Go, hardware, Go !

The BondMachine Project

Mirko Mariotti

Department of Physics and Geology - University of Perugia
INFN Perugia

October 22, 2018

@ GOLAB

Introduction

The BondMachine: a comprehensive approach to computing.

In this presentation i will talk about:

- Technological background of the project.
- Ideas behind the BondMachine.
- The Bondgo compiler and ancillary tools.
- Clustering.
- Conclusion.

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".

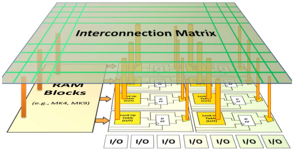
- Logic blocks can be configured to perform complex combinational functions.

- 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".



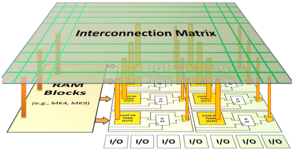
- Logic blocks can be configured to perform complex combinational functions.

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

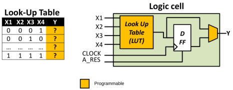
FPGA

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.

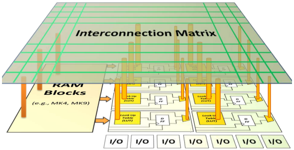


- The FPGA configuration is generally specified using a hardware description language (HDL).

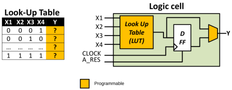
FPGA

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

Why is used ?

- Prototyping, hardware testing, etc.
- Low development cost and short time to market.
- Computing

FPGA

Why is used ?

- Prototyping, hardware testing, etc.
- Low development cost and short time to market.
- Computing

FPGA

Why is used ?

- Prototyping, hardware testing, etc.
- Low development cost and short time to market.
- Computing

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 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 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 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 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 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 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 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 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 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 scheduling has to be addressed.

The BondMachine

The idea

High level source code: Go

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 idea

High level source code: Go

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 Special-ization

The BondMachine

The idea

High level source code: Go

↓

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 idea

High level source code: Go

↓

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 idea

High level source code: Go

↓

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).

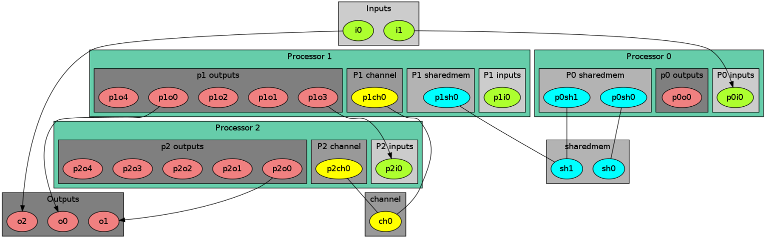
Introducing the BondMachine (BM)

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 the BM computer architecture

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 Register Tranfer Code (RTL)

Processor Builder

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

BondMachine Builder

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

Simulation Framework

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

Handle the BM computer architecture

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 Register Tranfer Code (RTL)

Processor Builder

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

BondMachine Builder

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

Simulation Framework

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

Handle the BM computer architecture

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 Register Tranfer Code (RTL)

Processor Builder

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

BondMachine Builder

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

Simulation Framework

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

Handle the BM computer architecture

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 Register Tranfer Code (RTL)

Processor Builder

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

BondMachine Builder

Connects CPs and SOs together in custom topologies, loads and saves on disk as JSON, create BM's RTL 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 expressoin to BM

Boolbond

Map boolean systems to BM

Matrixwork

Basic matrix computation

Neuralbond

Map neural networks to BM

Evolutive BM

Evolutionary computing to BM

tf2bm

Map TensorFlow graphs to BM

Use the BM computer architecture

Mapping specific computational problems to BMs

Symbond

Map symbolic mathematical expressoin to BM

Boolbond

Map boolean systems to BM

Matrixwork

Basic matrix computation

Neuralbond

Map neural networks to BM

Evolutive BM

Evolutionary computing to BM

tf2bm

Map TensorFlow graphs to BM

Use the BM computer architecture

Mapping specific computational problems to BMs

Symbond

Map symbolic mathematical expressoin to BM

Boolbond

Map boolean systems to BM

Matrixwork

Basic matrix computation

Neuralbond

Map neural networks to BM

Evolutive BM

Evolutionary computing to BM

tf2bm

Map TensorFlow graphs to BM

Use the BM computer architecture

Mapping specific computational problems to BMs

Symbond

Map symbolic mathematical expressoin to BM

Boolbond

Map boolean systems to BM

Matrixwork

Basic matrix computation

Neuralbond

Map neural networks to BM

Evolutive BM

Evolutionary computing to BM

tf2bm

Map TensorFlow graphs to BM

Use the BM computer architecture

Mapping specific computational problems to BMs

Symbond

Map symbolic mathematical expressoin to BM

Boolbond

Map boolean systems to BM

Matrixwork

Basic matrix computation

Neuralbond

Map neural networks to BM

Evolutive BM

Evolutionary computing to BM

tf2bm

Map TensorFlow graphs to BM

Use the BM computer architecture

Mapping specific computational problems to BMs

Symbond

Map symbolic mathematical expressoin to BM

Boolbond

Map boolean systems to BM

Matrixwork

Basic matrix computation

tf2bm

Map TensorFlow graphs to BM

Neuralbond

Map neural networks to BM

Evolutive BM

Evolutionary computing to BM

Use the BM computer architecture

Mapping specific computational problems to BMs

Symbond

Map symbolic mathematical expressoin to BM

Boolbond

Map boolean systems to BM

Matrixwork

Basic matrix computation

Neuralbond

Map neural networks to BM

Evolutive BM

Evolutionary computing to BM

tf2bm

Map TensorFlow 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

Bondgo does something different from standard compilers ...

