The BondMachine Toolkit

A novel moldable computer architecture

Mirko Mariotti Loriano Storchi Daniele Spiga

Department of Physics and Geology - University of Perugia
INFN Perugia

The International Conference on Parallel Computing
ParCo2019
Prague, Czech Republic, 10-13 September 2019

The BondMachine Toolkit: A novel moldable computer architecture

In this presentation i will talk about:

- Technological background of the project
- The BondMachine Project: the Architecture
- The BondMachine Project: the Tools
- Use cases

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

What is it ?

- A field-programmable gate array (FPGA) is an integrated circuit whose logic is re-programmable. It's used to build reconfigurable digital circuits.

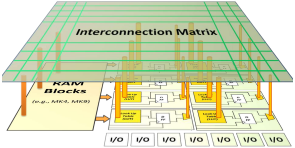
- FPGAs contain an array of programmable logic blocks, and a hierarchy of reconfigurable interconnects that allow the blocks to be "wired together".

- Logic blocks can be configured to perform complex combinational functions.

- The FPGA configuration is generally specified using a hardware description language (HDL).

FPGA
What is it ?

- A field-programmable gate array (FPGA) is an integrated circuit whose logic is re-programmable. It's used to build reconfigurable digital circuits.
- FPGAs contain an array of programmable logic blocks, and a hierarchy of reconfigurable interconnects that allow the blocks to be "wired together".



- Logic blocks can be configured to perform complex combinational functions.

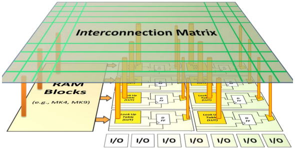
- The FPGA configuration is generally specified using a hardware description language (HDL).

FPGA

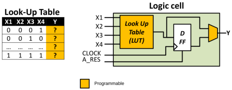
What is it ?

- A field-programmable gate array (FPGA) is an integrated circuit whose logic is re-programmable. It's used to build reconfigurable digital circuits.

- FPGAs contain an array of programmable logic blocks, and a hierarchy of reconfigurable interconnects that allow the blocks to be "wired together".



- Logic blocks can be configured to perform complex combinational functions.

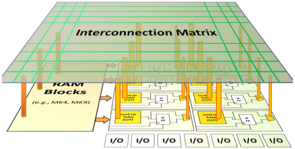


- The FPGA configuration is generally specified using a hardware description language (HDL).

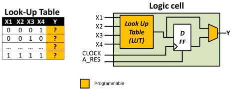
FPGA
What is it ?

- A field-programmable gate array (FPGA) is an integrated circuit whose logic is re-programmable. It’s used to build reconfigurable digital circuits.

- FPGAs contain an array of programmable logic blocks, and a hierarchy of reconfigurable interconnects that allow the blocks to be "wired together".



- Logic blocks can be configured to perform complex combinational functions.



- The FPGA configuration is generally specified using a hardware description language (HDL).

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

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.

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.

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.

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.

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.

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.

The BondMachine
The 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
The 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

The 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

The 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

The 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

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).

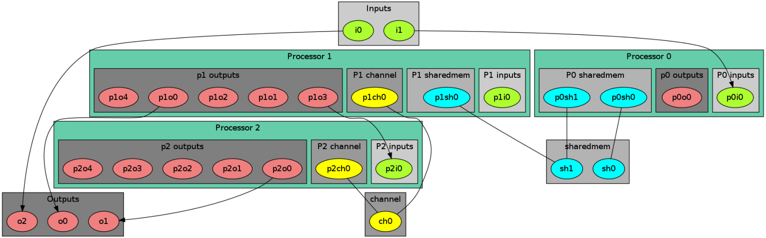
Introducing the BondMachine (BM)

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.
- There 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.
- There 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.
- There 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.
- There 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.
- There 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.
- There possible operating modes.

Shared Objects (SO)

The non-computational element of the BM

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

Examples of their purposes are:

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

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

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

Shared Objects (SO)

The non-computational element of the BM

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

Examples of their purposes are:

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

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

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

Shared Objects (SO)

The non-computational element of the BM

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

Examples of their purposes are:

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

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

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

Shared Objects (SO)

The non-computational element of the BM

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

Examples of their purposes are:

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

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

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

Shared Objects (SO)

The non-computational element of the BM

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

Examples of their purposes are:

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

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

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

Shared Objects (SO)

The non-computational element of the BM

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

Examples of their purposes are:

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

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

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

Handle BM computer architectures

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

- build a specify architecture
- modify a pre-existing architecture
- simulate or emulate the behavior
- Generate the Hardware Description 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 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 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 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

