The BondMachine project

Mirko Mariotti [1,2] Giulio Bianchini [1] Loriano Storchi [3,2] Giacomo Surace [2]
Daniele Spiga [2] Diego Ciangottini [2]

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

[2]INFN sezione di Perugia

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

Outline

1 Introduction
- Challenges
- FPGA
- Architectures
- Abstractions

2 The BondMachine project
- Architectures handling
- Architectures molding
- Bondgo
- Basm
- API

3 Clustering
- An example
- Video
- Distributed architecture

4 Accelerators
- Hardware
- Software
- Tests
- Benchmark

5 Misc
- Project timeline
- Supported boards
- Use cases

6 Machine Learning
- Train
- Benchmark

7 Optimizations
- Softmax example
- Results
- Model's compression
- Fragments compositions

8 Accelerator in a cloud
- Bring it to cloud level: why and how
- Implementing a KServe FPGA extension
- Where are we...

9 Conclusions and Future directions
- Conclusions
- Ongoing
- Future

Hands-on sessions

During the lecture some topic will have a hands-on session.
The code is available at the GitHub repository:
`https://github.com/BondMachineHQ/bmexamples`

To download the code, open a terminal and type:

```
git clone https://github.com/BondMachineHQ/bmexamples.git
```

Inside the folder `bmexamples` you will find the examples. They will work either on the terminal or on the Jupyter notebooks.
Each directory contains a project and is referred by a number in the slides (as for example shows the next slide).
Directories that not start with a number are not covered in the lecture but are part of the default BondMachine examples and available for you to play with.

Framework installation

Hands-on N.00

It will be shown how:

- To install the BondMachine framework
- Make it available in a Jupyter notebook

Introduction

1 Introduction
- Challenges
- FPGA
- Architectures
- Abstractions

2 The BondMachine project
- Architectures handling
- Architectures molding
- Bondgo
- Basm
- API

3 Clustering
- An example
- Video
- Distributed architecture

4 Accelerators
- Hardware
- Software
- Tests
- Benchmark

5 Misc
- Project timeline
- Supported boards
- Use cases

6 Machine Learning
- Train
- Benchmark

7 Optimizations
- Softmax example
- Results
- Model's compression
- Fragments compositions

8 Accelerator in a cloud
- Bring it to cloud level: why and how
- Implementing a KServe FPGA extension
- Where are we...

9 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

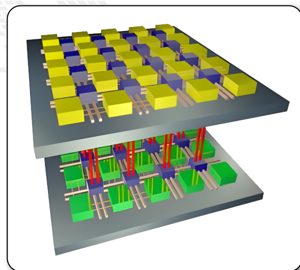
- Von Neumann Bottleneck:
  New computational problems show that current architectural models has to be improved or changed to address future payloads.

- Energy Efficient computation:
  Not wasting "resources" (silicon, time, energy, instructions).
  Using the right resource for the specific case

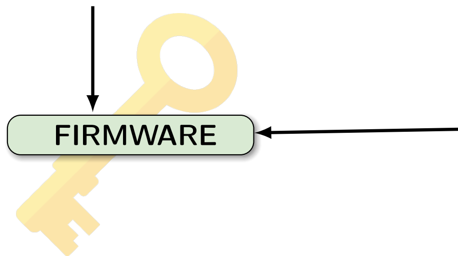
- Edge/Fog/Cloud Computing:
  Making the computation where it make sense
  Avoiding the transfer of unnecessary data
  Creating consistent interfaces for distributed systems

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

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

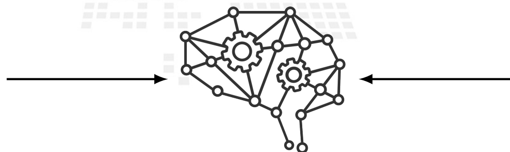
Use in computing

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 ...

FPGA

Challenges in computing

On the other hand the adoption on FPGA poses several challenges:

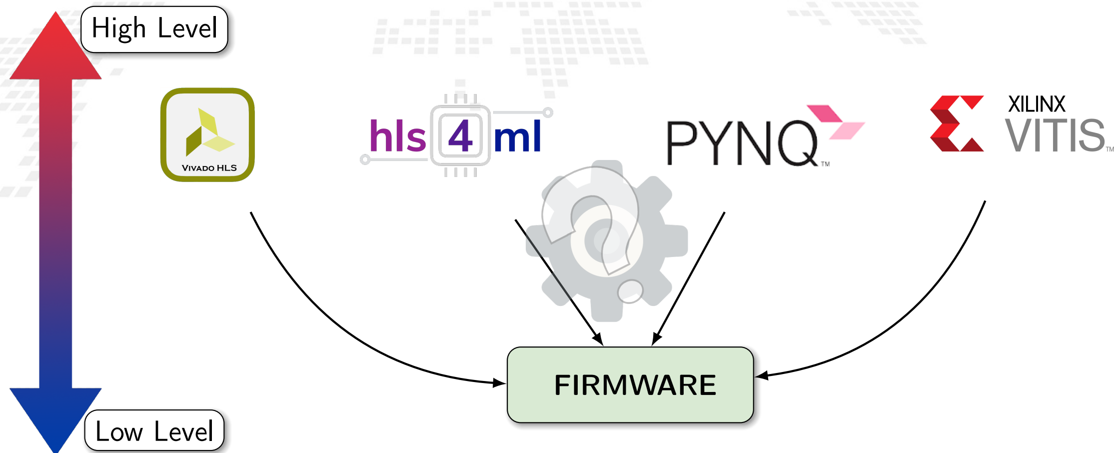
- Porting of legacy code is usually hard.
- Interoperability with standard applications is problematic.

On the other hand the adoption on FPGA poses several challenges:

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

Firmware generation

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



Today's computer architecture are:

- Multi-core,Two or more independent actual processing units execute multiple instructions at the same time.
  - The power is given by the number of cores.
  - Parallelism has to be addressed.
- Heterogeneous, different types of processing units.
  - Cell, GPU, Parallela, TPU.
  - The power is given by the specialization.
  - The units data transfer has to be addressed.
  - The payloads scheduling has to be addressed.

Today's computer architecture are:

- Multi-core,Two or more independent actual processing units execute multiple instructions at the same time.
  - The power is given by the number of cores.
  - Parallelism has to be addressed.
- Heterogeneous, different types of processing units.
  - Cell, GPU, Parallela, TPU.
  - The power is given by the specialization.
  - The units data transfer has to be addressed.
  - The payloads scheduling has to be addressed.

Today's computer architecture are:

- Multi-core,Two or more independent actual processing units execute multiple instructions at the same time.
  - The power is given by the number of cores.
  - Parallelism has to be addressed.
- Heterogeneous, different types of processing units.
  - Cell, GPU, Parallela, TPU.
  - The power is given by the specialization.
  - The units data transfer has to be addressed.
  - The payloads scheduling has to be addressed.

Computer Architectures

Multi-core and Heterogeneous

Today's computer architecture are:

- Multi-core,Two or more independent actual processing units execute multiple instructions at the same time.
  - The power is given by the number of cores.
  - Parallelism has to be addressed.
- Heterogeneous, different types of processing units.
  - Cell, GPU, Parallela, TPU.
  - The power is given by the specialization.
  - The units data transfer has to be addressed.
  - The payloads scheduling has to be addressed.

Computer Architectures

Multi-core and Heterogeneous

Today's computer architecture are:

- Multi-core,Two or more independent actual processing units execute multiple instructions at the same time.
  - The power is given by the number of cores.
  - Parallelism has to be addressed.
- Heterogeneous, different types of processing units.
  - Cell, GPU, Parallela, TPU.
  - The power is given by the specialization.
  - The units data transfer has to be addressed.
  - The payloads scheduling has to be addressed.

Computer Architectures

Multi-core and Heterogeneous

Today's computer architecture are:

- Multi-core,Two or more independent actual processing units execute multiple instructions at the same time.
  - The power is given by the number of cores.
  - Parallelism has to be addressed.
- Heterogeneous, different types of processing units.
  - Cell, GPU, Parallela, TPU.
  - The power is given by the specialization.
  - The units data transfer has to be addressed.
  - The payloads scheduling has to be addressed.

Today's computer architecture are:

- Multi-core,Two or more independent actual processing units execute multiple instructions at the same time.
  - The power is given by the number of cores.
  - Parallelism has to be addressed.
- Heterogeneous, different types of processing units.
  - Cell, GPU, Parallela, TPU.
  - The power is given by the specialization.
  - The units data transfer has to be addressed.
  - The payloads scheduling has to be addressed.

Today's computer architecture are:

- Multi-core,Two or more independent actual processing units execute multiple instructions at the same time.
  - The power is given by the number of cores.
  - Parallelism has to be addressed.
- Heterogeneous, different types of processing units.
  - Cell, GPU, Parallela, TPU.
  - The power is given by the specialization.
  - The units data transfer has to be addressed.
  - The payloads scheduling has to be addressed.

Computer Architectures

Multi-core and Heterogeneous

Today's computer architecture are:

- Multi-core,Two or more independent actual processing units execute multiple instructions at the same time.
  - The power is given by the number of cores.
  - Parallelism has to be addressed.
- Heterogeneous, different types of processing units.
  - Cell, GPU, Parallela, TPU.
  - The power is given by the specialization.
  - The units data transfer has to be addressed.
  - The payloads scheduling has to be addressed.

Today's computer architecture are:

- Multi-core,Two or more independent actual processing units execute multiple instructions at the same time.
  - The power is given by the number of cores.
  - Parallelism has to be addressed.
- Heterogeneous, different types of processing units.
  - Cell, GPU, Parallela, TPU.
  - The power is given by the specialization.
  - The units data transfer has to be addressed.
  - The payloads scheduling has to be addressed.

The BondMachine

First idea

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 Specialization

The BondMachine

First idea

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

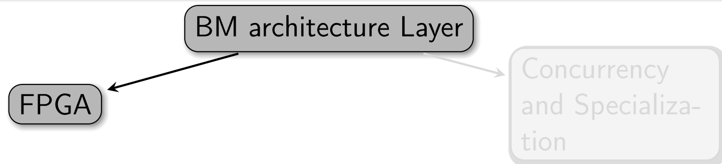
Concurrency and Specialization

The BondMachine

First idea

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.

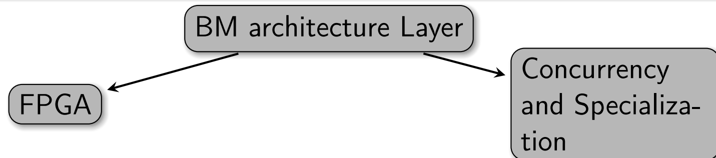


The BondMachine

First idea

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

First idea

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 Specialization

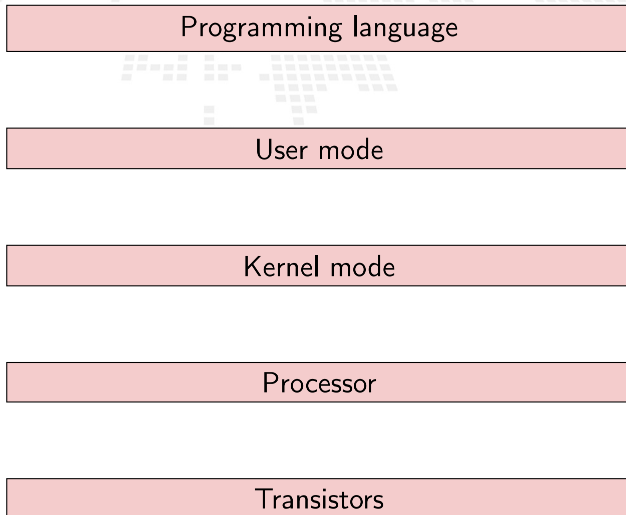
Layer, Abstractions and Interfaces

A Computing system is a matter of abstraction and interfaces. A lower layer exposes its functionalities (via interfaces) to the above layer hiding (abstraction) its inner details.

The quality of a computing system is determined by how abstractions are simple and how interfaces are clean.

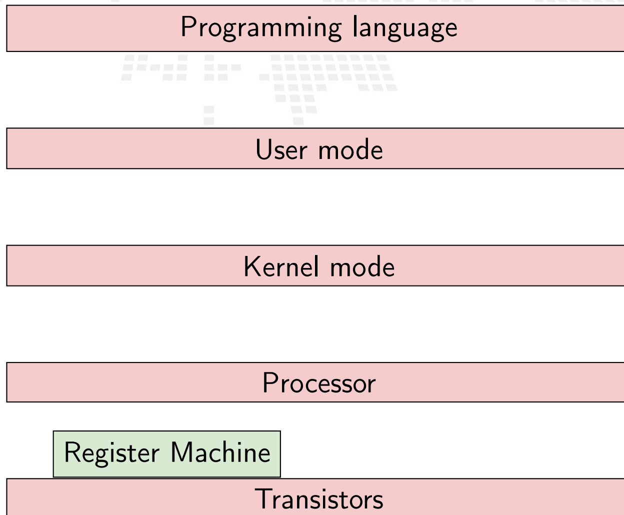
Layers, Abstractions and Interfaces

An example



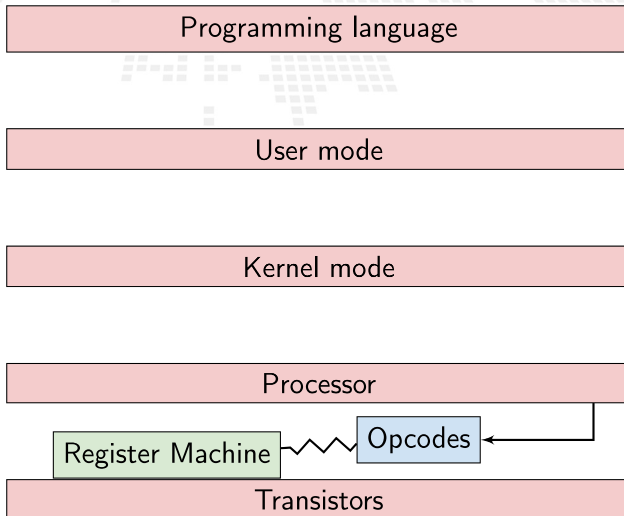
Layers, Abstractions and Interfaces

An example



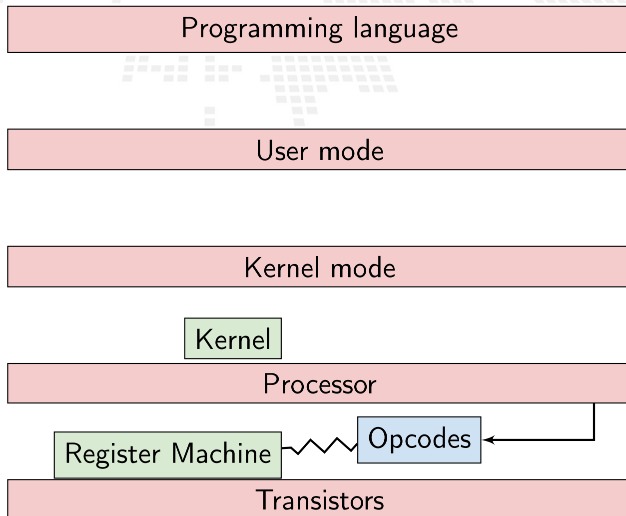
Layers, Abstractions and Interfaces

An example



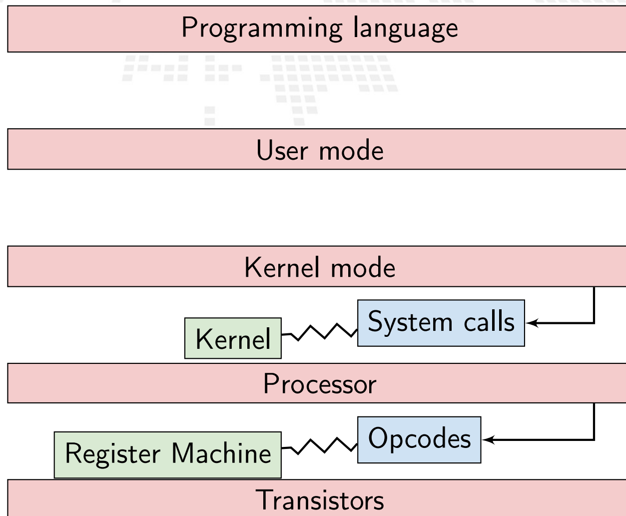
Layers, Abstractions and Interfaces

An example



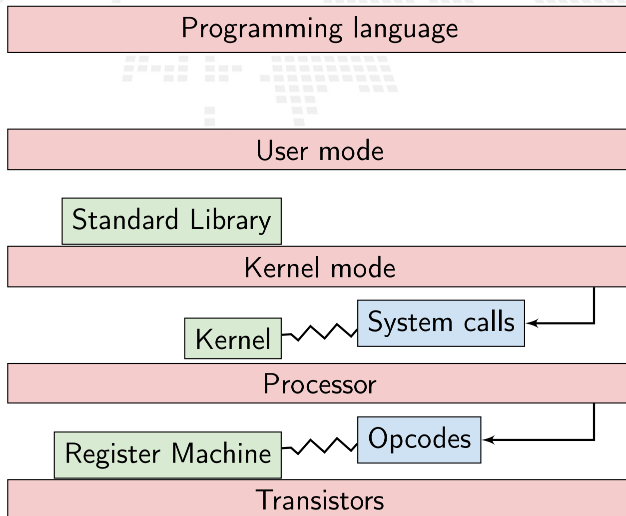
Layers, Abstractions and Interfaces

An example



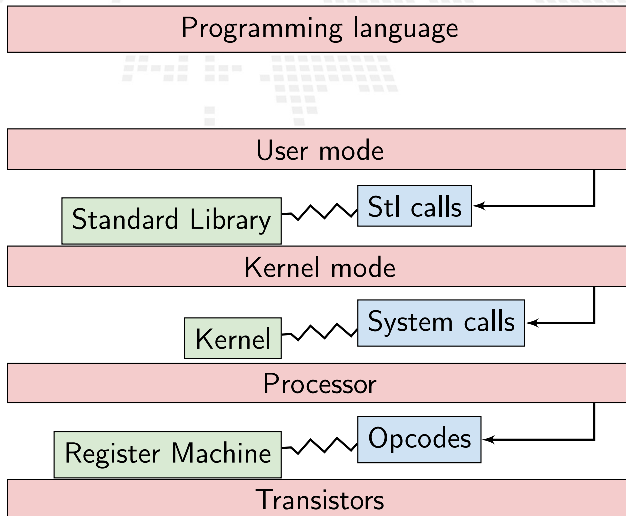
Layers, Abstractions and Interfaces

An example



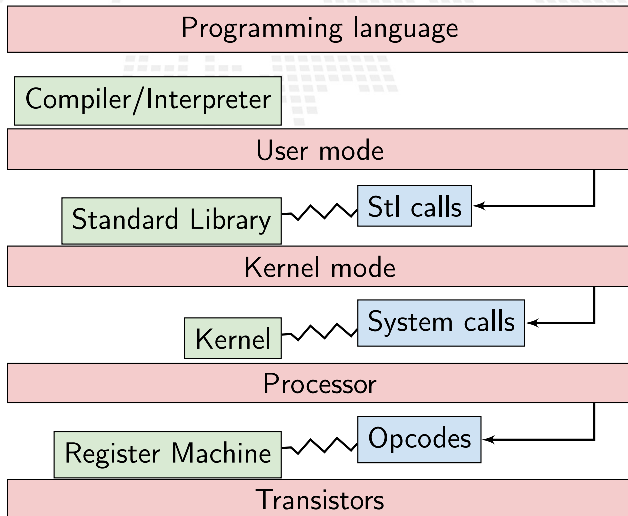
Layers, Abstractions and Interfaces

An example



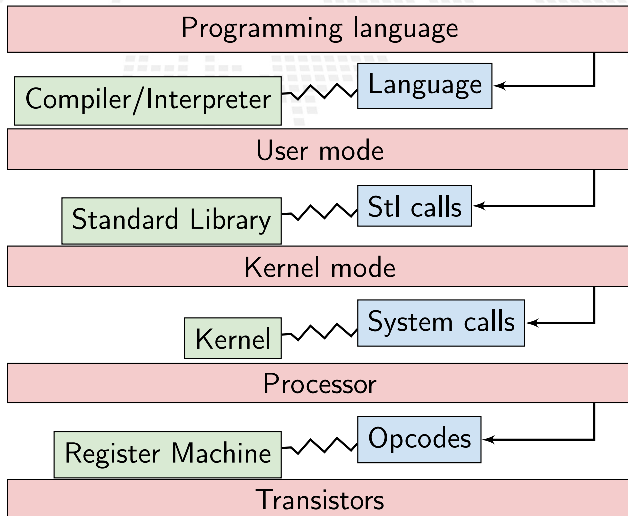
Layers, Abstractions and Interfaces

An example



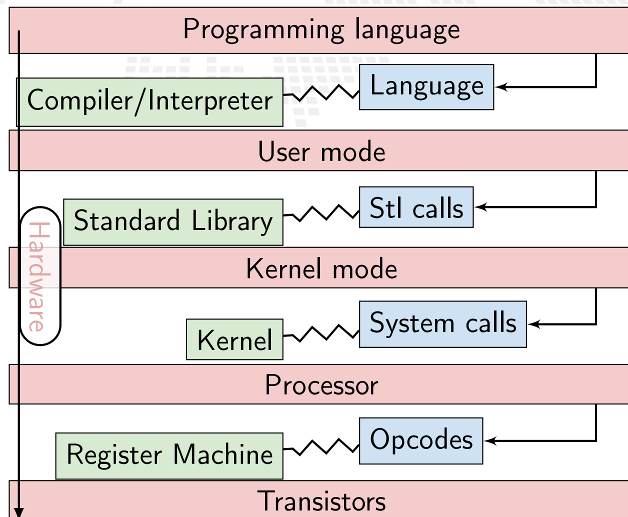
Layers, Abstractions and Interfaces

An example



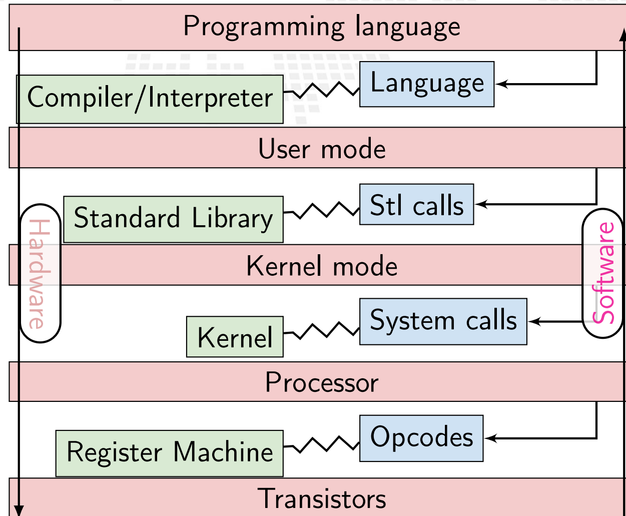
Layers, Abstractions and Interfaces

An example



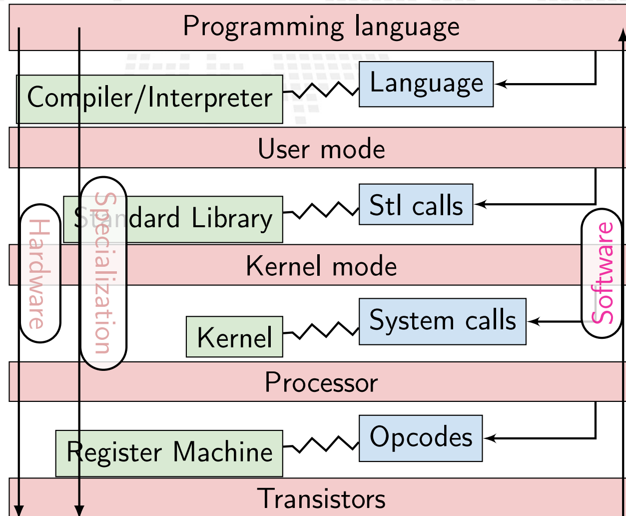
Layers, Abstractions and Interfaces

An example



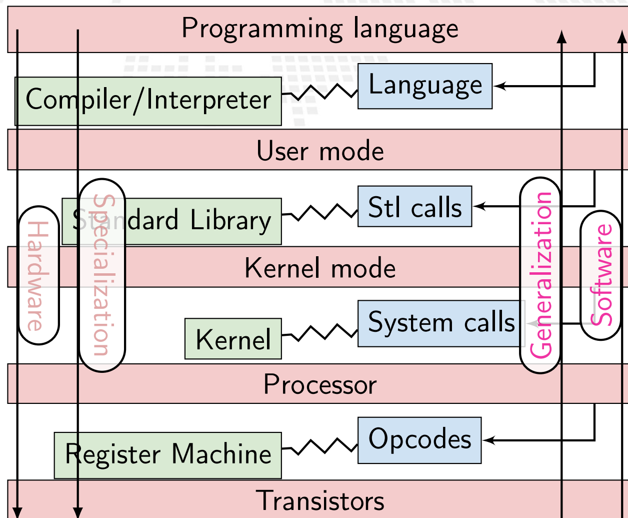
Layers, Abstractions and Interfaces

An example



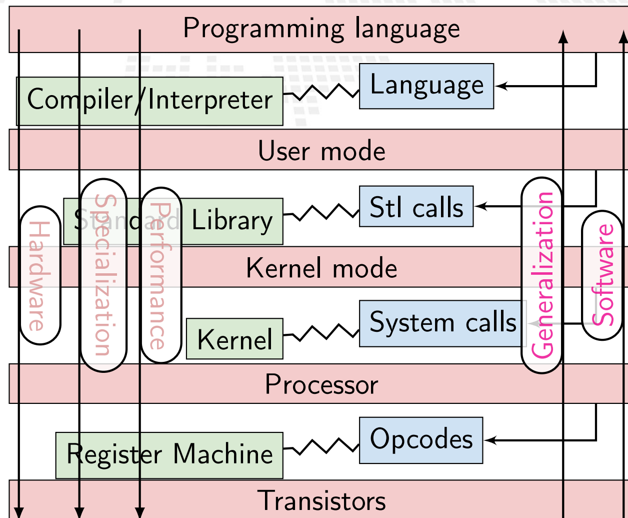
Layers, Abstractions and Interfaces

An example



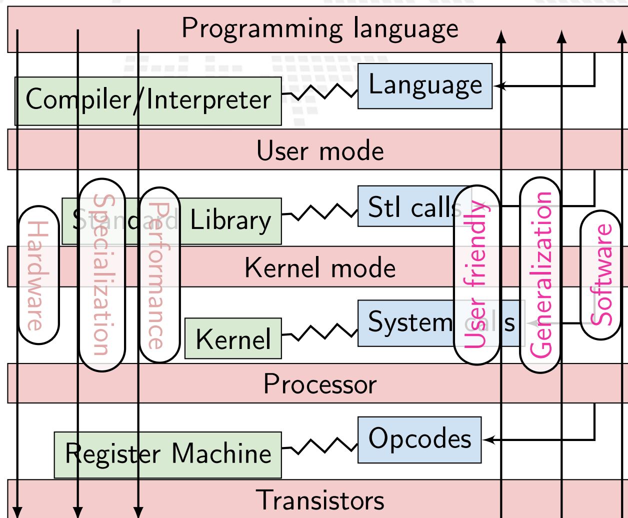
Layers, Abstractions and Interfaces

An example



Layers, Abstractions and Interfaces

An example



Layers, Abstractions and Interfaces

The second idea

Rethinking the stack

Build a computing system with a decreased number of layers resulting in a minor gap between HW and SW but keeping an user friendly way of programming it.

The BondMachine project

1 Introduction
- Challenges
- FPGA
- Architectures
- Abstractions

2 The BondMachine project
- Architectures handling
- Architectures molding
- Bondgo
- Basm
- API

3 Clustering
- An example
- Video
- Distributed architecture

4 Accelerators
- Hardware
- Software
- Tests
- Benchmark

5 Misc
- Project timeline
- Supported boards
- Use cases

6 Machine Learning
- Train
- Benchmark

7 Optimizations
- Softmax example
- Results
- Model's compression
- Fragments compositions

8 Accelerator in a cloud
- Bring it to cloud level: why and how
- Implementing a KServe FPGA extension
- Where are we...

9 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).

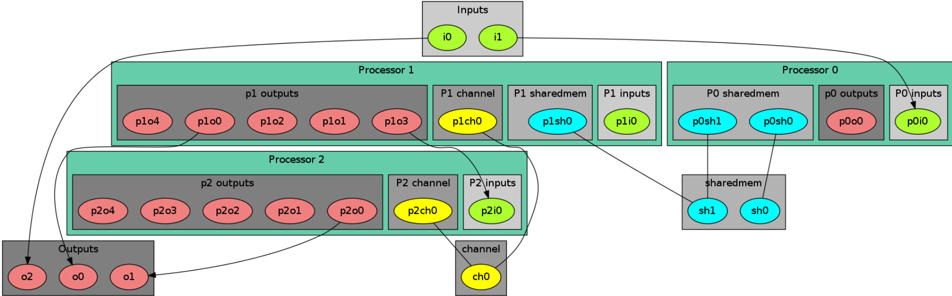
Introducing the BondMachine (BM)

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 ... r2^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 ... iN
M output registers: o0,o1 ... oM

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,addf16,addi,addp,and,chc,chw,cil,cilc,cir,cirn,clc,clr,cmpr,cpy,cset,dec,div
divf,divf16,divp,dpc,expf,hit,hlt,i2r,i2rw,incc,inc,j,ja,jc,jcmpa,jcmpl,jcmpo,jcmpria
jcmprio,je,jri,jria,jrio,jgt0f,jo,jz,k2r,lfsr82r,m2r,m2rri,mod,mulc,mult,multf,multf16
multp,nand,nop,nor,not,or,q2r,r2m,r2mri,r2o,r2owa,r2owaa,r2q,r2s,r2v,r2vri,r2t,r2u,ro2r
ro2rri,rsc,rset,sic,s2r,saj,sbc,sub,t2r,u2r,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.

Connecting Processor (CP)

More on instructions

The HDL code of the aforementioned opcodes is statically defined, and adding instructions to a CP includes the HDL code of the instructions in the CP HDL code.