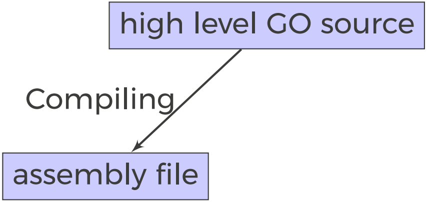
Bondgo

Bondgo does something different from standard compilers ...

high level GO source

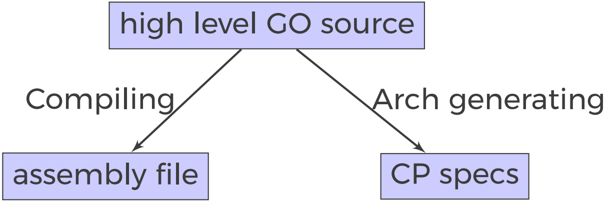
Bondgo

Bondgo does something different from standard compilers ...



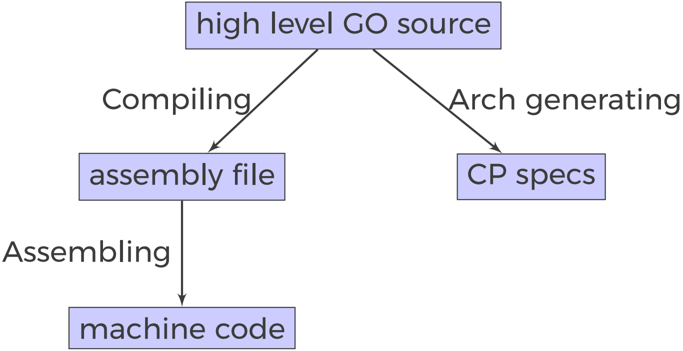
Bondgo

Bondgo does something different from standard compilers ...



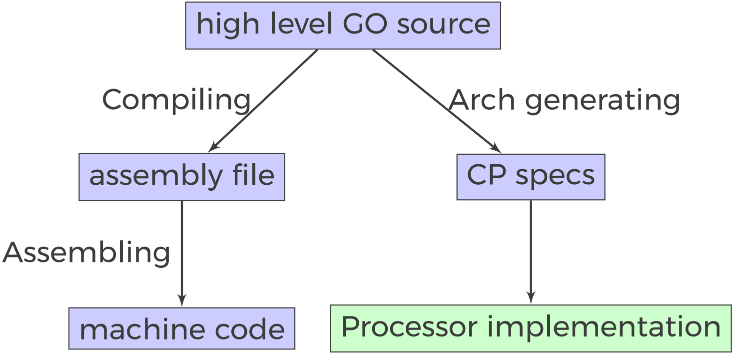
Bondgo

Bondgo does something different from standard compilers ...



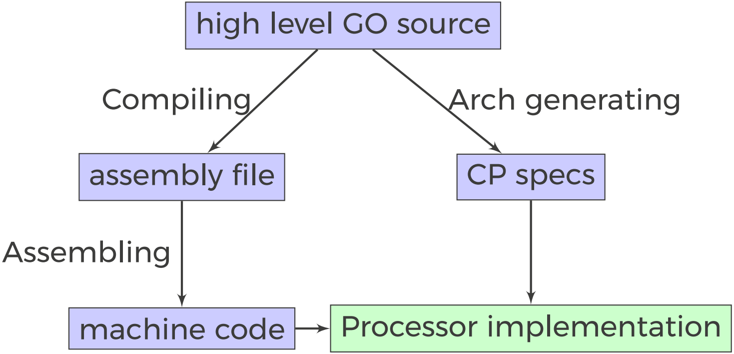
Bondgo

Bondgo does something different from standard compilers ...



Bondgo

Bondgo does something different from standard compilers ...