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

Evolutive BM

Evolutionary computing to BM

Bondgo

The architecture compiler

ML tools

Map computational graphs to BM

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

Evolutive BM

Evolutionary computing to BM

Bondgo

The architecture compiler

ML tools

Map computational graphs to BM

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

Evolutive BM

Evolutionary computing to BM

Bondgo

The architecture compiler

ML tools

Map computational graphs to BM

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

Evolutive BM

Evolutionary computing to BM

Bondgo

The architecture compiler

ML tools

Map computational graphs to BM

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

Evolutive BM

Evolutionary computing to BM

Bondgo

The architecture compiler

ML tools

Map computational graphs to BM

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

Evolutive BM

Evolutionary computing to BM

Bondgo

The architecture compiler

ML tools

Map computational graphs to BM

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

Evolutive BM

Evolutionary computing to BM

Bondgo

The architecture compiler

ML tools

Map computational graphs to BM

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

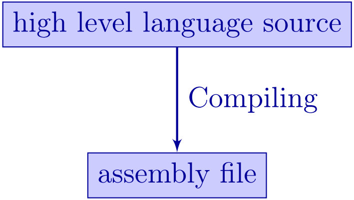
Bondgo

This is the standard flow when building computer programs

high level language source

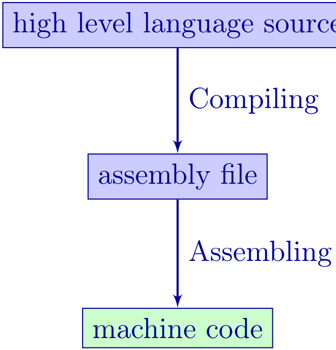
Bondgo

This is the standard flow when building computer programs



Bondgo

This is the standard flow when building computer programs



Bondgo

bondgo loop example

```
package main

import ()

func main() {
   var reg_aa uint8
   var reg_ab uint8
   for reg_aa = 10; reg_aa > 0; reg_aa-- {
      reg_ab = reg_aa
      break
   }
}
```

bondgo loop example in asm

```
clr aa
clr ab
rset ac 10
cpy aa ac
cpy ac aa
jz ac 11
cpy ac aa
cpy ab ac
j 11
dec aa
j 4
```

Bondgo

Bondgo does something different from standard compilers ...

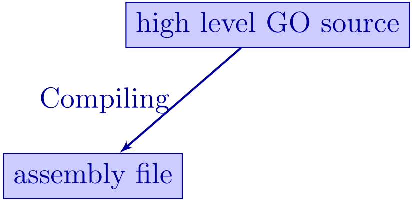
Bondgo

Bondgo does something different from standard compilers ...

high level GO source

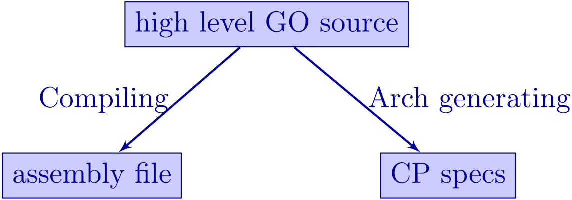
Bondgo

Bondgo does something different from standard compilers ...



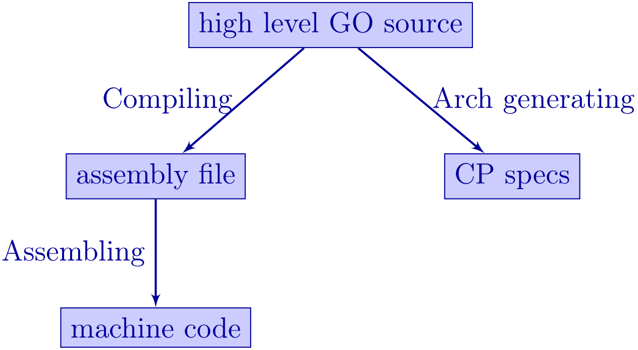
Bondgo

Bondgo does something different from standard compilers ...



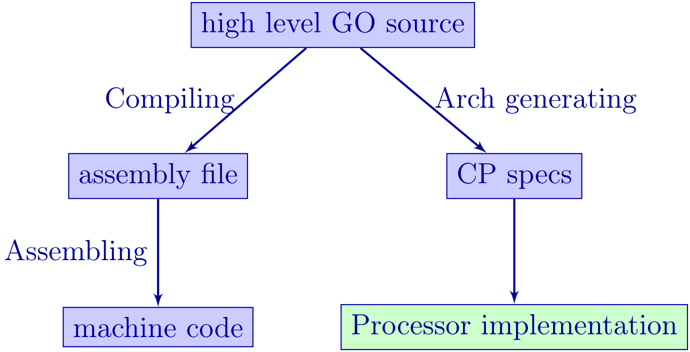
Bondgo

Bondgo does something different from standard compilers ...



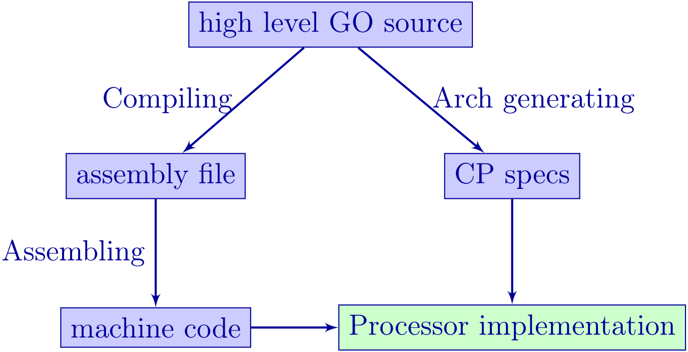
Bondgo

Bondgo does something different from standard compilers ...



Bondgo

Bondgo does something different from standard compilers ...



Bondgo

A first example

counter go source

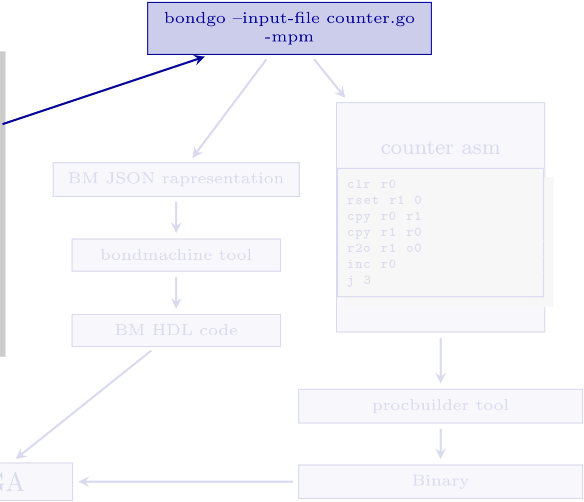
```
package main

import (
   "bondgo"
)

func main() {
   var outgoing bondgo.Output
   var reg_int uint8
   outgoing =
      bondgo.Make(bondgo.Output, 3)
   reg_int = 0
   for {
      bondgo.IOWrite(outgoing,
      reg_int)
      reg_int++
   }
}
```