Procbuilder also support the dynamic creation of new instructions created at runtime (the creation runtime not the FPGA). It can be for example HDL code generated by an external tool, or an instruction that changes according to some input data.
Here the list of current dynamic instructions:

- *FloPoCo*: A floating point unit generator.
- *Linear Quantizer*: A linear quantizer operation generator.
- *Rsets*: Static Register set with fixed size.
- *Call*: Call instruction with hardware based stack.
- *Stack*: Stack instruction with hardware based stack.
- *fixed point*: Fixed point arithmetic.

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.

Channel

The Channel SO is an hardware implementation of the CSP (communicating sequential processes) channel.

It is a model for inter-core communication and synchronization via message passing.

CPs use channels via 4 opcodes

- *wrd*: Want Read.
- *wwr*: Want Write.
- *chc*: Channel Check.
- *chw*: Channel Wait.

Channel

The Channel SO is an hardware implementation of the CSP (communicating sequential processes) channel.

It is a model for inter-core communication and synchronization via message passing.

CPs use channels via 4 opcodes

- *wrd*: Want Read.
- *wwr*: Want Write.
- *chc*: Channel Check.
- *chw*: Channel Wait.

Channel

The Channel SO is an hardware implementation of the CSP (communicating sequential processes) channel.

It is a model for inter-core communication and synchronization via message passing.

CPs use channels via 4 opcodes

- *wrd*: Want Read.
- *wwr*: Want Write.
- *chc*: Channel Check.
- *chw*: Channel Wait.

Shared Memory

The Shared Memory SO is a RAM block accessible from more than one CP.

Different Shared Memories can be used by different CP and not necessarily by all of them.

CPs use shared memories via 2 opcodes

- *s2r*: Shared memory read.
- *r2s*: Shared memory write.

Shared Memory

The Shared Memory SO is a RAM block accessible from more than one CP.

Different Shared Memories can be used by different CP and not necessarily by all of them.

CPs use shared memories via 2 opcodes

- *s2r*: Shared memory read.
- *r2s*: Shared memory write.

Shared Memory

The Shared Memory SO is a RAM block accessible from more than one CP.

Different Shared Memories can be used by different CP and not necessarily by all of them.

CPs use shared memories via 2 opcodes

- *s2r*: Shared memory read.
- *r2s*: Shared memory write.

Barrier

The Barrier SO is used to make CPs act synchronously.

When a CP hits a barrier, the execution stop until all the CPs that share the same barrier hit it.

CPs use barriers via 1 opcode

- *hit*: Hit the barrier.

Barrier

The Barrier SO is used to make CPs act synchronously.

When a CP hits a barrier, the execution stop until all the CPs that share the same barrier hit it.

CPs use barriers via 1 opcode

- *hit*: Hit the barrier.

Barrier

The Barrier SO is used to make CPs act synchronously.

When a CP hits a barrier, the execution stop until all the CPs that share the same barrier hit it.

CPs use barriers via 1 opcode

- *hit*: Hit the barrier.

Multicore and Heterogeneous

First idea on the BondMachine

The idea was:

Having a multi-core architecture completely heterogeneous both in cores types and interconnections.

The BondMachine may have many cores, eventually all different, arbitrarily interconnected and sharing non computing elements.

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

Processor Builder

Procbuilder is the CP manipulation tool.

- CP Creation
- CP Load/Save
- CP Assembler/Disassembler
- CP HDL

Examples

(32 bit registers counter machine)

```
procbuilder -register-size 32 -opcodes clr,cpy,dec,inc,je,jz
```

(Input and Output registers)

```
procbuilder -inputs 3 -outputs 2 ...
```

Processor Builder

Procbuilder is the CP manipulation tool.

- CP Creation
- CP Load/Save
- CP Assembler/Disassembler
- CP HDL

Examples

(Loading a CP)

procbuilder -load-machine conproc.json ...

(Saving a CP)

procbuilder -save-machine conproc.json ...

Processor Builder

Procbuilder is the CP manipulation tool.

- CP Creation
- CP Load/Save
- CP Assembler/Disassembler
- CP HDL

Examples

(Assembiling)

procbuilder -input-assembly program.asm ...

(Disassembling)

procbuilder -show-program-disassembled ...

Processor Builder

Procbuilder is the CP manipulation tool.

- CP Creation
- CP Load/Save
- CP Assembler/Disassembler
- CP HDL

Examples

(Create the CP RTL code in Verilog)

procbuilder -create-verilog ...

(Create testbench)

procbuilder -create-verilog-testbench test.v ...

Procbuilder Hands-on

Hands-on N.01

It will be shown how:

- To create a simple processor
- To assemble and disassemble code for it
- To produce its HDL code

BondMachine Builder

Bondmachine is the tool that compose CP and SO to form BondMachines.

- BM CP insert and remove
- BM SO insert and remove
- BM Inputs and Outputs
- BM Bonding Processors and/or IO
- BM Visualizing or HDL

Examples

(Add a processor)

bondmachine -add-domains proc.json ... ; ... -add-processor 0

(Remove a processor)

bondmachine -bondmachine-file bmach.json -del-processor n

BondMachine Builder

Bondmachine is the tool that compose CP and SO to form BondMachines.

- BM CP insert and remove
- BM SO insert and remove
- BM Inputs and Outputs
- BM Bonding Processors and/or IO
- BM Visualizing or HDL

Examples

(Add a Shared Object)

bondmachine -add-shared-objects specs ...

(Connect an SO to a processor)

bondmachine -connect-processor-shared-object ...

BondMachine Builder

Bondmachine is the tool that compose CP and SO to form BondMachines.

- BM CP insert and remove
- BM SO insert and remove
- BM Inputs and Outputs
- BM Bonding Processors and/or IO
- BM Visualizing or HDL

Examples

(Adding inputs or outputs)

bondmachine -add-inputs ... ; bondmachine -add-outputs ...

(Removing inputs or outputs)

bondmachine -del-input ... ; bondmachine -del-output ...

BondMachine Builder

Bondmachine is the tool that compose CP and SO to form BondMachines.

- BM CP insert and remove
- BM SO insert and remove
- BM Inputs and Outputs
- BM Bonding Processors and/or IO
- BM Visualizing or HDL

Examples

(Bonding processor)

bondmachine -add-bond p0i2,p1o4 ...

(Bonding IO)

bondmachine -add-bond i2,p0i6 ...

BondMachine Builder

Bondmachine is the tool that compose CP and SO to form BondMachines.

- BM CP insert and remove
- BM SO insert and remove
- BM Inputs and Outputs
- BM Bonding Processors and/or IO
- BM Visualizing or HDL

Examples

(Visualizing)

bondmachine -emit-dot ...

(Create RTL code)

bondmachine -create-verilog ...

BondMachine Hands-on

Hands-on N.02

It will be shown how:

- To create a single-core BondMachine
- To attach an external output
- To produce its HDL code

Toolchain and helper tool

bmhelper

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

Plus, an helper tool, called *bmhelper* has been developed to simplify the creation and maintenance of the BM Projects.

doctor
Checks whether the tools are correctly installed

create
Creates a new BM project

validate
Validates a BM project by checking the presence of all the necessary variables

apply
Finalizes the BM project by adding the necessary files

Toolchain and helper tool

Makefile

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 accelerator create an accelerator IP from the BM

make design create an accelerator design

make bitstream [design_bitstream] create the firwware [accelerator firmware]

make program flash the device into the destination target

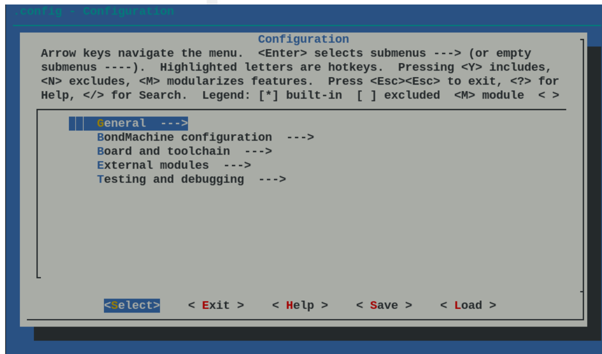
make xclbin create a platform firmware

make clean remove all the build files

Toolchain and helper tool

Kernel config style

Complementary to the Makefile and the local.mk file, a kernel config style file is used to specify the build operations.



Toolchain Hands-on

Hands-on N.03

It will be shown how:

- To explore the toolchain
- To flash the board with the code from the previous example

Shared Object Hands-on

Hands-on N.04

It will be shown how:

- To build a BondMachine with a processor and a shared object
- To flash the board

Dual core Hands-on

Hands-on N.05

It will be shown how:

- To build a dual-core BondMachine
- To connect cores
- To flash the board

BondMachine web front-end

Operations on BondMachines can also be performed via an under development web framework

Test | I/O and Bonds | Processors | Shared Objects

Inputs Management | Outputs Management | Bonds Management

Tools
Processors
Bondmachines

Examples
8ping
Ping Pong

Projects
Test

Bonds

Index	Endpoint 1 Name	Endpoint 2 Name	Actions
2	p1o0	o0	Delete bond
0	i0	p0i0	Delete bond
1	p0o0	p1i0	Delete bond

New

Select Endpoints

Endpoint 1 Name
p0i0
p1i0
o0

Endpoint 2 Name
p0o0
p1o0
i0

Layout

Inputs
i0
Processor 0
p0 outputs
p0o0
P0 channel
p0ch0
P0 inputs
p0i0
Processor 1
p1 outputs
p1o0
P1 channel
p1ch0
P1 inputs
p1i0
Outputs
o0
channel
ch0

Convergence Engine Version 2.0 - Copyright © 2016 - Mirko Mariotti - Theme design by Free CSS Templates

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 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 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 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 Hands-on

Hands-on N.06

It will be shown how:

- To show the simulation capabilities of the framework

Emulation

The simulation facility is not necessarily used for debug purposes, it can be used also to run payloads without having a real FPGA.

The same engine that simulate BondMachines can be used as emulator.

Through the emulator BondMachines can be used on Linux workstations.

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

ML tools

Map computational graphs to BM

Basm

The BondMachine assembler

Bondgo

The architecture compiler

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

Bondgo

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.

Bondgo

This is the standard flow when building computer programs

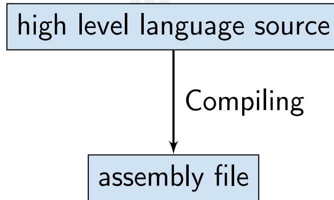
Bondgo

This is the standard flow when building computer programs

high level language source

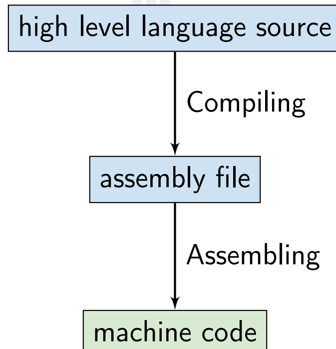
Bondgo

This is the standard flow when building computer programs



Bondgo

This is the standard flow when building computer programs



Bondgo

Bondgo does something different from standard compilers ...

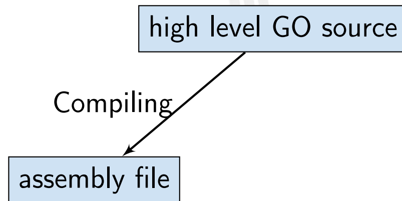
Bondgo

Bondgo does something different from standard compilers ...

high level GO source

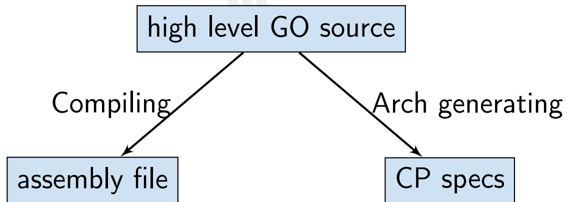
Bondgo

Bondgo does something different from standard compilers ...



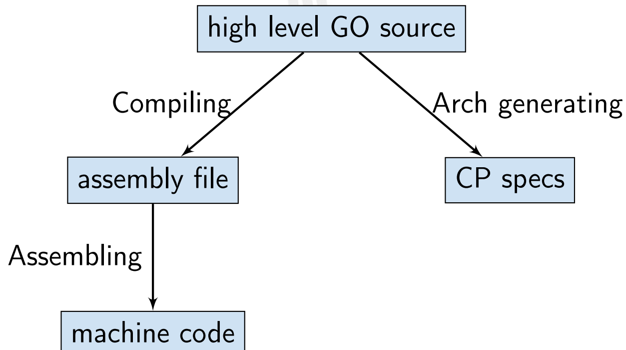
Bondgo

Bondgo does something different from standard compilers ...



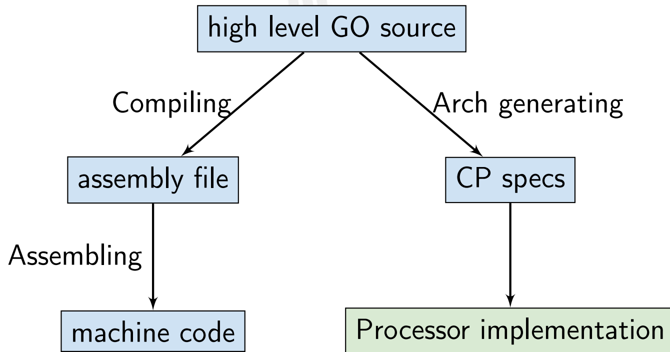
Bondgo

Bondgo does something different from standard compilers ...



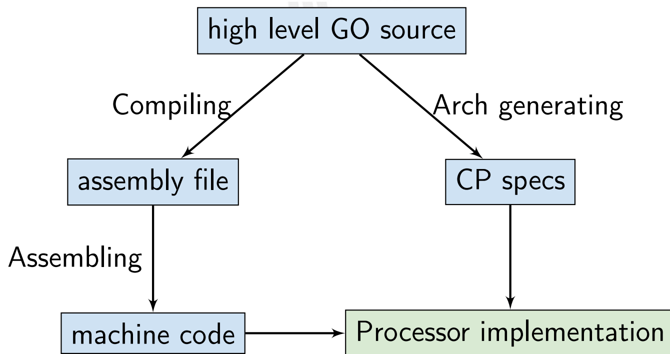
Bondgo

Bondgo does something different from standard compilers ...

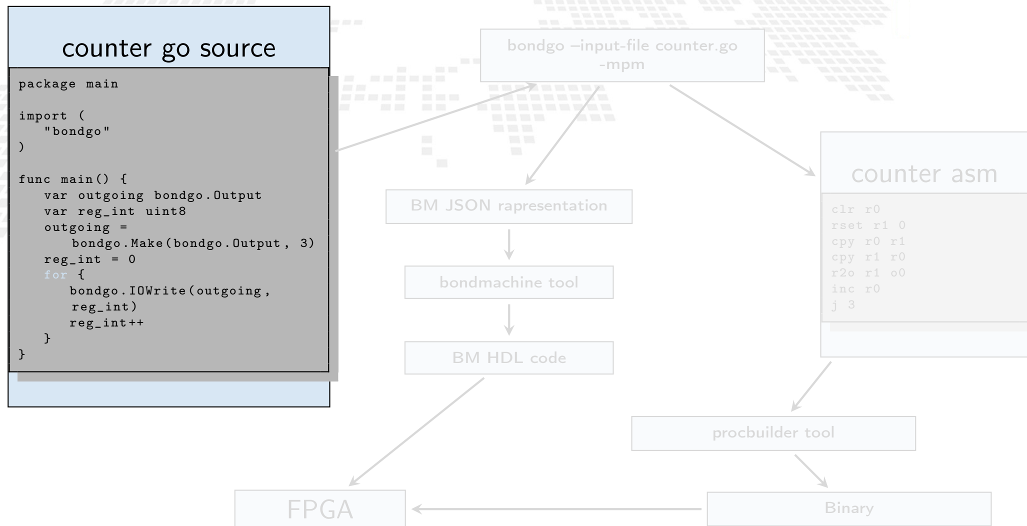


Bondgo

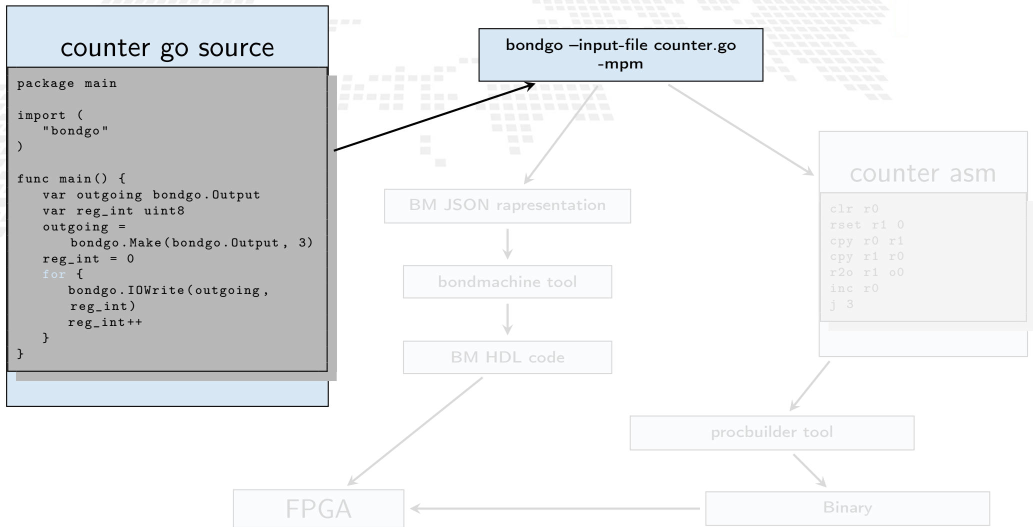
Bondgo does something different from standard compilers ...



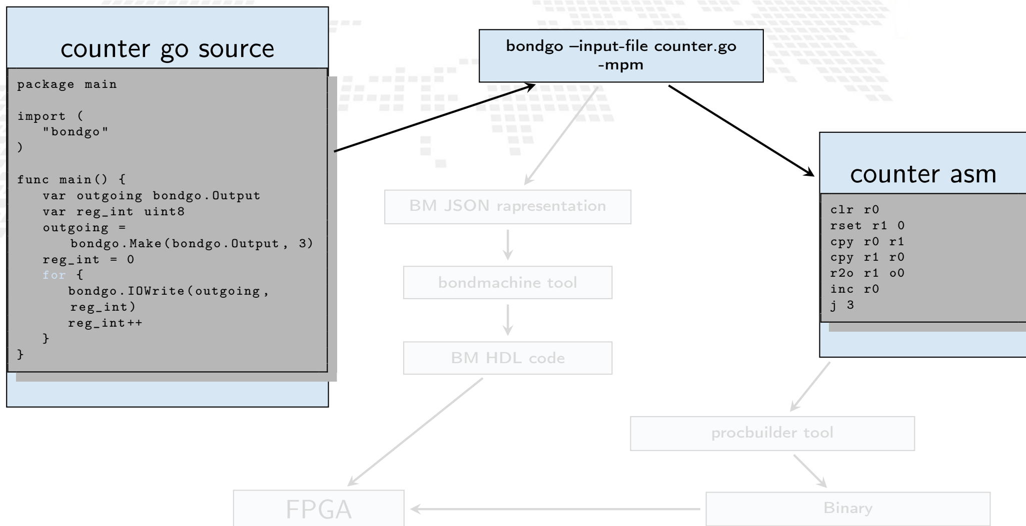
Bondgo workflow example



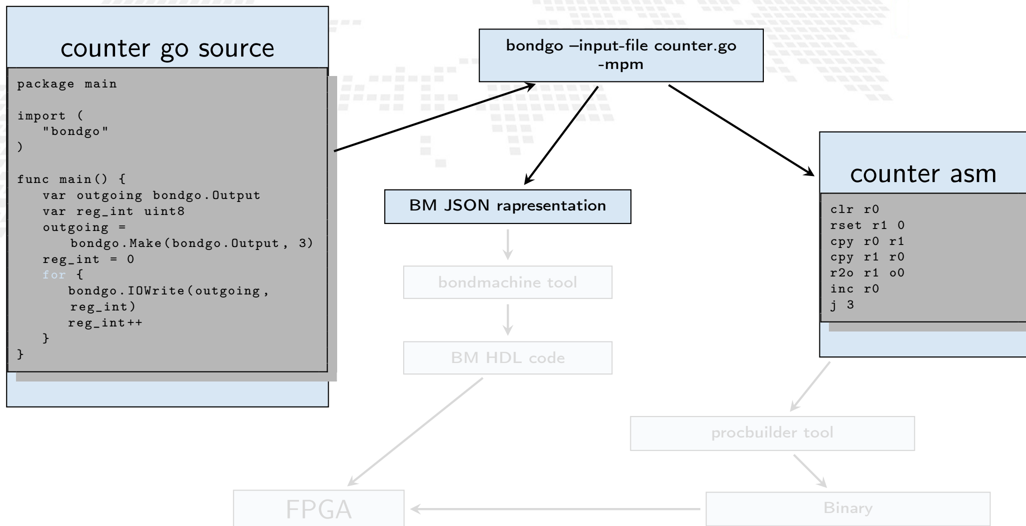
Bondgo workflow example



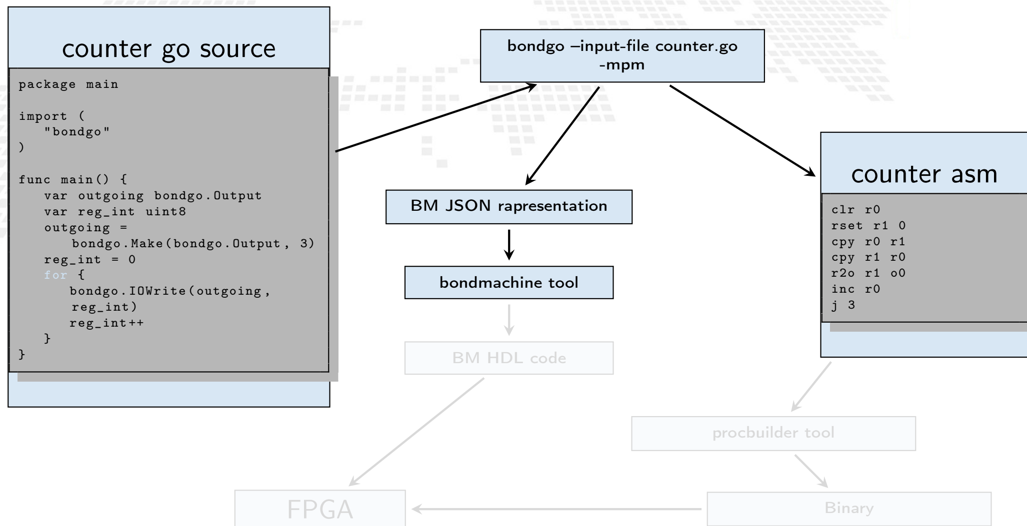
Bondgo workflow example



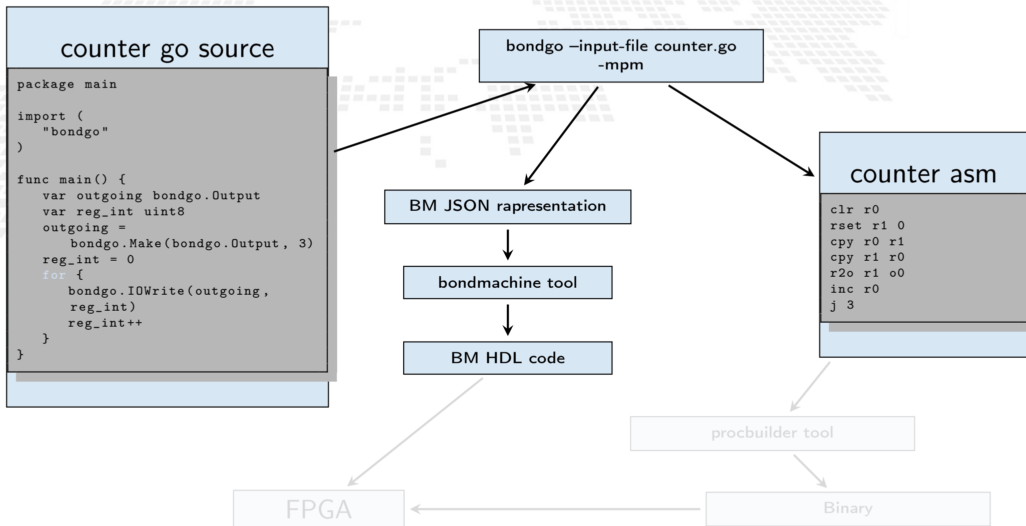
Bondgo workflow example



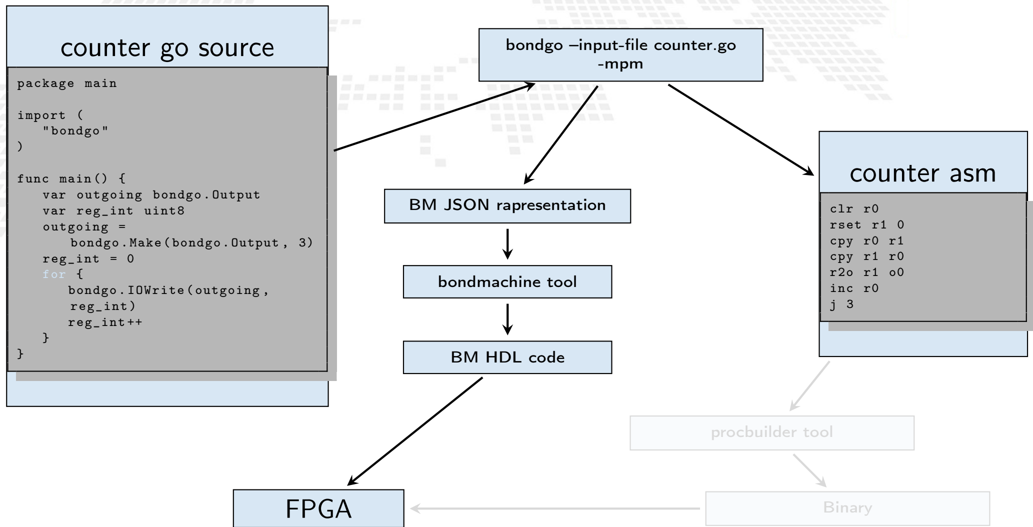
Bondgo workflow example



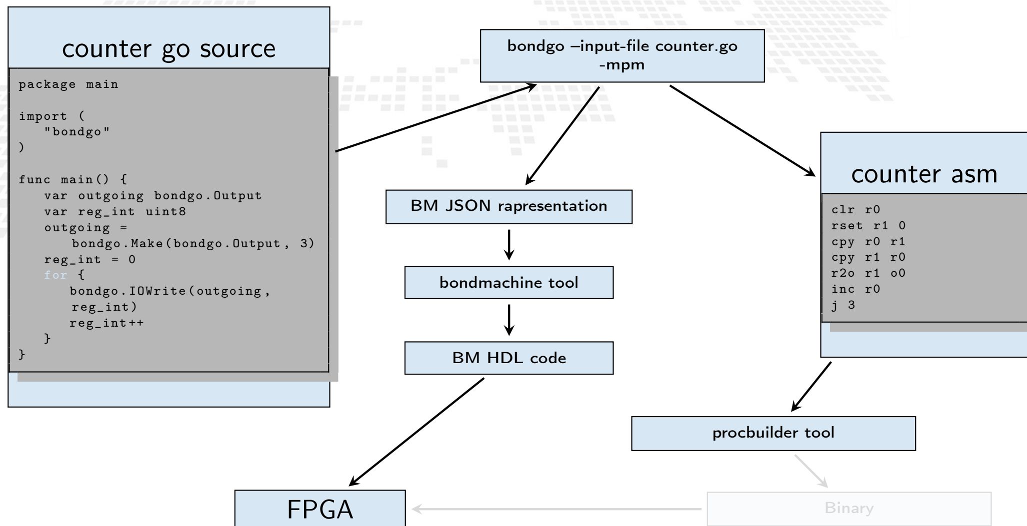
Bondgo workflow example



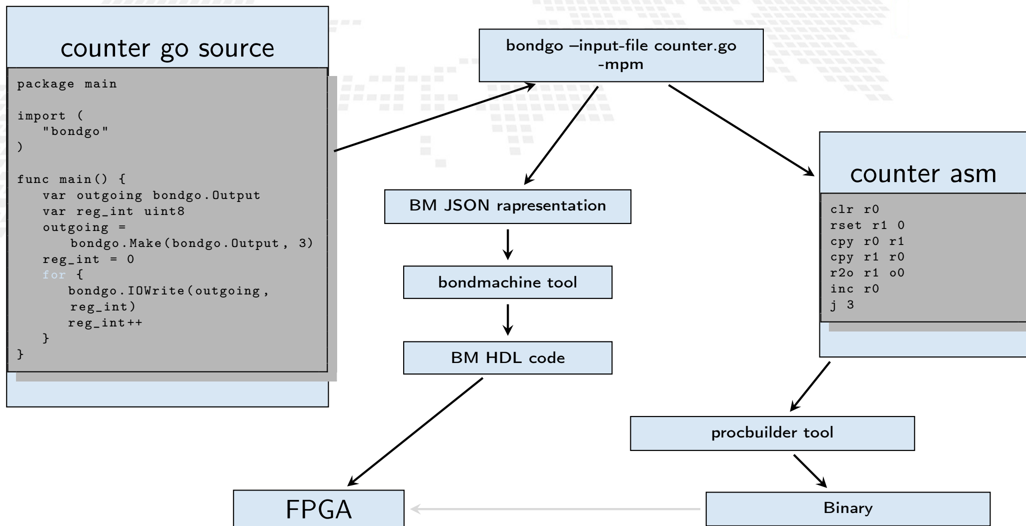
Bondgo workflow example



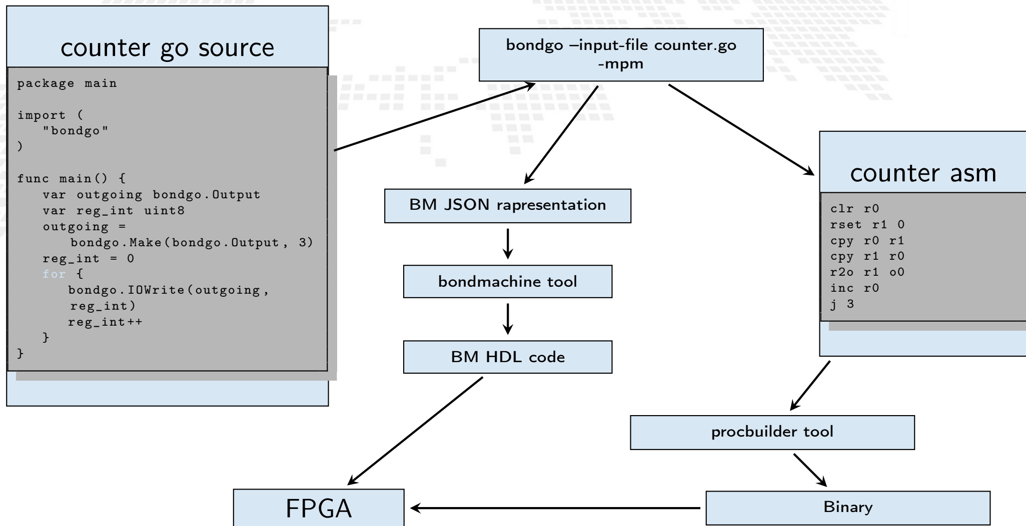
Bondgo workflow example



Bondgo workflow example



Bondgo workflow example



Bondgo

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

Bondgo Hands-on

Hands-on N.07

It will be shown how:

- To create a BondMachine from a Go source file
- To build the architecture
- To build the program
- To create the firmware and flash it to the board

Bondgo

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

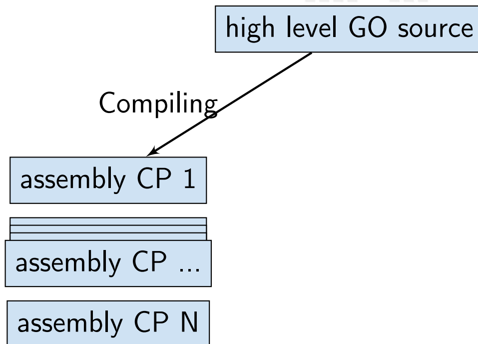
Bondgo

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

high level GO source

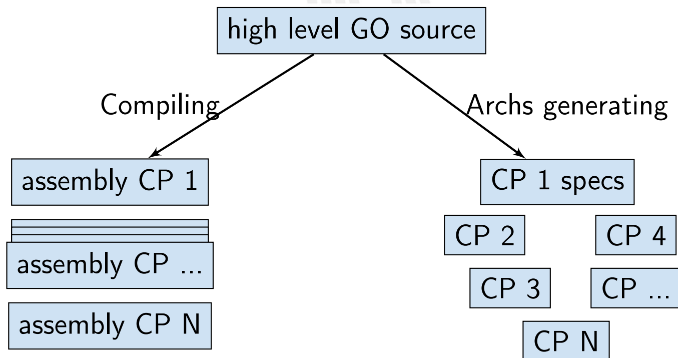
Bondgo

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



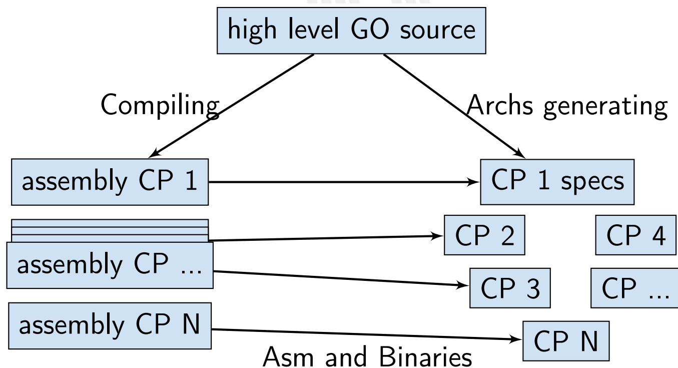
Bondgo

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



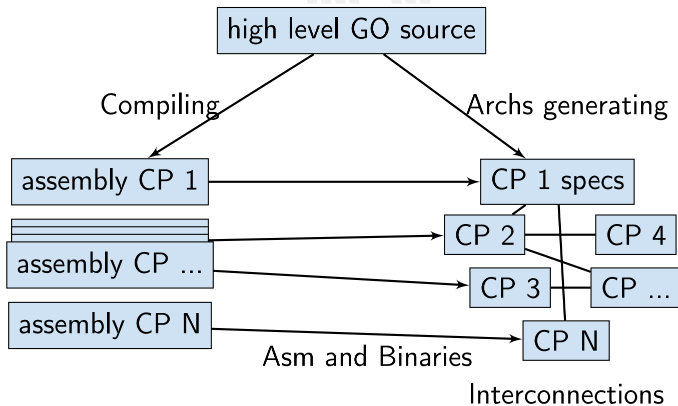
Bondgo

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



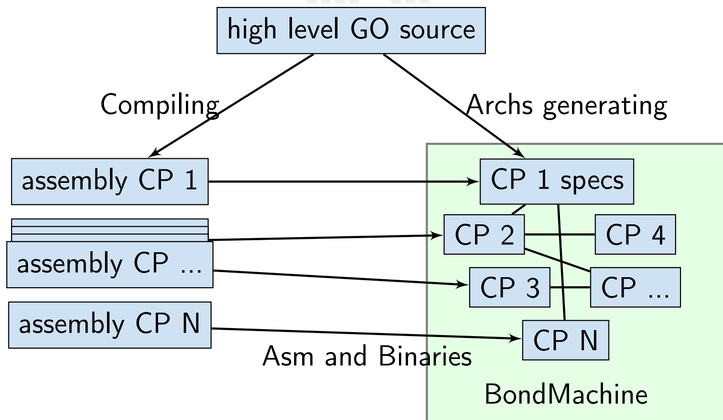
Bondgo

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



Bondgo

... 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.IORead(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.IORead(in0))
   }
}
```

Bondgo

A multi-core example

Compiling the code with the bondgo compiler:

bondgo -input-file ds.go -mpm

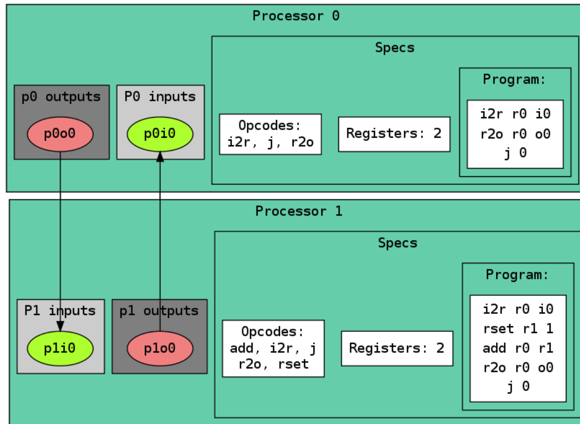
The toolchain perform the following steps:

- Map the two goroutines to two hardware cores.
- Creates two types of core, each one optimized to execute the assigned goroutine.
- Creates the two binaries.
- Connected the two core as inferred from the source code, using special IO registers.

The result is a multicore BondMachine:

Bondgo

A multi-core example



Compiling Architectures

One of the most important result

The architecture creation is a part of the compilation process.

Bondgo multi core Hands-on

Hands-on N.08

It will be shown how:

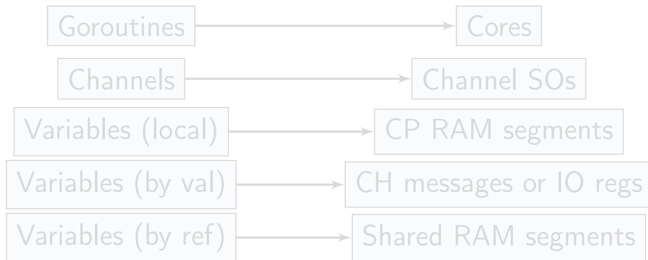
- To use bondgo to create a chain of interconnected processors
- To flash the firmware to the board

Bondgo

Go in hardware

Bondgo implements a sort of “Go in hardware”.

High level Go source code is directly mapped to interconnected processors without Operating Systems or runtimes.

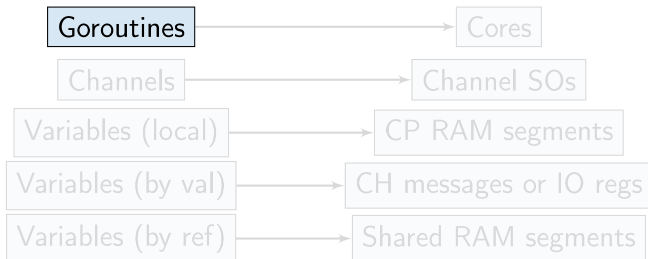


Bondgo

Go in hardware

Bondgo implements a sort of "Go in hardware".

High level Go source code is directly mapped to interconnected processors without Operating Systems or runtimes.

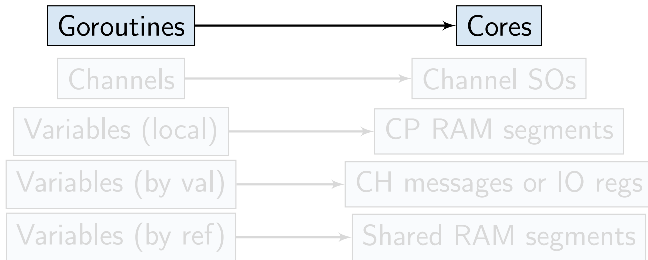


Bondgo

Go in hardware

Bondgo implements a sort of "Go in hardware".

High level Go source code is directly mapped to interconnected processors without Operating Systems or runtimes.

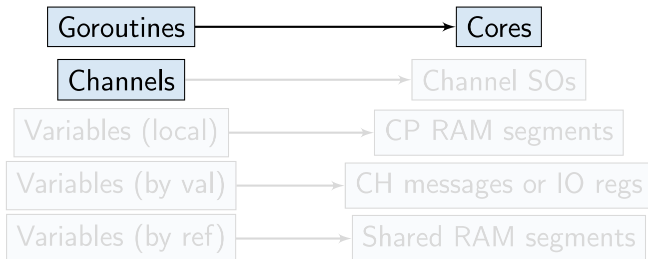


Bondgo

Go in hardware

Bondgo implements a sort of "Go in hardware".

High level Go source code is directly mapped to interconnected processors without Operating Systems or runtimes.

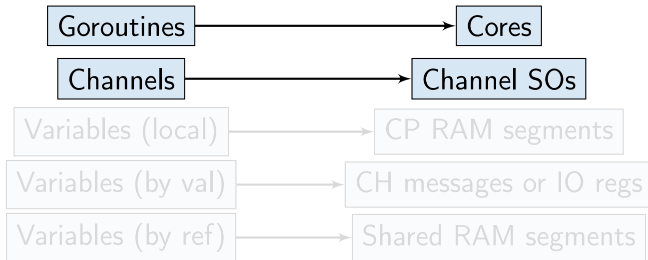


Bondgo

Go in hardware

Bondgo implements a sort of “Go in hardware”.

High level Go source code is directly mapped to interconnected processors without Operating Systems or runtimes.

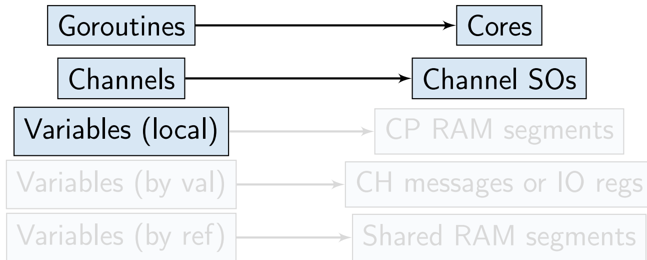


Bondgo

Go in hardware

Bondgo implements a sort of "Go in hardware".

High level Go source code is directly mapped to interconnected processors without Operating Systems or runtimes.

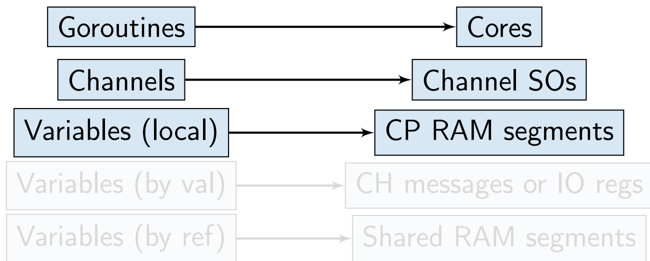


Bondgo

Go in hardware

Bondgo implements a sort of "Go in hardware".

High level Go source code is directly mapped to interconnected processors without Operating Systems or runtimes.

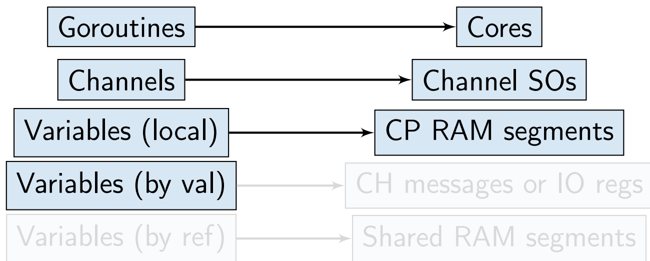


Bondgo

Go in hardware

Bondgo implements a sort of "Go in hardware".

High level Go source code is directly mapped to interconnected processors without Operating Systems or runtimes.

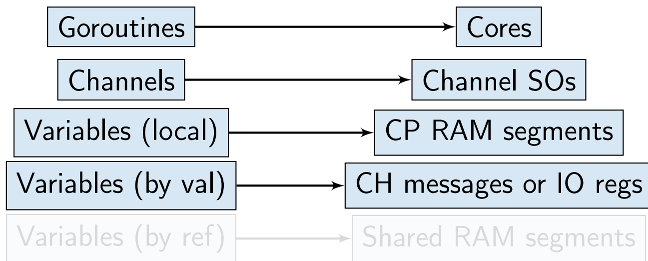


Bondgo

Go in hardware

Bondgo implements a sort of "Go in hardware".

High level Go source code is directly mapped to interconnected processors without Operating Systems or runtimes.

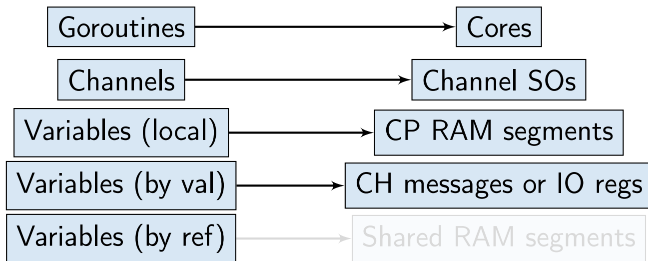


Bondgo

Go in hardware

Bondgo implements a sort of "Go in hardware".

High level Go source code is directly mapped to interconnected processors without Operating Systems or runtimes.

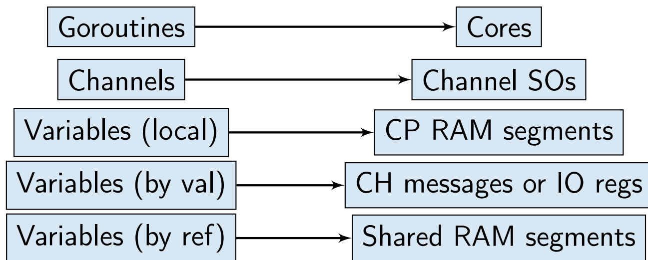


Bondgo

Go in hardware

Bondgo implements a sort of “Go in hardware”.

High level Go source code is directly mapped to interconnected processors without Operating Systems or runtimes.



Go in hardware

Second idea on the BondMachine

The idea was:
Build a computing system with a decreased number of layers resulting in a lower HW/SW gap.

This would raise the overall performances yet keeping an user friendly way of programming.

Between HW and SW there is only the processor abstraction, no Operating System nor runtimes. Despite that programming is done at high level.

Layers, Abstractions and Interfaces

and BondMachines

Bondgo

An example

bondgo stream processing example

```go
package main

import (
   "bondgo"
)

func streamprocessor(a *[]uint8, b *[]uint8,
   c *[]uint8, gid uint8) {
   (*c)[gid] = (*a)[gid] + (*b)[gid]
}
func main() {
   a := make([]uint8, 256)
   b := make([]uint8, 256)
   c := make([]uint8, 256)

   // ... some a and b values fill

   for i := 0; i < 256; i++ {
      go streamprocessor(&a, &b, &c, uint8(i))
   }
}
```

The compilation of this example results in the creation of a 257 CPs where 256 are the stream processors executing the code in the function called *streamprocessor*, and one is the coordinating CP. Each stream processor is optimized and capable only to make additions since it is the only operation requested by the source code. The three slices created on the main function are passed by reference to the Goroutines then a shared RAM is created by the *Bondgo* compiler available to the generated CPs.

Basm

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

It is a standard assembler that can be used to build code for the BondMachine. Given the "fluid" nature of the BM architectures, BASM has some unique features:

- Support for code fragments
- Support for template based assembly code
- Fragments composition: combining and rewriting
- Building hardware from assembly
- Software/Hardware rearrange capabilities
- LLVM IR import

Basm

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

It is a standard assembler that can be used to build code for the BondMachine. Given the "fluid" nature of the BM architectures, BASM has some unique features:

- Support for code fragments
- Support for template based assembly code
- Fragments composition: combining and rewriting
- Building hardware from assembly
- Software/Hardware rearrange capabilities
- LLVM IR import

Basm

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

It is a standard assembler that can be used to build code for the BondMachine. Given the "fluid" nature of the BM architectures, BASM has some unique features:

- Support for code fragments
- Support for template based assembly code
- Fragments composition: combining and rewriting
- Building hardware from assembly
- Software/Hardware rearrange capabilities
- LLVM IR import

Basm

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

It is a standard assembler that can be used to build code for the BondMachine. Given the "fluid" nature of the BM architectures, BASM has some unique features:

- Support for code fragments
- Support for template based assembly code
- Fragments composition: combining and rewriting
- Building hardware from assembly
- Software/Hardware rearrange capabilities
- LLVM IR import

Basm

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

It is a standard assembler that can be used to build code for the BondMachine. Given the "fluid" nature of the BM architectures, BASM has some unique features:

- Support for code fragments
- Support for template based assembly code
- Fragments composition: combining and rewriting
- Building hardware from assembly
- Software/Hardware rearrange capabilities
- LLVM IR import

Basm

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

It is a standard assembler that can be used to build code for the BondMachine. Given the "fluid" nature of the BM architectures, BASM has some unique features:

- Support for code fragments
- Support for template based assembly code
- Fragments composition: combining and rewriting
- Building hardware from assembly
- Software/Hardware rearrange capabilities
- LLVM IR import

Basm

File main structure

Metadata

.romtext

.romdata

Section

.ramtext

.ramdata

Macro

Chunk

Fragment

Sections are the main building blocks of the assembly file. They are used to group together code and data and are the objects that are actually loaded in the memory of the BondMachine (ROM or RAM).

Basm

File main structure

Metadata

.romtext

.romdata

Section

.ramtext

.ramdata

Macro

Chunk

Fragment

.romtext sections contains the code that will be hardwired in the ROM of the BondMachine.

Basm

File main structure

Metadata

.romtext

.romdata

Section

.ramtext

.ramdata

Macro

Chunk

Fragment

.romdata sections contains the data that will be hardwired in the ROM of the BondMachine.

Basm

File main structure

.ramtext sections contains the code that will form a CP executable. This code will be loaded in the RAM of the BondMachine once the CP is instantiated and can be replaced at runtime.

Metadata

.romtext

.romdata

Section

.ramtext

.ramdata

Macro

Chunk

Fragment

Basm

File main structure

Metadata

.romtext

.romdata

Section

.ramtext

.ramdata

Macro

Chunk

Fragment

.ramdata sections contains the data that accompanies the code in the .ramtext sections. This data will be loaded in the RAM of the BondMachine once the CP is instantiated and can be replaced at runtime.

Basm

File main structure

Metadata

.romtext

.romdata

Section

.ramtext

.ramdata

Macro

Chunk

Fragment

Macros are used to define code frequently used in the assembly file. They are expanded at assembly time like in most assemblers.

Basm

File main structure

Metadata

.romtext

.romdata

Section

.ramtext

.ramdata

Macro

Chunk

Fragment

Chunks are used to define data structures more complex than simple variables.

Basm

File main structure

Metadata

.romtext

.romdata

Section

.ramtext

.ramdata

Macro

Chunk

Fragment

Fragments small fragments of code that can be assembled in different ways to form different CPs. More on this later.

Basm

File main structure

Metadata

Metadata is used to define the properties of the bondmachine that will be built from the assembly file. In particular it contains details about the CPs and SOs interconnections.

.romtext

.romdata

Section

.ramtext

.ramdata

Macro

Chunk

Fragment

Basm

An example

basm example

```
%section code1 .romtext
        entry _start    ; Entry point
_start:

        clr     r0
        rset    r0,49
   rset  r1,45
        mov     vtm0:[r1], r0
        rset    r0, 50
        r2v     r0, 128
        clr     r0
        j _start

%endsection

%meta cpdef cpu1  romcode: code1, ramsize:8
%meta sodef videomemory    constraint:vtextmem:0:3:3:16:16
%meta soatt videomemory cp: cpu1, index:0
%meta bmdef global registersize:8
```

Basm

Fragments

The Basm fragments are the main feature of the assembler.

They are small pieces of code that can be assembled in different ways to form more complex code and in the end a CP.
T hey can for example:

- Be called as it were a function
- Be rewritten free or use a particular CP hardware (a register)
- Be logically combined with other fragments via metadata to for abstract graphs and . . .
- . . . part of these graphs can be placed in different CPs

Basm

Fragments

The Basm fragments are the main feature of the assembler.

They are small pieces of code that can be assembled in different ways to form more complex code and in the end a CP.
T hey can for example:

- Be called as it were a function
- Be rewritten free or use a particular CP hardware (a register)
- Be logically combined with other fragments via metadata to for abstract graphs and . . .
- . . . part of these graphs can be placed in different CPs

Basm

Fragments

The Basm fragments are the main feature of the assembler.

They are small pieces of code that can be assembled in different ways to form more complex code and in the end a CP.
T hey can for example:

- Be called as it were a function
- Be rewritten free or use a particular CP hardware (a register)
- Be logically combined with other fragments via metadata to for abstract graphs and . . .
- . . . part of these graphs can be placed in different CPs

Basm

Fragments

The Basm fragments are the main feature of the assembler.

They are small pieces of code that can be assembled in different ways to form more complex code and in the end a CP.
T hey can for example:

- Be called as it were a function
- Be rewritten free or use a particular CP hardware (a register)
- Be logically combined with other fragments via metadata to for abstract graphs and . . .
- . . . part of these graphs can be placed in different CPs

Basm

Fragments

The Basm fragments are the main feature of the assembler.

They are small pieces of code that can be assembled in different ways to form more complex code and in the end a CP.
T hey can for example:

- Be called as it were a function
- Be rewritten free or use a particular CP hardware (a register)
- Be logically combined with other fragments via metadata to for abstract graphs and . . .
- . . . part of these graphs can be placed in different CPs

Basm

Templates

Basm support templates. This allows to define generic code that can be instantiated in different ways from the tools that use the assembler.

```
%fragment weight template:true resin:r0 resout:r0
        rset r1,{{.Params.weight}}
        {{.Params.multop}}      r0, r1
%endfragment
```

For example the template above is used to define the weight of a neural network. The multiplication operation is specified via template. When filled with a specific operation the resulting hardware will be optimized for that.

Basm Hands-on

Hands-on N.09

It will be shown how:

- To create a BondMachine from a Basm source file
- To build the accelerator
- To build the xclbin
- To upload the xclbin to the board and use it

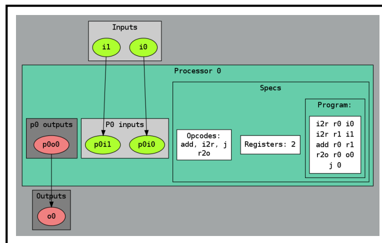
Basm

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 Basm 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.

Builders API

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

Builders API

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

Builders API

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

Builders API

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

Builders API

Symbond

A mathematical expression, or a system can be converted to a BondMachine:

sum(var(x),const(2))

Boolbond

```
symbond -expression "sum(var(x),const(2))" -save-bondmachine bondmachine.json
```

Resulting in:

Builders API

Symbond

A mathematical expression, or a system can be converted to a BondMachine:

sum(var(x),const(2))

Boolbond

```
symbond -expression "sum(var(x),const(2))" -save-bondmachine bondmachine.json
```

Resulting in:

Builders API

Symbond

A mathematical expression, or a system can be converted to a BondMachine:

sum(var(x),const(2))

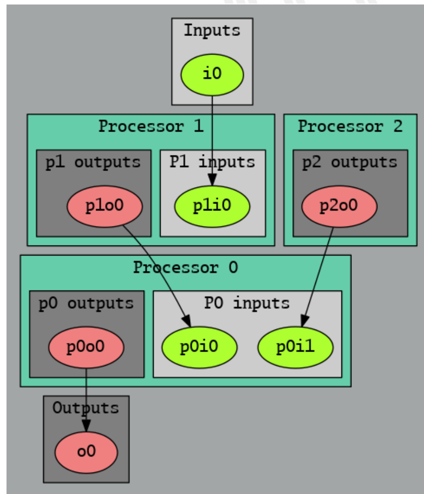
Boolbond

```
symbond -expression "sum(var(x),const(2))" -save-bondmachine bondmachine.json
```

Resulting in:

Builders API

Symbond



Builders API

Boolbond

A system of boolean equations, input and output variables are expressed as in the example file:

```
var(z)=or(var(x),not(var(y)))
var(t)=or(and(var(x),var(y)),var(z))
var(l)=and(xor(var(x),var(y)),var(t))
i:var(x)
i:var(y)
o:var(z)
o:var(t)
o:var(l)
```

Boolbond

```
boolbond -system-file expression.txt -save-bondmachine bondmachine.json
```

Resulting in:

Builders API

Boolbond

A system of boolean equations, input and output variables are expressed as in the example file:

```
var(z)=or(var(x),not(var(y)))
var(t)=or(and(var(x),var(y)),var(z))
var(l)=and(xor(var(x),var(y)),var(t))
i:var(x)
i:var(y)
o:var(z)
o:var(t)
o:var(l)
```

Boolbond

boolbond -system-file expression.txt -save-bondmachine bondmachine.json

Resulting in:

Builders API

Boolbond

A system of boolean equations, input and output variables are expressed as in the example file:

```
var(z)=or(var(x),not(var(y)))
var(t)=or(and(var(x),var(y)),var(z))
var(l)=and(xor(var(x),var(y)),var(t))
i:var(x)
i:var(y)
o:var(z)
o:var(t)
o:var(l)
```

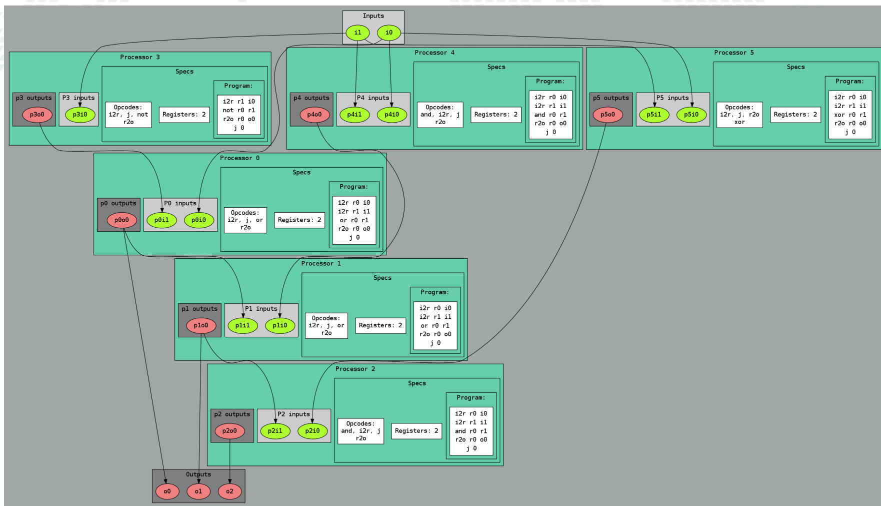
Boolbond

boolbond -system-file expression.txt -save-bondmachine bondmachine.json

Resulting in:

Builders API

Boolbond



Boolbond Hands-on

Hands-on N.10

It will be shown how:

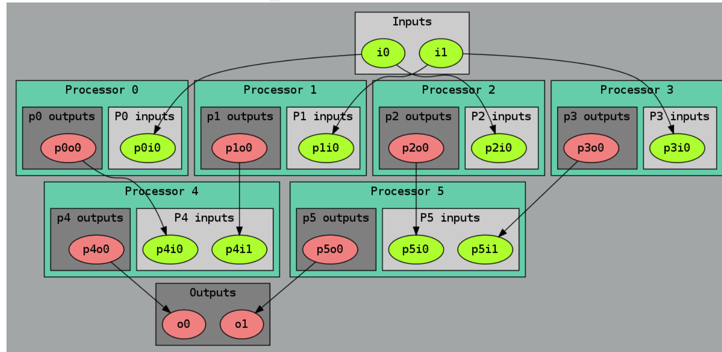
- To create complex multi-cores from boolean expressions

Builders API

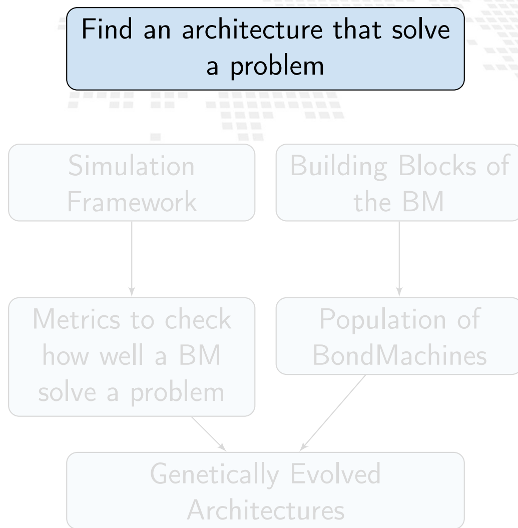
Matrixwork

Matrix multiplication

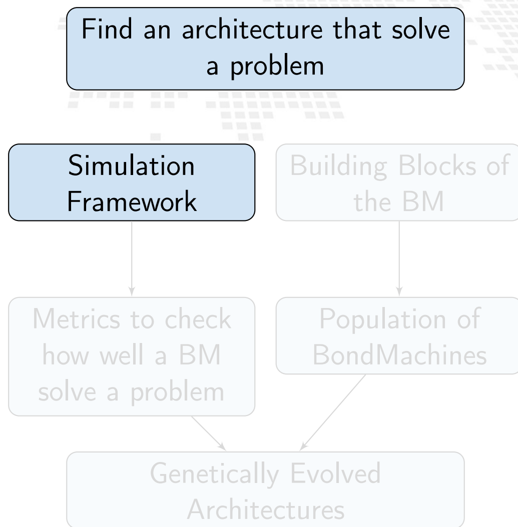
if mymachine, ok := matrixwork.Build_M(n, t); ok == nil ...



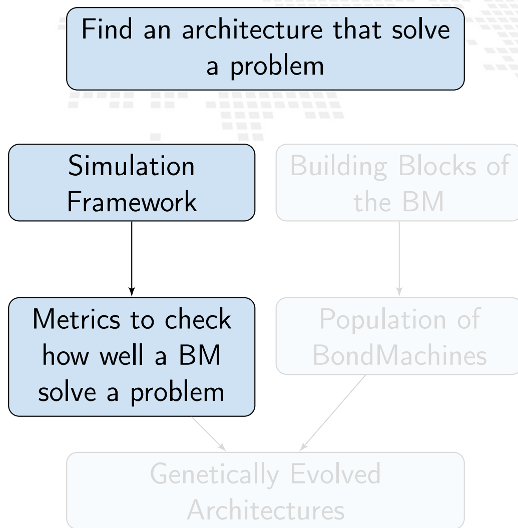
Evolutionary BondMachine



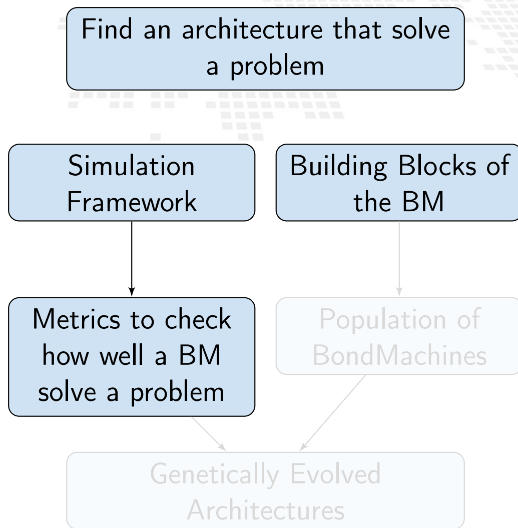
Evolutionary BondMachine



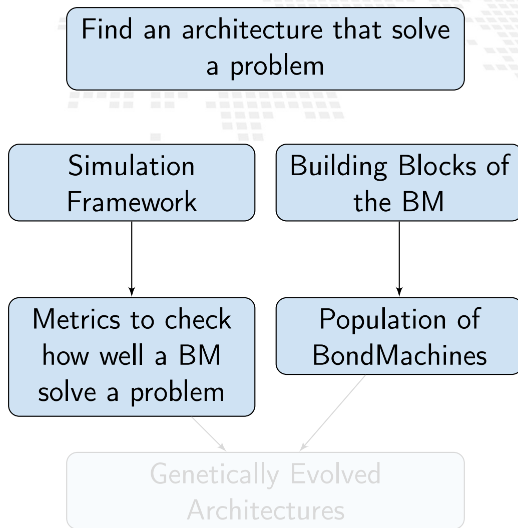
Evolutionary BondMachine



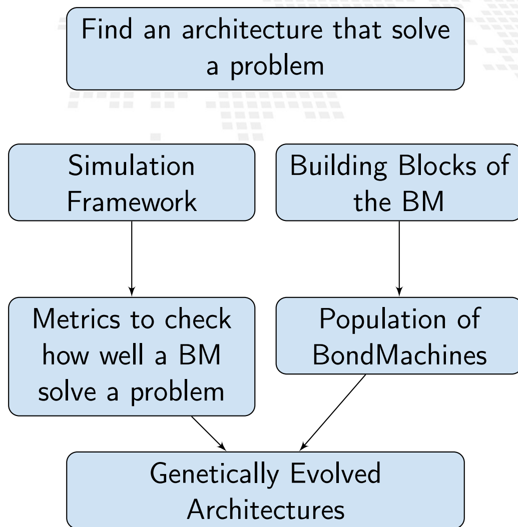
Evolutionary BondMachine



Evolutionary BondMachine



Evolutionary BondMachine



Clustering

1 Introduction
- Challenges
- FPGA
- Architectures
- Abstractions

2 The BondMachine project
- Architectures handling
- Architectures molding
- Bondgo
- Basm
- API

3 Clustering
- An example
- Video
- Distributed architecture

4 Accelerators
- Hardware
- Software
- Tests
- Benchmark

5 Misc
- Project timeline
- Supported boards
- Use cases

6 Machine Learning
- Train
- Benchmark

7 Optimizations
- Softmax example
- Results
- Model's compression
- Fragments compositions

8 Accelerator in a cloud
- Bring it to cloud level: why and how
- Implementing a KServe FPGA extension
- Where are we...

9 Conclusions and Future directions
- Conclusions
- Ongoing
- Future

BondMachine Clustering

So far we saw:

- An user friendly approach to create processors (single core).
- Optimizing a single device to support intricate computational work-flows (multi-cores) over an heterogeneous layer.

Interconnected BondMachines

What if we could extend the this layer to multiple interconnected devices ?

BondMachine Clustering

So far we saw:

- An user friendly approach to create processors (single core).
- Optimizing a single device to support intricate computational work-flows (multi-cores) over an heterogeneous layer.

Interconnected BondMachines

What if we could extend the this layer to multiple interconnected devices ?

BondMachine Clustering

So far we saw:

- An user friendly approach to create processors (single core).
- Optimizing a single device to support intricate computational work-flows (multi-cores) over an heterogeneous layer.

Interconnected BondMachines

What if we could extend the this layer to multiple interconnected devices ?

BondMachine Clustering

So far we saw:

- An user friendly approach to create processors (single core).
- Optimizing a single device to support intricate computational work-flows (multi-cores) over an heterogeneous layer.

Interconnected BondMachines

What if we could extend the this layer to multiple interconnected devices ?

BondMachine Clustering

The same logic existing among CP have been extended among different BondMachines organized in clusters.

Protocols, one ethernet called *etherbond* and one using UDP called *udpbond* have been created for the purpose.

FPGA based BondMachines, standard Linux Workstations, Emulated BondMachines might join a cluster an contribute to a single distributed computational problem.

BondMachine Clustering

The same logic existing among CP have been extended among different BondMachines organized in clusters.

Protocols, one ethernet called *etherbond* and one using UDP called *udpbond* have been created for the purpose.

FPGA based BondMachines, standard Linux Workstations, Emulated BondMachines might join a cluster an contribute to a single distributed computational problem.

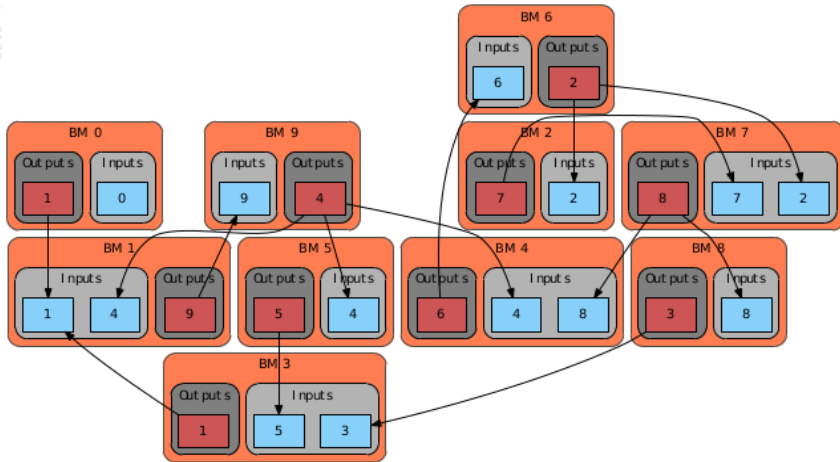
BondMachine Clustering

The same logic existing among CP have been extended among different BondMachines organized in clusters.

Protocols, one ethernet called *etherbond* and one using UDP called *udpbond* have been created for the purpose.

FPGA based BondMachines, standard Linux Workstations, Emulated BondMachines might join a cluster an contribute to a single distributed computational problem.

BondMachine Clustering



BondMachine Clustering

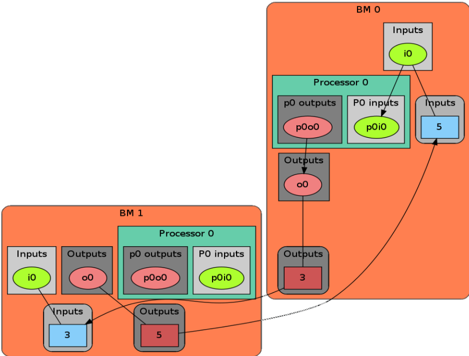
A distributed example

distributed 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.IORead(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_1:
   go pong()
   for {
      bondgo.IOWrite(out0, bondgo.IORead(in0))
   }
}
```

BondMachine Clustering

A distributed example



BondMachine Clustering

A distributed example

See it working:

https://youtube.com/embed/g9xYHK0zca4

A general result

Parts of the system can be redeployed among different devices without changing the system behavior (only the performances).

BondMachine Clustering

Results

Results

- User can deploy an entire HW/SW cluster starting from code written in a high level description (Go, NNEF, etc)
- Workstation with emulated BondMachines, workstation with etherbond drivers, standalone BondMachines (FPGA) may join these clusters.

BondMachine Clustering

Results

Results

- User can deploy an entire HW/SW cluster starting from code written in a high level description (Go, NNEF, etc)
- Workstation with emulated BondMachines, workstation with etherbond drivers, standalone BondMachines (FPGA) may join these clusters.

Accelerators

1 Introduction
- Challenges
- FPGA
- Architectures
- Abstractions

2 The BondMachine project
- Architectures handling
- Architectures molding
- Bondgo
- Basm
- API

3 Clustering
- An example
- Video
- Distributed architecture

4 Accelerators
- Hardware
- Software
- Tests
- Benchmark

5 Misc
- Project timeline
- Supported boards
- Use cases

6 Machine Learning
- Train
- Benchmark

7 Optimizations
- Softmax example
- Results
- Model's compression
- Fragments compositions

8 Accelerator in a cloud
- Bring it to cloud level: why and how
- Implementing a KServe FPGA extension
- Where are we...

9 Conclusions and Future directions
- Conclusions
- Ongoing
- Future

How to bring BM accelerators to the Linux system

Depending on the board, several ways of using BM as accelerators are possible:

- USB connection: BM and host connected via USB. A custom protocol over serial is used to communicate with the board (BMMRP).
- AXI MM on SoC (kernel): The BM and the PS are on the same chip and the communication is done via AXI MM. BMMRP is also used here but implemented in custom kernel module.
- AXI MM on Soc (Pynq): The BM and the PS are on the same chip and the communication is done via AXI MM. The Pynq framework is used for the BM.
- AXI Stream on Soc (Pynq): The BM and the PS are on the same chip and the communication is done via AXI Stream. The Pynq framework is used for the BM.
- AXI Stream on PCIe (Pynq): The BM is connected to the host PC via PCIe and the communication is done via AXI Stream, the XRT platform is used to communicate with the BM via Pynq.

How to bring BM accelerators to the Linux system

Depending on the board, several ways of using BM as accelerators are possible:

- USB connection: BM and host connected via USB. A custom protocol over serial is used to communicate with the board (BMMRP).
- AXI MM on SoC (kernel): The BM and the PS are on the same chip and the communication is done via AXI MM. BMMRP is also used here but implemented in custom kernel module.
- AXI MM on Soc (Pynq): The BM and the PS are on the same chip and the communication is done via AXI MM. The Pynq framework is used for the BM.
- AXI Stream on Soc (Pynq): The BM and the PS are on the same chip and the communication is done via AXI Stream. The Pynq framework is used for the BM.
- AXI Stream on PCIe (Pynq): The BM is connected to the host PC via PCIe and the communication is done via AXI Stream, the XRT platform is used to communicate with the BM via Pynq.

How to bring BM accelerators to the Linux system

Depending on the board, several ways of using BM as accelerators are possible:

- USB connection: BM and host connected via USB. A custom protocol over serial is used to communicate with the board (BMMRP).
- AXI MM on SoC (kernel): The BM and the PS are on the same chip and the communication is done via AXI MM. BMMRP is also used here but implemented in custom kernel module.
- AXI MM on Soc (Pynq): The BM and the PS are on the same chip and the communication is done via AXI MM. The Pynq framework is used for the BM.
- AXI Stream on Soc (Pynq): The BM and the PS are on the same chip and the communication is done via AXI Stream. The Pynq framework is used for the BM.
- AXI Stream on PCIe (Pynq): The BM is connected to the host PC via PCIe and the communication is done via AXI Stream, the XRT platform is used to communicate with the BM via Pynq.

How to bring BM accelerators to the Linux system

Depending on the board, several ways of using BM as accelerators are possible:

- USB connection: BM and host connected via USB. A custom protocol over serial is used to communicate with the board (BMMRP).
- AXI MM on SoC (kernel): The BM and the PS are on the same chip and the communication is done via AXI MM. BMMRP is also used here but implemented in custom kernel module.
- AXI MM on Soc (Pynq): The BM and the PS are on the same chip and the communication is done via AXI MM. The Pynq framework is used for the BM.
- AXI Stream on Soc (Pynq): The BM and the PS are on the same chip and the communication is done via AXI Stream. The Pynq framework is used for the BM.
- AXI Stream on PCIe (Pynq): The BM is connected to the host PC via PCIe and the communication is done via AXI Stream, the XRT platform is used to communicate with the BM via Pynq.

How to bring BM accelerators to the Linux system

Depending on the board, several ways of using BM as accelerators are possible:

- USB connection: BM and host connected via USB. A custom protocol over serial is used to communicate with the board (BMMRP).
- AXI MM on SoC (kernel): The BM and the PS are on the same chip and the communication is done via AXI MM. BMMRP is also used here but implemented in custom kernel module.
- AXI MM on Soc (Pynq): The BM and the PS are on the same chip and the communication is done via AXI MM. The Pynq framework is used for the BM.
- AXI Stream on Soc (Pynq): The BM and the PS are on the same chip and the communication is done via AXI Stream. The Pynq framework is used for the BM.
- AXI Stream on PCIe (Pynq): The BM is connected to the host PC via PCIe and the communication is done via AXI Stream, the XRT platform is used to communicate with the BM via Pynq.

How to bring BM accelerators to the Linux system

Depending on the board, several ways of using BM as accelerators are possible:

- USB connection: BM and host connected via USB. A custom protocol over serial is used to communicate with the board (BMMRP).
- AXI MM on SoC (kernel): The BM and the PS are on the same chip and the communication is done via AXI MM. BMMRP is also used here but implemented in custom kernel module.
- AXI MM on Soc (Pynq): The BM and the PS are on the same chip and the communication is done via AXI MM. The Pynq framework is used for the BM.
- AXI Stream on Soc (Pynq): The BM and the PS are on the same chip and the communication is done via AXI Stream. The Pynq framework is used for the BM.
- AXI Stream on PCIe (Pynq): The BM is connected to the host PC via PCIe and the communication is done via AXI Stream, the XRT platform is used to communicate with the BM via Pynq.

AXI MM on SoC (kernel)

Specs

FPGA

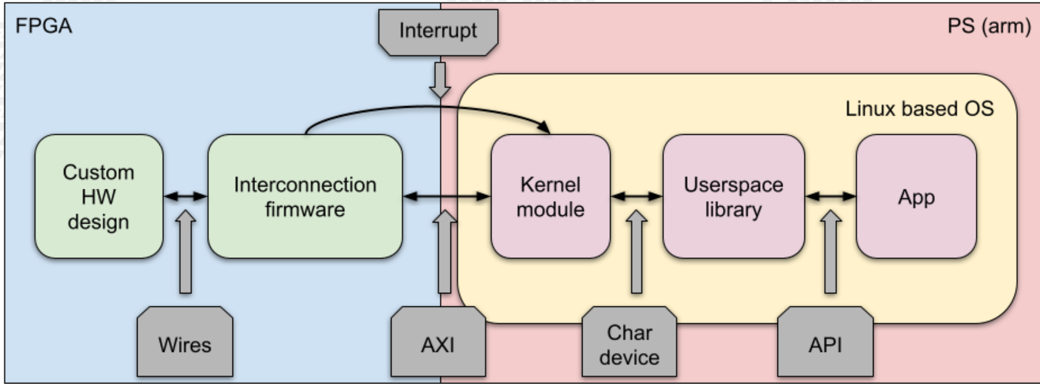
- Digilent Zedboard
- Soc: Zynq XC7Z020-CLG484-1
- 512 MB DDR3
- Vivado 2020.2

Workstations

- Dell Precision Tower 3620
- Intel(R) Xeon(R) CPU E3-1270 v5 @ 3.60GHz
- 16GB Ram
- Golang 1.18.1

- Intel(R) CPU I5-8500 v5 @ 3GHz
- 16GB Ram
- GCC with -O0

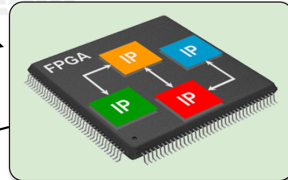
The whole system overview

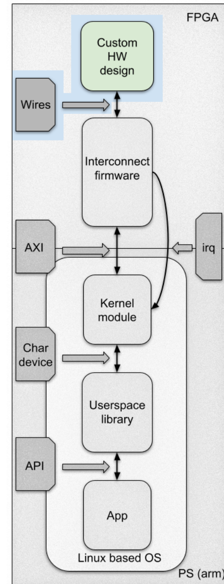


The Accelerator IP

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
```



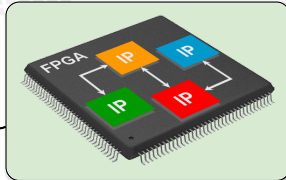


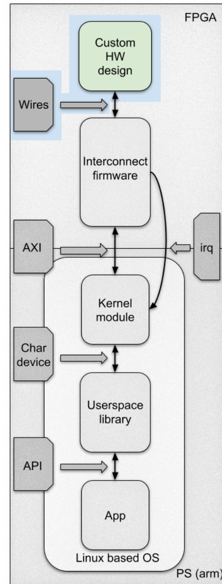
The Accelerator IP

Hardware Description Language

High Level Synthesis

```
template <typename T, int DIM>
void mmult_hw(T A[DIM][DIM], T B[DIM][DIM], T C[DIM][DIM])
{
    // matrix multiplication of a A*B matrix
    L1:for (int ia = 0; ia < DIM; ++ia)
    {
        L2:for (int ib = 0; ib < DIM; ++ib)
        {
            T sum = 0;
            L3:for (int id = 0; id < DIM; ++id)
            {
                sum += A[ia][id] * B[id][ib];
            }
            C[ia][ib] = sum;
        }
    }
}
```



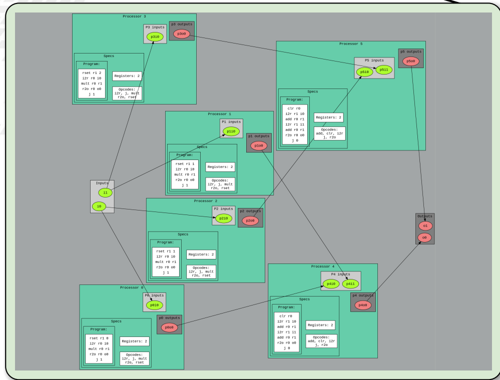


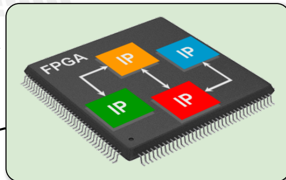
The Accelerator IP

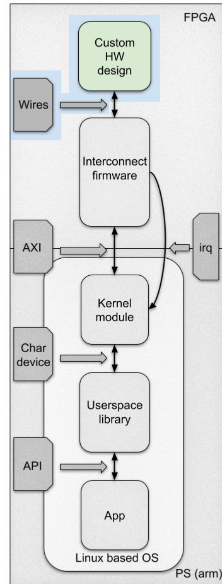
Hardware Description Language

High Level Synthesis

BondMachine





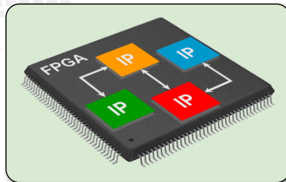


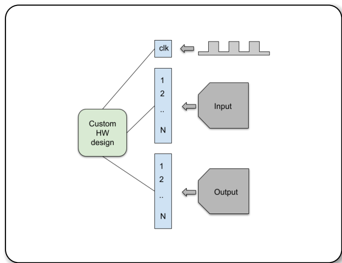
The Accelerator IP

Hardware Description Language

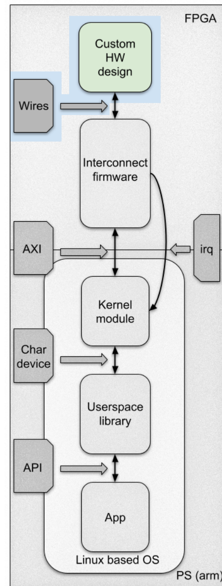
High Level Synthesis

BondMachine



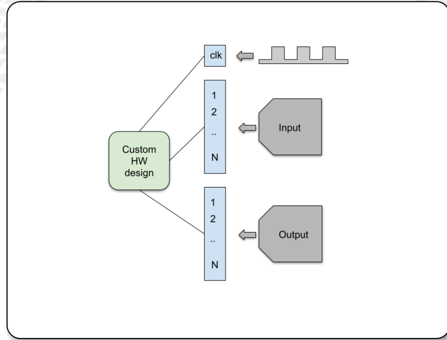


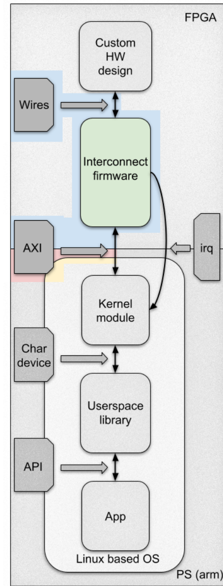
Wires:
- a clock signal,
- an input bus,
- an output bus for the result



Interconnection firmware

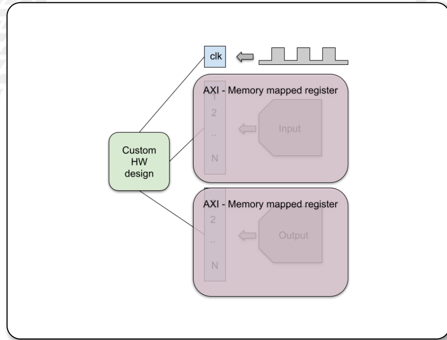
The input and output buses are the endpoints that we would like to have on the linux system.

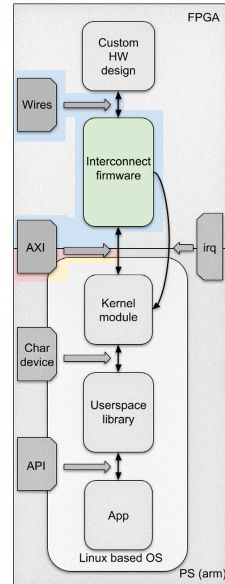




Interconnection firmware

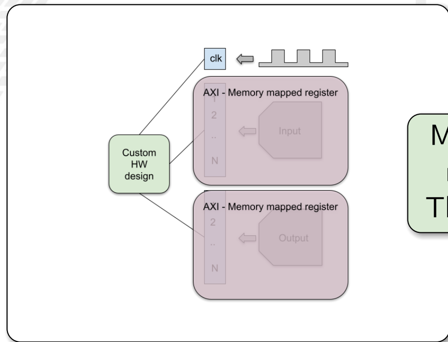
The input and output buses are the endpoints that we would like to have on the linux system.





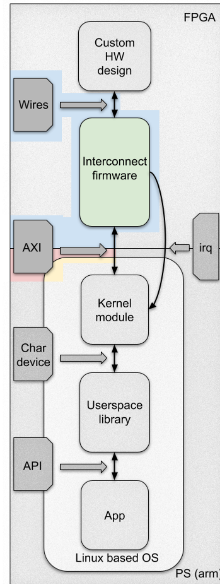
Interconnection firmware

The input and output buses are the endpoints that we would like to have on the linux system.



Memory mapped registers using The AXI protocol

```
wire [31:0] states;
wire [31:0] changes;
wire [31:0] DVDR_PS2PL;
wire [31:0] DVDR_PL2PS;
wire [31:0] port_o0;
wire [31:0] port_o1;
wire [31:0] port_o2;
wire [31:0] port_i0;
wire [31:0] port_i1;
wire [31:0] port_i2;

bondmachine_main bondmachine_inst(
    .clk(S_AXI_ACLK),
    .btnC(btnC),
    .A_DVDR_PS2PL(DVDR_PS2PL),
    .A_DVDR_PL2PS(DVDR_PL2PS),
    .A_changes(changes),
    .A_states(states),
    .A_port_o0(port_o0),
    .A_port_o1(port_o1),
    .A_port_o2(port_o2),
    .A_port_i0(port_i0),
    .A_port_i1(port_i1),
    .A_port_i2(port_i2),
    .interrupt(interrupt)
);
assign port_i0 = slv_reg0[31:0];
assign port_i1 = slv_reg1[31:0];
assign port_i2 = slv_reg2[31:0];
assign DVDR_PS2PL = slv_reg3[31:0];
assign states = slv_reg4[31:0];

always @( posedge S_AXI_ACLK )
begin
    slv_reg5 <= port_o0[31:0];
    slv_reg6 <= port_o1[31:0];
    slv_reg7 <= port_o2[31:0];
    sv_reg8 <= DVDR_PL2PS[31:0];
    slv_reg9 <= changes[31:0];
end
```

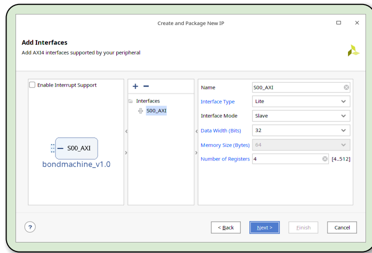


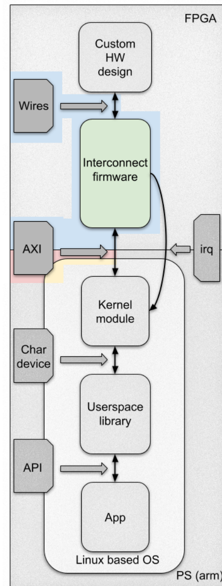
The Advanced eXtensible Interface Protocol

AXI is a communication bus protocol defined by ARM as part of the Advanced Microcontroller Bus Architecture (AMBA) standard.

There are 3 types of AXI Interfaces:

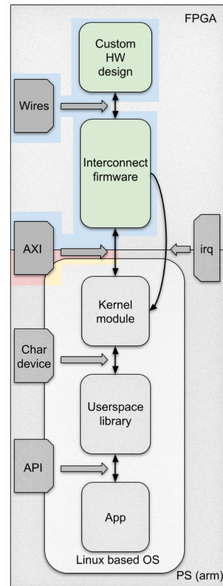
- AXI Full: for high-performance memory-mapped requirements.
- AXI Lite: for low-throughput memory-mapped communication.
- AXI Stream: for high-speed streaming data.





Block Design

rst_ps7_0_100M
slowest_sync_clk
ext_reset_in
aux_reset_in
mb_debug_sys_rst
dcm_locked
mb_reset
bus_struct_reset[0:0]
peripheral_reset[0:0]
interconnect_aresetn[0:0]
peripheral_aresetn[0:0]
Processor System Reset
processing_system7_0
M_AXI_GP0_ACLK
IRQ_F2P[0:0]
ZYNQ
DDR
FIXED_IO
USBIND_0
M_AXI_GP0
TTC0_WAVE0_OUT
TTC0_WAVE1_OUT
TTC0_WAVE2_OUT
FCLK_CLK0
FCLK_RESET0_N
ZYNQ7 Processing System
ps7_0_axi_periph
S00_AXI
ACLK
ARESETN
S00_ACLK
S00_ARESETN
M00_ACLK
M00_ARESETN
M00_AXI
AXI Interconnect
DDR
FIXED_IO
bondmachineip_0
S00_AXI
btnC
s00_axi_aclk
s00_axi_aresetn
interrupt
bondmachineip (Pre-Production)
btnC



Linux

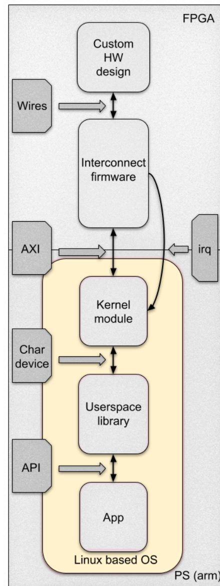
Now that we have a custom accelerated hardware, we need a Linux distro to run on it.

Common Features

Complete system build from source
Allow choice of kernel and bootloader
Support for modifying packages with patches or custom configuration files
Can build cross-toolchains for development
Convenient support for read-only root filesystems
Support offline builds
The build configuration files integrate well with SCM tools

- **Yocto**
  Convenient sharing of build configuration among similar projects (meta-layers)
  Larger community (Linux Foundation project)
  Can build a toolchain that runs on the target
  A package management system
- **Buildroot**
  Simple Makefile approach, easier to understand how the build system works
  Reduced resource requirements on the build machine
  Very easy to customize the final root filesystem (overlays)

Credits: https://jumpnowtek.com/linux/Choosing-an-embedded-linux-build-system.html



Ingredients to build the distro

Kernel config

Linux kernel configuration

Boot Loader config

U-boot Boot loader configuration

Device tree

Data structure describing hardware components

Used by: Kernel Boot loader

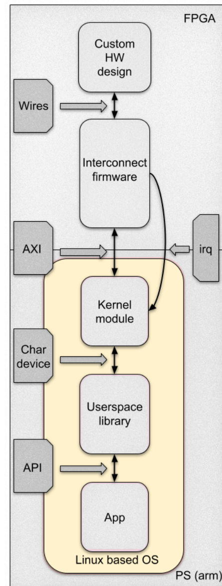
Created by: Vivado + GCC

Bitstream (firmware)

File describing Cells and routing of the FPGA

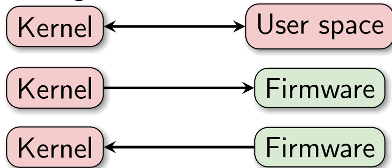
Used by Boot loader to programm FPGA

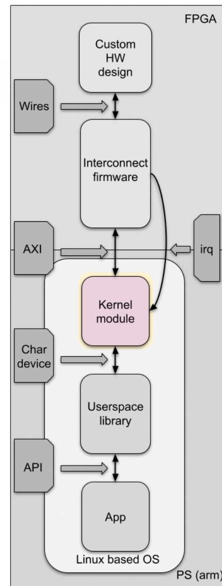
Created by: Vivado



kernel module

- The accelerator endpoints are exposed via AXI memory-mapped as memory location of the arm processor running Linux.
- To properly use the accelerator from user space, the kernel has to handle the accelerator endpoints and make them available to user space.
- We developed a kernel module for our accelerators. It manages 3 data flows:





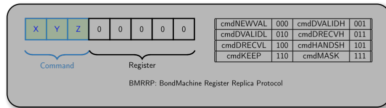
Kernel from and to user space: char device

The communication are through the standard read and write system call on a kernel generated char device

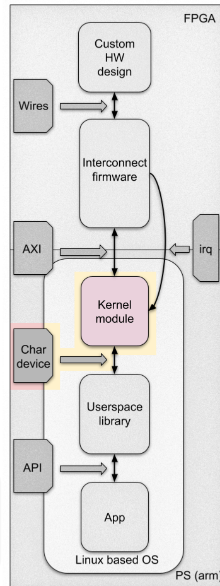
A language has been implemented for the desired operations

