The BondMachine project

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

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

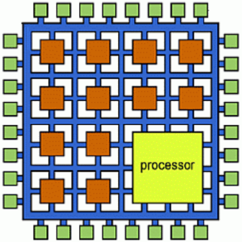
[2]INFN sezione di Perugia

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

Outline

Introduction

Processors in FPGA



It is a common practice to use FPGAs to implement processors. Some processors are directly created by the FPGA manufacturer, some are open-source, some are proprietary.

Now with the advent of the RISC-V architecture, it is clear that this will be more and more used in the future.

This is the also the case of the BondMachine project but with a different approach.

The Challenge

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 Challenge

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 Challenge

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 Challenge

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 Challenge

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 project

Introducing the BondMachine (BM)

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

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

Introducing the BondMachine (BM)

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

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

Introducing the BondMachine (BM)

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

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

Introducing the BondMachine (BM)

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

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

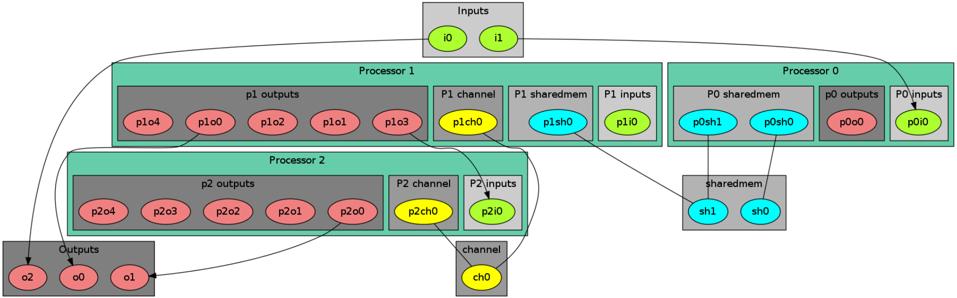
Introducing the BondMachine (BM)

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

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

The BondMachine

An example



Connecting Processor (CP)

The computational unit of the BM

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

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

Connecting Processor (CP)

The computational unit of the BM

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

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

General purpose registers

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

Connecting Processor (CP)

The computational unit of the BM

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

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

I/O specialized registers

N input registers: i0,i1 ... iN
M output registers: o0,o1 ... oM

Connecting Processor (CP)

The computational unit of the BM

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

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

Full set of possible opcodes

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

Connecting Processor (CP)

The computational unit of the BM

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

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

RAM and ROM

- 2^L RAM memory cells.
- 2^O ROM memory cells.

Connecting Processor (CP)

The computational unit of the BM

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

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

Operating modes

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

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.

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.

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.

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.

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.

Shared Objects (SO)

Currently implemented SOs

Shared Memory

A Memory shared among CPs

Channel

A CSP style channel for message passing

Barrier

A barrier for CP synchronization

Shared Stack

A stack shared among CPs

Shared Queue

A queue shared among CPs

Pseudo Random Number Generator

more about these

Handle BM computer architectures

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

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

Processor Builder

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

BondMachine Builder

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

Simulation Framework

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

Handle BM computer architectures

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

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

Processor Builder

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

BondMachine Builder

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

Simulation Framework

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

Handle BM computer architectures

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

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

Processor Builder

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

BondMachine Builder

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

Simulation Framework

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

Handle BM computer architectures

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

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

Processor Builder

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

BondMachine Builder

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

Simulation Framework

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

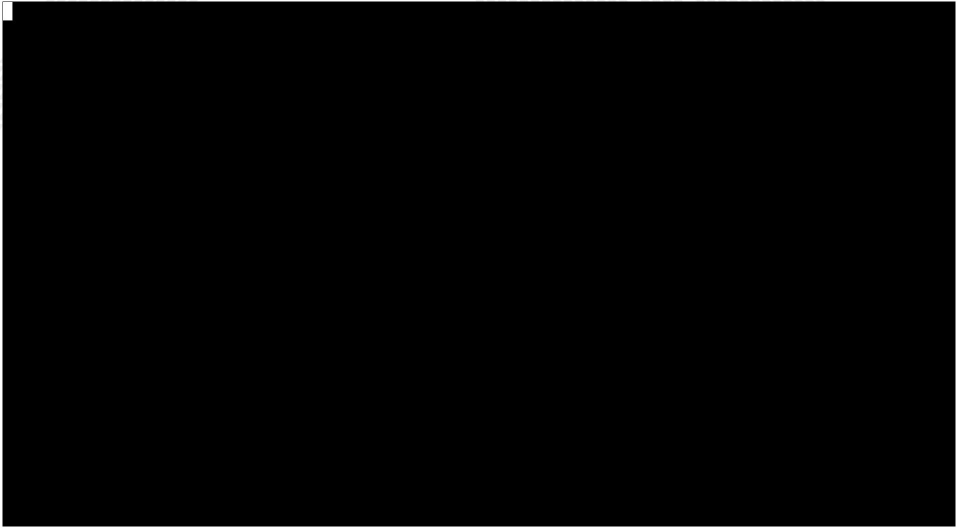
Toolchain and helper tool

A BondMachine Project is a directory containing all the necessary files to build a BondMachine.

A set of tools have been developed to simplify the creation and maintenance of the BM Projects.

bmhelper	Makefile	Kconfig
Project manteinance tool	Project targets	Kernel style configuration

A first example



A first example

```
[ Command >
bmhelper create --project_name Example
[ Output  >
 [ OK ]          Project has been successfully created.
```

A first example

```
[ Command >
bmhelper create --project_name Example
[ Output  >
 [ OK ]          Project has been successfully created.
[ Command >
cd Example
```

A first example

```
[ Command >
bmhelper create --project_name Example
[ Output  >
 [ OK ]          Project has been successfully created.
[ Command >
cd Example
[ Command >
cat <<EOF > cp0code.asm
clr r0
lfsr82r r0 lfsr80
r2o r0 o0
j 1
EOF
```

A first example

```
[ Command >
bmhelper create --project_name Example
[ Output  >
 [ OK ]          Project has been successfully created.
[ Command >
cd Example
[ Command >
cat <<EOF > cp0code.asm
clr r0
lfsr82r r0 lfsr80
r2o r0 o0
j 1
EOF
[ Command >
procbuilder -register-size 8 -registers 2 -inputs 0 -outputs 1 -opcodes clr,j,r2o,lfsr82r -save-mach
ine cp0.json -input-assembly cp0code.asm -shared-constraints "lfsr8:34"
```

A first example

```
[ Command >
bmhelper create --project_name Example
[ Output  >
 [ OK ]         Project has been successfully created.
[ Command >
cd Example
[ Command >
cat <<EOF > cp0code.asm
clr r0
lfsr82r r0 lfsr80
r2o r0 o0
j 1
EOF
[ Command >
procbuilder -register-size 8 -registers 2 -inputs 0 -outputs 1 -opcodes clr,j,r2o,lfsr82r -save-mach
ine cp0.json -input-assembly cp0code.asm -shared-constraints "lfsr8:34"
[ Command >
cat <<EOF > cp1code.asm
i2r r0 i0
r2o r0 o0
j 0
EOF
```

A first example

```
[ Command >
bmhelper create --project_name Example
[ Output  >
 [ OK ]         Project has been successfully created.
[ Command >
cd Example
[ Command >
cat <<EOF > cp0code.asm
clr r0
lfsr82r r0 lfsr80
r2o r0 o0
j 1
EOF
[ Command >
procbuilder -register-size 8 -registers 2 -inputs 0 -outputs 1 -opcodes clr,j,r2o,lfsr82r -save-mach
ine cp0.json -input-assembly cp0code.asm -shared-constraints "lfsr8:34"
[ Command >
cat <<EOF > cp1code.asm
i2r r0 i0
r2o r0 o0
j 0
EOF
[ Command >
procbuilder -register-size 8 -registers 2 -inputs 1 -outputs 1 -opcodes j,i2r,r2o -save-machine cp1.
json -input-assembly cp1code.asm
```

A first example

```
[ Command >
bmhelper create --project_name Example
[ Output  >
 [ OK ]          Project has been successfully created.
[ Command >
cd Example
[ Command >
cat <<EOF > cp0code.asm
clr r0
lfsr82r r0 lfsr80
r2o r0 o0
j 1
EOF
[ Command >
procbuilder -register-size 8 -registers 2 -inputs 0 -outputs 1 -opcodes clr,j,r2o,lfsr82r -save-mach
ine cp0.json -input-assembly cp0code.asm -shared-constraints "lfsr8:34"
[ Command >
cat <<EOF > cp1code.asm
i2r r0 i0
r2o r0 o0
j 0
EOF
[ Command >
procbuilder -register-size 8 -registers 2 -inputs 1 -outputs 1 -opcodes j,i2r,r2o -save-machine cp1.
json -input-assembly cp1code.asm
[ Command >
bondmachine -bondmachine-file bondmachine.json
```

A first example

```
clr r0
lfsr82r r0 lfsr80
r2o r0 o0
j 1
EOF
[ Command >
procbuilder -register-size 8 -registers 2 -inputs 0 -outputs 1 -opcodes clr,j,r2o,lfsr82r -save-mach
ine cp0.json -input-assembly cp0code.asm -shared-constraints "lfsr8:34"
[ Command >
cat <<EOF > cp1code.asm
i2r r0 i0
r2o r0 o0
j 0
EOF
[ Command >
procbuilder -register-size 8 -registers 2 -inputs 1 -outputs 1 -opcodes j,i2r,r2o -save-machine cp1.
json -input-assembly cp1code.asm
[ Command >
bondmachine -bondmachine-file bondmachine.json
[ Command >
bondmachine -bondmachine-file bondmachine.json -add-domains cp0.json
bondmachine -bondmachine-file bondmachine.json -add-processor 0
bondmachine -bondmachine-file bondmachine.json -add-domains cp1.json
bondmachine -bondmachine-file bondmachine.json -add-processor 1
[ Output  >
Processor 0 successfully added
Processor 1 successfully added
```

A first example

```
r2o r0 o0
j 1
EOF
[ Command >
procbuilder -register-size 8 -registers 2 -inputs 0 -outputs 1 -opcodes clr,j,r2o,lfsr82r -save-mach
ine cp0.json -input-assembly cp0code.asm -shared-constraints "lfsr8:34"
[ Command >
cat <<EOF > cp1code.asm
i2r r0 i0
r2o r0 o0
j 0
EOF
[ Command >
procbuilder -register-size 8 -registers 2 -inputs 1 -outputs 1 -opcodes j,i2r,r2o -save-machine cp1.
json -input-assembly cp1code.asm
[ Command >
bondmachine -bondmachine-file bondmachine.json
[ Command >
bondmachine -bondmachine-file bondmachine.json -add-domains cp0.json
bondmachine -bondmachine-file bondmachine.json -add-processor 0
bondmachine -bondmachine-file bondmachine.json -add-domains cp1.json
bondmachine -bondmachine-file bondmachine.json -add-processor 1
[ Output  >
Processor 0 successfully added
Processor 1 successfully added
[ Command >
bondmachine -bondmachine-file bondmachine.json -add-outputs 1
```