Bondgo

A first example

counter go source

```
package main

import (
   "bondgo"
)

func main() {
   var outgoing bondgo.Output
   var reg_int uint8
   outgoing =
      bondgo.Make(bondgo.Output, 3)
   reg_int = 0
   for {
      bondgo.IOWrite(outgoing,
      reg_int)
      reg_int++
   }
}
```

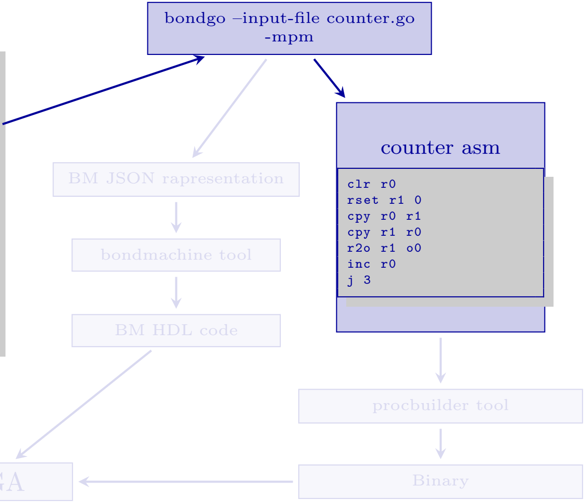


Bondgo

A first example

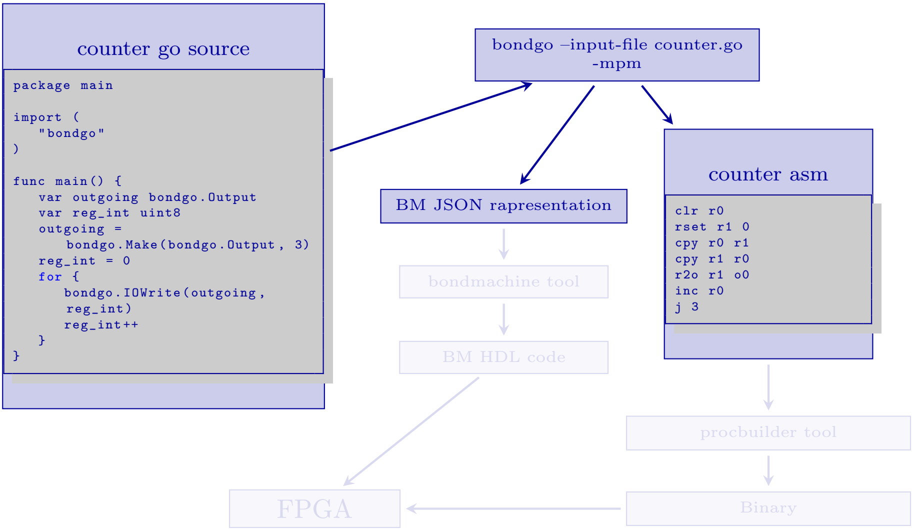
counter go source

```
package main

import (
   "bondgo"
)

func main() {
   var outgoing bondgo.Output
   var reg_int uint8
   outgoing =
      bondgo.Make(bondgo.Output, 3)
   reg_int = 0
   for {
      bondgo.IOWrite(outgoing,
      reg_int)
      reg_int++
   }
}
```



Bondgo

A first example



counter go source

```
package main

import (
   "bondgo"
)

func main() {
   var outgoing bondgo.Output
   var reg_int uint8
   outgoing =
      bondgo.Make(bondgo.Output, 3)
   reg_int = 0
   for {
      bondgo.IOWrite(outgoing,
      reg_int)
      reg_int++
   }
}
```

counter asm

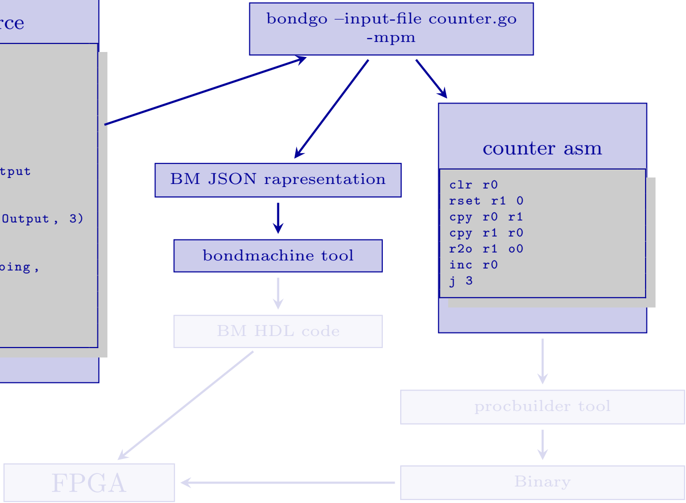
```
clr r0
rset r1 0
cpy r0 r1
cpy r1 r0
r2o r1 o0
inc r0
j 3
```

Bondgo

A first example

counter go source

```
package main

import (
   "bondgo"
)

func main() {
   var outgoing bondgo.Output
   var reg_int uint8
   outgoing =
      bondgo.Make(bondgo.Output, 3)
   reg_int = 0
   for {
      bondgo.IOWrite(outgoing,
      reg_int)
      reg_int++
   }
}
```