```c
static ssize_t bm_read(struct file *filp, char __user *buf, size_t len, loff_t *off)
{
        struct work_data *writework;

        wait_event_interruptible(wait_queue_bm, wait_queue_flag != 0);

        switch (wait_queue_flag)
        {
        case 1:
                switch (bmacc_state)
                {
                case stateHSRECV:
                        if (copy_to_user(buf, &hsmask, 1))
                        {
                                pr_err("Data Read : Err!\n");
                        }
                        //pr_info("Sent MASK");
                        bmacc_state = stateMASKSENT;
                        wait_queue_flag = 0;
                        return 1;
                        brealk
// ....
```

```c
static ssize_t bm_write(struct file *filp, const char __user *buf, size_t len, loff_t *off)
{
        struct work_data *writework;

        if (copy_from_user(write_buffer, buf, len))
        {
                pr_err("data write error\n");
        }
        else
        {
                for (i = 0; i < len; i++)
                {
                        switch (bmacc_state)
                        {
                        case stateWAIT:
                                switch (write_buffer[i] & cmdMASK)
                                {
                                case cmdHANDSH:
                                        //pr_info("HandShake");
                                        hs = write_buffer[i];
                                        bmacc_state = stateHSRECV;
                                        wait_queue_flag = 0;
                                        writework = kmalloc(sizeof(struct work_data), GFP_KERNEL);
                                        INIT_WORK(&writework->work, work_handler);
                                        writework->cs = 1;
                                        queue_work(wq, &writework->work);
                                        break;
```

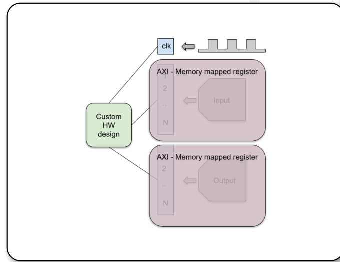


cmdNEWVAL	000	cmdDVALIDH	001
cmdDVALIDL	010	cmdDRECVH	011
cmdDRECVL	100	cmdHANDSH	101
cmdKEEP	110	cmdMASK	111



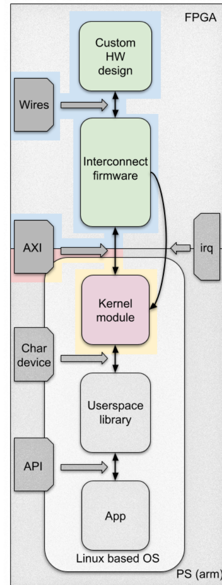
Kernel to firmware

Once the kernel has correctly decoded the data from the char device, it can directly write on AXI registers.



AXI registers are directly written by the kernel

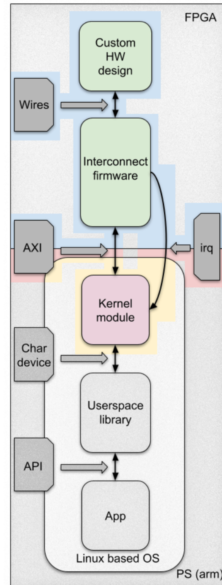
AXI guarantees consistency and transfer to the firmware input ports. Moreover the data flow from kernel cannot saturate the PL part.



Firmware to kernel: IRQ

Different story is the data flow from the FPGA to the PS part. Data can easily flow so fast to saturate and make the PS part completely unusable.

The firmware collect all the changes to send and fill in a list using a dedicated AXI register

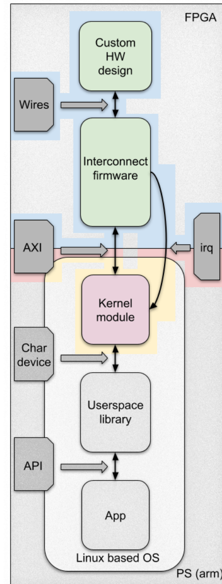


Firmware to kernel: IRQ

Different story is the data flow from the FPGA to the PS part. Data can easily flow so fast to saturate and make the PS part completely unusable.

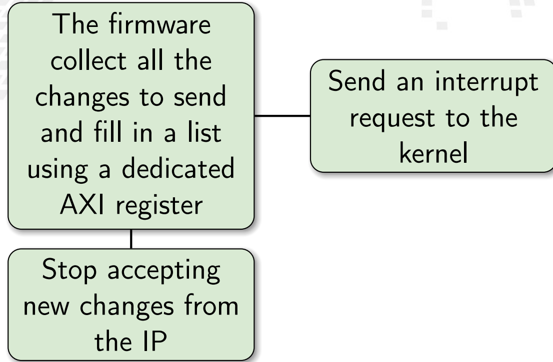
The firmware collect all the changes to send and fill in a list using a dedicated AXI register

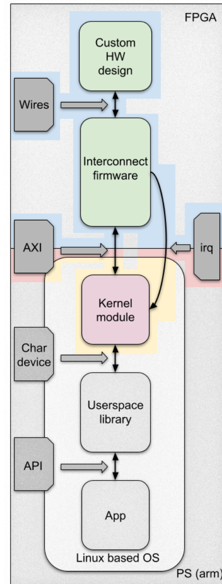
Stop accepting new changes from the IP



Firmware to kernel: IRQ

Different story is the data flow from the FPGA to the PS part. Data can easily flow so fast to saturate and make the PS part completely unusable.





Firmware to kernel: IRQ

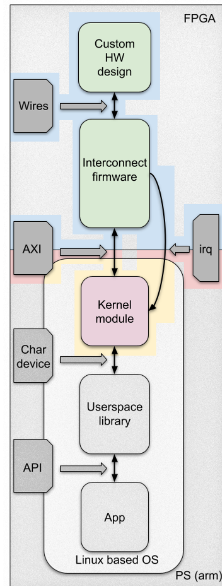
Different story is the data flow from the FPGA to the PS part. Data can easily flow so fast to saturate and make the PS part completely unusable.

The firmware collect all the changes to send and fill in a list using a dedicated AXI register

Send an interrupt request to the kernel

```c
// Interrupt Request number
#define IRQ_NO 46

// Interrupt handler for BM IRQ 46.
static irqreturn_t irq_handler(int irq, void *dev_id)
{
        struct work_data *irqwork;
        if (bmacc_state == stateCONNECT)
        {
                irqwork = kmalloc(sizeof(struct work_data), GFP_KERNEL);
                INIT_WORK(&irqwork->work, work_handler);
                irqwork->cs = 3;
                queue_work(wq, &irqwork->work);
        }
        return IRQ_HANDLED;
}
```

Stop accepting new changes from the IP



Firmware to kernel: IRQ

Different story is the data flow from the FPGA to the PS part. Data can easily flow so fast to saturate and make the PS part completely unusable.

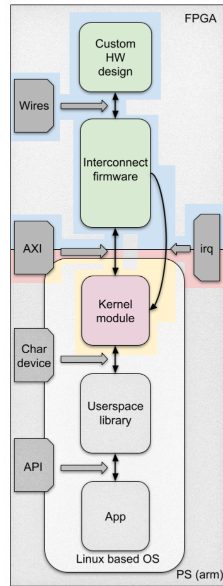
The firmware collect all the changes to send and fill in a list using a dedicated AXI register

Stop accepting new changes from the IP

Send an interrupt request to the kernel

```c
// Interrupt Request number
#define IRQ_NO 46

// Interrupt handler for BM IRQ 46.
static irqreturn_t irq_handler(int irq, void *dev_id)
{
        struct work_data *irqwork;
        if (bmacc_state == stateCONNECT)
        {
                irqwork = kmalloc(sizeof(struct work_data), GFP_KERNEL);
                INIT_WORK(&irqwork->work, work_handler);
                irqwork->cs = 3;
                queue_work(wq, &irqwork->work);
        }
        return IRQ_HANDLED;
}
```

The kernel get the IRQ, read the list of changes and send each of they through the char dev



Firmware to kernel: IRQ

Different story is the data flow from the FPGA to the PS part. Data can easily flow so fast to saturate and make the PS part completely unusable.

The firmware collect all the changes to send and fill in a list using a dedicated AXI register

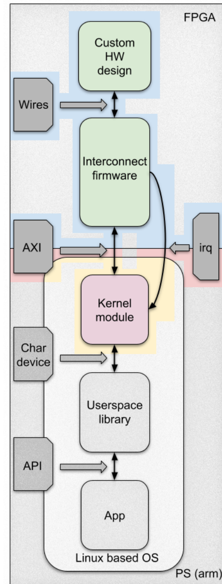
Send an interrupt request to the kernel

```c
// Interrupt Request number
#define IRQ_NO 46

// Interrupt handler for BM IRQ 46.
static irqreturn_t irq_handler(int irq, void *dev_id)
{
        struct work_data *irqwork;
        if (bmacc_state == stateCONNECT)
        {
                irqwork = kmalloc(sizeof(struct work_data), GFP_KERNEL);
                INIT_WORK(&irqwork->work, work_handler);
                irqwork->cs = 3;
                queue_work(wq, &irqwork->work);
        }
        return IRQ_HANDLED;
}
```

The kernel get the IRQ, read the list of changes and send each of they through the char dev

The kernel notify the firmware when done

Stop accepting new changes from the IP



Library

The char device created by the kernel is opened by the BMAPI user space library that implements the BMMRP.

/dev/bm

BMAPI Library

(*BMAPI) BMr2owa

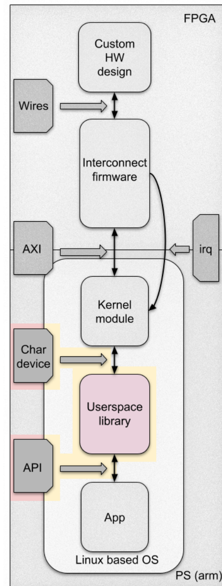
(*BMAPI) BMr2ow

(*BMAPI) BMr2o

(*BMAPI) BMi2rw

(*BMAPI) BMi2r

The library functions can be used by the application



Accelerated application: an example

```go
package main

import (

    "os"
    tr "github.com/BondMachineHQ/bmapiaximmtransceiver"
)   "github.com/BondMachineHQ/fakebmapi"

func main() {

    hwport, present := os.LookupEnv("BMAPIHWPORT")
    if !(present) {
        hwport = "/dev/bm"
    }
    if ba, err := fakebmapi.AcceleratorInit(hwport, tr.AXImmTransceiver); err !=
        log.Printf("AcceleratorInit failed: %v", err)
        return
    } else {
        ba.WaitConnection()
        var icheck uint8 = 0
        for i := 0; i < 256; i++ {
            for {
                if err := ba.BMr2o(0, uint8(i)); err == nil {
                    break
                }
            }
            for {
                if check, err := ba.BMi2r(0); err == nil {
                    icheck = check
                    break
                }
            }
            if uint8(i+1) != icheck {
                log.Printf("Got the wrong responce from FPGA expected %d but go
            } else {
                log.Printf("test: ok expected %d found %d\n", i+1, icheck)
            }
        }
        ba.AcceleratorStop()
}   }
```

Output

1 - the user application through the library sends a value to the accelerator

Input

Input

2 - the user application through the library read the value from the accelerator

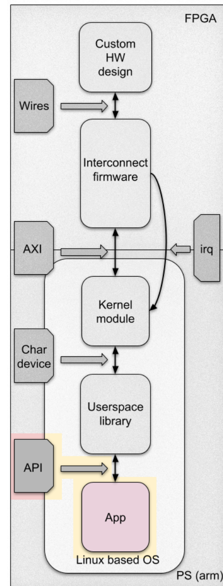
Output

```
1970/01/01 00:23:20 test: ok expected 239 found 239
1970/01/01 00:23:20 test: ok expected 240 found 240
1970/01/01 00:23:20 test: ok expected 241 found 241
1970/01/01 00:23:20 test: ok expected 242 found 242
1970/01/01 00:23:20 test: ok expected 243 found 243
1970/01/01 00:23:20 test: ok expected 244 found 244
1970/01/01 00:23:20 test: ok expected 245 found 245
1970/01/01 00:23:20 test: ok expected 246 found 246
1970/01/01 00:23:21 test: ok expected 247 found 247
1970/01/01 00:23:21 test: ok expected 248 found 248
1970/01/01 00:23:21 test: ok expected 249 found 249
1970/01/01 00:23:21 test: ok expected 250 found 250
1970/01/01 00:23:21 test: ok expected 251 found 251
1970/01/01 00:23:21 test: ok expected 252 found 252
1970/01/01 00:23:21 test: ok expected 253 found 253
1970/01/01 00:23:21 test: ok expected 254 found 254
1970/01/01 00:23:21 test: ok expected 255 found 255
```

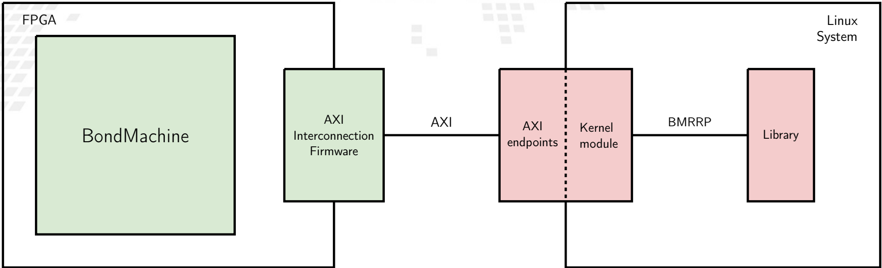
Library

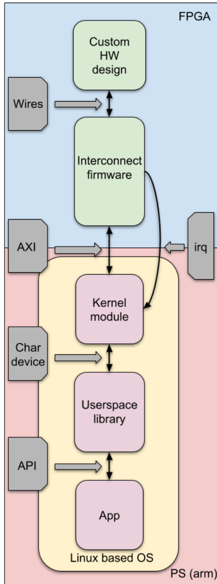
BondMachine

Accelerator



Accelerated Application





An example

- Definition of an example
- Check of the correctness of the accelerator results
- Benchmark of the execution

Squared Matrix-vector multiplication

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} = \left[c_i\right]_{i=1}^{n} = \left[\sum_{k=1}^{n} a_{ik} b_k\right]_{i=1}^{n}$$

Squared Matrix-vector multiplication

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} = \left[c_i\right]_{i=1}^{n} = \left[\sum_{k=1}^{n} a_{ik} b_k\right]_{i=1}^{n}$$


```
"A": [
      [6,5],
      [1,2]
],
"B": [
      [3,1,1],
      [6,7,2],
      [7,1,4]
],
"C": [
      [6,3,7,1],
      [1,6,4,2],
      [3,2,1,7],
      [5,3,1,7]
],
```

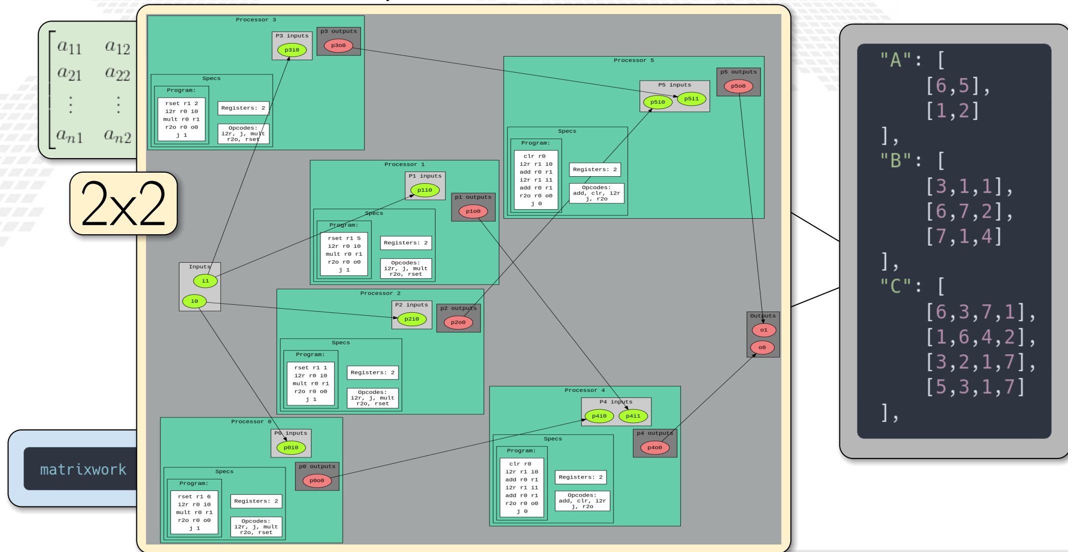
Squared Matrix-vector multiplication

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} = [c_i]_{i=1}^{n} = \left[\sum_{k=1}^{n} a_{ik} b_k\right]_{i=1}^{n}$$

```
"A": [
      [6,5],
      [1,2]
],
"B": [
      [3,1,1],
      [6,7,2],
      [7,1,4]
],
"C": [
      [6,3,7,1],
      [1,6,4,2],
      [3,2,1,7],
      [5,3,1,7]
],
```

```
matrixwork -constants constants.json -constant-matrix A -numerical-type uint8 ...
```

Squared Matrix-vector multiplication

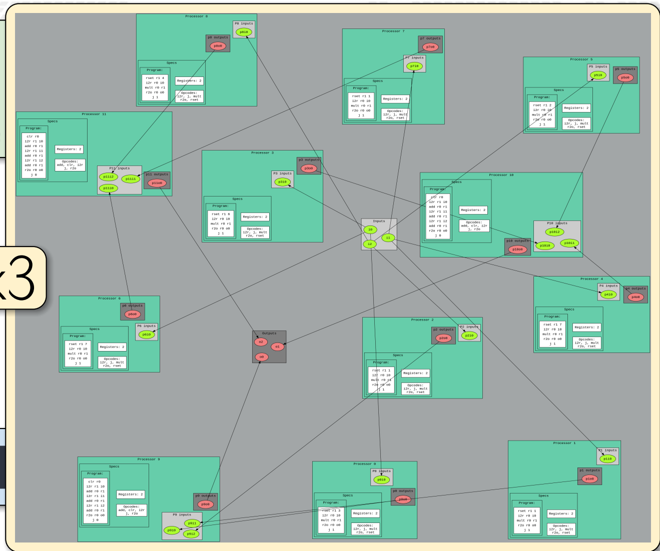


Squared Matrix-vector multiplication

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ \vdots & \vdots \\ a_{n1} & a_{n2} \end{bmatrix}$$

3x3

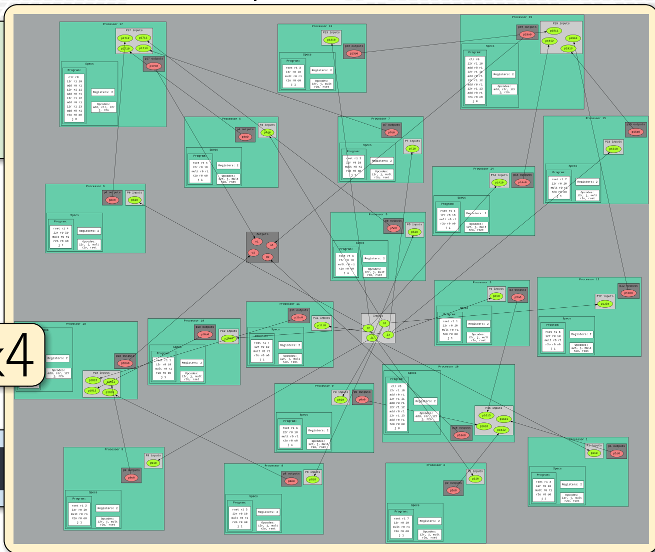
matrixwork



```
"A": [
    [6,5],
    [1,2]
],
"B": [
    [3,1,1],
    [6,7,2],
    [7,1,4]
],
"C": [
    [6,3,7,1],
    [1,6,4,2],
    [3,2,1,7],
    [5,3,1,7]
],
```

Squared Matrix-vector multiplication

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ \vdots & \vdots \\ a_{n1} & a_{n2} \end{bmatrix}$$



4x4

matrixwork

```
"A": [
      [6,5],
      [1,2]
],
"B": [
      [3,1,1],
      [6,7,2],
      [7,1,4]
],
"C": [
      [6,3,7,1],
      [1,6,4,2],
      [3,2,1,7],
      [5,3,1,7]
],
```

Correctness and module debug

To verify the correct computation of the accelerator:

- a tool to monitor the AXI memory
- write directly to AXI memory mapped input addresses (through devmem)
- check the AXI memory mapped output addresses

```
# ./monitor -g 0x43c00000 -n 8
i0:     00000000 (0x43c00003) 00000000 (0x43c00002) 00000000 (0x43c00001) 11111010 (0x43c00000)
i1:     00000000 (0x43c00007) 00000000 (0x43c00006) 00000000 (0x43c00005) 00000000 (0x43c00004)
i2:     00000000 (0x43c0000b) 00000000 (0x43c0000a) 00000000 (0x43c00009) 00000000 (0x43c00008)
i3:     00000000 (0x43c0000f) 00000000 (0x43c0000e) 00000000 (0x43c0000d) 00000000 (0x43c0000c)
i4:     00000000 (0x43c00013) 00000000 (0x43c00012) 00000000 (0x43c00011) 00000000 (0x43c00010)
i5:     00000000 (0x43c00017) 00000000 (0x43c00016) 00000000 (0x43c00015) 00000000 (0x43c00014)
i6:     00000000 (0x43c0001b) 00000000 (0x43c0001a) 00000000 (0x43c00019) 00000000 (0x43c00018)
i7:     00000000 (0x43c0001f) 00000000 (0x43c0001e) 00000000 (0x43c0001d) 00000000 (0x43c0001c)
PS2PL:  00000000 (0x43c00023) 00000000 (0x43c00022) 00000000 (0x43c00021) 00000000 (0x43c00020)
STATES: 00000000 (0x43c00027) 00000000 (0x43c00026) 00000000 (0x43c00025) 00000000 (0x43c00024)
o0:     00000000 (0x43c0002b) 00000000 (0x43c0002a) 00000000 (0x43c00029) 11011100 (0x43c00028)
o1:     00000000 (0x43c0002f) 00000000 (0x43c0002e) 00000000 (0x43c0002d) 11101110 (0x43c0002c)
o2:     00000000 (0x43c00033) 00000000 (0x43c00032) 00000000 (0x43c00031) 11011100 (0x43c00030)
o3:     00000000 (0x43c00037) 00000000 (0x43c00036) 00000000 (0x43c00035) 11101000 (0x43c00034)
o4:     00000000 (0x43c0003b) 00000000 (0x43c0003a) 00000000 (0x43c00039) 11011100 (0x43c00038)
o5:     00000000 (0x43c0003f) 00000000 (0x43c0003e) 00000000 (0x43c0003d) 11100010 (0x43c0003c)
o6:     00000000 (0x43c00043) 00000000 (0x43c00042) 00000000 (0x43c00041) 11110100 (0x43c00040)
o7:     00000000 (0x43c00047) 00000000 (0x43c00046) 00000000 (0x43c00045) 11011100 (0x43c00044)
bench:  00000000 (0x43c0004b) 00000000 (0x43c0004a) 00000000 (0x43c00049) 00011101 (0x43c00048)
PL2PS:  00000000 (0x43c0004f) 11111111 (0x43c0004e) 10000000 (0x43c0004d) 00000000 (0x43c0004c)
CHANGE: 00000000 (0x43c00053) 11111111 (0x43c00052) 11111111 (0x43c00051) 11000000 (0x43c00050)
```

Correctness and module debug

To verify the correct computation of the accelerator:

- a tool to monitor the AXI memory
- write directly to AXI memory mapped input addresses (through devmem)
- check the AXI memory mapped output addresses

```
# ./monitor -g 0x43c00000 -n 8
i0:     00000000 (0x43c00003) 00000000 (0x43c00002) 00000000 (0x43c00001) 11111010 (0x43c00000)
i1:     00000000 (0x43c00007) 00000000 (0x43c00006) 00000000 (0x43c00005) 00000000 (0x43c00004)
i2:     00000000 (0x43c0000b) 00000000 (0x43c0000a) 00000000 (0x43c00009) 00000000 (0x43c00008)
i3:     00000000 (0x43c0000f) 00000000 (0x43c0000e) 00000000 (0x43c0000d) 00000000 (0x43c0000c)
i4:     00000000 (0x43c00013) 00000000 (0x43c00012) 00000000 (0x43c00011) 00000000 (0x43c00010)
i5:     00000000 (0x43c00017) 00000000 (0x43c00016) 00000000 (0x43c00015) 00000000 (0x43c00014)
i6:     00000000 (0x43c0001b) 00000000 (0x43c0001a) 00000000 (0x43c00019) 00000000 (0x43c00018)
i7:     00000000 (0x43c0001f) 00000000 (0x43c0001e) 00000000 (0x43c0001d) 00000000 (0x43c0001c)
PS2PL:  00000000 (0x43c00023) 00000000 (0x43c00022) 00000000 (0x43c00021) 00000000 (0x43c00020)
STATES: 00000000 (0x43c00027) 00000000 (0x43c00026) 00000000 (0x43c00025) 00000000 (0x43c00024)
o0:     00000000 (0x43c0002b) 00000000 (0x43c0002a) 00000000 (0x43c00029) 11011100 (0x43c00028)
o1:     00000000 (0x43c0002f) 00000000 (0x43c0002e) 00000000 (0x43c0002d) 11101110 (0x43c0002c)
o2:     00000000 (0x43c00033) 00000000 (0x43c00032) 00000000 (0x43c00031) 11011100 (0x43c00030)
o3:     00000000 (0x43c00037) 00000000 (0x43c00036) 00000000 (0x43c00035) 11101000 (0x43c00034)
o4:     00000000 (0x43c0003b) 00000000 (0x43c0003a) 00000000 (0x43c00039) 11011100 (0x43c00038)
o5:     00000000 (0x43c0003f) 00000000 (0x43c0003e) 00000000 (0x43c0003d) 11100010 (0x43c0003c)
o6:     00000000 (0x43c00043) 00000000 (0x43c00042) 00000000 (0x43c00041) 11110100 (0x43c00040)
o7:     00000000 (0x43c00047) 00000000 (0x43c00046) 00000000 (0x43c00045) 11011100 (0x43c00044)
bench:  00000000 (0x43c0004b) 00000000 (0x43c0004a) 00000000 (0x43c00049) 00011101 (0x43c00048)
PL2PS:  00000000 (0x43c0004f) 11111111 (0x43c0004e) 10000000 (0x43c0004d) 00000000 (0x43c0004c)
CHANGE: 00000000 (0x43c00053) 11111111 (0x43c00052) 11111111 (0x43c00051) 11000000 (0x43c00050)
```

```
devmem 0x43c00000 b 1
```

Correctness and module debug

To verify the correct computation of the accelerator:

- a tool to monitor the AXI memory
- write directly to AXI memory mapped input addresses (through devmem)
- check the AXI memory mapped output addresses

```
# ./monitor -g 0x43c00000 -n 8
i0:     00000000 (0x43c00003) 00000000 (0x43c00002) 00000000 (0x43c00001) 11111010 (0x43c00000)
i1:     00000000 (0x43c00007) 00000000 (0x43c00006) 00000000 (0x43c00005) 00000000 (0x43c00004)
i2:     00000000 (0x43c0000b) 00000000 (0x43c0000a) 00000000 (0x43c00009) 00000000 (0x43c00008)
i3:     00000000 (0x43c0000f) 00000000 (0x43c0000e) 00000000 (0x43c0000d) 00000000 (0x43c0000c)
i4:     00000000 (0x43c00013) 00000000 (0x43c00012) 00000000 (0x43c00011) 00000000 (0x43c00010)
i5:     00000000 (0x43c00017) 00000000 (0x43c00016) 00000000 (0x43c00015) 00000000 (0x43c00014)
i6:     00000000 (0x43c0001b) 00000000 (0x43c0001a) 00000000 (0x43c00019) 00000000 (0x43c00018)
i7:     00000000 (0x43c0001f) 00000000 (0x43c0001e) 00000000 (0x43c0001d) 00000000 (0x43c0001c)
PS2PL:  00000000 (0x43c00023) 00000000 (0x43c00022) 00000000 (0x43c00021) 00000000 (0x43c00020)
STATES: 00000000 (0x43c00027) 00000000 (0x43c00026) 00000000 (0x43c00025) 00000000 (0x43c00024)
o0:     00000000 (0x43c0002b) 00000000 (0x43c0002a) 00000000 (0x43c00029) 11011100 (0x43c00028)
o1:     00000000 (0x43c0002f) 00000000 (0x43c0002e) 00000000 (0x43c0002d) 11101110 (0x43c0002c)
o2:     00000000 (0x43c00033) 00000000 (0x43c00032) 00000000 (0x43c00031) 11011100 (0x43c00030)
o3:     00000000 (0x43c00037) 00000000 (0x43c00036) 00000000 (0x43c00035) 11101000 (0x43c00034)
o4:     00000000 (0x43c0003b) 00000000 (0x43c0003a) 00000000 (0x43c00039) 11011100 (0x43c00038)
o5:     00000000 (0x43c0003f) 00000000 (0x43c0003e) 00000000 (0x43c0003d) 11100010 (0x43c0003c)
o6:     00000000 (0x43c00043) 00000000 (0x43c00042) 00000000 (0x43c00041) 11110100 (0x43c00040)
o7:     00000000 (0x43c00047) 00000000 (0x43c00046) 00000000 (0x43c00045) 11011100 (0x43c00044)
bench:  00000000 (0x43c0004b) 00000000 (0x43c0004a) 00000000 (0x43c00049) 00011101 (0x43c00048)
PL2PS:  00000000 (0x43c0004f) 11111111 (0x43c0004e) 10000000 (0x43c0004d) 00000000 (0x43c0004c)
CHANGE: 00000000 (0x43c00053) 11111111 (0x43c00052) 11111111 (0x43c00051) 11000000 (0x43c00050)
```