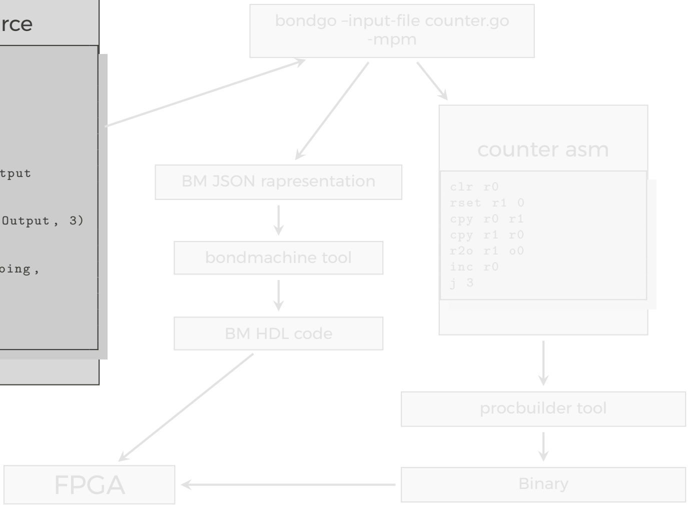


Bondgo

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++
   }
}
```

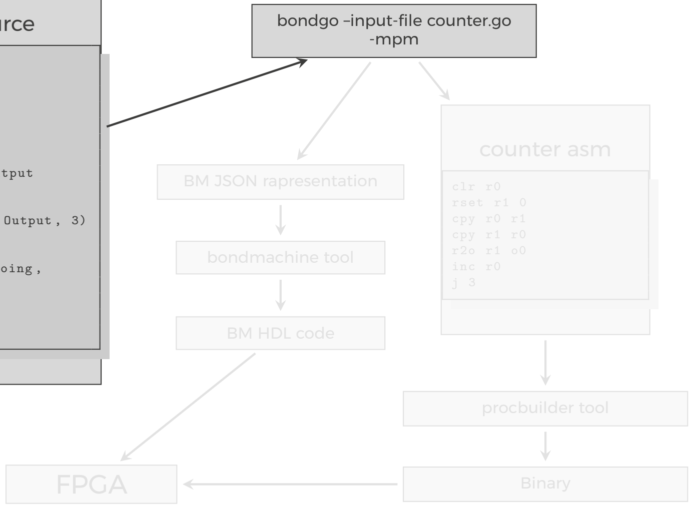


Bondgo

A first example

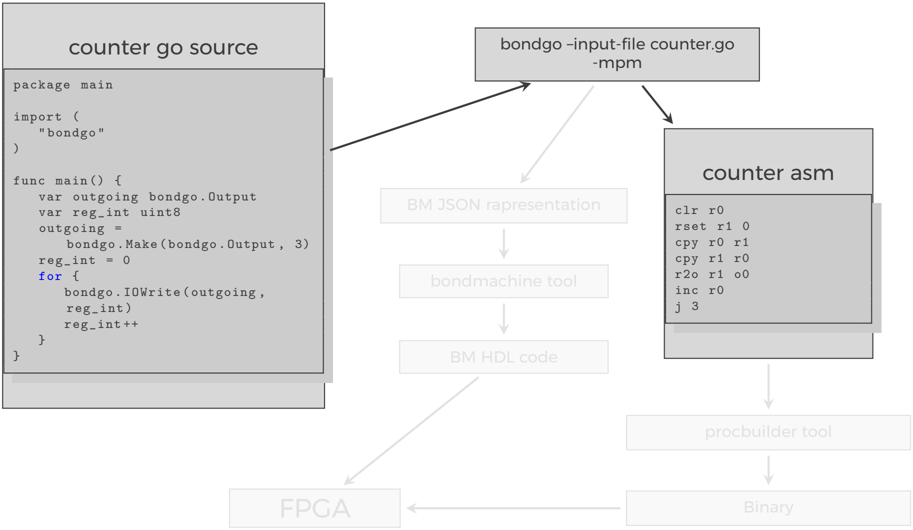
counter go source

```
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

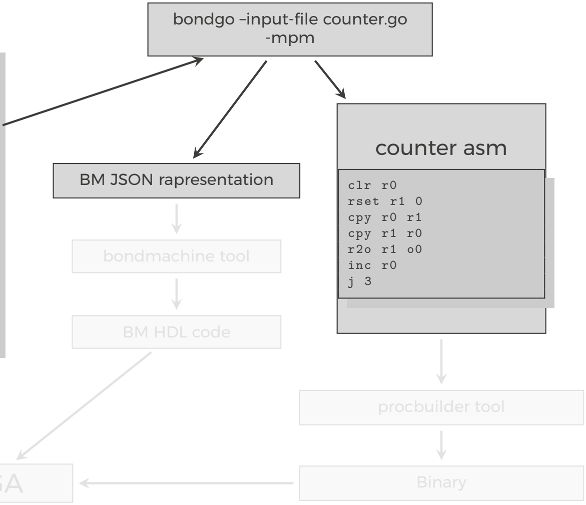


Bondgo

A first example

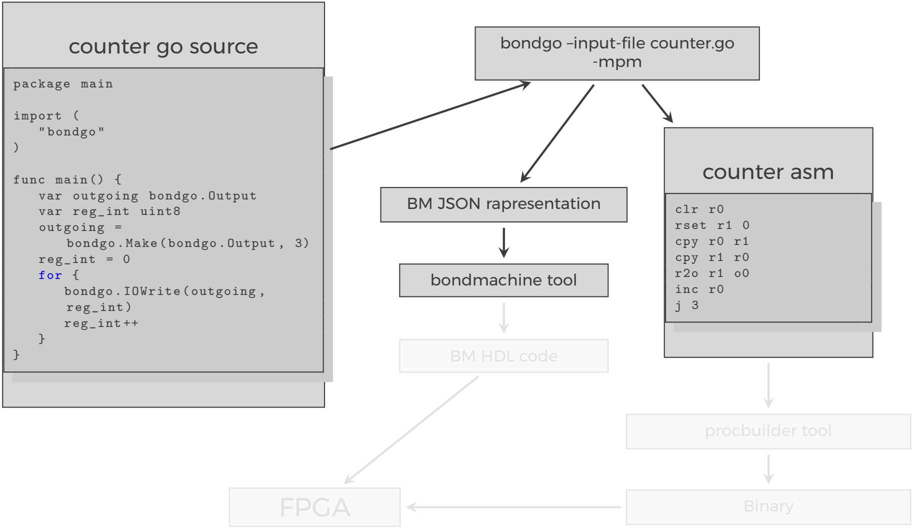
counter go source

```
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++
   }
}
```