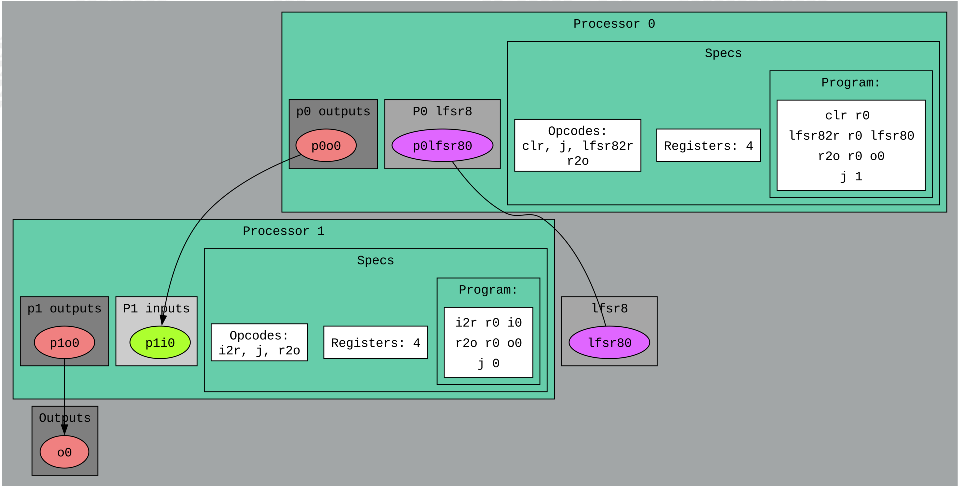
A first example

```
[ Command >
procbuilder -register-size 8 -registers 2 -inputs 0 -outputs 1 -opcodes clr,j,r2o,lfsr82r -save-mach
ine cp0.json -input-assembly cp0code.asm -shared-constraints "lfsr8:34"
[ Command >
cat <<EOF > cp1code.asm
i2r r0 i0
r2o r0 o0
j 0
EOF
[ Command >
procbuilder -register-size 8 -registers 2 -inputs 1 -outputs 1 -opcodes j,i2r,r2o -save-machine cp1.
json -input-assembly cp1code.asm
[ Command >
bondmachine -bondmachine-file bondmachine.json
[ Command >
bondmachine -bondmachine-file bondmachine.json -add-domains cp0.json
bondmachine -bondmachine-file bondmachine.json -add-processor 0
bondmachine -bondmachine-file bondmachine.json -add-domains cp1.json
bondmachine -bondmachine-file bondmachine.json -add-processor 1
[ Output  >
Processor 0 successfully added
Processor 1 successfully added
[ Command >
bondmachine -bondmachine-file bondmachine.json -add-outputs 1
[ Command >
bondmachine -bondmachine-file bondmachine.json -add-bond p0o0,p1i0
bondmachine -bondmachine-file bondmachine.json -add-bond p1o0,o0
```

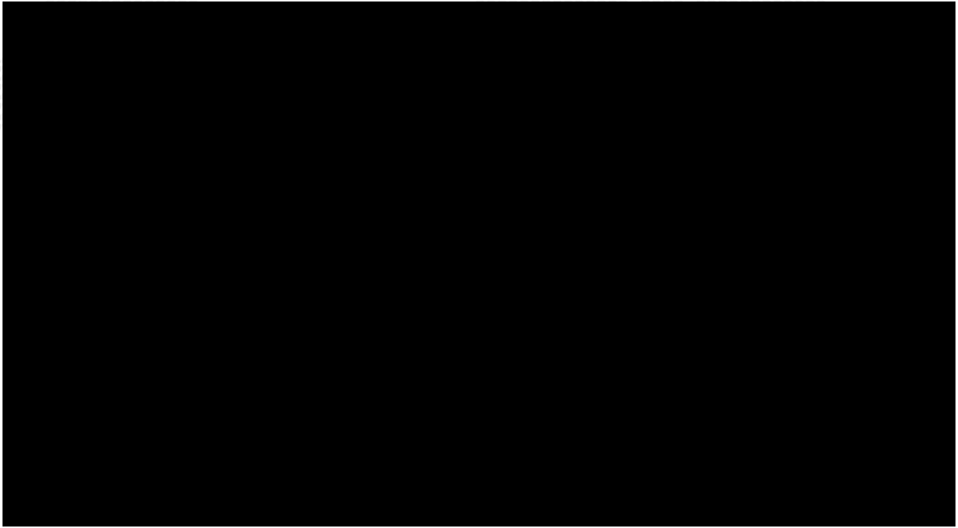
A first example

```
[ Command >
cat <<EOF > cp1code.asm
i2r r0 i0
r2o r0 o0
j 0
EOF
[ Command >
procbuilder -register-size 8 -registers 2 -inputs 1 -outputs 1 -opcodes j,i2r,r2o -save-machine cp1.
json -input-assembly cp1code.asm
[ Command >
bondmachine -bondmachine-file bondmachine.json
[ Command >
bondmachine -bondmachine-file bondmachine.json -add-domains cp0.json
bondmachine -bondmachine-file bondmachine.json -add-processor 0
bondmachine -bondmachine-file bondmachine.json -add-domains cp1.json
bondmachine -bondmachine-file bondmachine.json -add-processor 1
[ Output  >
Processor 0 successfully added
Processor 1 successfully added
[ Command >
bondmachine -bondmachine-file bondmachine.json -add-outputs 1
[ Command >
bondmachine -bondmachine-file bondmachine.json -add-bond p0o0,p1i0
bondmachine -bondmachine-file bondmachine.json -add-bond p1o0,o0
[ Command >
bondmachine -bondmachine-file bondmachine.json -add-shared-objects "lfsr8:34"
bondmachine -bondmachine-file bondmachine.json -connect-processor-shared-object 0,0
```

A first example



A first example



A first example

```
[ Command > bmhelper validate
[ Output  >
 [ ERROR ]       No workflows could be identified based on the analyzed files
```

A first example

```
[ Command > bmhelper validate
[ Output  >
 [ ERROR ]       No workflows could be identified based on the analyzed files
[ Command >
cat <<EOF > local.mk
WORKING_DIR=working_dir
CURRENT_DIR=$(shell pwd)
SOURCE_JSON=bondmachine.json
BOARD=icefun
MAPFILE=icefun_maps.json
BOARD_SLOW=1
BOARD_SLOW_FACTOR=19
IF_LEDS=1
IF_LEDS_MAP=o0
EOF
```

A first example

```
BOARD=icefun
MAPFILE=icefun_maps.json
BOARD_SLOW=1
BOARD_SLOW_FACTOR=19
IF_LEDS=1
IF_LEDS_MAP=o0
EOF
[ Command >
cat <<EOF > icefun.pcf
set_io --warn-no-port led0      C10
set_io --warn-no-port led1      A10
set_io --warn-no-port led2      D7
set_io --warn-no-port led3      D6
set_io --warn-no-port led4      A7
set_io --warn-no-port led5      C7
set_io --warn-no-port led6      A4
set_io --warn-no-port led7      C4
set_io --warn-no-port lcol1     A12
set_io --warn-no-port lcol2     D10
set_io --warn-no-port lcol3     A6
set_io --warn-no-port lcol4     C5
set_io --warn-no-port spkp      M12
set_io --warn-no-port spkm      M6

set_io --warn-no-port btn       A5
set_io --warn-no-port clk       P7
EOF
```

A first example

```
set_io --warn-no-port led1      A10
set_io --warn-no-port led2      D7
set_io --warn-no-port led3      D6
set_io --warn-no-port led4      A7
set_io --warn-no-port led5      C7
set_io --warn-no-port led6      A4
set_io --warn-no-port led7      C4
set_io --warn-no-port lcol1     A12
set_io --warn-no-port lcol2     D10
set_io --warn-no-port lcol3     A6
set_io --warn-no-port lcol4     C5
set_io --warn-no-port spkp      M12
set_io --warn-no-port spkm      M6

set_io --warn-no-port btn       A5
set_io --warn-no-port clk       P7
EOF
[ Command >
cat <<EOF > icefun_maps.json
{
"Assoc" : {
        "logic": "negative",
        "clk" : "clk",
        "reset" : "btn"
        }
}
EOF
```

A first example

```
set_io --warn-no-port led6      A4
set_io --warn-no-port led7      C4
set_io --warn-no-port lcol1     A12
set_io --warn-no-port lcol2     D10
set_io --warn-no-port lcol3     A6
set_io --warn-no-port lcol4     C5
set_io --warn-no-port spkp      M12
set_io --warn-no-port spkm      M6

set_io --warn-no-port btn       A5
set_io --warn-no-port clk       P7
EOF
[ Command >
cat <<EOF > icefun_maps.json
{
"Assoc" : {
        "logic": "negative",
        "clk" : "clk",
        "reset" : "btn"
        }
}
EOF
[ Command >
bmhelper validate
[ Output  >
 [ OK ]         Workflow detected: json.
 [ OK ]         Project has been successfully validate.
```

A first example

```
set_io --warn-no-port lcol4      C5
set_io --warn-no-port spkp       M12
set_io --warn-no-port spkm       M6

set_io --warn-no-port btn        A5
set_io --warn-no-port clk        P7
EOF
[ Command >
cat <<EOF > icefun_maps.json
{
"Assoc" : {
        "logic": "negative",
        "clk" : "clk",
        "reset" : "btn"
        }
}
EOF
[ Command >
bmhelper validate
[ Output  >
 [ OK ]          Workflow detected: json.
 [ OK ]          Project has been successfully validate.
[ Command >
bmhelper apply
[ Output  >
 [ OK ]          Workflow detected: json.
 [ OK ]          Project has been successfully initialized.
```

A first example

```
        "logic": "negative",
        "clk" : "clk",
        "reset" : "btn"
        }
}
EOF
[ Command >
bmhelper validate
[ Output  >
 [ OK ]          Workflow detected: json.
 [ OK ]          Project has been successfully validate.
[ Command >
bmhelper apply
[ Output  >
 [ OK ]          Workflow detected: json.
 [ OK ]          Project has been successfully initialized.
[ Command >
make bondmachine
[ Output  >
[Project: Example] - [Working directory creation begin] - [Target: working_dir]
mkdir -p working_dir
[Project: Example] - [Working directory creation end]

[Project: Example] - [BondMachine generation begin] - [Target: working_dir/bondmachine_target]
cp bondmachine.json working_dir/bondmachine.json
[Project: Example] - [BondMachine generation end]
```

A first example

```
[ Command >
make bondmachine
[ Output  >
[Project: Example] - [Working directory creation begin] - [Target: working_dir]
mkdir -p working_dir
[Project: Example] - [Working directory creation end]

[Project: Example] - [BondMachine generation begin] - [Target: working_dir/bondmachine_target]
cp bondmachine.json working_dir/bondmachine.json
[Project: Example] - [BondMachine generation end]

[ Command >
make hdl
[ Output  >
[Project: Example] - [HDL generation begin] - [Target: working_dir/hdl_target]
bondmachine -bondmachine-file working_dir/bondmachine.json -create-verilog -verilog-mapfile icefun_m
aps.json -verilog-flavor icefun   -board-slow -board-slow-factor 19   -icefun-leds -icefun-leds-map
o0
echo > working_dir/bondmachine.sv
for i in `ls *.v | sort -d` ; do cat $i >> working_dir/bondmachine.sv ; done
rm -f *.v
echo > working_dir/bondmachine.vhd
for i in `ls *.vhd | sort -d` ; do cat $i >> working_dir/bondmachine.vhd ; done
ls: cannot access '*.vhd': No such file or directory
rm -f *.vhd
[Project: Example] - [HDL generation end]
```