```
devmem 0x43c00000 b 1
```

An example of error

```
#  ./monitor -g 0x43c00000 -n 13
i0:     00000000 (0x43c00003) 00000000 (0x43c00002) 00000000 (0x43c00001) 00000001 (0x43c00000)
i1:     00000000 (0x43c00007) 00000000 (0x43c00006) 00000000 (0x43c00005) 00000000 (0x43c00004)
i2:     00000000 (0x43c0000b) 00000000 (0x43c0000a) 00000000 (0x43c00009) 00000000 (0x43c00008)
i3:     00000000 (0x43c0000f) 00000000 (0x43c0000e) 00000000 (0x43c0000d) 00000000 (0x43c0000c)
i4:     00000000 (0x43c00013) 00000000 (0x43c00012) 00000000 (0x43c00011) 00000000 (0x43c00010)
i5:     00000000 (0x43c00017) 00000000 (0x43c00016) 00000000 (0x43c00015) 00000000 (0x43c00014)
i6:     00000000 (0x43c0001b) 00000000 (0x43c0001a) 00000000 (0x43c00019) 00000000 (0x43c00018)
i7:     00000000 (0x43c0001f) 00000000 (0x43c0001e) 00000000 (0x43c0001d) 00000000 (0x43c0001c)
i8:     00000000 (0x43c00023) 00000000 (0x43c00022) 00000000 (0x43c00021) 00000000 (0x43c00020)
i9:     00000000 (0x43c00027) 00000000 (0x43c00026) 00000000 (0x43c00025) 00000000 (0x43c00024)
i10:    00000000 (0x43c0002b) 00000000 (0x43c0002a) 00000000 (0x43c00029) 00000000 (0x43c00028)
i11:    00000000 (0x43c0002f) 00000000 (0x43c0002e) 00000000 (0x43c0002d) 00000000 (0x43c0002c)
i12:    00000000 (0x43c00033) 00000000 (0x43c00032) 00000000 (0x43c00031) 00000000 (0x43c00030)
PS2PL:  00000000 (0x43c00037) 00000000 (0x43c00036) 00000000 (0x43c00035) 00000000 (0x43c00034)
STATES: 00000000 (0x43c0003b) 00000000 (0x43c0003a) 00000000 (0x43c00039) 00000000 (0x43c00038)
o0:     00000000 (0x43c0003f) 00000000 (0x43c0003e) 00000000 (0x43c0003d) 00000111 (0x43c0003c)
o1:     00000000 (0x43c00043) 00000000 (0x43c00042) 00000000 (0x43c00041) 00000110 (0x43c00040)
o2:     00000000 (0x43c00047) 00000000 (0x43c00046) 00000000 (0x43c00045) 00000110 (0x43c00044)
o3:     00000000 (0x43c0004b) 00000000 (0x43c0004a) 00000000 (0x43c00049) 00000100 (0x43c00048)
o4:     00000000 (0x43c0004f) 00000000 (0x43c0004e) 00000000 (0x43c0004d) 00000001 (0x43c0004c)
o5:     00000000 (0x43c00053) 00000000 (0x43c00052) 00000000 (0x43c00051) 00110100 (0x43c00050)
o6:     00000000 (0x43c00057) 00000000 (0x43c00056) 00000000 (0x43c00055) 00000010 (0x43c00054)
o7:     00000000 (0x43c0005b) 00000000 (0x43c0005a) 00000000 (0x43c00059) 00000010 (0x43c00058)
o8:     00000000 (0x43c0005f) 00000000 (0x43c0005e) 00000000 (0x43c0005d) 00000100 (0x43c0005c)
o9:     00000000 (0x43c00063) 00000000 (0x43c00062) 00000000 (0x43c00061) 00000011 (0x43c00060)
o10:    00000000 (0x43c00067) 00000000 (0x43c00066) 00000000 (0x43c00065) 00000010 (0x43c00064)
o11:    00000000 (0x43c0006b) 00000000 (0x43c0006a) 00000000 (0x43c00069) 00000110 (0x43c00068)
o12:    00000000 (0x43c0006f) 00000000 (0x43c0006e) 00000000 (0x43c0006d) 00000011 (0x43c0006c)
o13 bcm 00000000 (0x43c00073) 00000000 (0x43c00072) 00000000 (0x43c00071) 00000101 (0x43c00070)
PL2PS:  00000000 (0x43c00077) 00000111 (0x43c00076) 11111111 (0x43c00075) 11100000 (0x43c00074)
CHANGE: 00000000 (0x43c0007b) 00000111 (0x43c0007a) 11111111 (0x43c00079) 11111111 (0x43c00078)
```

An example of error

```
# ./monitor -g 0x43c00000 -n 13
i0:     00000000 (0x43c00003) 00000000 (0x43c00002) 00000000 (0x43c00001) 00000001 (0x43c00000)
i1:     00000000 (0x43c00007) 00000000 (0x43c00006) 00000000 (0x43c00005) 00000000 (0x43c00004)
i2:     00000000 (0x43c0000b) 00000000 (0x43c0000a) 00000000 (0x43c00009) 00000000 (0x43c00008)
i3:     00000000 (0x43c0000f) 00000000 (0x43c0000e) 00000000 (0x43c0000d) 00000000 (0x43c0000c)
i4:     00000000 (0x43c00013) 00000000 (0x43c00012) 00000000 (0x43c00011) 00000000 (0x43c00010)
i5:     00000000 (0x43c00017) 00000000 (0x43c00016) 00000000 (0x43c00015) 00000000 (0x43c00014)
i6:     00000000 (0x43c0001b) 00000000 (0x43c0001a) 00000000 (0x43c00019) 00000000 (0x43c00018)
i7:     00000000 (0x43c0001f) 00000000 (0x43c0001e) 00000000 (0x43c0001d) 00000000 (0x43c0001c)
i8:     00000000 (0x43c00023) 00000000 (0x43c00022) 00000000 (0x43c00021) 00000000 (0x43c00020)
i9:     00000000 (0x43c00027) 00000000 (0x43c00026) 00000000 (0x43c00025) 00000000 (0x43c00024)
i10:    00000000 (0x43c0002b) 00000000 (0x43c0002a) 00000000 (0x43c00029) 00000000 (0x43c00028)
i11:    00000000 (0x43c0002f) 00000000 (0x43c0002e) 00000000 (0x43c0002d) 00000000 (0x43c0002c)
i12:    00000000 (0x43c00033) 00000000 (0x43c00032) 00000000 (0x43c00031) 00000000 (0x43c00030)
PS2PL:  00000000 (0x43c00037) 00000000 (0x43c00036) 00000000 (0x43c00035) 00000000 (0x43c00034)
STATES: 00000000 (0x43c0003b) 00000000 (0x43c0003a) 00000000 (0x43c00039) 00000000 (0x43c00038)
o0:     00000000 (0x43c0003f) 00000000 (0x43c0003e) 00000000 (0x43c0003d) 00000111 (0x43c0003c)
o1:     00000000 (0x43c00043) 00000000 (0x43c00042) 00000000 (0x43c00041) 00000110 (0x43c00040)
o2:     00000000 (0x43c00047) 00000000 (0x43c00046) 00000000 (0x43c00045) 00000110 (0x43c00044)
o3:     00000000 (0x43c0004b) 00000000 (0x43c0004a) 00000000 (0x43c00049) 00000100 (0x43c00048)
o4:     00000000 (0x43c0004f) 00000000 (0x43c0004e) 00000000 (0x43c0004d) 00000001 (0x43c0004c)
o5:     00000000 (0x43c00053) 00000000 (0x43c00052) 00000000 (0x43c00051) 00110100 (0x43c00050)
o6:     00000000 (0x43c00057) 00000000 (0x43c00056) 00000000 (0x43c00055) 00000010 (0x43c00054)
o7:     00000000 (0x43c0005b) 00000000 (0x43c0005a) 00000000 (0x43c00059) 00000010 (0x43c00058)
o8:     00000000 (0x43c0005f) 00000000 (0x43c0005e) 00000000 (0x43c0005d) 00000100 (0x43c0005c)
o9:     00000000 (0x43c00063) 00000000 (0x43c00062) 00000000 (0x43c00061) 00000011 (0x43c00060)
o10:    00000000 (0x43c00067) 00000000 (0x43c00066) 00000000 (0x43c00065) 00000010 (0x43c00064)
o11:    00000000 (0x43c0006b) 00000000 (0x43c0006a) 00000000 (0x43c00069) 00000110 (0x43c00068)
o12:    00000000 (0x43c0006f) 00000000 (0x43c0006e) 00000000 (0x43c0006d) 00000011 (0x43c0006c)
o13 bcm 00000000 (0x43c00073) 00000000 (0x43c00072) 00000000 (0x43c00071) 00000101 (0x43c00070)
PL2PS:  00000000 (0x43c00077) 00000111 (0x43c00076) 11111111 (0x43c00075) 11100000 (0x43c00074)
CHANGE: 00000000 (0x43c0007b) 00000111 (0x43c0007a) 11111111 (0x43c00079) 11111111 (0x43c00078)
```

```
o0    7    7
o1    6    6

o10        6
o11        4
o12        1
o13        52

o2    2    2
o3    2    2
o4    4    4
o5    3    3
o6    2    2
o7    6    6
o8    3    3
o9    5    5
o10   6
o11   4
o12   1
o13   52
```

Benchmark: caveats

This is a preliminary work.

We trust some tools:

- Vivado reports
- perf

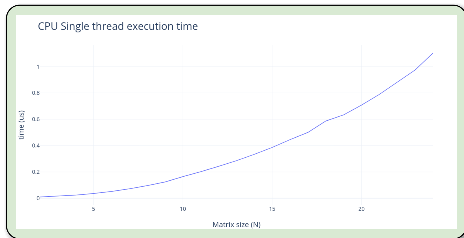
The FPGA benchmarks do not include the PS part overhead (the comparisons are not really fair)

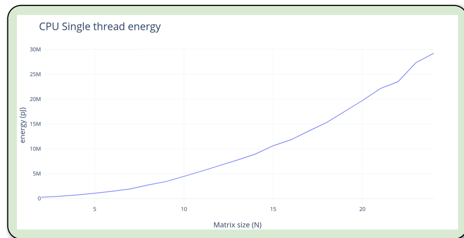
Benchmark: the CPU (Golang)

```go
func matrixtest(n int, iter int64) float32 {
//...
    start := time.Now()

    for k := 0; int64(k) < iter; k++ {
        for i := 0; i < n; i++ {
            output[i] = uint8(0)
        }

        for i := 0; i < n; i++ {
            for j := 0; j < n; j++ {
                output[i] += input[j] * matrix[i+j*n]
            }
        }
    }
    return float32(time.Since(start).Microseconds()) / float32(iter)
}
func main() {
    for i := 2; i <= 32; i++ {
        fmt.Println(i, ",", matrixtest(i, 100000000))
    }
}
```

- Time measures: built-in golang facilities
- Energy measures: perf
- Intel(R) Xeon(R) CPU E3-1270 v5 @ 3.60GHz
- Go 1.18.2

	N	single op time (us)	single op energy (pJ)	Energy eff
1	2	0.00940269	259200	3.85802E-06
2	3	0.01631968	454200	2.20167E-06
3	4	0.02399984	722200	1.38466E-06
4	5	0.03632906	1070400	9.3423E-07
5	6	0.05176583	1471400	6.79625E-07
6	7	0.07183811	1935800	5.16582E-07
7	8	0.09597738	2737000	3.65364E-07
8	9	0.12337612	3429200	2.91613E-07
9	10	0.16450378	4465900	2.23919E-07
10	11	0.20173332	5530300	1.80822E-07
11	12	0.24265632	6643300	1.50528E-07
12	13	0.28554472	7762600	1.28823E-07
13	14	0.33400026	8954000	1.11682E-07
14	15	0.3861176	10633500	9.40424E-08
15	16	0.44550508	11832200	8.45151E-08
16	17	0.5004054	13604700	7.3504E-08
17	18	0.5867083	15324500	6.5255E-08
18	19	0.63376665	17524400	5.70633E-08
19	20	0.708154	19716100	5.072E-08
20	21	0.7893206	22133800	4.51798E-08
21	22	0.8830365	23525100	4.25078E-08
22	23	0.97467226	27346700	3.65675E-08
23	24	1.1031791	29235100	3.42055E-08

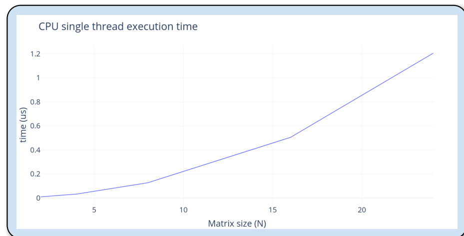


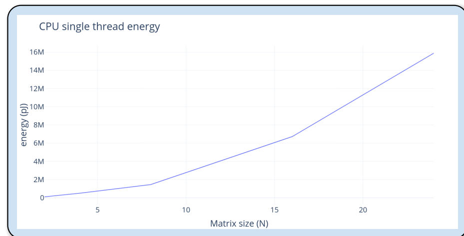


Benchmark: the CPU (C)

- Time measures: time
- Energy measures: perf
- Intel(R) CPU I5-8500 v5 @ 3GHz
- gcc with -O0

	n	single op energy (pJ)	single op time (us)	energy eff
1	2	100000	0.01	0.000008333333333
2	4	500000	0.033	0.000002702702703
3	8	1450000	0.127	0.0000005524861878
4	16	6720000	0.505	0.0000001326259947
5	24	15880000	1.205	0.00000006854009596



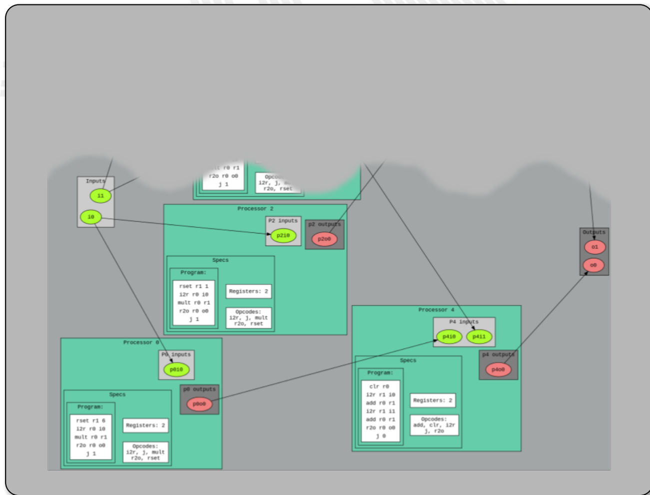


Benchmark: the FPGA

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.

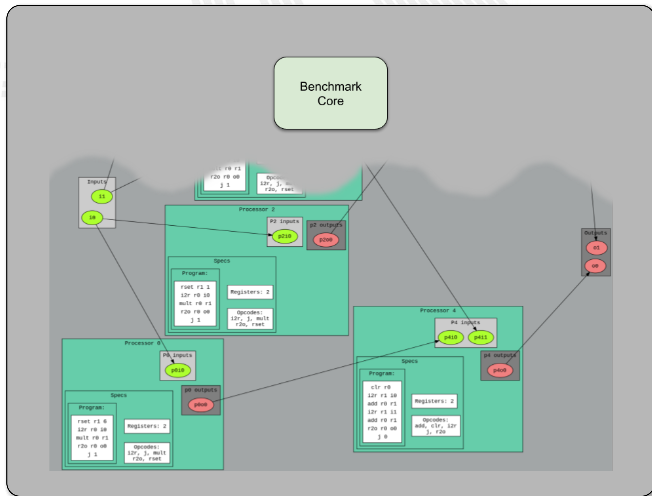


Benchmark: the FPGA

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.

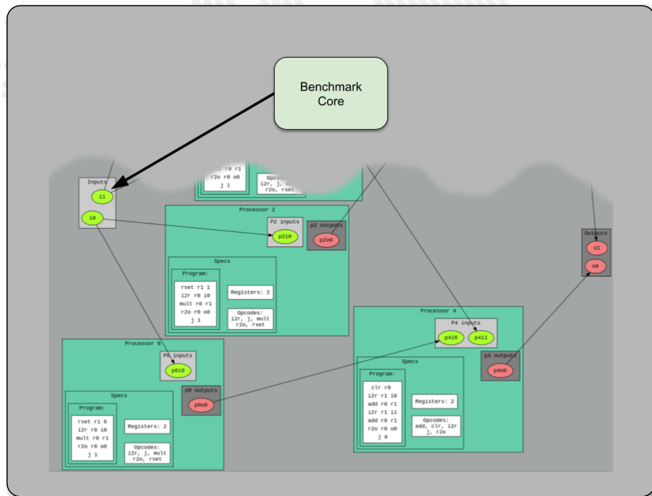


Benchmark: the FPGA

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.

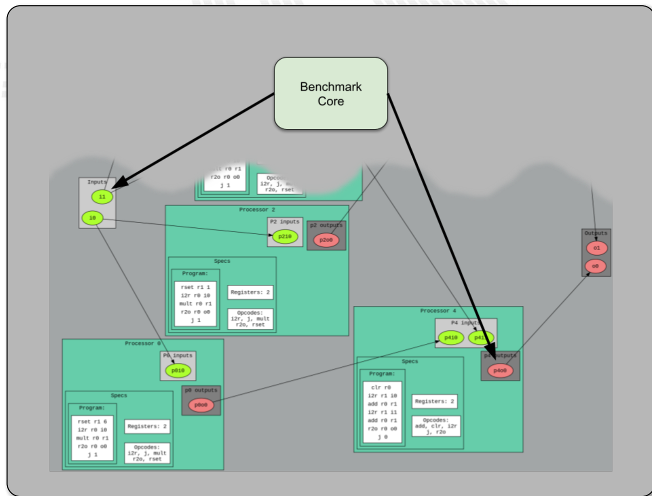


Benchmark: the FPGA

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.

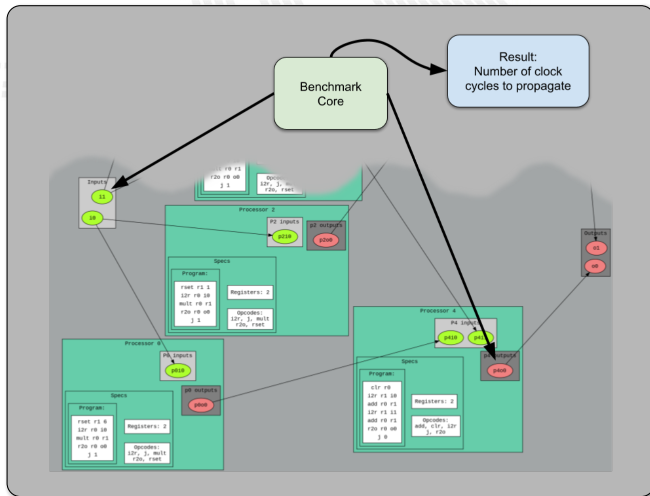


Benchmark: the FPGA

Benchmark an IP is not an easy task.

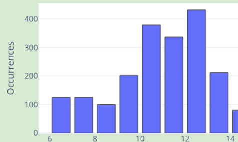
Fortunately we have a custom design and an FPGA.

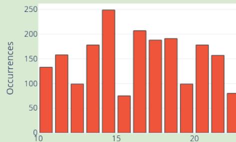
We can put the benchmarks tool inside the accelerator.

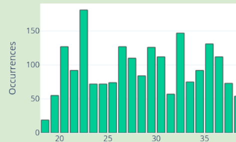


Benchmark core clock cycles distributions

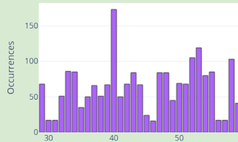
Clock cycles distributions

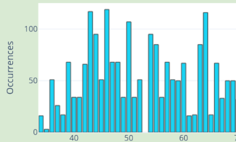


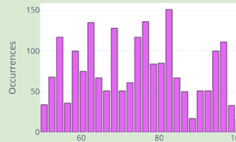




- 2x2
- 4x4
- 8x8
- 13x13
- 16x16
- 24x24



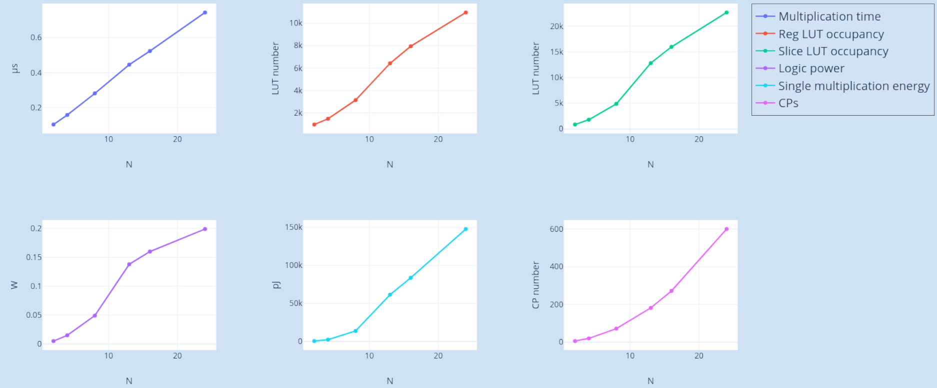




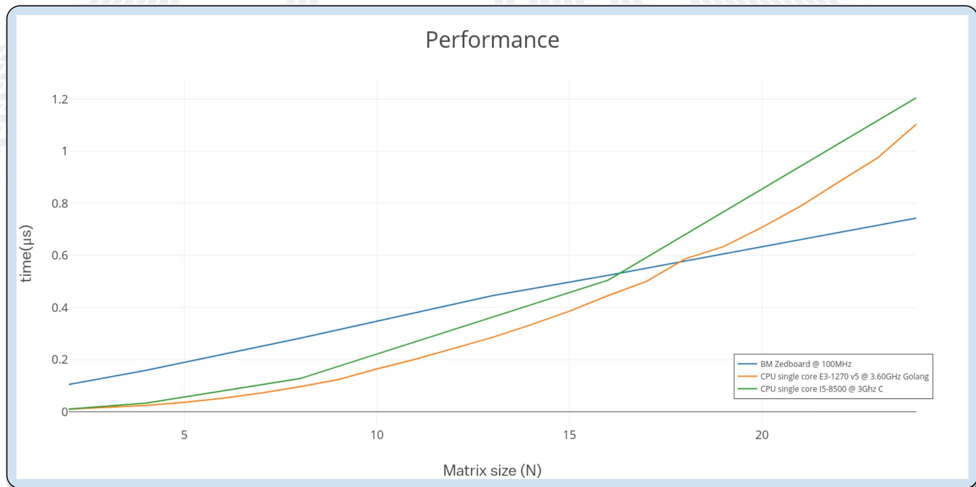
FPGA benchmark summary

	N	single op time (us)	Register LUTs	Slice LUTs	Power	single op energy (pJ)	CPs
1	2	0.1044	947	875	0.005	522	6
2	4	0.1587	1457	1813	0.015	2380.5	20
3	8	0.2819	3131	4897	0.049	13813.1	72
4	13	0.4456	6422	12819	0.138	61492.8	182
5	16	0.5234	7950	15979	0.160	83744	272
6	24	0.7432	10974	22669	0.199	147896.8	600

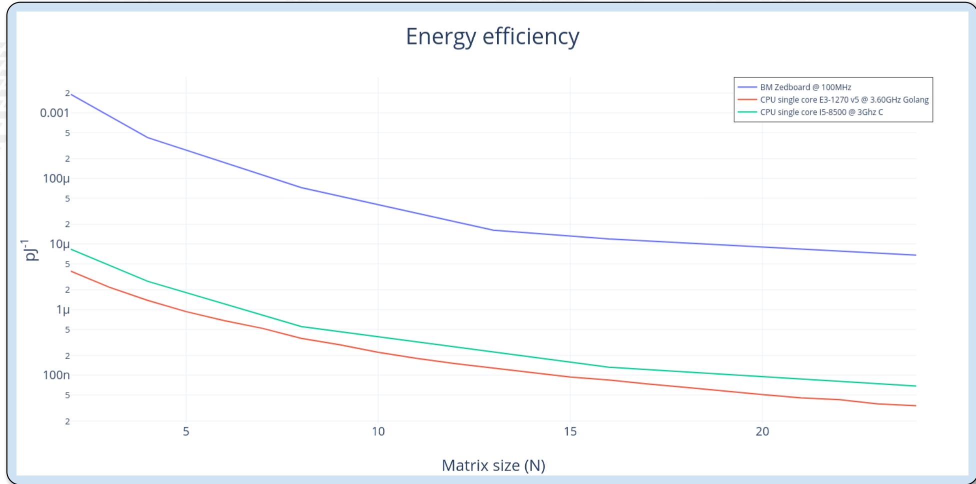
Benchmark core



Comparisons: Performace



Comparisons: Energy



Misc

1 Introduction
- Challenges
- FPGA
- Architectures
- Abstractions

2 The BondMachine project
- Architectures handling
- Architectures molding
- Bondgo
- Basm
- API

3 Clustering
- An example
- Video
- Distributed architecture

4 Accelerators
- Hardware
- Software
- Tests
- Benchmark

5 Misc
- Project timeline
- Supported boards
- Use cases

6 Machine Learning
- Train
- Benchmark

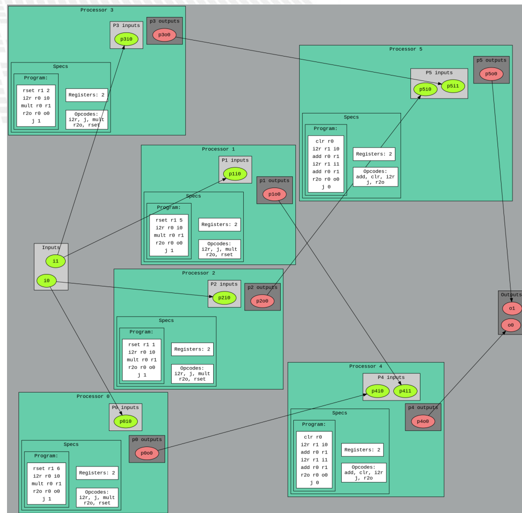
7 Optimizations
- Softmax example
- Results
- Model's compression
- Fragments compositions

8 Accelerator in a cloud
- Bring it to cloud level: why and how
- Implementing a KServe FPGA extension
- Where are we...

9 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.

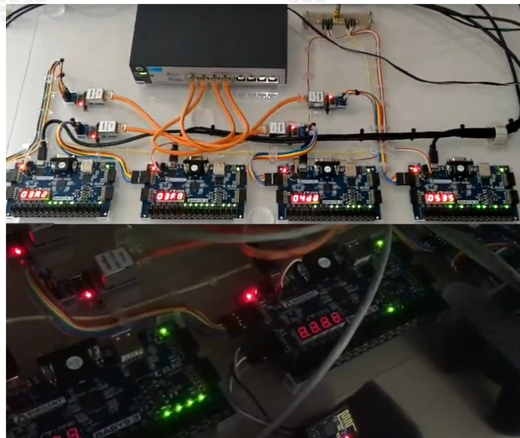
BondMachine recap

- 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 2022 2023 Talk
- InnovateFPGA 2018 Iron Award, Grand Final at Intel Campus (CA) USA
- Invited lectures: "Advanced Workshop on Modern FPGA Based Technology for Scientific Computing", ICTP 2019 and 2022
- Invited lectures: "NiPS Summer School 2019"
- Golab 2018 talk
- Several other talks and posters, ISGC 2019, SOSC 2022, 2023, INFN ML Hackathon 2022
- Article published on Parallel Computing, Elsevier 2022

Project timeline

- CCR 2015 First ideas, 2016 Poster, 2017 2022 2023 Talk
- InnovateFPGA 2018 Iron Award, Grand Final at Intel Campus (CA) USA
- Invited lectures: "Advanced Workshop on Modern FPGA Based Technology for Scientific Computing", ICTP 2019 and 2022
- Invited lectures: "NiPS Summer School 2019"
- Golab 2018 talk
- Several other talks and posters, ISGC 2019, SOSC 2022, 2023, INFN ML Hackathon 2022
- Article published on Parallel Computing, Elsevier 2022

Project timeline

- CCR 2015 First ideas, 2016 Poster, 2017 2022 2023 Talk
- InnovateFPGA 2018 Iron Award, Grand Final at Intel Campus (CA) USA
- Invited lectures: "Advanced Workshop on Modern FPGA Based Technology for Scientific Computing", ICTP 2019 and 2022
- Invited lectures: "NiPS Summer School 2019"
- Golab 2018 talk
- Several other talks and posters, ISGC 2019, SOSC 2022, 2023, INFN ML Hackathon 2022
- Article published on Parallel Computing, Elsevier 2022

Project timeline

- CCR 2015 First ideas, 2016 Poster, 2017 2022 2023 Talk
- InnovateFPGA 2018 Iron Award, Grand Final at Intel Campus (CA) USA
- Invited lectures: "Advanced Workshop on Modern FPGA Based Technology for Scientific Computing", ICTP 2019 and 2022
- Invited lectures: "NiPS Summer School 2019"
- Golab 2018 talk
- Several other talks and posters, ISGC 2019, SOSC 2022, 2023, INFN ML Hackathon 2022
- Article published on Parallel Computing, Elsevier 2022

The BondMachine Toolkit
Enabling Machine Learning on FPGA

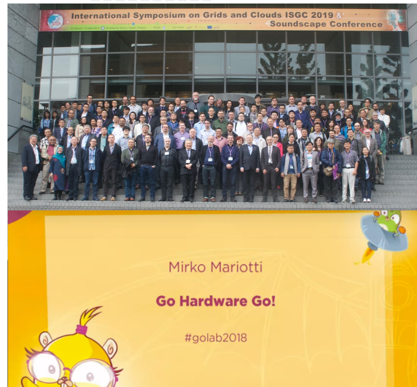
Mirko Mariotti

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 2022 2023 Talk
- InnovateFPGA 2018 Iron Award, Grand Final at Intel Campus (CA) USA
- Invited lectures: "Advanced Workshop on Modern FPGA Based Technology for Scientific Computing", ICTP 2019 and 2022
- Invited lectures: "NiPS Summer School 2019"
- Golab 2018 talk
- Several other talks and posters, ISGC 2019, SOSC 2022, 2023, INFN ML Hackathon 2022
- Article published on Parallel Computing, Elsevier 2022



Project timeline

- CCR 2015 First ideas, 2016 Poster, 2017 2022 2023 Talk
- InnovateFPGA 2018 Iron Award, Grand Final at Intel Campus (CA) USA
- Invited lectures: "Advanced Workshop on Modern FPGA Based Technology for Scientific Computing", ICTP 2019 and 2022
- Invited lectures: "NiPS Summer School 2019"
- Golab 2018 talk
- Several other talks and posters, ISGC 2019, SOSC 2022, 2023, INFN ML Hackathon 2022
- Article published on Parallel Computing, Elsevier 2022

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, b]

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 2022 2023 Talk
- InnovateFPGA 2018 Iron Award, Grand Final at Intel Campus (CA) USA
- Invited lectures: "Advanced Workshop on Modern FPGA Based Technology for Scientific Computing", ICTP 2019 and 2022
- Invited lectures: "NiPS Summer School 2019"
- Golab 2018 talk
- Several other talks and posters, ISGC 2019, SOSC 2022, 2023, INFN ML Hackathon 2022
- Article published on Parallel Computing, Elsevier 2022

Fabrics

The HDL code for the BondMachine has been tested on these devices/system:

- Digilent Basys3 - Xilinx Artix-7 - Vivado
- Kintex7 Evaluation Board - Vivado
- Digilent Zedboard and ebaz4205- Xilinx Zynq 7020 - Vivado
- ZC702 - Xilinx Zynq 7020 - Vivado
- Alveo boards - Xilinx - Vivado/Vitis
- Linux - Iverilog
- ice40lp1k icefun icebreaker icesugarnano - Lattice - Icestorm
- Terasic De10nano - Intel Cyclone V - Quartus
- Arrow Max1000 - Intel Max10 - Quartus

Within the project other firmware have been written or tested:

- Microchip ENC28J60 Ethernet interface controller.
- Microchip ENC424J600 10/100 Base-T Ethernet interface controller.
- ESP8266 Wi-Fi chip.

Use cases

Two use cases in Physics experiments are currently being developed:

- Real time pulse shape analysis in neutron detectors
  - bringing the intelligence to the edge
- Test beam for space experiments
  - increasing testbed operations efficiency

And not only in Physics:

- Machine learning accelerators
  - Ultra low latency inference
- Edge computing
  - Power efficiency for IoT
  - Heterogeneous computing
- Exotic HW/SW/OS architectures
  - Research in innovative OS design

Machine Learning

1 Introduction
- Challenges
- FPGA
- Architectures
- Abstractions

2 The BondMachine project
- Architectures handling
- Architectures molding
- Bondgo
- Basm
- API

3 Clustering
- An example
- Video
- Distributed architecture

4 Accelerators
- Hardware
- Software
- Tests
- Benchmark

5 Misc
- Project timeline
- Supported boards
- Use cases

6 Machine Learning
- Train
- Benchmark

7 Optimizations
- Softmax example
- Results
- Model's compression
- Fragments compositions

8 Accelerator in a cloud
- Bring it to cloud level: why and how
- Implementing a KServe FPGA extension
- Where are we...

9 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 this structures to BondMachine has 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 this structures to BondMachine has 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.

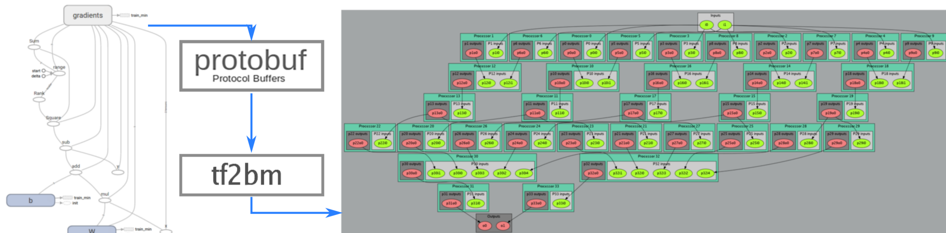
```
layers := []int{2, 5, 2}
weights := make([]neuralbond.Weight, 0)