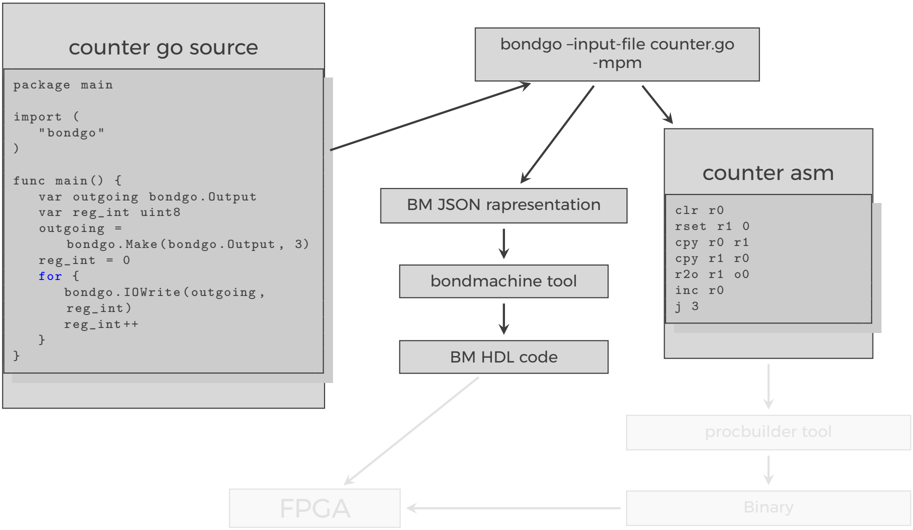
Bondgo

A first example



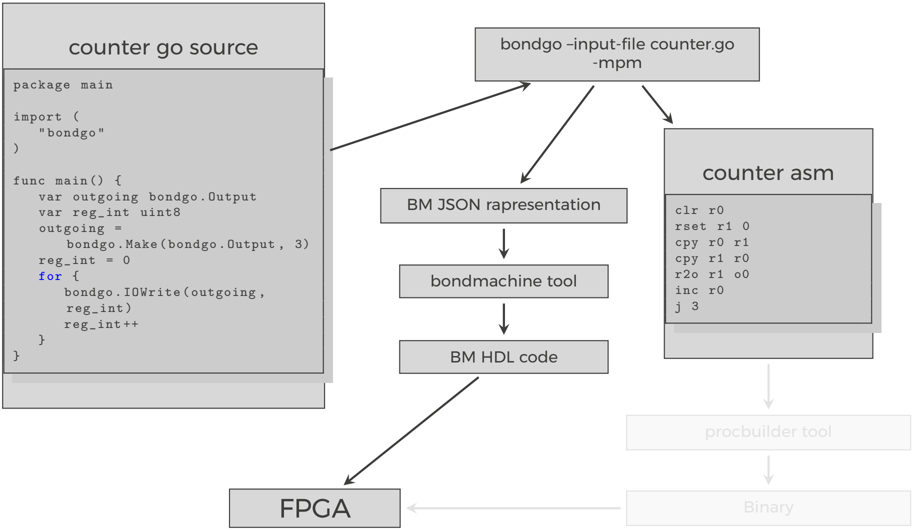
Bondgo

A first example



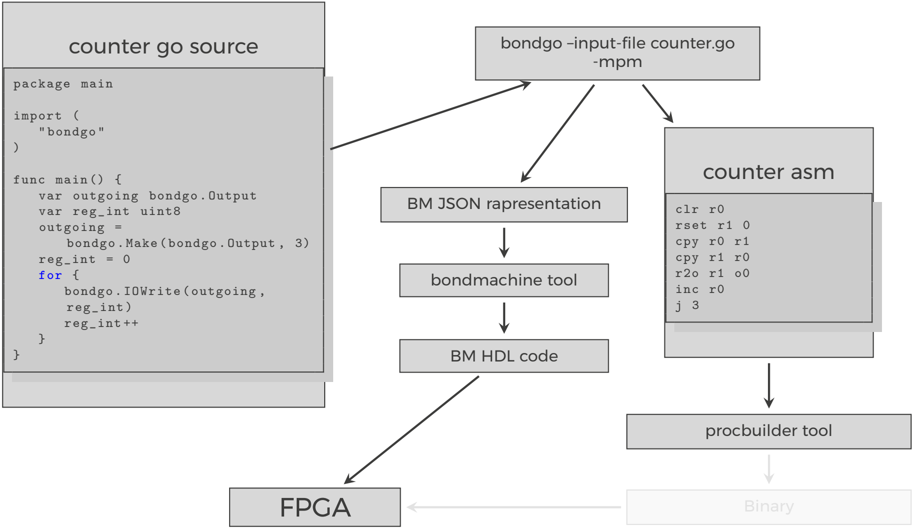
Bondgo

A first example



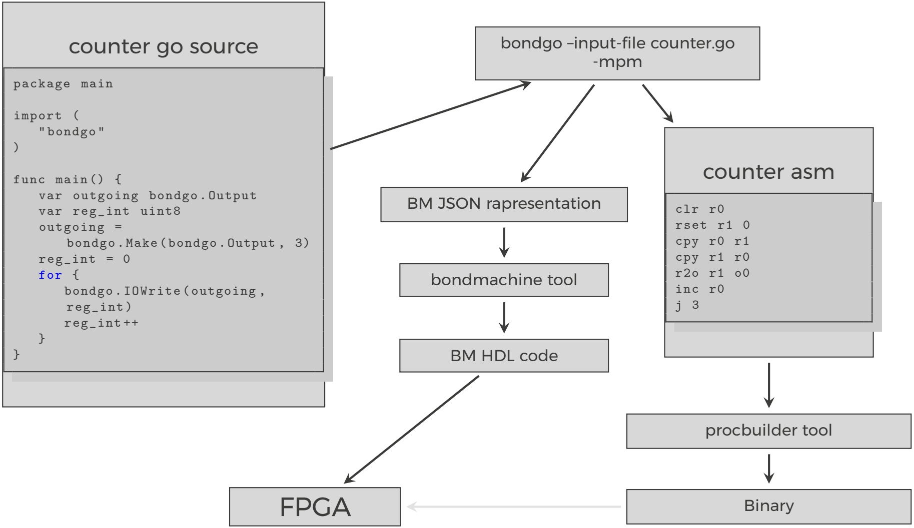
Bondgo

A first example



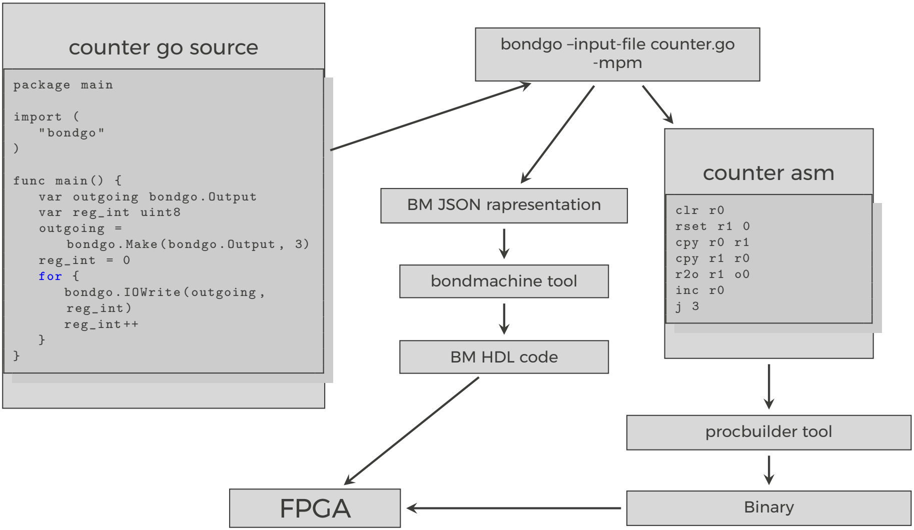
Bondgo

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.