A first example

```
make hdl
[ Output  >
[Project: Example] - [HDL generation begin] - [Target: working_dir/hdl_target]
bondmachine -bondmachine-file working_dir/bondmachine.json -create-verilog -verilog-mapfile icefun_m
aps.json -verilog-flavor icefun   -board-slow -board-slow-factor 19   -icefun-leds -icefun-leds-map
o0
echo > working_dir/bondmachine.sv
for i in `ls *.v | sort -d` ; do cat $i >> working_dir/bondmachine.sv ; done
rm -f *.v
echo > working_dir/bondmachine.vhd
for i in `ls *.vhd | sort -d` ; do cat $i >> working_dir/bondmachine.vhd ; done
ls: cannot access '*.vhd': No such file or directory
rm -f *.vhd
[Project: Example] - [HDL generation end]

[ Command >
make project
[ Output  >
[Project: Example] - [Icestorm toolchain - copy constraints begin] - [Target: working_dir/icefun.pcf
]
cp icefun.pcf working_dir
[Project: Example] - [Icestorm toolchain - copy constraints end]

[Project: Example] - [Icestorm toolchain - project creation begin] - [Target: working_dir/icestorm_c
reation]
[Project: Example] - [Icestorm toolchain - project creation end]
```

A first example

```
[Project: Example] - [HDL generation begin] - [Target: working_dir/hdl_target]
bondmachine -bondmachine-file working_dir/bondmachine.json -create-verilog -verilog-mapfile icefun_m
aps.json -verilog-flavor icefun   -board-slow -board-slow-factor 19   -icefun-leds -icefun-leds-map
o0
echo > working_dir/bondmachine.sv
for i in `ls *.v | sort -d` ; do cat $i >> working_dir/bondmachine.sv ; done
rm -f *.v
echo > working_dir/bondmachine.vhd
for i in `ls *.vhd | sort -d` ; do cat $i >> working_dir/bondmachine.vhd ; done
ls: cannot access '*.vhd': No such file or directory
rm -f *.vhd
[Project: Example] - [HDL generation end]

[ Command >
make project
[ Output  >
[Project: Example] - [Icestorm toolchain - copy constraints begin] - [Target: working_dir/icefun.pcf
]
cp icefun.pcf working_dir
[Project: Example] - [Icestorm toolchain - copy constraints end]

[Project: Example] - [Icestorm toolchain - project creation begin] - [Target: working_dir/icestorm_c
reation]
[Project: Example] - [Icestorm toolchain - project creation end]

[ Command >
make synthesis
```

A first example

```
   Number of port bits:             14
   Number of memories:               0
   Number of memory bits:            0
   Number of processes:              0
   Number of cells:                219
     $print                         30
     $scopeinfo                     10
     SB_CARRY                       30
     SB_DFF                         20
     SB_DFFE                        16
     SB_DFFER                       24
     SB_DFFR                        16
     SB_DFFSR                        8
     SB_LUT4                        65

2.50. Executing CHECK pass (checking for obvious problems).
Checking module bondmachine_main...
Found and reported 0 problems.

2.51. Executing JSON backend.

Warnings: 3 unique messages, 3 total
End of script. Logfile hash: 6e1abd25a9, CPU: user 0.81s system 0.04s, MEM: 24.88 MB peak
Yosys 0.41+108 (git sha1 557968567, clang++ 14.0.0-1ubuntu1.1 -fPIC -Os)
Time spent: 47% 21x read_verilog (0 sec), 10% 29x opt_expr (0 sec), ...
[Project: Example] - [Icestorm toolchain - synthesis end]
```

A first example

```
   Number of memory bits:            0
   Number of processes:              0
   Number of cells:                219
     $print                         30
     $scopeinfo                     10
     SB_CARRY                       30
     SB_DFF                         20
     SB_DFFE                        16
     SB_DFFER                       24
     SB_DFFR                        16
     SB_DFFSR                        8
     SB_LUT4                        65

2.50. Executing CHECK pass (checking for obvious problems).
Checking module bondmachine_main...
Found and reported 0 problems.

2.51. Executing JSON backend.

Warnings: 3 unique messages, 3 total
End of script. Logfile hash: 6e1abd25a9, CPU: user 0.81s system 0.04s, MEM: 24.88 MB peak
Yosys 0.41+108 (git sha1 557968567, clang++ 14.0.0-1ubuntu1.1 -fPIC -Os)
Time spent: 47% 21x read_verilog (0 sec), 10% 29x opt_expr (0 sec), ...
[Project: Example] - [Icestorm toolchain - synthesis end]

[ Command >
make implementation
```

A first example

```
Info:  legend: * represents 1 endpoint(s)
Info:          + represents [1,1) endpoint(s)
Info: [ 78692,  78851) |***
Info: [ 78851,  79010) |*******
Info: [ 79010,  79169) |*****
Info: [ 79169,  79328) |*
Info: [ 79328,  79487) |***
Info: [ 79487,  79646) |***
Info: [ 79646,  79805) |*
Info: [ 79805,  79964) |**************************
Info: [ 79964,  80123) |**
Info: [ 80123,  80282) |***********
Info: [ 80282,  80441) |*****
Info: [ 80441,  80600) |****
Info: [ 80600,  80759) |****
Info: [ 80759,  80918) |*********************
Info: [ 80918,  81077) |******
Info: [ 81077,  81236) |*********
Info: [ 81236,  81395) |********
Info: [ 81395,  81554) |**
Info: [ 81554,  81713) |
Info: [ 81713,  81872) |*****************************************
2 warnings, 0 errors

Info: Program finished normally.
[Project: Example] - [Icestorm toolchain - implementation end]
```

A first example

```
Info: [ 78692,  78851) |***
Info: [ 78851,  79010) |*******
Info: [ 79010,  79169) |*****
Info: [ 79169,  79328) |*
Info: [ 79328,  79487) |***
Info: [ 79487,  79646) |***
Info: [ 79646,  79805) |*
Info: [ 79805,  79964) |**************************
Info: [ 79964,  80123) |**
Info: [ 80123,  80282) |***********
Info: [ 80282,  80441) |*****
Info: [ 80441,  80600) |****
Info: [ 80600,  80759) |****
Info: [ 80759,  80918) |*********************
Info: [ 80918,  81077) |******
Info: [ 81077,  81236) |*********
Info: [ 81236,  81395) |********
Info: [ 81395,  81554) |**
Info: [ 81554,  81713) |
Info: [ 81713,  81872) |*****************************************
2 warnings, 0 errors

Info: Program finished normally.
[Project: Example] - [Icestorm toolchain - implementation end]

[ Command >
make bitstream
```

A first example

```
Info: [ 79646,  79805) |*
Info: [ 79805,  79964) |**************************
Info: [ 79964,  80123) |**
Info: [ 80123,  80282) |***********
Info: [ 80282,  80441) |*****
Info: [ 80441,  80600) |****
Info: [ 80600,  80759) |****
Info: [ 80759,  80918) |*********************
Info: [ 80918,  81077) |******
Info: [ 81077,  81236) |*********
Info: [ 81236,  81395) |********
Info: [ 81395,  81554) |**
Info: [ 81554,  81713) |
Info: [ 81713,  81872) |*****************************************
2 warnings, 0 errors

Info: Program finished normally.
[Project: Example] - [Icestorm toolchain - implementation end]

[ Command >
make bitstream
[ Output  >
[Project: Example] - [Icestorm toolchain - write bitstream begin] - [Target: working_dir/icestorm_bi
tstream]
icepack working_dir/bondmachine.asc working_dir/bondmachine.bin
[Project: Example] - [Icestorm toolchain - write bitstream end]
```

A first example

```
Info: [ 79964,  80123) |**
Info: [ 80123,  80282) |***********
Info: [ 80282,  80441) |*****
Info: [ 80441,  80600) |****
Info: [ 80600,  80759) |****
Info: [ 80759,  80918) |*********************
Info: [ 80918,  81077) |******
Info: [ 81077,  81236) |*********
Info: [ 81236,  81395) |********
Info: [ 81395,  81554) |**
Info: [ 81554,  81713) |
Info: [ 81713,  81872) |******************************************
2 warnings, 0 errors

Info: Program finished normally.
[Project: Example] - [Icestorm toolchain - implementation end]

[ Command >
make bitstream
[ Output  >
[Project: Example] - [Icestorm toolchain - write bitstream begin] - [Target: working_dir/icestorm_bi
tstream]
icepack working_dir/bondmachine.asc working_dir/bondmachine.bin
[Project: Example] - [Icestorm toolchain - write bitstream end]

[ Command >
make program
```

A first example

```
[Project: Example] - [Icestorm toolchain - write bitstream end]

[ Command >
make program
[ Output  >
[Project: Example] - [Icestorm toolchain - programming begin] - [Target: working_dir/icestorm_progra
m]
icefunprog /dev/ttyACM0 working_dir/bondmachine.bin
Flash ID 0x1f 0x85 0x1
Program length 135100
Erase pages 3
Erasing sector 0x00000
Erase sector response 0xb0
Erasing sector 0x10000
Erase sector response 0xb0
Erasing sector 0x20000
Erase sector response 0xb0
####################################################################################################
####################################################################################################
####################################################################################################
####################################################################################################
####################################################################################################
############################
Flash ok
Release response 0x0
[Project: Example] - [Icestorm toolchain - programming end]
```

Molding the BondMachine

Main tools

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

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

- *bondgo*: A new type of compiler that create not only the CPs assembly but also the architecture itself.
- *basm*: The BondMachine Assembler.
- A set of tools to use BondMachine in Machine Learning.
- *bmqsim*: A quantum computer simulator.

Molding the BondMachine

Main tools

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

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

- *bondgo*: A new type of compiler that create not only the CPs assembly but also the architecture itself.
- *basm*: The BondMachine Assembler.
- A set of tools to use BondMachine in Machine Learning.
- *bmqsim*: A quantum computer simulator.

Molding the BondMachine

Main tools

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

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

- *bondgo*: A new type of compiler that create not only the CPs assembly but also the architecture itself.
- *basm*: The BondMachine Assembler.
- A set of tools to use BondMachine in Machine Learning.
- *bmqsim*: A quantum computer simulator.

Molding the BondMachine

Main tools

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

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

- *bondgo*: A new type of compiler that create not only the CPs assembly but also the architecture itself.
- *basm*: The BondMachine Assembler.
- A set of tools to use BondMachine in Machine Learning.
- *bmqsim*: A quantum computer simulator.

Molding the BondMachine

Main tools

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

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

- *bondgo*: A new type of compiler that create not only the CPs assembly but also the architecture itself.
- *basm*: The BondMachine Assembler.
- A set of tools to use BondMachine in Machine Learning.
- *bmqsim*: A quantum computer simulator.