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

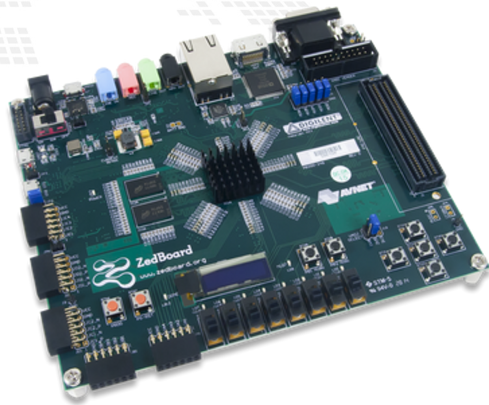
This approch has several advandages over the previous:

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

Specs

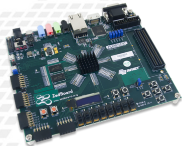
FPGA

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



Different boards

All tests were done using the **Zedboard** device, but BondMachine supports different boards also from different vendors (Intel lattice).

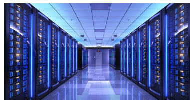


Xilinx Zynq-7000 SoC
85000 logic cells
53200 look-up tables (LUTs)

PCIe card
2800000 logic cells
1732000 Look-Up Tables (LUTs)

FPGA cluster ICSC
Xilinx and Intel FPGAs

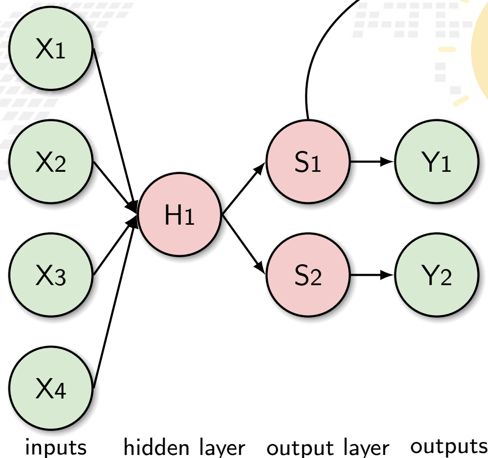
National supercomputing center (ICSC)

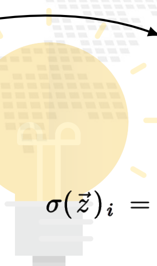


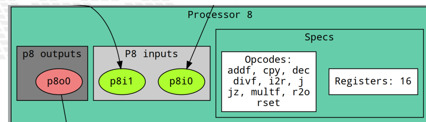
Resources are a key aspect
and often a bottleneck ...

BM inference: A first tentative idea

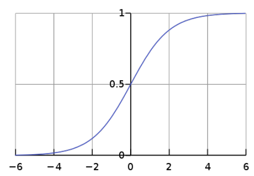
A neuron of a neural network
can be seen as Connecting Processor of BM







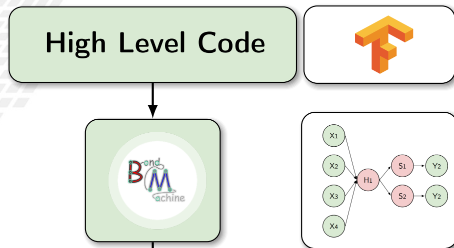
$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$



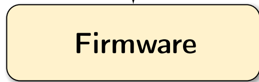

```
%section softmax .romtext iomode:sync
          entry _start    ; Entry point
_start:
 mov r8, 0f0.0
{{range $y := intRange "0" .Params.inputs}}
{{printf "i2r r1,i%d\n" $y}}
          mov      r0, 0f1.0
          mov      r2, 0f1.0
          mov      r3, 0f1.0
          mov      r4, 0f1.0
          mov      r5, 0f1.0
          mov      r7, {{$.Params.expprec}}
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
```

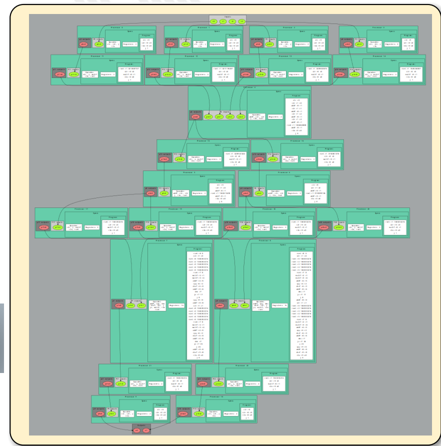
From idea to implementation

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.




```
neuralbond -net-file banknote.json -neuron-lib-path neurons -save-basm working_dir/bondmachine.basm -config-file
neuralbondconfig.json ; basm  -o working_dir/bondmachine.json working_dir/bondmachine.basm neurons/*.basm
```





A first test

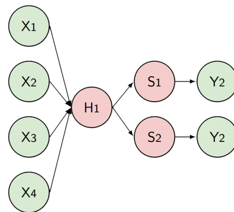
Dataset info:

- **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:



It will be shown how:

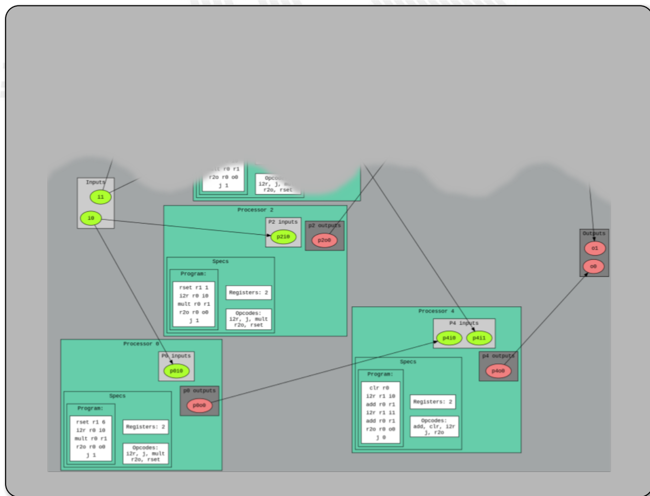
- To build a BondMachine with a trained Neural Network
- Interact with the BondMachine via Jupyter

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.

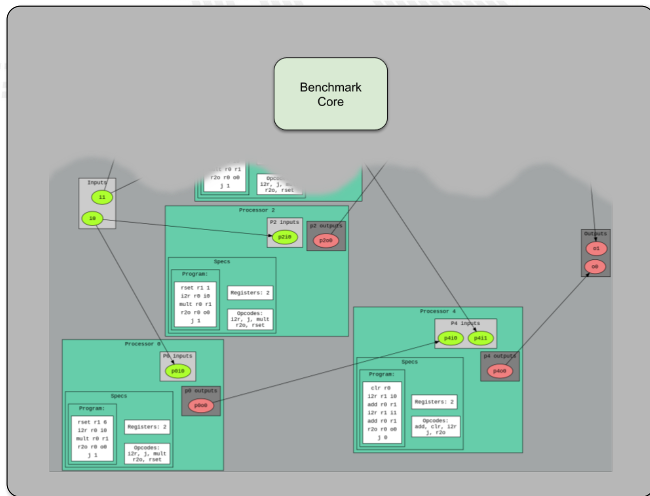


Benchcore

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.

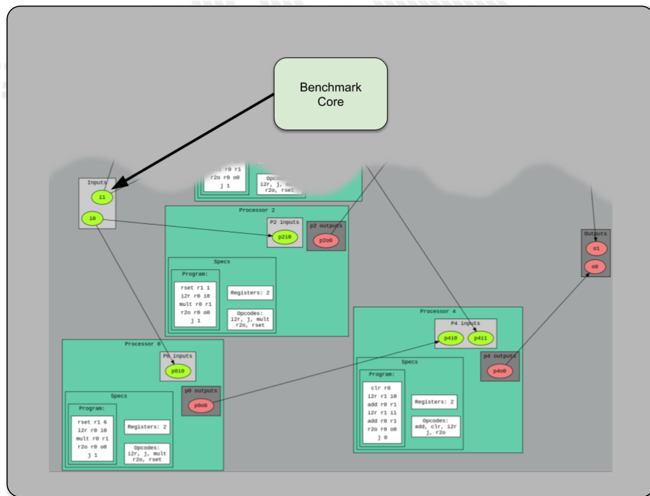


Benchcore

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.

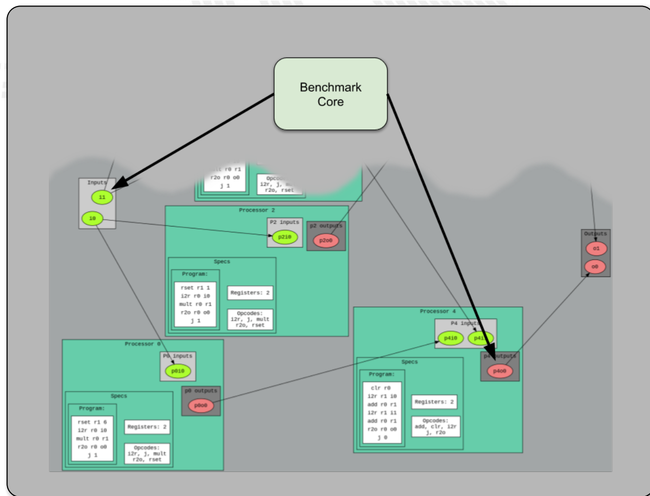


Benchcore

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.

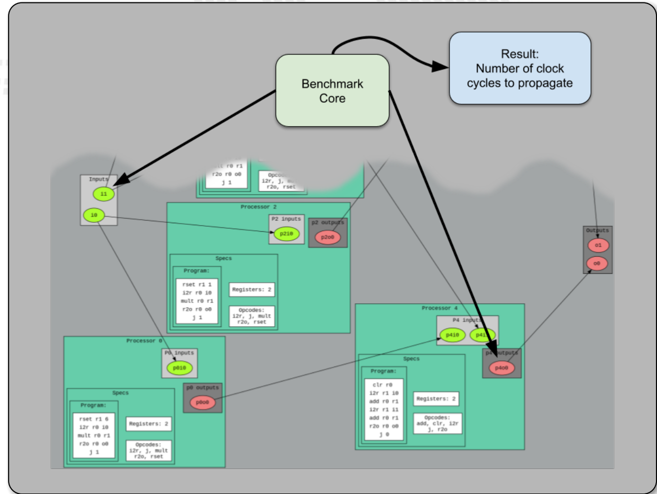


Benchcore

Benchmark an IP is not an easy task.

Fortunately we have a custom design and an FPGA.

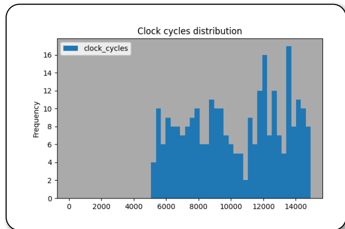
We can put the benchmarks tool inside the accelerator.



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 percentage of error on predictions.



- σ: 2875.94
- Mean: 10268.45
- Latency: 102.68 µs

Resource usage

resource	value	occupancy
regs	15122	28.42%
luts	11192	10.51%

Optimizations

1 Introduction
- Challenges
- FPGA
- Architectures
- Abstractions

2 The BondMachine project
- Architectures handling
- Architectures molding
- Bondgo
- Basm
- API

3 Clustering
- An example
- Video
- Distributed architecture

4 Accelerators
- Hardware
- Software
- Tests
- Benchmark

5 Misc
- Project timeline
- Supported boards
- Use cases

6 Machine Learning
- Train
- Benchmark

7 Optimizations
- Softmax example
- Results
- Model's compression
- Fragments compositions

8 Accelerator in a cloud
- Bring it to cloud level: why and how
- Implementing a KServe FPGA extension
- Where are we...

9 Conclusions and Future directions
- Conclusions
- Ongoing
- Future

A first example of optimization

Remember the **softmax function**?

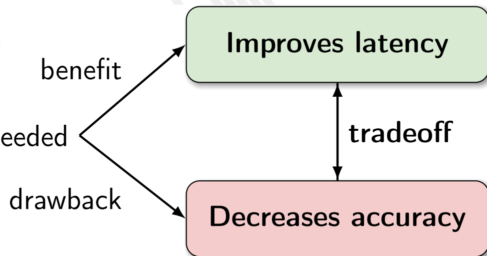
$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}}$$

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

```
%section softmax .romtext iomode:sync
        entry _start    ; Entry point
_start:
 mov r8, 0f0.0
{{range $y := intRange "0" .Params.inputs}}
{{printf "i2r r1,i%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}}
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
```

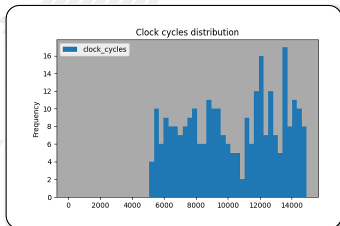
A first example of optimization

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



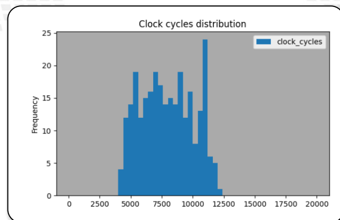
Results of optimization

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



- K: 20
- σ: 2875.94
- Mean: 10268.45
- Latency: 102 µs
- Prediction: 100%

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

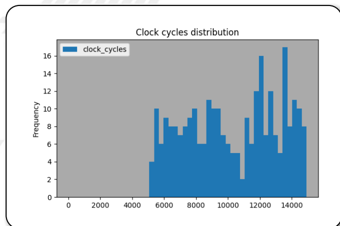


- K: 16
- σ: 2106.32
- Mean: 7946.16
- Latency: 79 µs
- Prediction: 100%

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

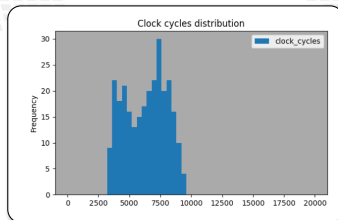
Results of optimization

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



- K: 20
- σ: 2875.94
- Mean: 10268.45
- Latency: 102 µs
- Prediction: 100%

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

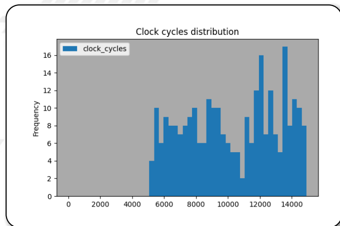


- K: 13
- σ: 1669.88
- Mean: 6312.26
- Latency: 63 µs
- Prediction: 100%

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

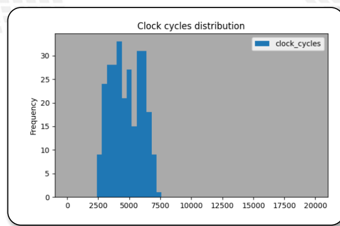
Results of optimization

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



- K: 20
- σ: 2875.94
- Mean: 10268.45
- Latency: 102 μs
- Prediction: 100%

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

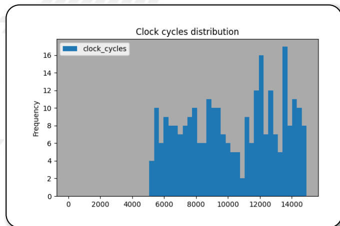


- K: 10
- σ: 1232.47
- Mean: 4766.75
- Latency: 47 μs
- Prediction: 100%

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

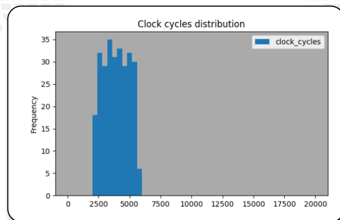
Results of optimization

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



- K: 20
- σ: 2875.94
- Mean: 10268.45
- Latency: 102 µs
- Prediction: 100%

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

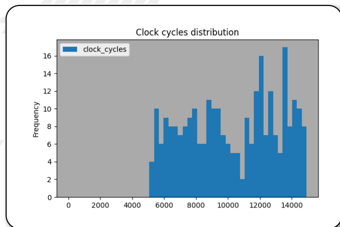


- K: 8
- σ: 1015.50
- Mean: 3913.66
- Latency: 39 µs
- Prediction: 100%

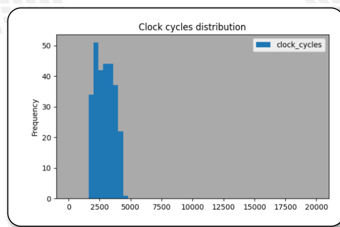
	mean	σ
prob0	6.5562E-05	1.7607E-05
prob1	6.6098E-05	1.7609E-05

Results of optimization

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



- K: 20
- σ: 2875.94
- Mean: 10268.45
- Latency: 102 µs
- Prediction: 100%



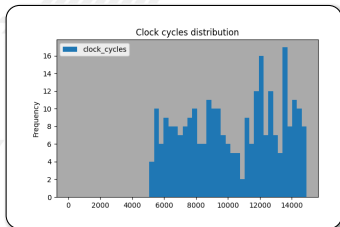
- K: 5
- σ: 740
- Mean: 2911
- Latency: 29 µs
- Prediction: 100%

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

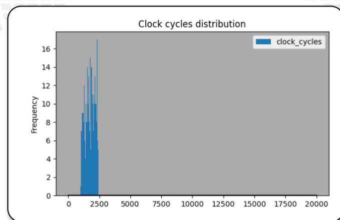
	mean	σ
prob0	3.1070E-05	7.5290E-05
prob1	3.1070E-05	7.5290E-05

Results of optimization

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



- K: 20
- σ: 2875.94
- Mean: 10268.45
- Latency: 102 μs
- Prediction: 100%



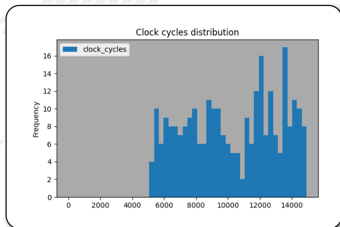
- K: 3
- σ: 394.10
- Mean: 1750.93
- Latency: 17 μs
- Prediction: 100%

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

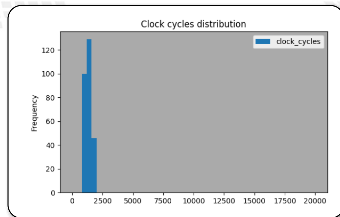
	mean	σ
prob0	0.0053	0.0090
prob1	0.0053	0.0090

Results of optimization

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



- K: 20
- σ: 2875.94
- Mean: 10268.45
- Latency: 102 µs
- Prediction: 100%



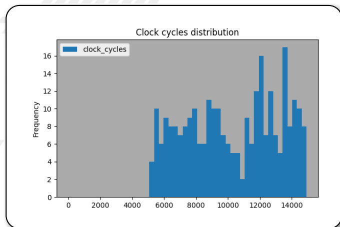
- K: 2
- σ: 268.69
- Mean: 1311.11
- Latency: 13.11 µs
- Prediction: 100%

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

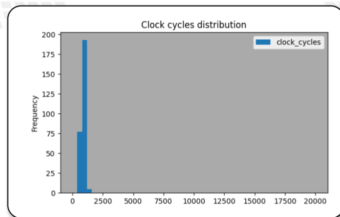
	mean	σ
prob0	0.0193	0.0232
prob1	0.0193	0.0232

Results of optimization

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



- K: 20
- σ: 2875.94
- Mean: 10268.45
- Latency: 102 µs
- Prediction: 100%

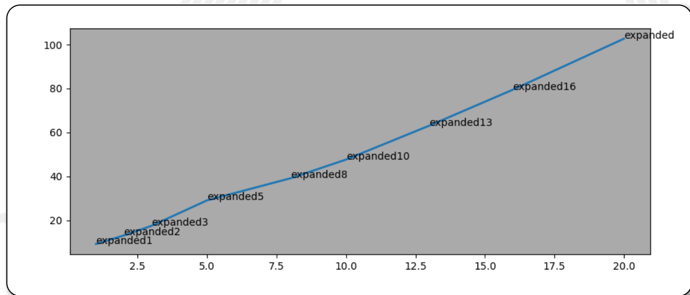


- K: 1
- σ: 173.25
- Mean: 923.71
- Latency: 9.23 µs
- Prediction: 100%

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

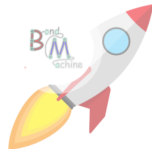
	mean	σ
prob0	0.0990	0.1641
prob1	0.0990	0.1641

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.



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

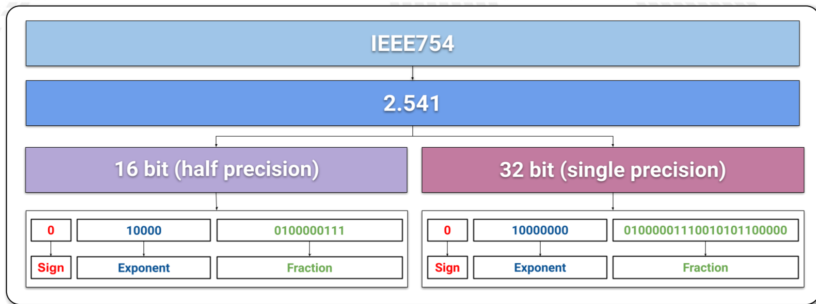
prob0	prob1	class
0.6895	0.3104	0
0.5748	0.4251	0
0.4009	0.5990	1

The output of the bm corresponds to the software output

Open the notebook

Why change numerical precision?

The same floating point number can be represented in different ways



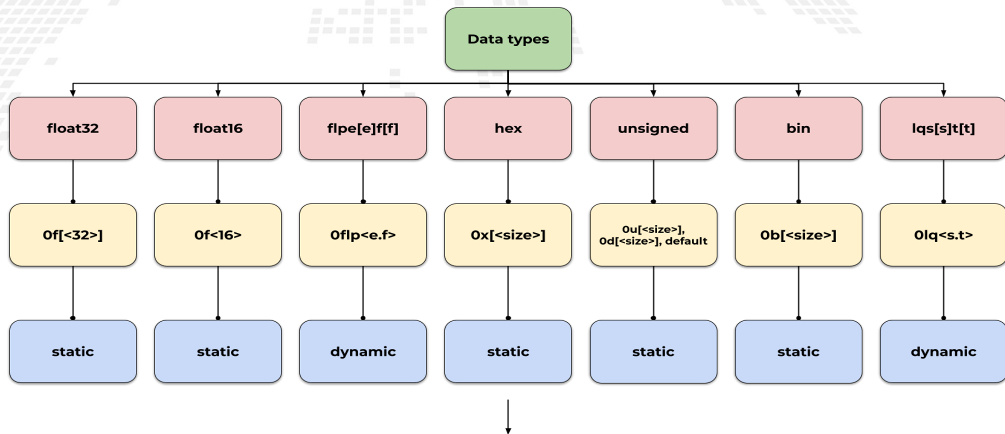
Pro

- Reduced memory usage
- Increased computational speed
- Lower power consumption

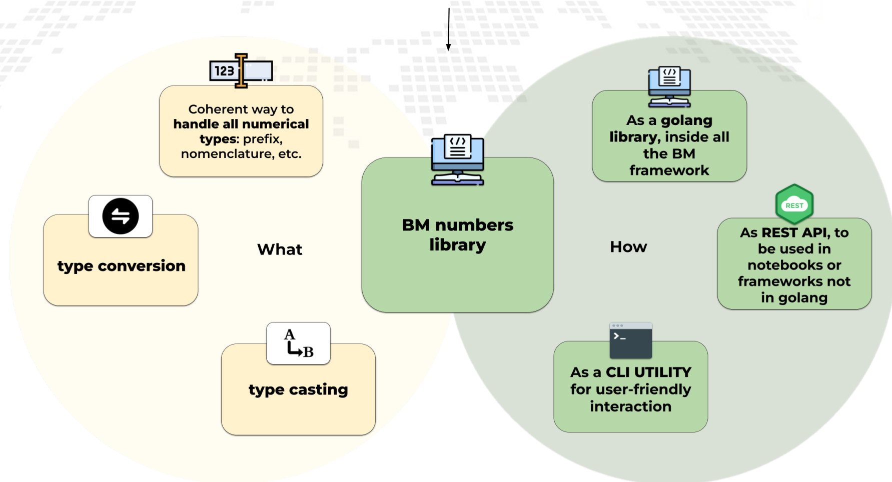
Cons

- Reduced accuracy
- Increased rounding errors
- Limited range

Data types in BondMachine: BMnumbers



Data types in BondMachine: BMnumbers

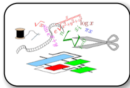


Floating point FloPoCo

FloPoCo is an open source software project that provides a toolchain for automatically generating floating-point arithmetic operators implemented in hardware.

