2 The BondMachine project

- Architectures handling
- Architectures molding
- Bondgo
- Basm
- API

3 Misc

- Project timeline

4 Machine Learning

- Train
- BondMachine creation
- Simulation
- Accelerator
- Benchmark

5 Optimizations

6 Conclusions and Future directions

- Conclusions
- Ongoing
- Future

Conclusions

The BondMachine is a new kind of computing device made possible in practice only by the emerging of new re-programmable hardware technologies such as FPGA.

The result of this process is the construction of a computer architecture that is not anymore a static constraint where computing occurs but its creation becomes a part of the computing process, gaining computing power and flexibility.

Over this abstraction is it possible to create a full computing Ecosystem, ranging from small interconnected IoT devices to Machine Learning accelerators.

Ongoing

The project

- Move all the code to github
- Documentation
- First DAQ use case
- Complete the inclusion of Intel and Lattice FPGAs
- ML inference in a cloud workflow

Ongoing Accelerators

- Different data types and operations, especially low and trans-precision
- Different boards support, especially data center accelerator
- Compare with GPUs
- Include some real power consumption measures

Ongoing Machine Learning

With ML we are still at the beginning ...

- **Quantization**
- **More datasets:** test on other datasets with more features and multiclass classification
- **Neurons:** increase the library of neurons to support other activation functions
- **Evaluate results:** compare the results obtained with other technologies (CPU and GPU) in terms of inference speed and energy efficiency

Future work

- Include new processor shared objects and currently unsupported opcodes
- Extend the compiler to include more data structures
- Assembler improvements, fragments optimization and others
- Improve the networking including new kind of interconnection firmware

What would an OS for BondMachines look like ?

Future work

- Include new processor shared objects and currently unsupported opcodes
- Extend the compiler to include more data structures
- Assembler improvements, fragments optimization and others
- Improve the networking including new kind of interconnection firmware

What would an OS for BondMachines look like ?

Future work

- Include new processor shared objects and currently unsupported opcodes
- Extend the compiler to include more data structures
- Assembler improvements, fragments optimization and others
- Improve the networking including new kind of interconnection firmware

What would an OS for BondMachines look like ?

Future work

- Include new processor shared objects and currently unsupported opcodes
- Extend the compiler to include more data structures
- Assembler improvements, fragments optimization and others
- Improve the networking including new kind of interconnection firmware

What would an OS for BondMachines look like ?

Future work

- Include new processor shared objects and currently unsupported opcodes
- Extend the compiler to include more data structures
- Assembler improvements, fragments optimization and others
- Improve the networking including new kind of interconnection firmware

What would an OS for BondMachines look like ?



website: <http://bondmachine.fisica.unipg.it>

code: <https://github.com/BondMachineHQ>

parallel computing paper: link

contact email: mirko.mariotti@unipg.it