Molding the BondMachine

Main tools

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

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

- *bondgo*: A new type of compiler that create not only the CPs assembly but also the architecture itself.
- *basm*: The BondMachine Assembler.
- A set of tools to use BondMachine in Machine Learning.
- *bmqsim*: A quantum computer simulator.

Bondgo

The major innovation of the BondMachine Project is its compiler.

Bondgo is the name chosen for the compiler developed for the BondMachine.

The compiler source language is Go as the name suggest.

Bondgo

This is the standard flow when building computer programs

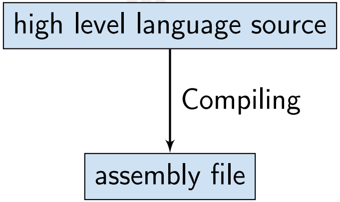
Bondgo

This is the standard flow when building computer programs

high level language source

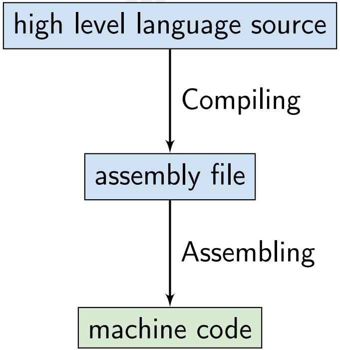
Bondgo

This is the standard flow when building computer programs



Bondgo

This is the standard flow when building computer programs



Bondgo

Bondgo does something different from standard compilers ...

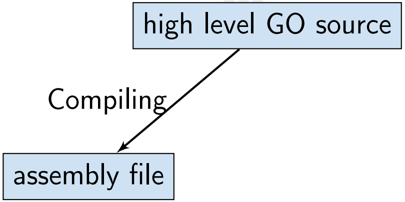
Bondgo

Bondgo does something different from standard compilers ...

high level GO source

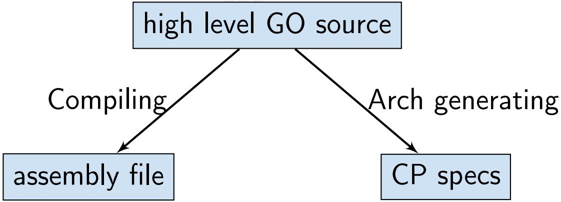
Bondgo

Bondgo does something different from standard compilers ...



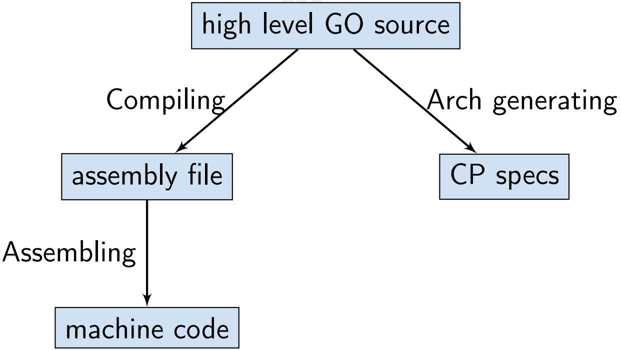
Bondgo

Bondgo does something different from standard compilers ...



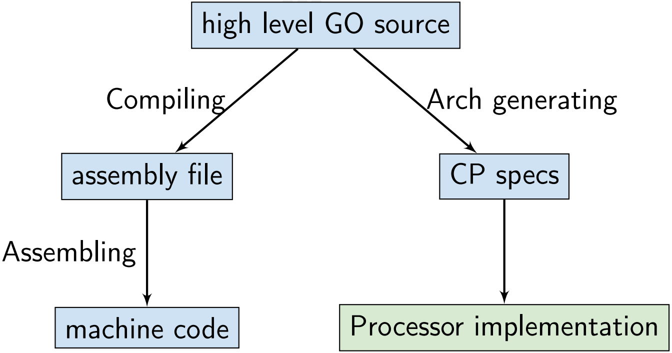
Bondgo

Bondgo does something different from standard compilers ...



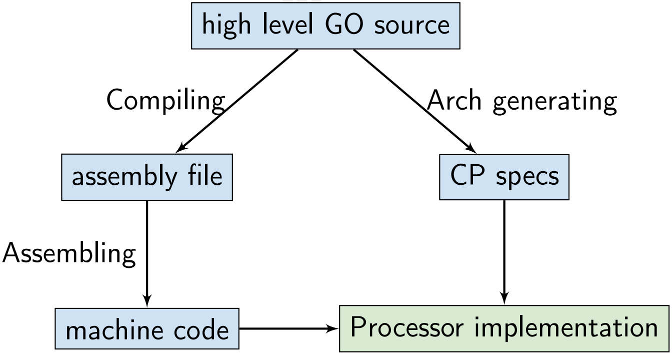
Bondgo

Bondgo does something different from standard compilers ...



Bondgo

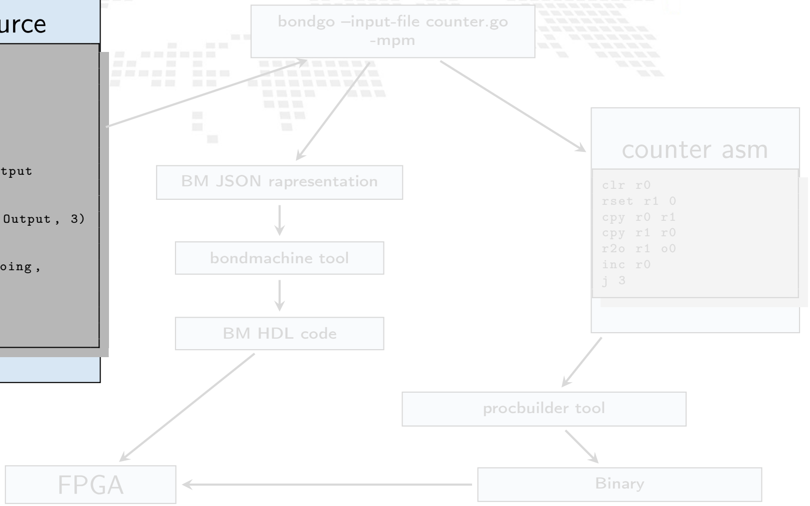
Bondgo does something different from standard compilers ...



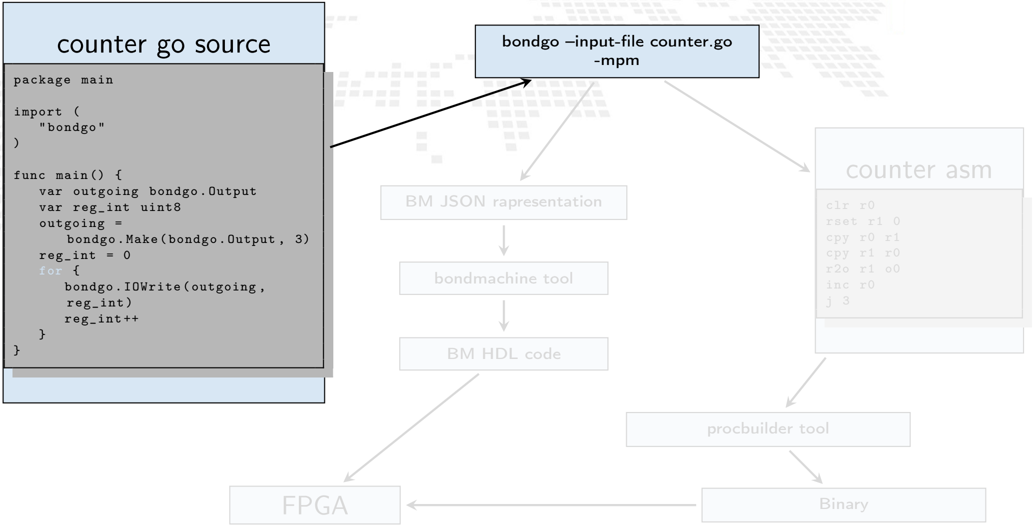
Bondgo workflow example

counter go source

```go
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 workflow example



Bondgo workflow example

counter go source

```go
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 –input-file counter.go -mpm

counter asm

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

BM JSON rapresentation

bondmachine tool

BM HDL code

procbuilder tool

FPGA

Binary

Bondgo workflow example

counter go source

```go
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 –input-file counter.go -mpm

BM JSON rapresentation

bondmachine tool

BM HDL code

counter asm

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

procbuilder tool

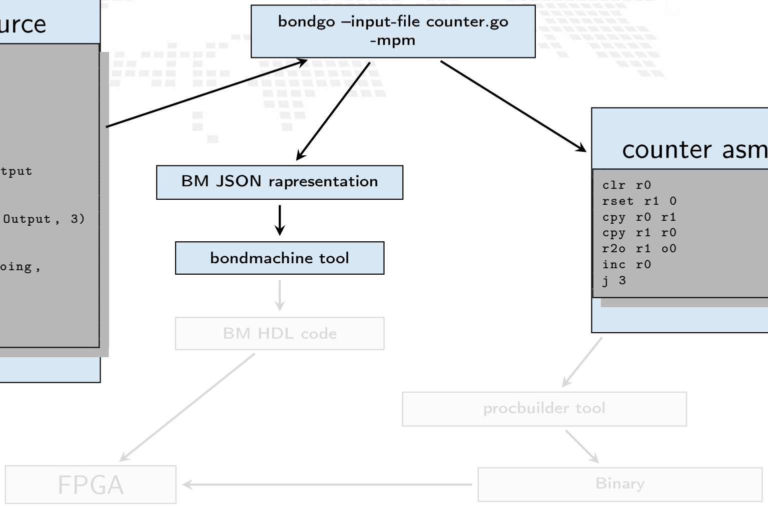
Binary

FPGA

Bondgo workflow example

counter go source

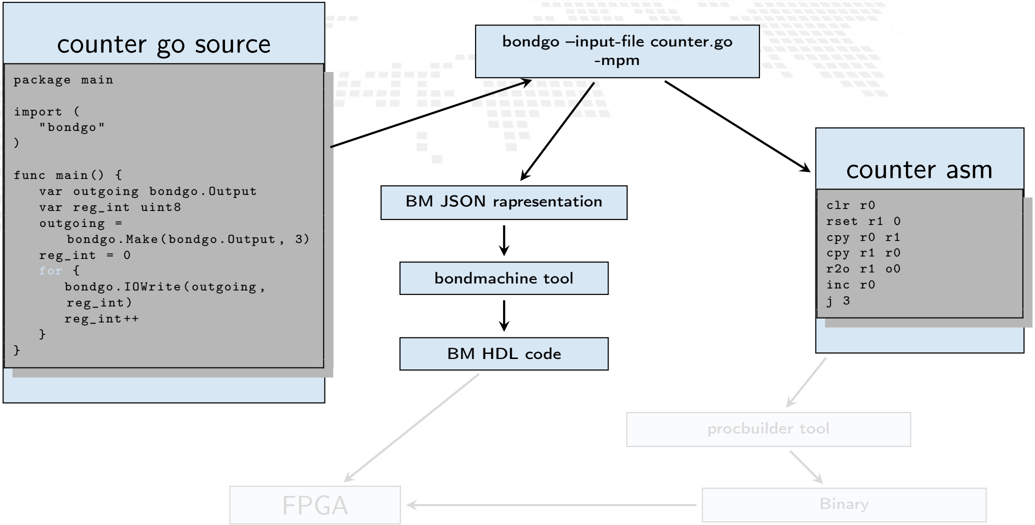
```go
package main