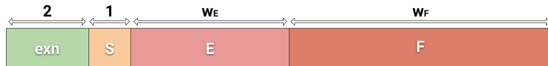
```
./flopoco pipeline=yes frequency=300 FPAdd wE=8 wF=23
```

```
Final report:
|---Entity RightShifter_24_by_max_26_F300_uid4
|      Pipeline depth = 1
|---Entity IntAdder_27_f300_uid8
|      Not pipelined
|---Entity LZCShifter_28_to_28_counting_32_F300_uid16
|      Pipeline depth = 2
|---Entity IntAdder_34_f300_uid20
|      Not pipelined
Entity FPAdd_8_23_F300_uid2
   Pipeline depth = 6
Output file: flopoco.vhdl
```



Features:

- exponent size and mantissa size can take arbitrary values
- 0, ∞ and NaN in explicit exception bits
  - not as special exponent values
  - two more exponent values available in FloPoCo
  - hardware efficient



Tests FloPoCo implementation

We've already seen the pros and cons of changing the numerical precision

Pro

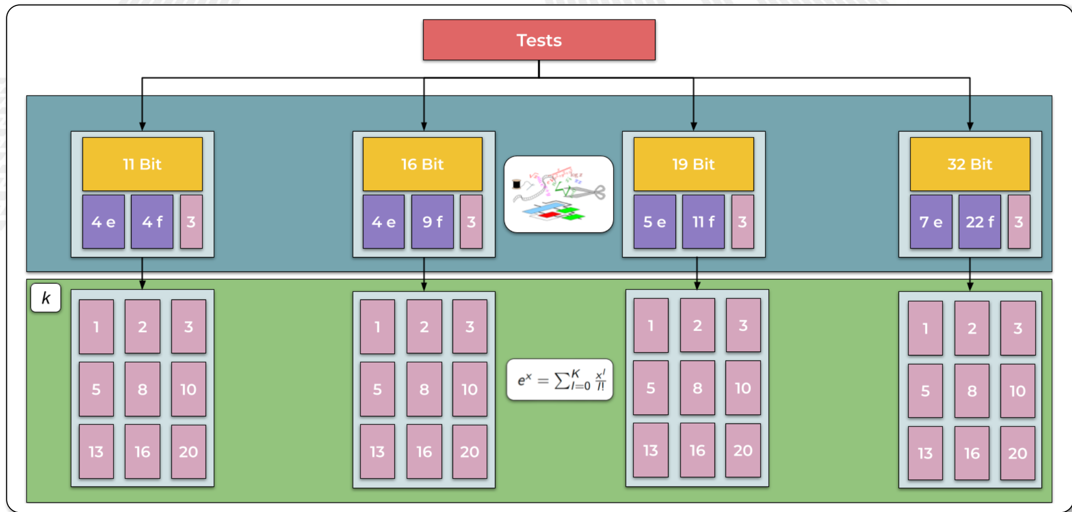
- Reduced memory usage
- **Increased computational speed**
- Lower power consumption

Cons

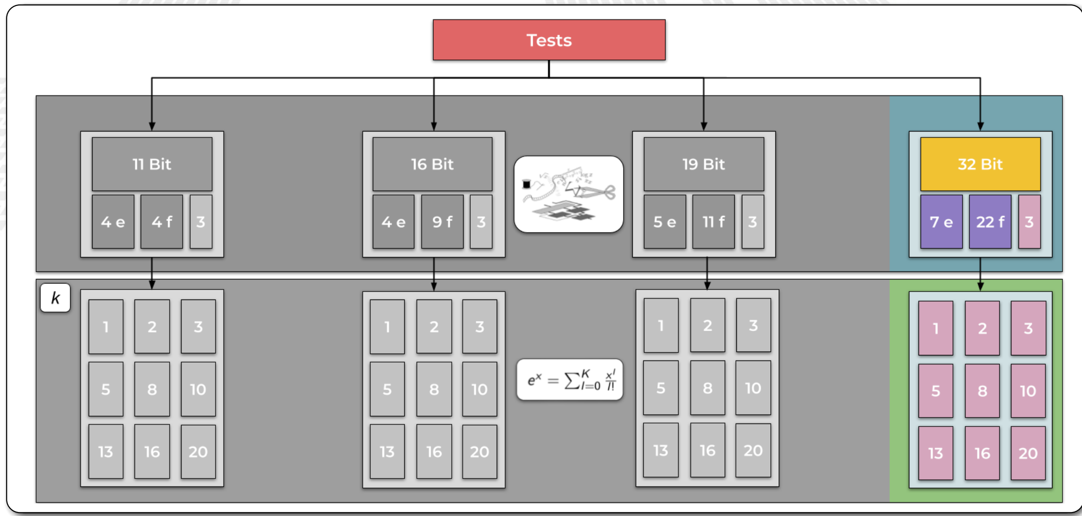
- **Reduced accuracy**
- Increased rounding errors
- Limited range

- How much computationally **faster** are the arithmetic operations implemented by FloPoCo?
- How do **latency**, **accuracy**, occupancy and power consumption vary by changing the numerical precision and the exponent of the exponential?

Tests and results with FloPoCo

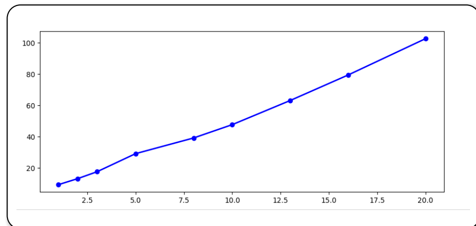


Tests and results with FloPoCo



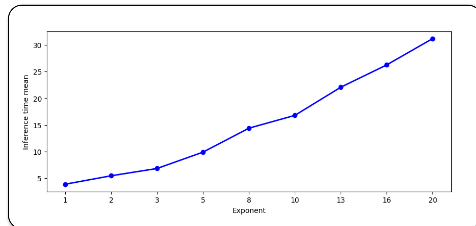
Tests and results with FloPoCo

32bit IEEE754



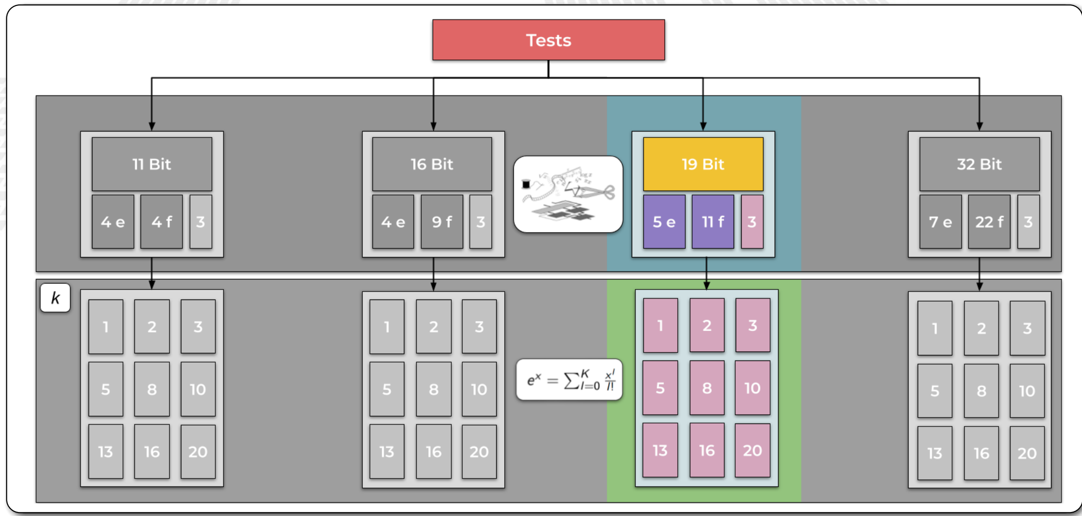
K	Latency	Err prob0	Err prob1
1	9.23 μs	0.0990	0.0990
2	13.11 μs	0.0193	0.0193
3	17.50 μs	0.0053	0.0053
5	29.11 μs	3.1070E-05	3.1071E-05
8	39.13 μs	6.5562E-07	6.6098E-07
10	47.66 μs	1.6162E-07	1.6525E-07
13	63.12 μs	1.6470E-07	1.6652E-07
16	79.46 μs	1.6470E-07	1.6652E-07
20	102.68 μs	1.6470E-07	1.6652E-07

32bit FloPoCo



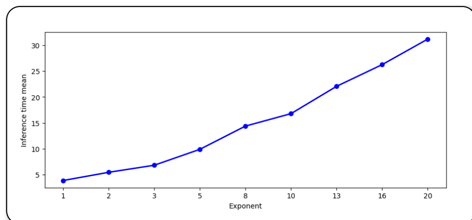
K	Latency	Err prob0	Err prob1
1	3.89 μs	0.0990	0.0990
2	5.47 μs	0.0193	0.0193
3	6.84 μs	0.0053	0.0053
5	9.90 μs	0.0001	0.0001
8	14.39 μs	6.5890E-07	6.5425E-07
10	16.79 μs	1.7316E-07	1.7770E-07
13	22.07 μs	1.7610E-07	1.8029E-07
16	26.25 μs	1.7610E-07	1.8029E-07
20	31.18 μs	1.7610E-07	1.8029E-07

Tests and results with FloPoCo



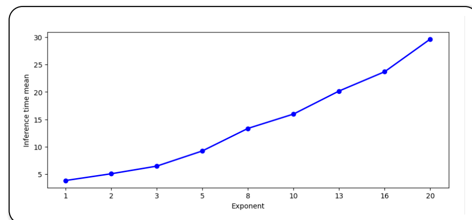
Tests and results with FloPoCo

32bit FloPoCo



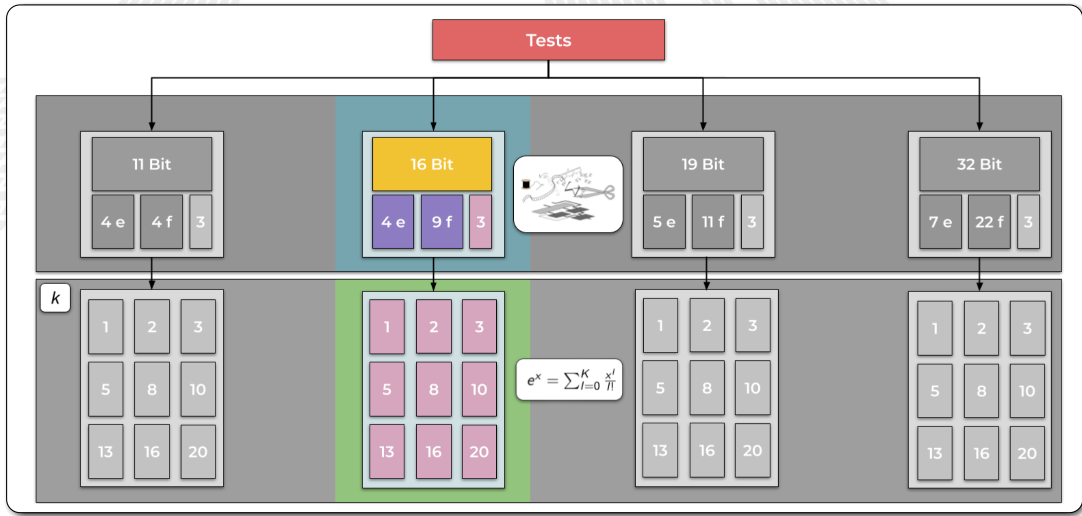
K	Latency	Err prob0	Err prob1
1	3.89 µs	0.0990	0.0990
2	5.47 µs	0.0193	0.0193
3	6.84 µs	0.0053	0.0053
5	9.90 µs	0.0001	0.0001
8	14.39 µs	6.5890E-07	6.5425E-07
10	16.79 µs	1.7316E-07	1.7770E-07
13	22.07 µs	1.7610E-07	1.8029E-07
16	26.25 µs	1.7610E-07	1.8029E-07
20	31.18 µs	1.7610E-07	1.8029E-07

19bit FloPoCo



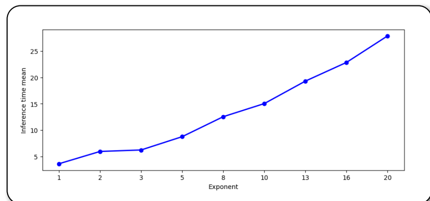
K	Latency	Err prob0	Err prob1
1	3.80 µs	0.1229	0.009
2	5.04 µs	0.0193	0.0193
3	6.44 µs	0.0054	0.0054
5	9.21 µs	0.00024	0.00025
8	13.33 µs	0.00010	9.9151E-05
10	15.95 µs	0.00010	9.9151E-05
13	20.17 µs	0.00010	9.9151E-05
16	23.70 µs	0.00010	9.9151E-05
20	29.67 µs	0.00010	9.9151E-05

Tests and results with FloPoCo



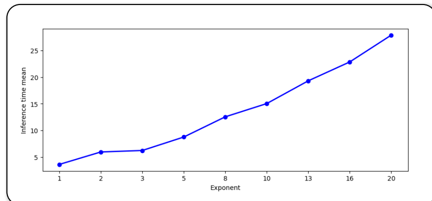
Tests and results with FloPoCo

19bit FloPoCo



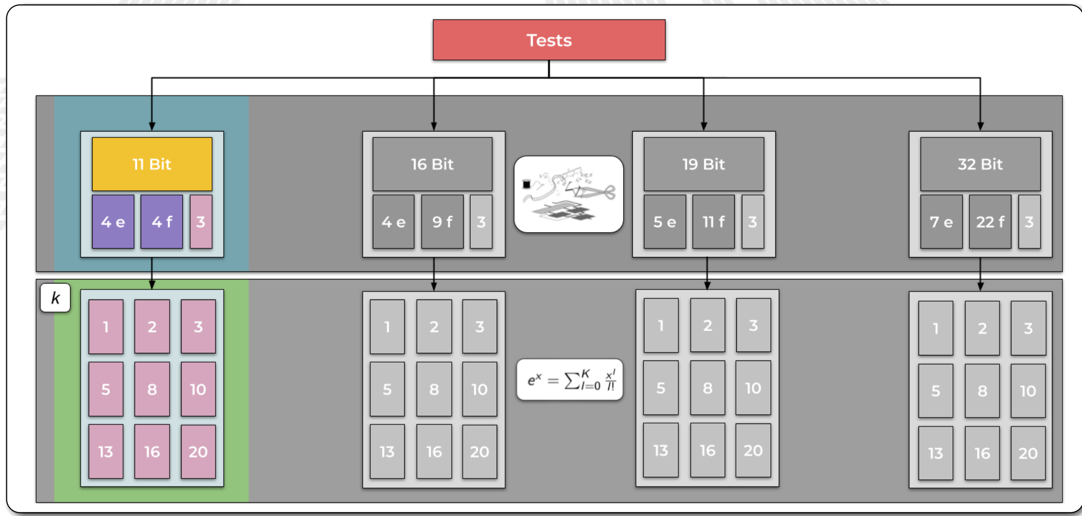
K	Latency	Err prob0	Err prob1
1	3.80 µs	0.1229	0.009
2	5.04 µs	0.0193	0.0193
3	6.44 µs	0.0054	0.0054
5	9.21 µs	0.00024	0.00025
8	13.33 µs	0.00010	9.9151E-05
10	15.95 µs	0.00010	9.9151E-05
13	20.17 µs	0.00010	9.9151E-05
16	23.70 µs	0.00010	9.9151E-05
20	29.67 µs	0.00010	9.9151E-05

16bit FloPoCo



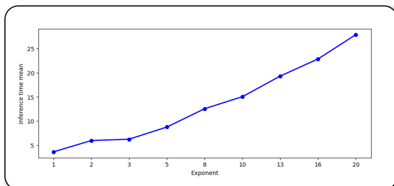
K	Latency	Err prob0	Err prob1	Pred
1	3.59 µs	1.3935	0.099	99.27%
2	5.93 µs	0.0192	0.0191	100%
3	6.21 µs	0.0057	0.0057	100%
5	8.74 µs	0.00125	0.0019	100%
8	12.54 µs	0.00125	0.0019	100%
10	15.04 µs	0.0012	0.0019	100%
13	19.32 µs	0.0026	0.0025	99.63%
16	22.87 µs	0.0037	1.8113	99.63%
20	27.91 µs	0.0060	4.1385	98.54%

Tests and results with FloPoCo



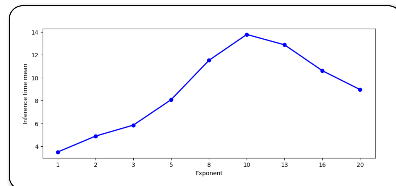
Tests and results with FloPoCo

16bit FloPoCo



K	Latency	Err prob0	Err prob1	Pred
1	3.59 μs	1.3935	0.099	99.27%
2	5.93 μs	0.0192	0.0191	100%
3	6.21 μs	0.0057	0.0057	100%
5	8.74 μs	0.00125	0.0019	100%
8	12.54 μs	0.00125	0.0019	100%
10	15.04 μs	0.0012	0.0019	100%
13	19.32 μs	0.0026	0.0025	99.63%
16	22.87 μs	0.0037	1.8113	99.63%
20	27.91 μs	0.0060	4.1385	98.54%

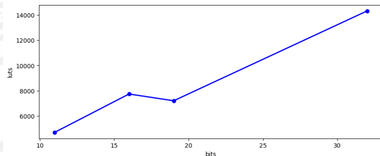
11bit FloPoCo



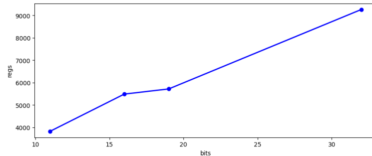
K	Latency	Err prob0	Err prob1	Pred
1	3.49 μs	0.2235	0.0992	97.45%
2	4.88 μs	0.0221	0.0168	98.54%
3	5.84 μs	0.0156	0.0126	98.54%
5	8.07 μs	0.0138	0.0110	98.54%
8	11.51 μs	0.0138	0.0110	98.54%
10	13.78 μs	0.0138	0.0110	98.54%
13	12.87 μs	0.0175	1.5069	97.09%
16	10.61 μs	0.0187	2.5789	96.72%
20	8.95 μs	0.0273	1.223	94.90%

Results with FloPoCo

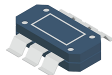
How do latency, accuracy, **occupancy** and **power consumption** vary by changing the numerical precision ?

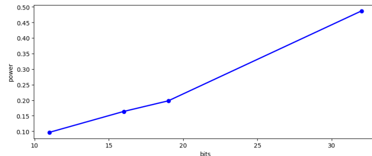


Bits	Luts	Usage
11	4704	8.84%
16	7738	14.54%
19	7202	13.54%
32	14306	26.89%



Bits	Regs	Usage
11	3828	3.59%
16	5487	5.15%
19	5717	5.37%
32	9264	8.70%

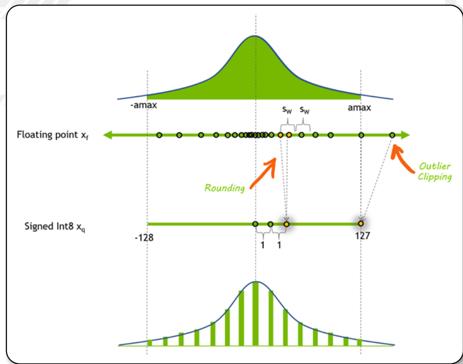




Bits	Power
11	0.096 W
16	0.163 W
19	0.198 W
32	0.487 W

Linear quantization

Linear quantization is a widely used technique in signal processing, in particular in neural networks **reduces memory usage and computational complexity** by representing values with fewer bits, enabling **efficient deployment on resource-constrained devices** (but it may introduce some loss of accuracy).



BMnumbers translates the floating point number into the quantized equivalent using the data type `lqs[s]t[t]`

```
bmnumbers --show native -cast lqs16t1 -linear-data-range 1,ranges.txt "0b<16>010010110"

0lq<16.1>13.73291015625
```

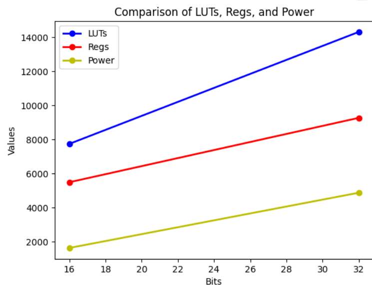
Corrected signed integer instructions are used in hardware

Quantized networks can be simulated to check if the precision is acceptable.

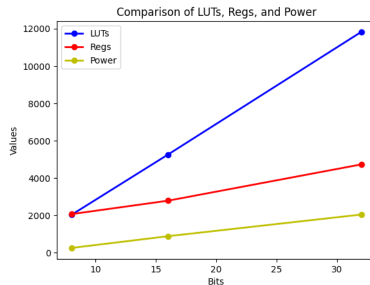
Quantization: tests, results and analysis

Linear quantization **reduces memory usage and computational complexity** by representing values with fewer bits, enabling efficient deployment on resource constrained devices (but it may introduce some loss of accuracy)

FloPoCo



Quantization



FloPoCo					
Bits	**Luts**	**Regs**	**Power**	**Latency**	**Pred**
16	7738 (14%)	5487 (5%)	0.163W	6.21 µs	100%
32	14306 (26%)	9264 (8%)	0.487W	6.84 µs	100%

Bits	Luts	Regs	Power	Latency	Pred
8	2013 (3%)	2054 (2%)	0.024W	1.60 µs	91%
16	5259 (9%)	2774 (3%)	0.087W	1.60 µs	99%
32	11823 (22%)	4718 (5%)	0.203W	1.61 µs	99%

ML Hands-on

Hands-on N.12

It will be shown how:

- To build a BondMachine with a trained Neural Network ...
- ... with floating point 16bit precision

- Interact with the BondMachine via Jupyter

ML Hands-on

Hands-on N.13

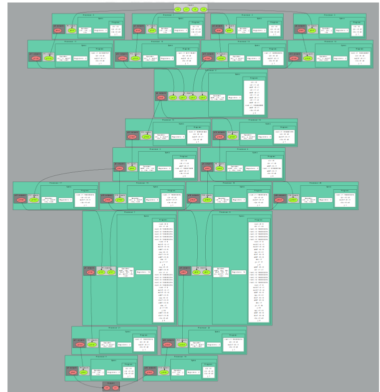
It will be shown how:

- To build a BondMachine with a trained Neural Network ...
- ... with fixed point 16bit

- Interact with the BondMachine via Jupyter

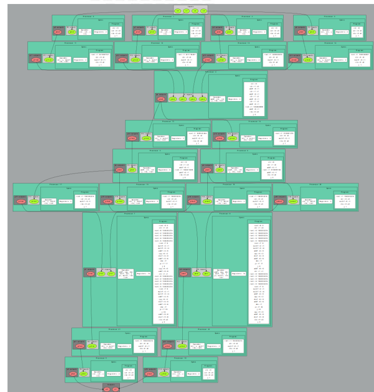
Fragments composition

- 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.



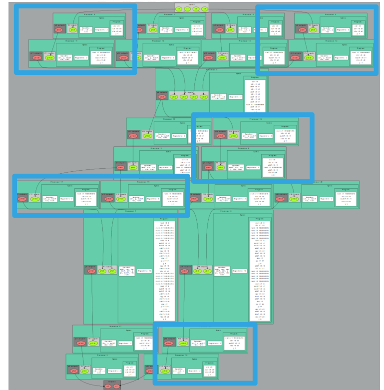
Fragments composition

- 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.



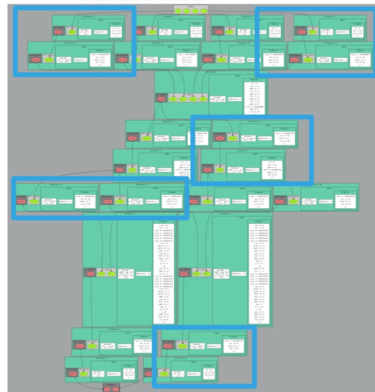
Fragments composition

- 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.



CP pruning hands-on

Hands-on N.14

Goals are:

- Prune a processor and find out the outcomes

CP collapsing hands-on

Hands-on N.15

Goals are:

- Collapse processors and find out the outcomes

Hands-on N.16

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

Accelerator in a cloud

1 Introduction
- Challenges
- FPGA
- Architectures
- Abstractions

2 The BondMachine project
- Architectures handling
- Architectures molding
- Bondgo
- Basm
- API

3 Clustering
- An example
- Video
- Distributed architecture

4 Accelerators
- Hardware
- Software
- Tests
- Benchmark

5 Misc
- Project timeline
- Supported boards
- Use cases

6 Machine Learning
- Train
- Benchmark

7 Optimizations
- Softmax example
- Results
- Model's compression
- Fragments compositions

8 Accelerator in a cloud
- Bring it to cloud level: why and how
- Implementing a KServe FPGA extension
- Where are we...

9 Conclusions and Future directions
- Conclusions
- Ongoing
- Future

Bring it to cloud level: why?

So we “know” how to build firmware for ML inference in a vendor agnostic way. Can we integrate it with cloud-native inference as-a-service solution to get any advantage?

- **Ease of usage and flexibility**
  - Being able to deploy an inference algorithm on FPGA without caring for “where” the resources are
  - Accessing ML predictions from a remote computing resource without having in place any specialized hardware or software piece
    - At the cost of increased latency $\rightarrow$ to be carefully evaluated case by case
  - Sharing the access to the same model predictions with other collaborators
- **Democratic access and management**
  - Leveraging cloud/k8s native tools, you can reuse a well established way to orchestrate the bookkeeping and distribution of the payloads
- **Easy Prototyping**
  - Automation of the build and load process -> the framework take care of vendor specific details

CHEP 2023

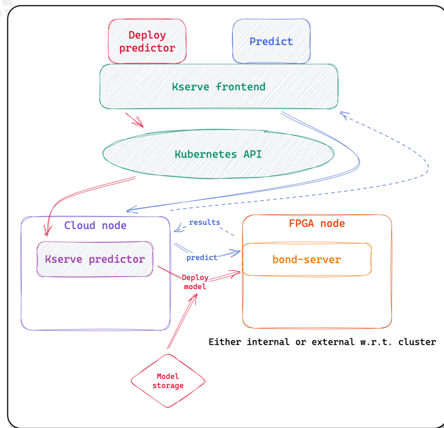
Implementing a KServe FPGA extension

The remote inference still an open field on many aspects, regardless we started from one of the main emerging ecosystems for ML: **Kubeflow**

KServe in particular is the component responsible for providing inference endpoint as-a-Service

Our simple workflow:

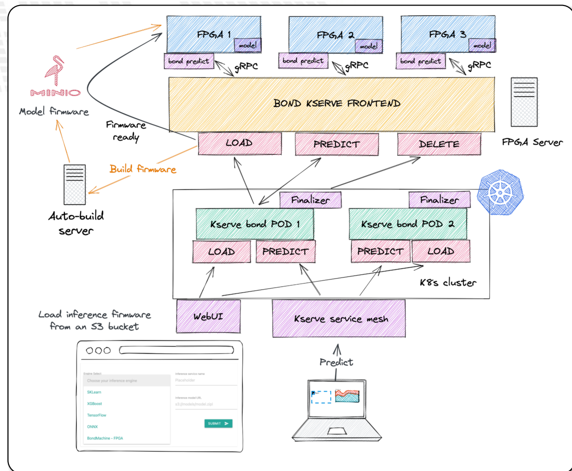
1. **Train your model** with your preferred framework (e.g. TF)
2. **Store the model** on a remote storage
   - S3 storage is the one used for our tests
3. Deploying the **same model on a remote FPGA via a user friendly UI**
4. Get back the **details of the endpoint to interact with**
   - Either via HTTP or grpc protocols



Kserve extension implementation

The main components that we developed are:

- **Custom WebUI** to hide complexity to the user
  - A Kubeflow managed solution exists, we are planning to integrate this work eventually
    - We need additional metadata to be passed (e.g. board model, provider, hls engine etc)
- Translate a **model load** request into conditional actions
  - Load the bitstream file from the remote location directly
    - Pre built by the user on its own
  - **building a firmware** "seamlessly" on an external building machine
- **Eventually load the firmware** on the FPGA board via the development of a grpc server installed on the machine that have access to the board

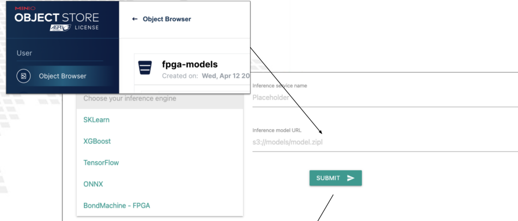


CHEP 2023

Where are we...

We have **validated an end to end workflow with a generic ML** algorithm.
With the following steps:

- Load the model description to an S3 bucket
- Report the model URL and name in the WebUI
  - Selecting HLS engine (BM in this case)
- Wait for the build server to build and store your firmware for the available FPGAs
  - Store back the firmware on S3 bucket for further reuse
  - Load the created firmware on a FPGA
- Publish the endpoint to send the prediction requests to and then do your prediction.




```
2023-05-04 11:54:13  * Request arrived to build firmware *
2023-05-04 11:54:13  * HLS tool requested bondmachine *
2023-05-04 11:54:13  * Requirements check completed successfully, going to build firmware *
2023-05-04 11:54:14  * Before exec command: make bondmachine *
2023-05-04 11:54:15  * Command executed successfully: make bondmachine *
2023-05-04 11:54:15  * Before exec command: make hdl *
ls: impossibile accedere a '*.vhd': File o directory non esistente
2023-05-04 11:54:17  * Command executed successfully: make hdl *
2023-05-04 11:54:17  * Before exec command: make design_synthesis *
2023-05-04 12:01:02  * Command executed successfully: make design_synthesis *
2023-05-04 12:01:02  * Before exec command: make design_implementation *
2023-05-04 12:05:40  * Command executed successfully: make design_implementation *
2023-05-04 12:05:40  * Before exec command: make design_bitstream *
2023-05-04 12:06:57  * Command executed successfully: make design_bitstream *
2023-05-04 12:06:57  * Going to upload firmware to MINIO: make design_bitstream *
2023-05-04 12:06:59  * Metadata 9cc92d82-9053-4cc7-8ab5-075ce53eb7af_firmware.json successfully uploaded to MINIO *
2023-05-04 12:06:59  * Hardare description file 9cc92d82-9053-4cc7-8ab5-075ce53eb7af_firmware.hwh successfully uploaded to
2023-05-04 12:06:59  * Firmware 9cc92d82-9053-4cc7-8ab5-075ce53eb7af_firmware.bit successfully uploaded to MINIO *
2023-05-04 12:06:59 * Going to clean temporary files *
2023-05-04 12:06:59 * Temporary files removed *
2023-05-04 12:06:59 * Firmware generation completed successfully in 12.75906725 minutes *
```

SERVICE_TYPE	API_VERSION	INFERENCE_SERVICE_NAME	SERVICE_HOSTNAME	MODEL URL
fpga-model	v1	test01	test01.default.fpga.infn.it	ghcr.io/bondmachinehq/bond-server

Editor

```
1 {
2   "inputs": "{'inputs':[{'name':'input_1'
3 }
```

Response

```
{
  "success": true,
  "outputs": "[{\"classification\":
  {\"probabilities\": [[0.6895769834518433, 0
  .31042304635047911, [0.5748991370201111, 0
  .4251008629798889]], \"classification\": [0
  .0, 0.0]}}]"
}
```

0.057s

CHEP 2023

Conclusions and Future directions

1 Introduction
- Challenges
- FPGA
- Architectures
- Abstractions

2 The BondMachine project
- Architectures handling
- Architectures molding
- Bondgo
- Basm
- API

3 Clustering
- An example
- Video
- Distributed architecture

4 Accelerators
- Hardware
- Software
- Tests
- Benchmark

5 Misc
- Project timeline
- Supported boards
- Use cases

6 Machine Learning
- Train
- Benchmark

7 Optimizations
- Softmax example
- Results
- Model's compression
- Fragments compositions

8 Accelerator in a cloud
- Bring it to cloud level: why and how
- Implementing a KServe FPGA extension
- Where are we...

9 Conclusions and Future directions
- Conclusions
- Ongoing
- Future

Towards an OS Hands-on

Hands-on N.17

It will be shown:

- How to build a BondMachine with a close interaction with the host machine
- A shell-like BM application from Jupyter

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
- Fist steps in the direction of a full OS

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