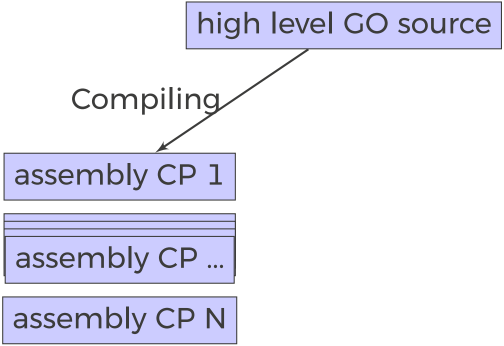
Bondgo

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

high level GO source

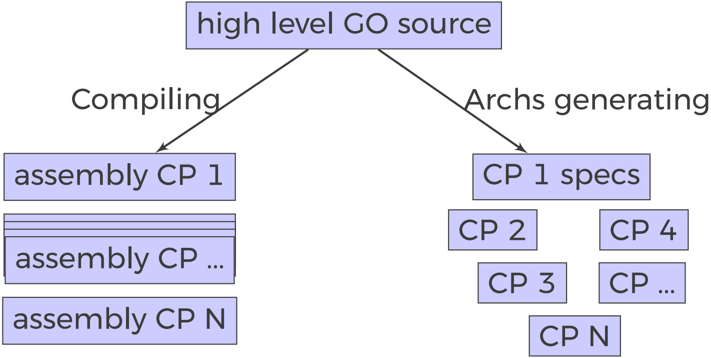
Bondgo

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



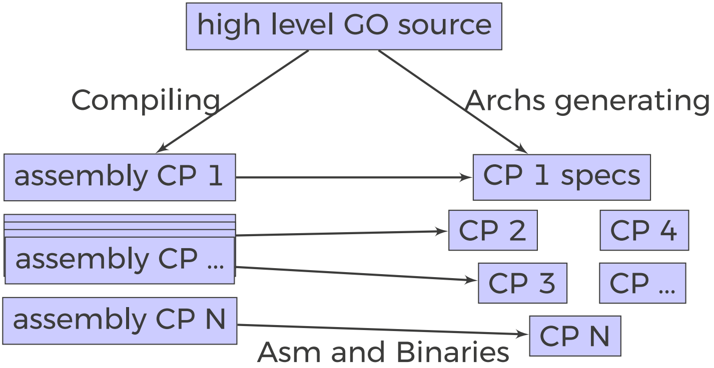
Bondgo

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



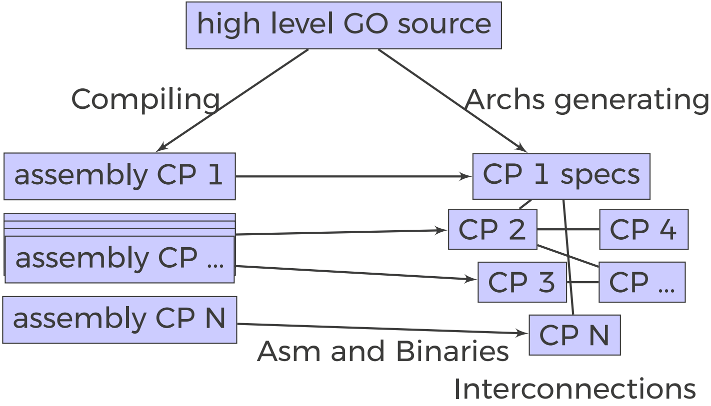
Bondgo

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



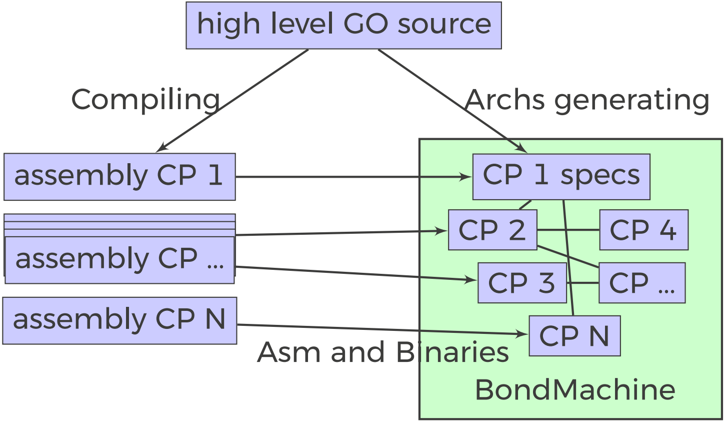
Bondgo

... 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

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

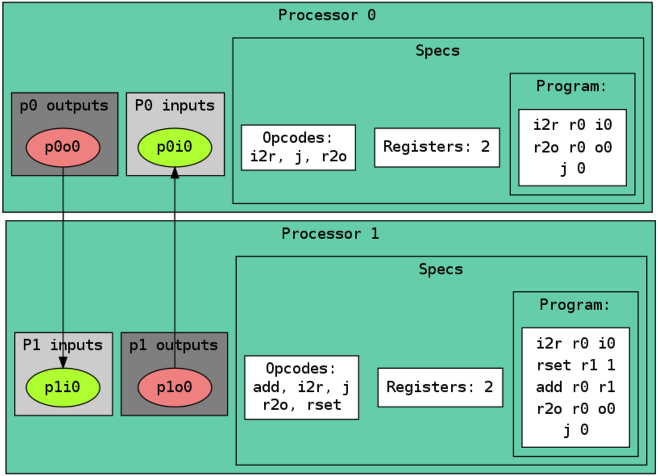
The toolchain perform the following steps:

- 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

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.

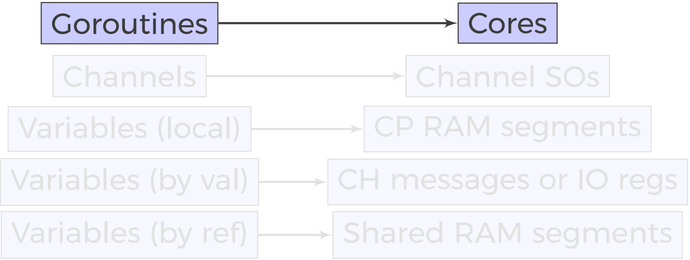
- **Goroutines** → Cores
- Channels → Channel SOs
- Variables (local) → CP RAM segments
- Variables (by val) → CH messages or IO regs
- Variables (by ref) → Shared RAM segments

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.

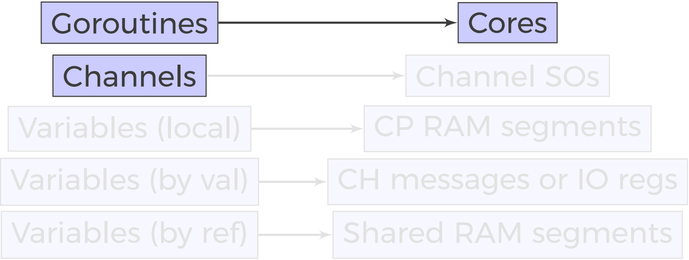


Bondgo

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.

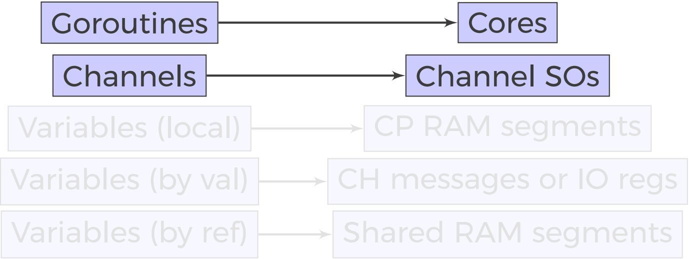


Bondgo

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.

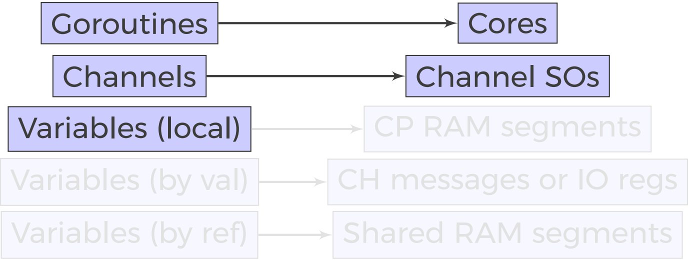


Bondgo

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.

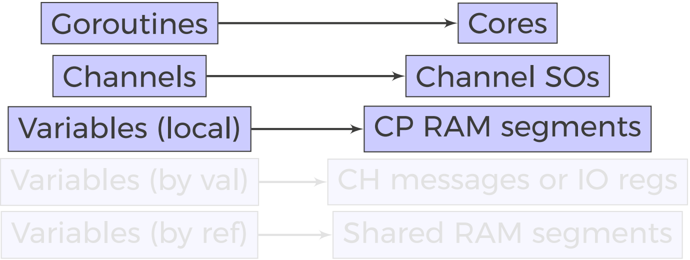


Bondgo

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.