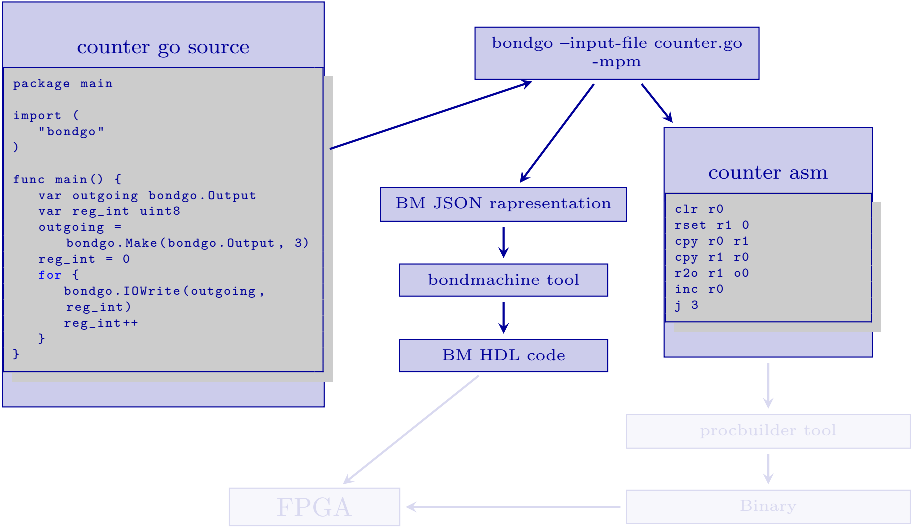


counter asm

```
clr r0
rset r1 0
cpy r0 r1
cpy r1 r0
r2o r1 o0
inc r0
j 3
```

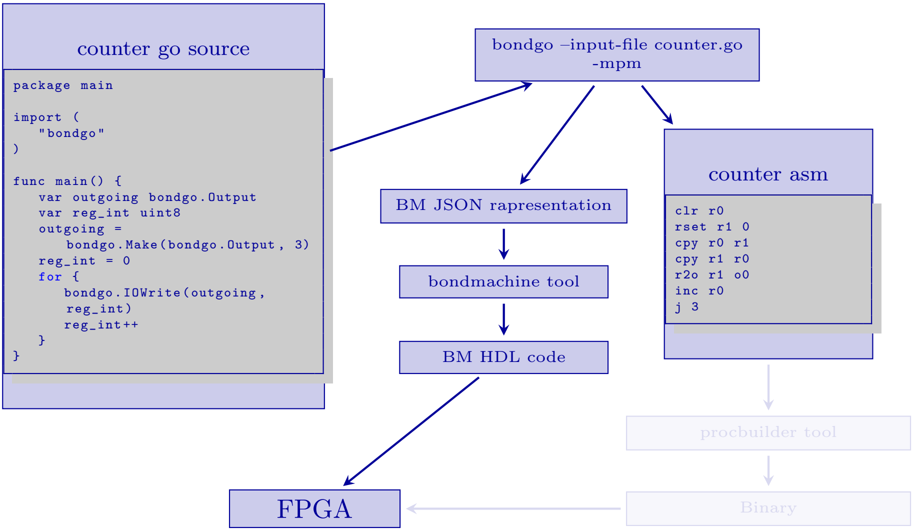
Bondgo

A first example



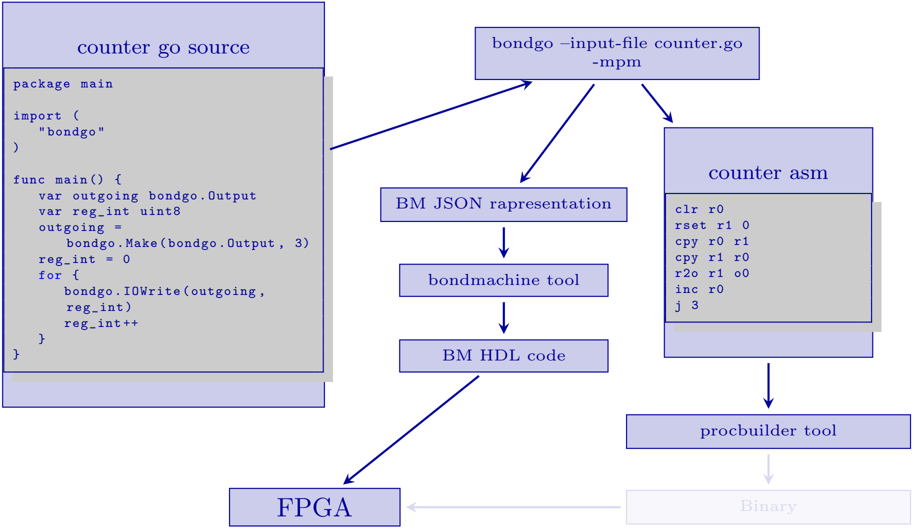
Bondgo

A first example



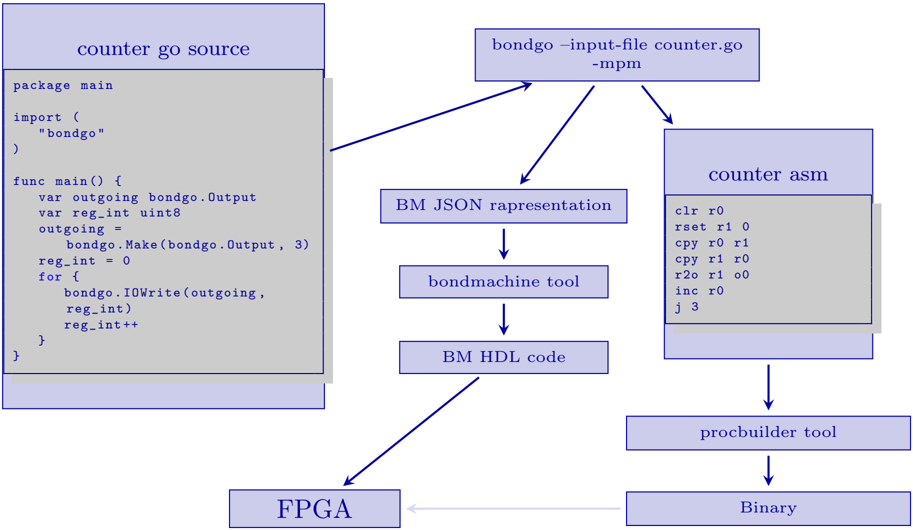
Bondgo

A first example



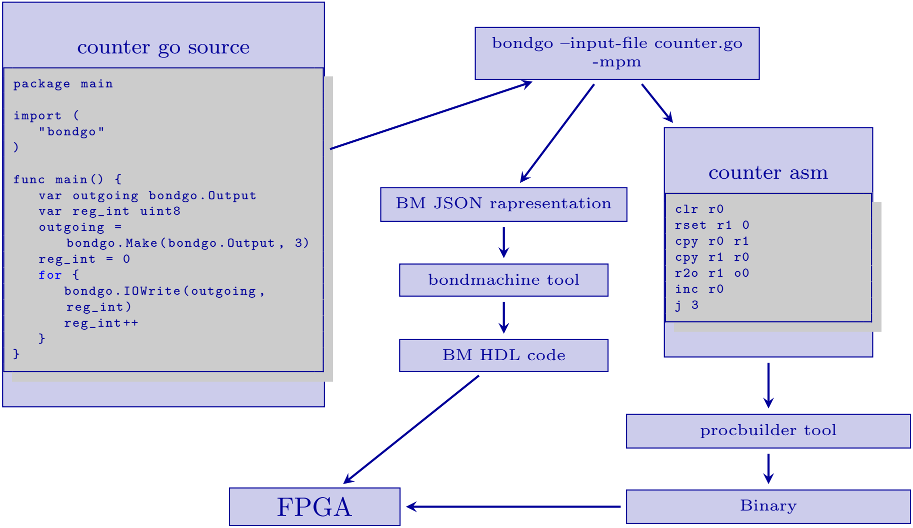
Bondgo

A first example



Bondgo

A first example



Bondgo

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

Bondgo

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

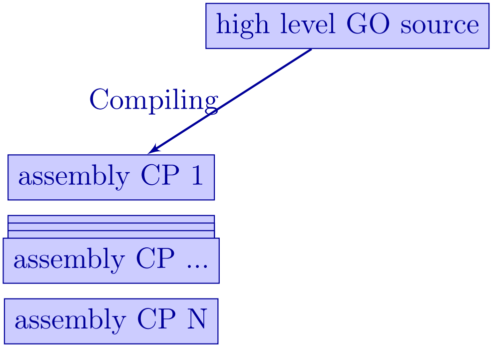
Bondgo

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

high level GO source

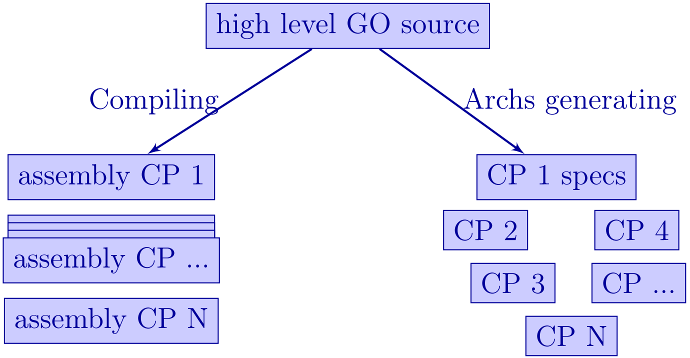
Bondgo

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



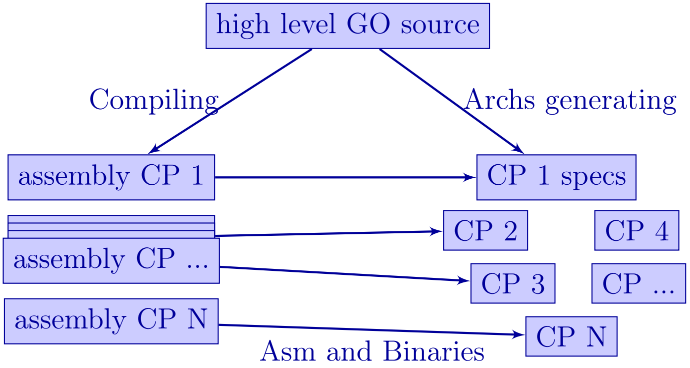
Bondgo

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



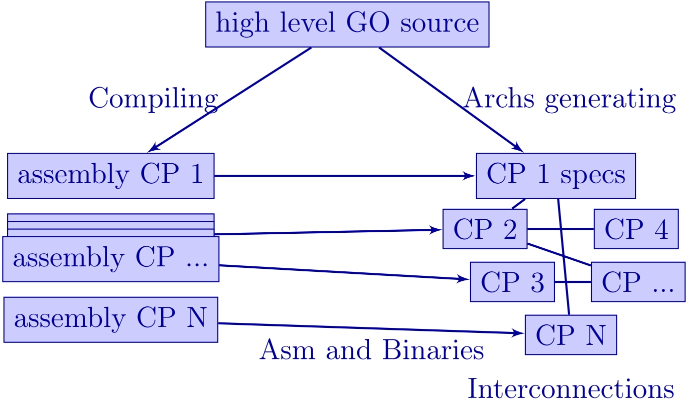
Bondgo

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



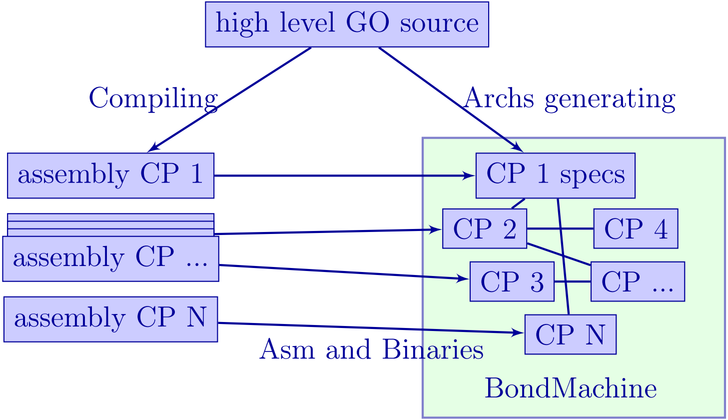
Bondgo

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



Bondgo

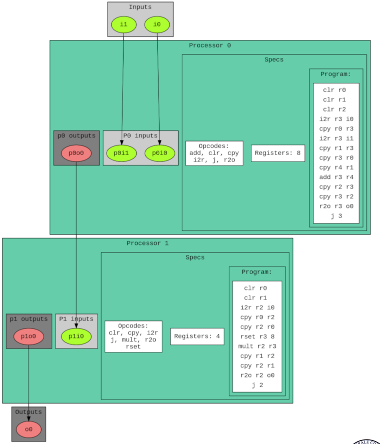
... it can do even much more interesting things when compiling concurrent programs.



Bondgo