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++
   }
}
```



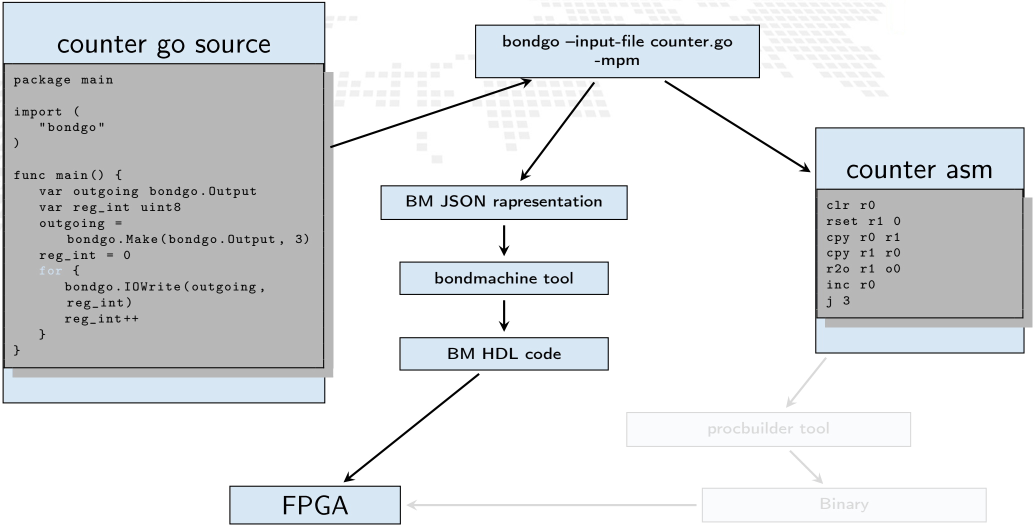
counter asm

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

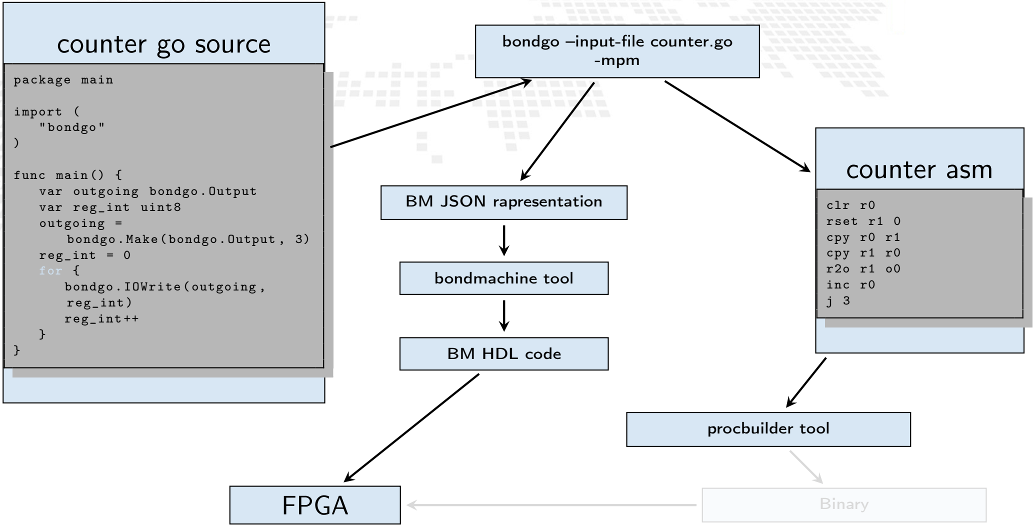
Bondgo workflow example



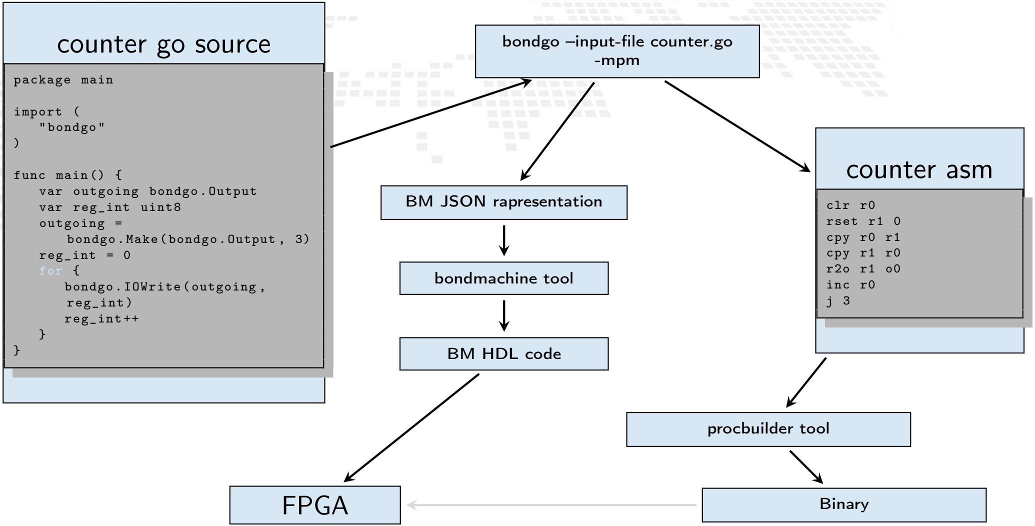
Bondgo workflow example



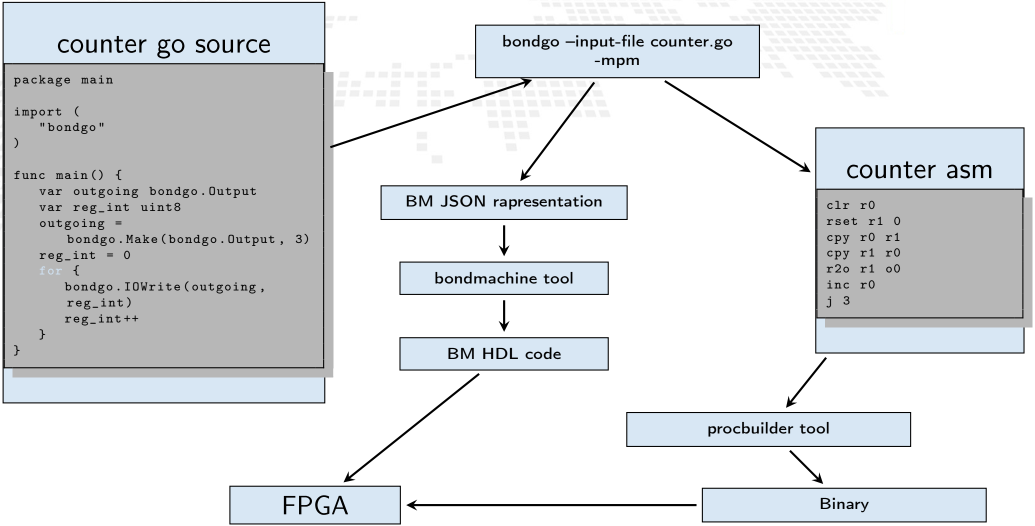
Bondgo workflow example



Bondgo workflow example



Bondgo workflow example



Bondgo

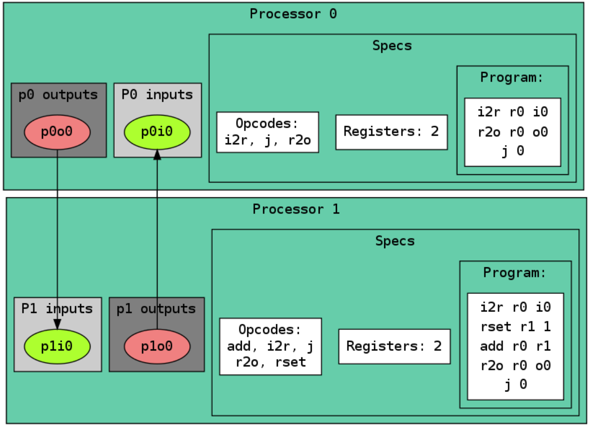
A multi-core example

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 Architectures

One of the most important result

The architecture creation is a part of the compilation process.

Basm

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

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

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

Basm

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

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

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

Basm

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

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

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

Basm

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

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

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

Basm

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

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

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

Basm

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

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

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

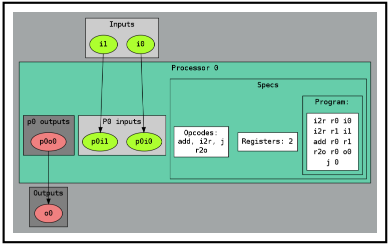
Basm

Abstract Assembly

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

Given a specific piece of assembly code Basm has the ability to compute the "minimum CP" that can execute that code.

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



These are Building Blocks for complex BondMachines.

Machine Learning

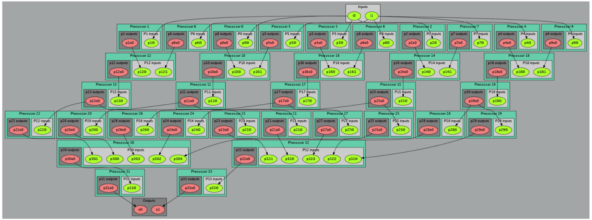
Machine Learning with BondMachine

Native Neural Network library

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

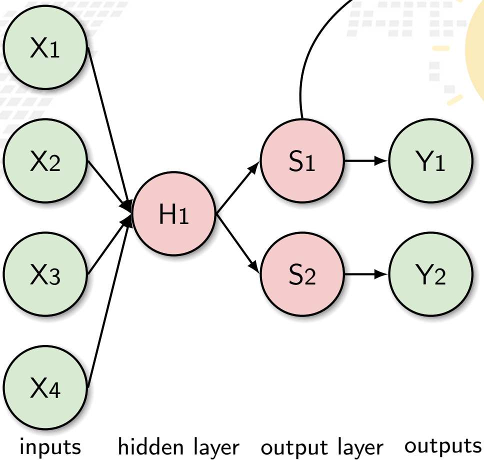
- Neurons are converted to BondMachine connecting processors.
- Tensors are mapped to CP connections.

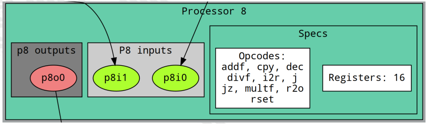
```go
layers := []int{2, 5, 2}
weights := make([]neuralbond.Weight, 0)

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



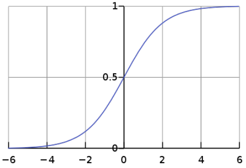
BM inference: A first tentative idea

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





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

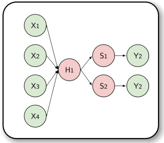



```
%section softmax .romtext iomode:sync
        entry _start    ; Entry point
_start:
 mov r8, 0f0.0
{{range $y := intRange "0" .Params.inputs}}
{{printf "i2r r1,i%d\n" $y}}
        mov     r0, 0f1.0
        mov     r2, 0f1.0
        mov     r3, 0f1.0
        mov     r4, 0f1.0
        mov     r5, 0f1.0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

        addf    r0, r6

        dec     r7
        jz      r7,exit{{printf "%d" $y}}
        j       loop{{printf "%d" $y}}
exit{{printf "%d" $y}}:
{{$z := atoi $.Params.pos}}
{{if eq $y $z}}
 mov r9, r0
%endsection
```

From idea to implementation

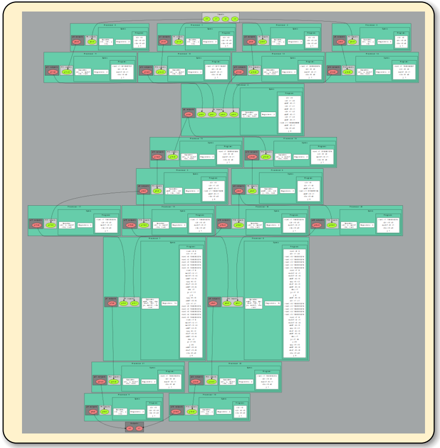
Starting from High Level Code, a NN model trained with **TensorFlow** and exported in a standard interpreted by **neuralbond** that converts nodes and weights of the network into a set of heterogeneous processors.