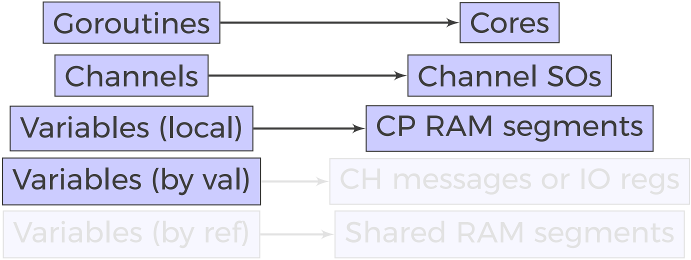


Bondgo

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.

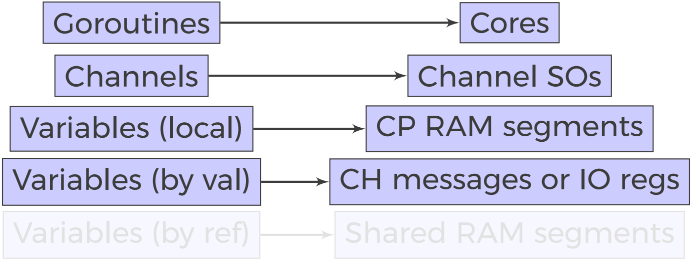


Bondgo

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.

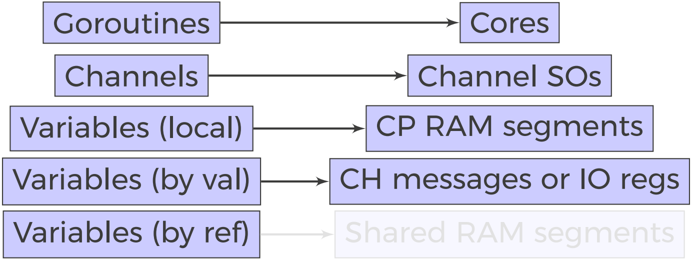


Bondgo

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.