A multi-core example

```
func multproc() {
  var in0 bondgo.Input
  var out0 bondgo.Output
  var reg_d_in uint8
  var reg_d_out uint8
  in0 = bondgo.Make(bondgo.Input, 4)
  out0 = bondgo.Make(bondgo.Output, 3)
  for {
    reg_d_in = bondgo.IORead(in0)
    reg_d_out = reg_d_in * 8
    bondgo.IOWrite(out0, reg_d_out)
  }
}
func main() {
  var in0 bondgo.Input
  var in1 bondgo.Input
  var out0 bondgo.Output
  var reg_d_in0 uint8
  var reg_d_in1 uint8
  var reg_d_out0 uint8
  in0 = bondgo.Make(bondgo.Input, 1)
  in1 = bondgo.Make(bondgo.Input, 2)
  out0 = bondgo.Make(bondgo.Output, 4)
device_0:
  go multproc()
  for {
    reg_d_in0 = bondgo.IORead(in0)
    reg_d_in1 = bondgo.IORead(in1)
    reg_d_out0 = reg_d_in0 + reg_d_in1
    bondgo.IOWrite(out0, reg_d_out0)
  }
}
```



Compiling Architectures

One of the most important result

The architecture creation is a part of the compilation process.

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

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.

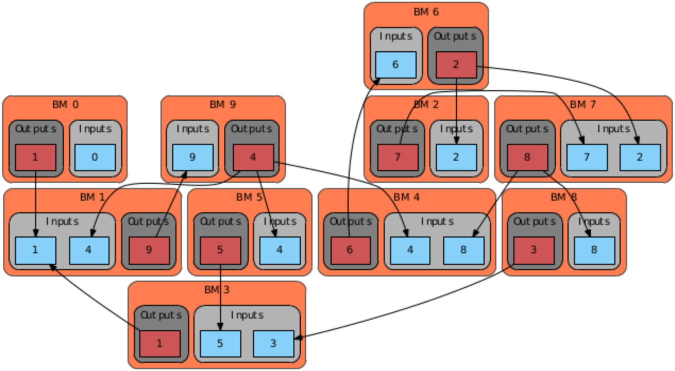
BondMachine Clustering

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.

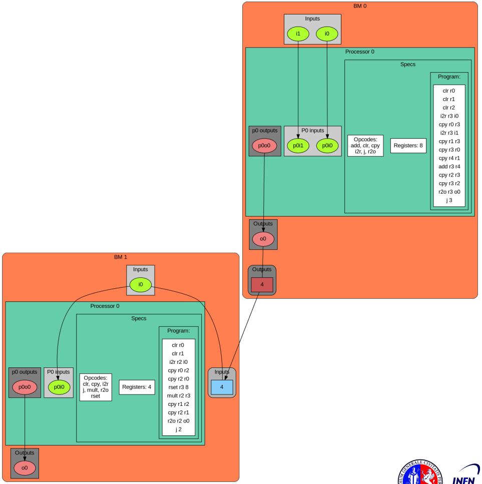
BondMachine Clustering



Bondgo

A cluster creation example