High Level Code




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

Firmware

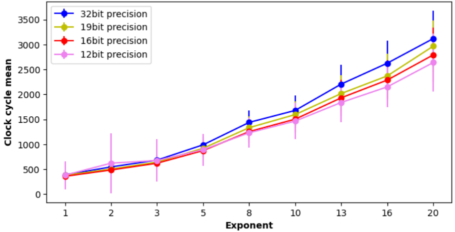


Analysis and evaluation

A key aspect of the BondMachine inference engine is **the high degree of configurability**: choose the desired trade-off between inference speed, accuracy and resource usage.

- **numerical precision**: floating, fixed point or custom operator to change arbitrary the bit-width.
- **hw/sw function swap**: choose the best trade-off between hardware and software.
- **architecture optimization** collapse, prune or even customize CPs based on the requirements.

Bit	Luts	Regs
32	14306	9264
19	7202	5717
16	7738	5487
12	4133	5094



Bit	Static-Power (W)	Dynamic-Power (W)	Time / Inf (s)	En. / Inf (J)
32	0.037	0.055	6.84E-06	3.78E-07
19	0.013	0.022	6.44E-06	1.39E-07
16	0.017	0.024	6.21E-06	1.49E-07
12	0.020	0.012	6.76E-06	8.11E-08

Many studies have been conducted to evaluate the performance of the BM inference engine regarding power consumption, latency and resources usage.

Quantum Computing

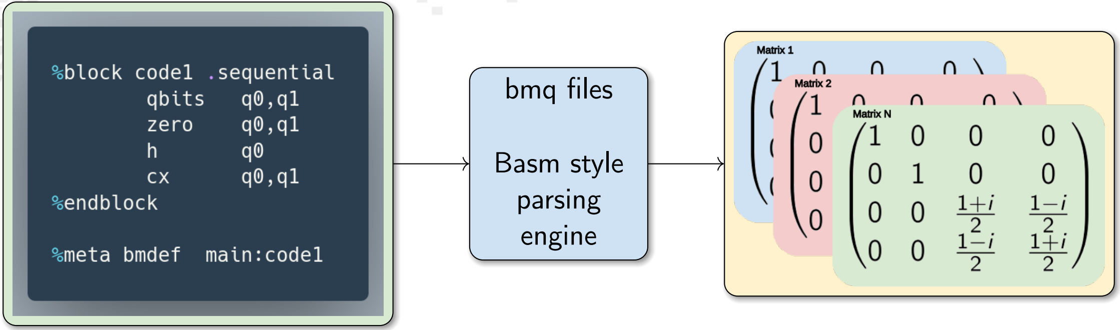
Quantum Computing

With all the capabilities of the BondMachine in terms of parallelism and speed, of customizability of the instruction set and the numerical precision, it is a natural question to ask whether the BondMachine could be used to simulate quantum computers.

A quantum computer simulator called bmqsim has been developed and is available within the BondMachine project.

Quantum Circuit

The first ingredient for bmqsim is a quantum circuit. The quantum circuit is a sequence of quantum gates represented by a sequence of matrices. the “program” is a .bmq file that contains code similar to the Qasm code.

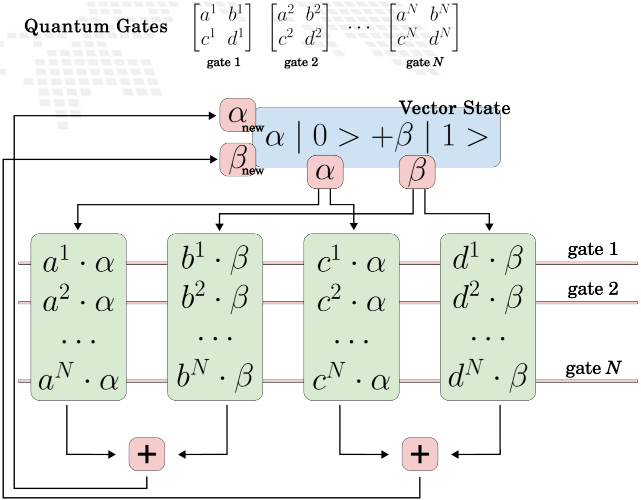


Independently of the backend, bmqsim translates the .bmq file into N matrices.

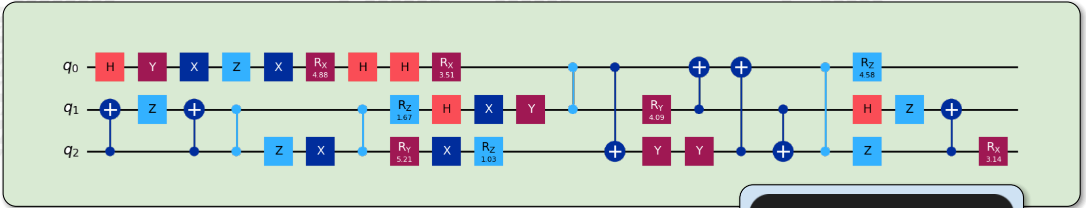
Backend: Hardcoded matrices sequence

This backend creates a hardware that for each state of the quantum register, it applies the sequence of matrices.

For each matrix operation a dedicated processor is used. Within the processor, the matrix elements of all the gates are hardcoded.



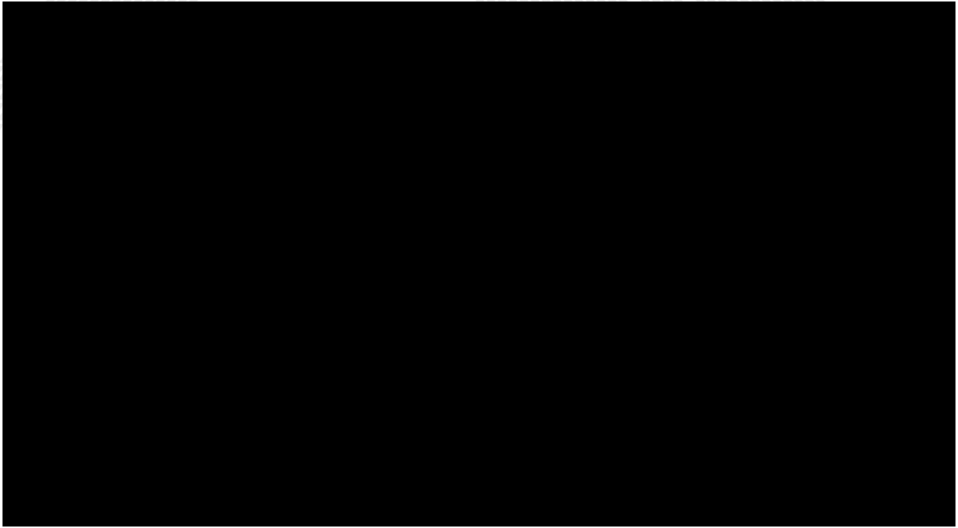
Validation




```
Overall: Passed

Detailed results:
        pennylane: Passed
        qiskitrot: Passed
        quest: Passed
        bmqsim_hw: Passed
        bmqsim: Passed
        bmqsim_alveo: Passed
```

The validation is done by comparing the results of the simulation with the results of the same quantum circuit simulated by a well-known quantum simulator. Randomizing both the quantum circuit and the input state.

FPGA Quantum Teleportation



FPGA Quantum Teleportation

```
[ Command > bmhelper create --project_name Example
```

FPGA Quantum Teleportation

```
[ Command > bmhelper create --project_name Example
[ Command > cd Example
```

FPGA Quantum Teleportation

```
[ Command > bmhelper create --project_name Example
[ Command > cd Example
[ Command >
source /tools/Xilinx/Vitis/2023.2/settings64.sh
[ Output  >
```

FPGA Quantum Teleportation

```
[ Command > bmhelper create --project_name Example
[ Command > cd Example
[ Command >
source /tools/Xilinx/Vitis/2023.2/settings64.sh
[ Output  >
[ Command > cat <<EOF > program.bmq
%block code1 .sequential
        qbits   s,a,b
        zero    s,a,b
        h       a
        cx      a,b
        cx      s,a
        h       s
        cx      a,b
        cz      s,b
%endblock

%meta bmdef global main:code1
EOF
```

FPGA Quantum Teleportation

```
%meta bmdef global main:code1
EOF
[ Command >
cat <<EOF > local.mk
WORKING_DIR=working_dir
CURRENT_DIR=$(shell pwd)
SOURCE_QUANTUM=program.bmq
QUANTUM_APP=working_dir/circuit.c
QUANTUM_ARGS=-build-matrix-seq-hardcoded -hw-flavor seq_hardcoded_complex -app-flavor cpp_opencl_com
plex -build-app -app-file $(QUANTUM_APP) -emit-bmapi-maps -bmapi-maps-file bmapi.json
BOARD=alveou55c
SHOWARGS=-dot-detail 5
SHOWRENDERER=dot -Txlib
VERILOG_OPTIONS=-comment-verilog -bcof-file $(WORKING_DIR)/bondmachine.bcof
BMREQS=$(WORKING_DIR)/requirements.json
HWOPTIMIZATIONS=onlydestregs,onlysrcregs
BASM_ARGS=-disable-dynamical-matching -bo $(WORKING_DIR)/bondmachine.bcof -chooser-min-word-size -ch
ooser-force-same-name -dump-requirements $(WORKING_DIR)/requirements.json
HDL_REGRESSION=bondmachine.sv
BM_REGRESSION=bondmachine.json
PLATFORM=xilinx_u55c_gen3x16_xdma_3_202210_1
MAPFILE=alveou55c_maps.json
include bmapi.mk
include deploy.mk
EOF
```