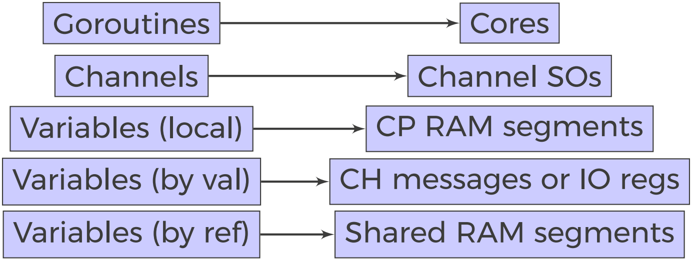


Bondgo

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

An example

bondgo stream processing example

```
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.

Hardware implementation

FPGA

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

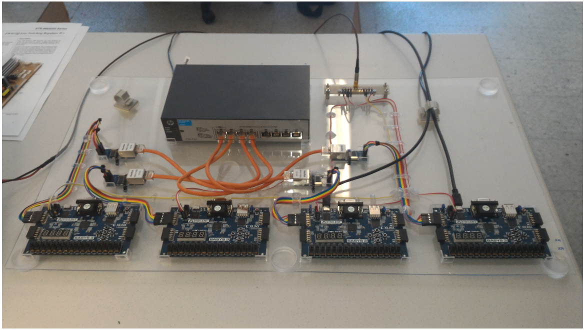
- Digilent Basys3 - Xilinx Artix-7 - Vivado.
- Kintex7 Evaluation Board - Vivado.
- Digilent Zedboard - Xilinx Zynq 7020 - Vivado.
- Linux - Iverilog.
- Terasic De10nano - Intel Cyclone V - Quartus

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:

https://youtube.com/embed/hukTrGxTb7A

Toolchains

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

Bondgo Toolchain example

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 show displays a graphical representation of the BM.
make simulate start a simulation.
make videosim create a simulation video.
make flash the device into the destination target.

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.

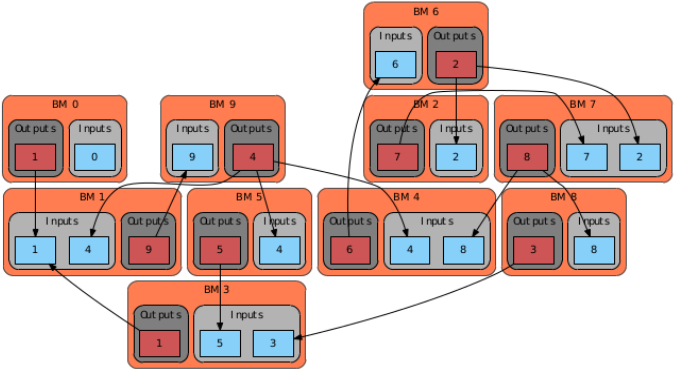
BondMachine Clustering

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

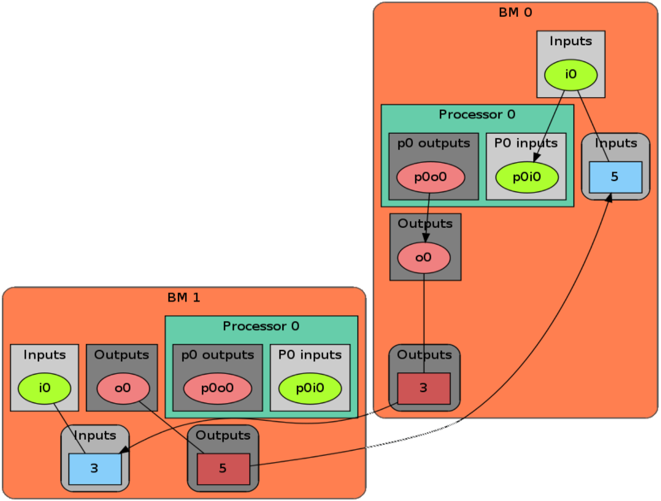
A distributed example

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

The result is:

`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 a code written in Go.
- 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 a code written in Go.
- Workstation with emulated BondMachines, workstation with etherbond drivers, standalone BondMachines (FPGA) may join these clusters.

Project History

- May 2016 - Idea presented at INFN-computing and Networking Commission Workshop 2016.
- September 2016 - The first prototype is built.
- October 2016 - It is Selected and the prototype is presented at "Makerfaire 2016 Rome (The European edition)".
- November 2016 - Presented at "Umbria Business Match 2016".
- March 2017 - First tests for Physics applications.
- November 2017 - Presented at "Umbria Business Match 2017".
- December 2107 - Submitted at InnovateFPGA 2018

Project History

- May 2016 - Idea presented at INFN-computing and Networking Commission Workshop 2016.
- September 2016 - The first prototype is built.
- October 2016 - It is Selected and the prototype is presented at “Makerfaire 2016 Rome (The European edition)”.
- November 2016 - Presented at “Umbria Business Match 2016”.
- March 2017 - First tests for Physics applications.
- November 2017 - Presented at “Umbria Business Match 2017”.
- December 2107 - Submitted at InnovateFPGA 2018

Project History

- May 2016 - Idea presented at INFN-computing and Networking Commission Workshop 2016.
- September 2016 - The first prototype is built.
- October 2016 - It is Selected and the prototype is presented at "Makerfaire 2016 Rome (The European edition)".
- November 2016 - Presented at "Umbria Business Match 2016".
- March 2017 - First tests for Physics applications.
- November 2017 - Presented at "Umbria Business Match 2017".
- December 2107 - Submitted at InnovateFPGA 2018

Project History

InnovateFPGA 2018

- Feb 2018 - Reached the EMEA Semifinal.
- Jun 2018 - Reached the EMEA Regional final.
- Jul 2018 - EMEA Silver Award, Reached the Grand Final.
- Aug 2018 - Presented at Intel Campus, Santa Jose (CA) .
- Aug 2018 - Won the Iron Award in the Grand Final.

Project History

InnovateFPGA 2018

- Feb 2018 - Reached the EMEA Semifinal.