```
func multproc() {
  var in0 bondgo.Input
  var out0 bondgo.Output
  var reg_d_in uint8
  var reg_d_out uint8
  in0 = bondgo.Make(bondgo.Input, 4)
  out0 = bondgo.Make(bondgo.Output, 3)
  for {
    reg_d_in = bondgo.IORead(in0)
    reg_d_out = reg_d_in * 8
    bondgo.IOWrite(out0, reg_d_out)
  }
}
func main() {
  var in0 bondgo.Input
  var in1 bondgo.Input
  var out0 bondgo.Output
  var reg_d_in0 uint8
  var reg_d_in1 uint8
  var reg_d_out0 uint8
  in0 = bondgo.Make(bondgo.Input, 1)
  in1 = bondgo.Make(bondgo.Input, 2)
  out0 = bondgo.Make(bondgo.Output, 4)
device_1:
  go multproc()
  for {
    reg_d_in0 = bondgo.IORead(in0)
    reg_d_in1 = bondgo.IORead(in1)
    reg_d_out0 = reg_d_in0 + reg_d_in1
    bondgo.IOWrite(out0, reg_d_out0)
  }
}
```



BondMachine Clustering

A distributed example

The result is:
BondMachine Clustering Youtube video

A general result

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

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 (DAMPE, HERD)
  - increasing testbed operations efficiency

Possible other uses

The BondMachine could be used in several types of real world applications, some of them being:

- IoT and CyberPhysical systems.
- Computer Science educational applications.