FPGA Quantum Teleportation

```
SHOWARGS=-dot-detail 5
SHOWRENDERER=dot -Txlib
VERILOG_OPTIONS=-comment-verilog -bcof-file $(WORKING_DIR)/bondmachine.bcof
BMREQS=$(WORKING_DIR)/requirements.json
HWOPTIMIZATIONS=onlydestregs,onlysrcregs
BASM_ARGS=-disable-dynamical-matching -bo $(WORKING_DIR)/bondmachine.bcof -chooser-min-word-size -ch
ooser-force-same-name -dump-requirements $(WORKING_DIR)/requirements.json
HDL_REGRESSION=bondmachine.sv
BM_REGRESSION=bondmachine.json
PLATFORM=xilinx_u55c_gen3x16_xdma_3_202210_1
MAPFILE=alveou55c_maps.json
include bmapi.mk
include deploy.mk
EOF
[ Command >
cat <<EOF > bmapi.mk
USE_BMAPI=yes
BMAPI_LANGUAGE=python
BMAPI_FLAVOR=axist
BMAPI_FLAVOR_VERSION=basic
BMAPI_MAPFILE=bmapi.json
BMAPI_LIBOUTDIR=working_dir/bmapi
BMAPI_MODOUTDIR=working_dir/rtl_bondmachine
BMAPI_FRAMEWORK=pynq
BMAPI_GENERATE_EXAMPLE=notebook.ipynb
EOF
```

FPGA Quantum Teleportation

```
PLATFORM=xilinx_u55c_gen3x16_xdma_3_202210_1
MAPFILE=alveou55c_maps.json
include bmapi.mk
include deploy.mk
EOF
[ Command >
cat <<EOF > bmapi.mk
USE_BMAPI=yes
BMAPI_LANGUAGE=python
BMAPI_FLAVOR=axist
BMAPI_FLAVOR_VERSION=basic
BMAPI_MAPFILE=bmapi.json
BMAPI_LIBOUTDIR=working_dir/bmapi
BMAPI_MODOUTDIR=working_dir/rtl_bondmachine
BMAPI_FRAMEWORK=pynq
BMAPI_GENERATE_EXAMPLE=notebook.ipynb
EOF
[ Command >
cat <<EOF > deploy.mk
DEPLOY_TYPE=local
DEPLOY_PATH=/home/mirko/alveoruns/$(PROJECT_NAME)
DEPLOY_CLONE=/home/mirko/alveoruns/template
DEPLOY_APP=working_dir/circuit.c
DEPLOY_OVERRIDE=true
DEPLOY_BITTYPE=xclbin
EOF
```

FPGA Quantum Teleportation

```
BMAPI_MAPFILE=bmapi.json
BMAPI_LIBOUTDIR=working_dir/bmapi
BMAPI_MODOUTDIR=working_dir/rtl_bondmachine
BMAPI_FRAMEWORK=pynq
BMAPI_GENERATE_EXAMPLE=notebook.ipynb
EOF
[ Command >
cat <<EOF > deploy.mk
DEPLOY_TYPE=local
DEPLOY_PATH=/home/mirko/alveoruns/$(PROJECT_NAME)
DEPLOY_CLONE=/home/mirko/alveoruns/template
DEPLOY_APP=working_dir/circuit.c
DEPLOY_OVERRIDE=true
DEPLOY_BITTYPE=xclbin
EOF
[ Command >
bmhelper validate
[ Output  >
 [ OK ]         Workflow detected: quantum.
 [ OK ]         Mandatory variable found SOURCE_QUANTUM
 [ OK ]         Mandatory variable found WORKING_DIR
 [ OK ]         Mandatory variable found MAPFILE
 [ OK ]         Optional variable found: SHOWARGS
 [ OK ]         Source file program.bmq found
 [ OK ]         Found target board: alveou55c
 [ OK ]         Project has been successfully validate.
```

FPGA Quantum Teleportation

```
DEPLOY_APP=working_dir/circuit.c
DEPLOY_OVERRIDE=true
DEPLOY_BITTYPE=xclbin
EOF
[ Command >
bmhelper validate
[ Output  >
 [ OK ]         Workflow detected: quantum.
 [ OK ]         Mandatory variable found SOURCE_QUANTUM
 [ OK ]         Mandatory variable found WORKING_DIR
 [ OK ]         Mandatory variable found MAPFILE
 [ OK ]         Optional variable found: SHOWARGS
 [ OK ]         Source file program.bmq found
 [ OK ]         Found target board: alveou55c
 [ OK ]         Project has been successfully validate.
[ Command >
bmhelper apply
[ Output  >
 [ OK ]         Workflow detected: quantum.
 [ OK ]         Mandatory variable found SOURCE_QUANTUM
 [ OK ]         Mandatory variable found WORKING_DIR
 [ OK ]         Mandatory variable found MAPFILE
 [ OK ]         Optional variable found: SHOWARGS
 [ OK ]         Source file program.bmq found
 [ OK ]         Found target board: alveou55c
 [ OK ]         Project has been successfully initialized.
```

FPGA Quantum Teleportation

```
[ Command >
bmhelper apply
[ Output  >
 [ OK ]          Workflow detected: quantum.
 [ OK ]          Mandatory variable found SOURCE_QUANTUM
 [ OK ]          Mandatory variable found WORKING_DIR
 [ OK ]          Mandatory variable found MAPFILE
 [ OK ]          Optional variable found: SHOWARGS
 [ OK ]          Source file program.bmq found
 [ OK ]          Found target board: alveou55c
 [ OK ]          Project has been successfully initialized.
[ Command >
make bondmachine
[ Output  >
[Project: Example] - [Working directory creation begin] - [Target: working_dir]
mkdir -p working_dir
[Project: Example] - [Working directory creation end]

[Project: Example] - [BondMachine generation begin] - [Target: working_dir/bondmachine_target]
bmqsim  -save-basm working_dir/bondmachine.basm -build-matrix-seq-hardcoded -hw-flavor seq_hardcoded
_complex -app-flavor cpp_opencl_complex -build-app -app-file working_dir/circuit.c -emit-bmapi-maps
-bmapi-maps-file bmapi.json  program.bmq ; basm -disable-dynamical-matching -bo working_dir/bondmach
ine.bcof -chooser-min-word-size -chooser-force-same-name -dump-requirements working_dir/requirements
.json  -o working_dir/bondmachine.json working_dir/bondmachine.basm
[Project: Example] - [BondMachine generation end]
```

FPGA Quantum Teleportation

```
bmqsim  -save-basm working_dir/bondmachine.basm -build-matrix-seq-hardcoded -hw-flavor seq_hardcoded
_complex -app-flavor cpp_opencl_complex -build-app -app-file working_dir/circuit.c -emit-bmapi-maps
-bmapi-maps-file bmapi.json  program.bmq ; basm -disable-dynamical-matching -bo working_dir/bondmach
ine.bcof -chooser-min-word-size -chooser-force-same-name -dump-requirements working_dir/requirements
.json  -o working_dir/bondmachine.json working_dir/bondmachine.basm
[Project: Example] - [BondMachine generation end]

[ Command >
make hdl
[ Output  >
[Project: Example] - [HDL generation begin] - [Target: working_dir/hdl_target]
bondmachine -bondmachine-file working_dir/bondmachine.json -create-verilog -verilog-mapfile alveou55
c_maps.json -verilog-flavor alveou55c            -use-bmapi -bmapi-flavor axist -bmapi-language pyth
on -bmapi-mapfile bmapi.json -bmapi-liboutdir working_dir/bmapi -bmapi-framework pynq -bmapi-flavor-
version basic -bmapi-modoutdir working_dir/rtl_bondmachine -bmapi-generate-example notebook.ipynb  -
comment-verilog -bcof-file working_dir/bondmachine.bcof  -bmrequirements-file working_dir/requiremen
ts.json -hw-optimizations onlydestregs,onlysrcregs
echo > working_dir/bondmachine.sv
for i in `ls *.v | sort -d` ; do cat $i >> working_dir/bondmachine.sv ; done
rm -f *.v
echo > working_dir/bondmachine.vhd
for i in `ls *.vhd | sort -d` ; do cat $i >> working_dir/bondmachine.vhd ; done
ls: cannot access '*.vhd': No such file or directory
rm -f *.vhd
[Project: Example] - [HDL generation end]
```

FPGA Quantum Teleportation

```
.json  -o working_dir/bondmachine.json working_dir/bondmachine.basm
[Project: Example] - [BondMachine generation end]

[ Command >
make hdl
[ Output  >
[Project: Example] - [HDL generation begin] - [Target: working_dir/hdl_target]
bondmachine -bondmachine-file working_dir/bondmachine.json -create-verilog -verilog-mapfile alveou55
c_maps.json -verilog-flavor alveou55c           -use-bmapi -bmapi-flavor axist -bmapi-language pyth
on -bmapi-mapfile bmapi.json -bmapi-liboutdir working_dir/bmapi -bmapi-framework pynq -bmapi-flavor-
version basic -bmapi-modoutdir working_dir/rtl_bondmachine -bmapi-generate-example notebook.ipynb  -
comment-verilog -bcof-file working_dir/bondmachine.bcof  -bmrequirements-file working_dir/requiremen
ts.json -hw-optimizations onlydestregs,onlysrcregs
echo > working_dir/bondmachine.sv
for i in `ls *.v | sort -d` ; do cat $i >> working_dir/bondmachine.sv ; done
rm -f *.v
echo > working_dir/bondmachine.vhd
for i in `ls *.vhd | sort -d` ; do cat $i >> working_dir/bondmachine.vhd ; done
ls: cannot access '*.vhd': No such file or directory
rm -f *.vhd
[Project: Example] - [HDL generation end]

[ Command >
make xclbin
[ Output  >
```

FPGA Quantum Teleportation

```
INFO: [v++ 60-1306] Additional information associated with this v++ package can be found at:
        Reports: /tmp/tmp577nekug/Example/working_dir/rtl_bondmachine/_x/reports/package
        Log files: /tmp/tmp577nekug/Example/working_dir/rtl_bondmachine/_x/logs/package
Running Dispatch Server on port: 46409
INFO: [v++ 60-1548] Creating build summary session with primary output /tmp/tmp577nekug/Example/work
ing_dir/rtl_bondmachine/build_dir.hw.xilinx_u55c_gen3x16_xdma_3_202210_1/bondmachine.xclbin.package_
summary, at Wed Jun  5 20:23:31 2024
INFO: [v++ 60-1315] Creating rulecheck session with output '/tmp/tmp577nekug/Example/working_dir/rtl
_bondmachine/_x/reports/package/v++_package_bondmachine_guidance.html', at Wed Jun  5 20:23:31 2024
INFO: [v++ 60-895]   Target platform: /opt/xilinx/platforms/xilinx_u55c_gen3x16_xdma_3_202210_1/xili
nx_u55c_gen3x16_xdma_3_202210_1.xpfm
INFO: [v++ 60-1578]   This platform contains Xilinx Shell Archive '/opt/xilinx/platforms/xilinx_u55c
_gen3x16_xdma_3_202210_1/hw/hw.xsa'
INFO: [v++ 74-78] Compiler Version string: 2023.2
INFO: [v++ 60-2256] Packaging for hardware
INFO: [v++ 60-2460] Successfully copied a temporary xclbin to the output xclbin: /tmp/tmp577nekug/Ex
ample/working_dir/rtl_bondmachine/./build_dir.hw.xilinx_u55c_gen3x16_xdma_3_202210_1/bondmachine.xcl
bin
INFO: [v++ 60-2343] Use the vitis_analyzer tool to visualize and navigate the relevant reports. Run
the following command.
    vitis_analyzer /tmp/tmp577nekug/Example/working_dir/rtl_bondmachine/build_dir.hw.xilinx_u55c_gen
3x16_xdma_3_202210_1/bondmachine.xclbin.package_summary
INFO: [v++ 60-791] Total elapsed time: 0h 0m 6s
INFO: [v++ 60-1653] Closing dispatch client.
[Project: Example] - [Vivado toolchain - xclbin creation end]
```