- Jun 2018 - Reached the EMEA Regional final.
- Jul 2018 - EMEA Silver Award, Reached the Grand Final.
- Aug 2018 - Presented at Intel Campus, Santa Jose (CA) .
- Aug 2018 - Won the Iron Award in the Grand Final.

Project History

InnovateFPGA 2018

- Feb 2018 - Reached the EMEA Semifinal.
- Jun 2018 - Reached the EMEA Regional final.
- Jul 2018 - EMEA Silver Award, Reached the Grand Final.
- Aug 2018 - Presented at Intel Campus, Santa Jose (CA) .
- Aug 2018 - Won the Iron Award in the Grand Final.

Project History

InnovateFPGA 2018


- Feb 2018 - Reached the EMEA Semifinal.
- Jun 2018 - Reached the EMEA Regional final.
- Jul 2018 - EMEA Silver Award, Reached the Grand Final.
- Aug 2018 - Presented at Intel Campus, Santa Jose (CA) .
- Aug 2018 - Won the Iron Award in the Grand Final.

Project History

InnovateFPGA 2018

- Feb 2018 - Reached the EMEA Semifinal.
- Jun 2018 - Reached the EMEA Regional final.
- Jul 2018 - EMEA Silver Award, Reached the Grand Final.
- Aug 2018 - Presented at Intel Campus, Santa Jose (CA) .
- Aug 2018 - Won the 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.

Keeping the register machine abstraction it is possible to use language like Go, removing the need of having a general purpose architecture.

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.

Future work

- The project is a prototype.
- Include new processor shared objects and currently unsupported opcodes.
- Extend the compiler to include more data structures.
- Improve the networking including new interconnection firmwares.
- Work on BondMachine as accelerators.
- What would an OS for BondMachines look like ?

Future work

- The project is a prototype.
- Include new processor shared objects and currently unsupported opcodes.
- Extend the compiler to include more data structures.
- Improve the networking including new interconnection firmwares.
- Work on BondMachine as accelerators.
- What would an OS for BondMachines look like ?

Future work

- The project is a prototype.
- Include new processor shared objects and currently unsupported opcodes.
- Extend the compiler to include more data structures.
- Improve the networking including new interconnection firmwares.
- Work on BondMachine as accelerators.
- What would an OS for BondMachines look like ?

Future work

- The project is a prototype.
- Include new processor shared objects and currently unsupported opcodes.
- Extend the compiler to include more data structures.
- Improve the networking including new interconnection firmwares.
- Work on BondMachine as accelerators.
- What would an OS for BondMachines look like ?

Future work

- The project is a prototype.
- Include new processor shared objects and currently unsupported opcodes.
- Extend the compiler to include more data structures.
- Improve the networking including new interconnection firmwares.
- Work on BondMachine as accelerators.
- What would an OS for BondMachines look like ?

Future work

- The project is a prototype.
- Include new processor shared objects and currently unsupported opcodes.
- Extend the compiler to include more data structures.
- Improve the networking including new interconnection firmwares.
- Work on BondMachine as accelerators.
- What would an OS for BondMachines look like ?

If you have question/curiosity/interest for joining the project:

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