Computing Accelerator

Our effort is now in enabling the possibility of building computing accelerators to be used from within standard (Linux) applications.

Possible other uses

The BondMachine could be used in several types of real world applications, some of them being:

- IoT and CyberPhysical systems.
- Computer Science educational applications.

Computing Accelerator

Our effort is now in enabling the possibility of building computing accelerators to be used from within standard (Linux) applications.

Possible other uses

The BondMachine could be used in several types of real world applications, some of them being:

- IoT and CyberPhysical systems.
- Computer Science educational applications.

Computing Accelerator

Our effort is now in enabling the possibility of building computing accelerators to be used from within standard (Linux) applications.

Possible other uses

The BondMachine could be used in several types of real world applications, some of them being:

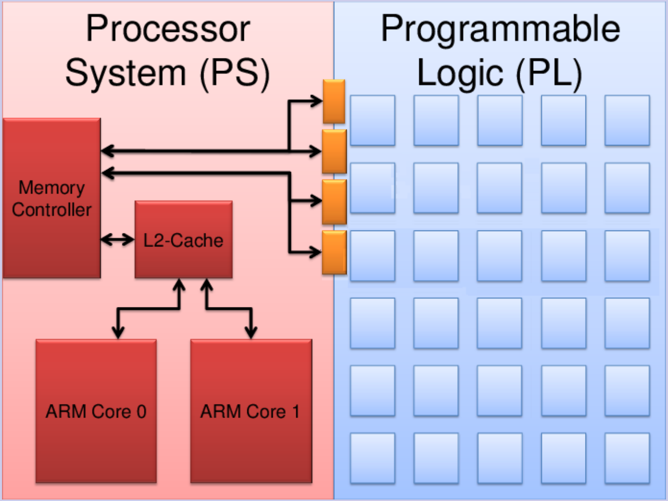
- IoT and CyberPhysical systems.
- Computer Science educational applications.

Computing Accelerator

Our effort is now in enabling the possibility of building computing accelerators to be used from within standard (Linux) applications.

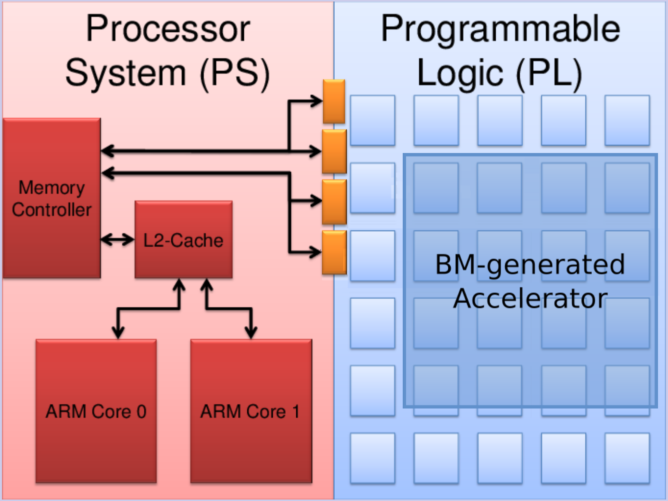
Accelerators

Hybrid chips



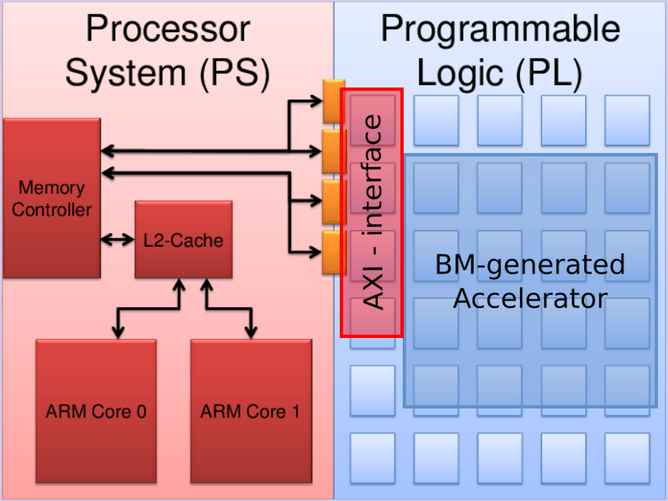
Accelerators

Hybrid chips

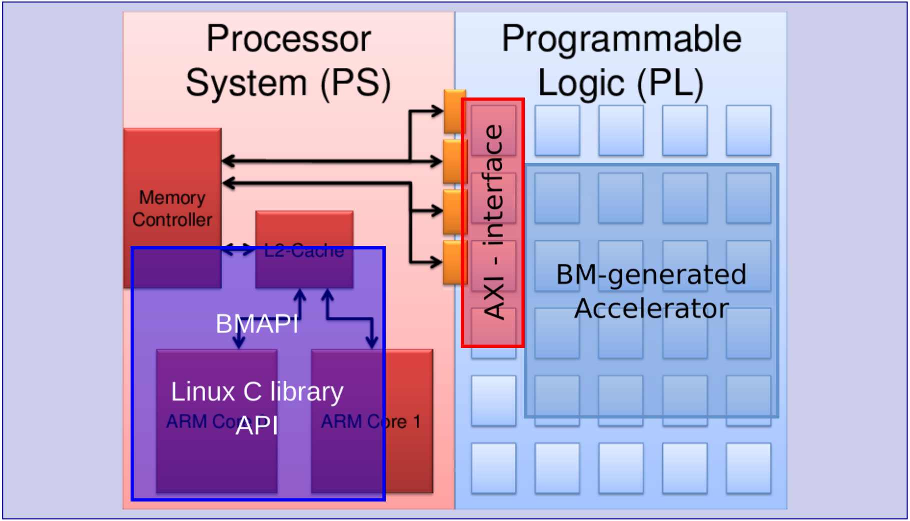


Accelerators

Hybrid chips



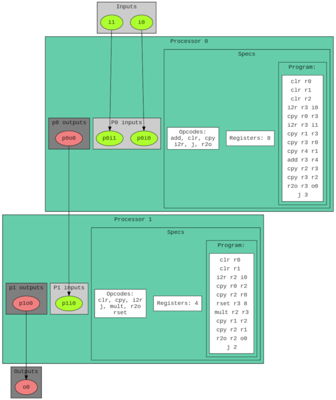
Accelerators
Hybrid chips



Accelerators

Example

```c
#include "bondmachineip1.h"
#include "bmapi.h"

/* Define the base memaddr of the BM IP core */
#define BM_BASE XPAR_BONDMACHINEIP1_0_S00_AXI_BASEADDR

int main(void)
{
  u32 input0 = 0, input1 = 0, output = 0;
  int i = 0, retval, input0_id = 0,  input1_id = 1,
      output_id = 0;

  /* Loop on input0 */
  for (input0 = 0 ; input0 < 5 ; input0 = input0 + 1 )
  {
    /* Loop on input1 */
    for (input1 = 0 ; input1 < 5 ; input1 = input1 + 1 )
    {
      /* Write value to the two accelerator inputs  */
      retval = BM_r2o(&input0, input0_id);
      retval = BM_r2o(&input1, input1_id);

      /* run a simple delay to allow changes on output  */
      for(i=0;i<DELAY;i++);

      /* Read the value produced by the accelerator  */
      retval = BM_i2r(&output, output_id);
    }
  }
  return 1;
}
```



Hardware implementation

FPGA

The HDL code for the BondMachine is written in Verilog and System Verilog, and has been tested on these devices/system:

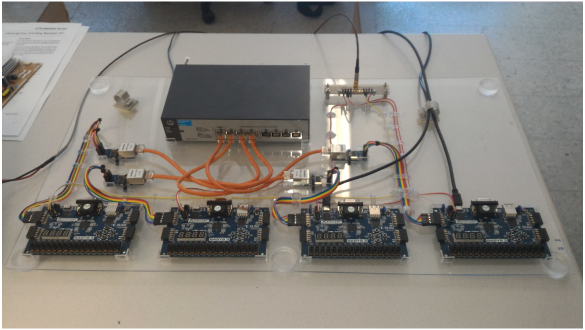
- Digilent Basys3 - Xilinx Artix-7 - Vivado.
- Kintex7 Evaluation Board - Vivado.
- Digilent Zedboard - Xilinx Zynq 7020 - Vivado.
- Linux - Iverilog.
- Terasic De10nano - Intel Cyclone V - Quartus

Within the project other firmwares have been written or tested:

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

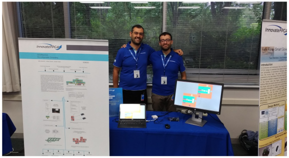
The Prototype

The project has been selected for the participation at MakerFaire 2016 Rome (The Europen Edition) and a prototype has been assembled and presented.

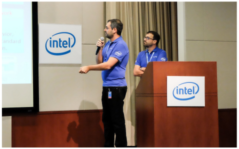


First run Youtube video

Project History



- May 2016 - First tests on the idea.
- October 2016 - Prototype at “Makerfaire 2016 Rome”
- Jul 2018 - InnovateFPGA EMEA Silver Award.
- Aug 2018 - Presented at Intel Campus, Santa Jose (CA) .
- Aug 2018 - InnovateFPGA Iron Award in the Grand Final.



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.

Next few months goals

- Improve the use of BondMachines as accelerators, integrating them into the ecosystem

Next few months goals

- Improve the use of BondMachines as accelerators, integrating them into the ecosystem
- Start making benchmarks

Next few months goals

- Improve the use of BondMachines as accelerators, integrating them into the ecosystem
- Start making benchmarks
- Integrate low and trans-precision instructions

Next few months goals

- Improve the use of BondMachines as accelerators, integrating them into the ecosystem
- Start making benchmarks
- Integrate low and trans-precision instructions
- Find a way to sustain the project

Next few months goals

- Improve the use of BondMachines as accelerators, integrating them into the ecosystem
- Start making benchmarks
- Integrate low and trans-precision instructions
- Find a way to sustain the project
- Move the repositories to github and open the code to the community

Future work

The project is at the stage of a working prototype, so work has to be done in several areas:

Future work

The project is at the stage of a working prototype, so work has to be done in several areas:

- Include new processor shared objects and currently unsupported opcodes.

Future work

The project is at the stage of a working prototype, so work has to be done in several areas:

- Include new processor shared objects and currently unsupported opcodes.
- Extend the compiler to include more data structures.

Future work

The project is at the stage of a working prototype, so work has to be done in several areas:

- Include new processor shared objects and currently unsupported opcodes.
- Extend the compiler to include more data structures.
- Improve the networking including new interconnection firmwares.

Future work

The project is at the stage of a working prototype, so work has to be done in several areas:

- Include new processor shared objects and currently unsupported opcodes.
- Extend the compiler to include more data structures.
- Improve the networking including new interconnection firmwares.

What would an OS for BondMachines look like ?

If you have question/curiosity on the project:

Mirko Mariotti
mirko.mariotti@unipg.it
http://bondmachine.fisica.unipg.it