FPGA Quantum Teleportation

```
INFO: [v++ 74-78] Compiler Version string: 2023.2
INFO: [v++ 60-2256] Packaging for hardware
INFO: [v++ 60-2460] Successfully copied a temporary xclbin to the output xclbin: /tmp/tmp577nekug/Ex
ample/working_dir/rtl_bondmachine/./build_dir.hw.xilinx_u55c_gen3x16_xdma_3_202210_1/bondmachine.xcl
bin
INFO: [v++ 60-2343] Use the vitis_analyzer tool to visualize and navigate the relevant reports. Run
the following command.
    vitis_analyzer /tmp/tmp577nekug/Example/working_dir/rtl_bondmachine/build_dir.hw.xilinx_u55c_gen
3x16_xdma_3_202210_1/bondmachine.xclbin.package_summary
INFO: [v++ 60-791] Total elapsed time: 0h 0m 6s
INFO: [v++ 60-1653] Closing dispatch client.
[Project: Example] - [Vivado toolchain - xclbin creation end]

[ Command >
make deploy_xclbin
[ Output  >
[Project: Example] - [BondMachine deploy xclbin begin] - [Target: deploy_xclbin]
[Project: Example] - [BondMachine deploy local]
if [ -d /home/mirko/alveoruns/Example ]; then rm -rf /home/mirko/alveoruns/Example; fi
if [ -d /home/mirko/alveoruns/template ]; then cp -a /home/mirko/alveoruns/template /home/mirko/alve
oruns/Example; fi
cp working_dir/rtl_bondmachine/build_dir.hw.xilinx_u55c_gen3x16_xdma_3_202210_1/bondmachine.xclbin /
home/mirko/alveoruns/Example/firmware.xclbin
cp working_dir/circuit.c /home/mirko/alveoruns/Example/
[Project: Example] - [BondMachine deploy xclbin end]
```

FPGA Quantum Teleportation

```
the following command.
    vitis_analyzer /tmp/tmp577nekug/Example/working_dir/rtl_bondmachine/build_dir.hw.xilinx_u55c_gen
3x16_xdma_3_202210_1/bondmachine.xclbin.package_summary
INFO: [v++ 60-791] Total elapsed time: 0h 0m 6s
INFO: [v++ 60-1653] Closing dispatch client.
[Project: Example] - [Vivado toolchain - xclbin creation end]

[ Command >
make deploy_xclbin
[ Output  >
[Project: Example] - [BondMachine deploy xclbin begin] - [Target: deploy_xclbin]
[Project: Example] - [BondMachine deploy local]
if [ -d /home/mirko/alveoruns/Example ]; then rm -rf /home/mirko/alveoruns/Example; fi
if [ -d /home/mirko/alveoruns/template ]; then cp -a /home/mirko/alveoruns/template /home/mirko/alve
oruns/Example; fi
cp working_dir/rtl_bondmachine/build_dir.hw.xilinx_u55c_gen3x16_xdma_3_202210_1/bondmachine.xclbin /
home/mirko/alveoruns/Example/firmware.xclbin
cp working_dir/circuit.c /home/mirko/alveoruns/Example/
[Project: Example] - [BondMachine deploy xclbin end]

[ Command >
bmqsim -software-simulation program.bmq
[ Output  >
[{"Vector":[{"Real":0.49999997,"Imag":0},{"Real":0,"Imag":0},{"Real":0.49999997,"Imag":0},{"Real":0,
"Imag":0},{"Real":0.49999997,"Imag":0},{"Real":0,"Imag":0},{"Real":0.49999997,"Imag":0},{"Real":0,"I
mag":0}]}]
```

FPGA Quantum Teleportation

```
INFO: [v++ 60-791] Total elapsed time: 0h 0m 6s
INFO: [v++ 60-1653] Closing dispatch client.
[Project: Example] - [Vivado toolchain - xclbin creation end]

[ Command >
make deploy_xclbin
[ Output  >
[Project: Example] - [BondMachine deploy xclbin begin] - [Target: deploy_xclbin]
[Project: Example] - [BondMachine deploy local]
if [ -d /home/mirko/alveoruns/Example ]; then rm -rf /home/mirko/alveoruns/Example; fi
if [ -d /home/mirko/alveoruns/template ]; then cp -a /home/mirko/alveoruns/template /home/mirko/alve
oruns/Example; fi
cp working_dir/rtl_bondmachine/build_dir.hw.xilinx_u55c_gen3x16_xdma_3_202210_1/bondmachine.xclbin /
home/mirko/alveoruns/Example/firmware.xclbin
cp working_dir/circuit.c /home/mirko/alveoruns/Example/
[Project: Example] - [BondMachine deploy xclbin end]

[ Command >
bmqsim -software-simulation program.bmq
[ Output  >
[{"Vector":[{"Real":0.49999997,"Imag":0},{"Real":0,"Imag":0},{"Real":0.49999997,"Imag":0},{"Real":0,
"Imag":0},{"Real":0.49999997,"Imag":0},{"Real":0,"Imag":0},{"Real":0.49999997,"Imag":0},{"Real":0,"I
mag":0}]}]
[ Command >
cd /home/mirko/alveoruns/proj_alveou55c_teleport/
source /opt/xilinx/xrt/setup.sh
```

FPGA Quantum Teleportation

```
ls/Xilinx/Vitis/2023.2/gnu/microblaze/linux_toolchain/lin64_le/bin:/tools/Xilinx/Vitis/2023.2/gnu/aa
rch32/lin/gcc-arm-linux-gnueabi/bin:/tools/Xilinx/Vitis/2023.2/gnu/aarch32/lin/gcc-arm-none-eabi/bin
:/tools/Xilinx/Vitis/2023.2/gnu/aarch64/lin/aarch64-linux/bin:/tools/Xilinx/Vitis/2023.2/gnu/aarch64
/lin/aarch64-none/bin:/tools/Xilinx/Vitis/2023.2/gnu/armr5/lin/gcc-arm-none-eabi/bin:/tools/Xilinx/V
itis/2023.2/tps/lnx64/cmake-3.3.2/bin:/tools/Xilinx/Vitis/2023.2/aietools/bin:/tools/Xilinx/Vitis/20
23.2/gnu/riscv/lin/riscv64-unknown-elf/bin:/tools/Xilinx/Vivado/2023.2/bin:/tools/Xilinx/DocNav:/opt
/xilinx/xrt/bin:/tools/Xilinx/Vitis_HLS/2023.2/bin:/tools/Xilinx/Model_Composer/2023.2/bin:/tools/Xi
linx/Vitis/2023.2/bin:/tools/Xilinx/Vitis/2023.2/gnu/microblaze/lin/bin:/tools/Xilinx/Vitis/2023.2/g
nu/microblaze/linux_toolchain/lin64_le/bin:/tools/Xilinx/Vitis/2023.2/gnu/aarch32/lin/gcc-arm-linux-
gnueabi/bin:/tools/Xilinx/Vitis/2023.2/gnu/aarch32/lin/gcc-arm-none-eabi/bin:/tools/Xilinx/Vitis/202
3.2/gnu/aarch64/lin/aarch64-linux/bin:/tools/Xilinx/Vitis/2023.2/gnu/aarch64/lin/aarch64-none/bin:/t
ools/Xilinx/Vitis/2023.2/gnu/armr5/lin/gcc-arm-none-eabi/bin:/tools/Xilinx/Vitis/2023.2/tps/lnx64/cm
ake-3.3.2/bin:/tools/Xilinx/Vitis/2023.2/aietools/bin:/tools/Xilinx/Vitis/2023.2/gnu/riscv/lin/riscv
64-unknown-elf/bin:/tools/Xilinx/Vivado/2023.2/bin:/tools/Xilinx/DocNav:/tools/Xilinx/Vitis_HLS/2023
.2/bin:/tools/Xilinx/Model_Composer/2023.2/bin:/tools/Xilinx/Vitis/2023.2/bin:/tools/Xilinx/Vitis/20
23.2/gnu/microblaze/lin/bin:/tools/Xilinx/Vitis/2023.2/gnu/microblaze/linux_toolchain/lin64_le/bin:/
tools/Xilinx/Vitis/2023.2/gnu/aarch32/lin/gcc-arm-linux-gnueabi/bin:/tools/Xilinx/Vitis/2023.2/gnu/a
arch32/lin/gcc-arm-none-eabi/bin:/tools/Xilinx/Vitis/2023.2/gnu/aarch64/lin/aarch64-linux/bin:/tools
/Xilinx/Vitis/2023.2/gnu/aarch64/lin/aarch64-none/bin:/tools/Xilinx/Vitis/2023.2/gnu/armr5/lin/gcc-a
rm-none-eabi/bin:/tools/Xilinx/Vitis/2023.2/tps/lnx64/cmake-3.3.2/bin:/tools/Xilinx/Vitis/2023.2/aie
tools/bin:/tools/Xilinx/Vitis/2023.2/gnu/riscv/lin/riscv64-unknown-elf/bin:/tools/Xilinx/Vivado/2023
.2/bin:/tools/Xilinx/DocNav:/usr/lib/xpra:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/b
in:/usr/games:/usr/local/games:/snap/bin:/home/mirko/.go/bin:/home/mirko/Workarea/Scripts:/usr/local
/go/bin
LD_LIBRARY_PATH    : /opt/xilinx/xrt/lib:/opt/xilinx/xrt/lib
PYTHONPATH         : /opt/xilinx/xrt/python:/opt/xilinx/xrt/python
```

FPGA Quantum Teleportation

```
linx/Vitis/2023.2/bin:/tools/Xilinx/Vitis/2023.2/gnu/microblaze/lin/bin:/tools/Xilinx/Vitis/2023.2/g
nu/microblaze/linux_toolchain/lin64_le/bin:/tools/Xilinx/Vitis/2023.2/gnu/aarch32/lin/gcc-arm-linux-
gnueabi/bin:/tools/Xilinx/Vitis/2023.2/gnu/aarch32/lin/gcc-arm-none-eabi/bin:/tools/Xilinx/Vitis/202
3.2/gnu/aarch64/lin/aarch64-linux/bin:/tools/Xilinx/Vitis/2023.2/gnu/aarch64/lin/aarch64-none/bin:/t
ools/Xilinx/Vitis/2023.2/gnu/armr5/lin/gcc-arm-none-eabi/bin:/tools/Xilinx/Vitis/2023.2/tps/lnx64/cm
ake-3.3.2/bin:/tools/Xilinx/Vitis/2023.2/aietools/bin:/tools/Xilinx/Vitis/2023.2/gnu/riscv/lin/riscv
64-unknown-elf/bin:/tools/Xilinx/Vivado/2023.2/bin:/tools/Xilinx/DocNav:/tools/Xilinx/Vitis_HLS/2023
.2/bin:/tools/Xilinx/Model_Composer/2023.2/bin:/tools/Xilinx/Vitis/2023.2/bin:/tools/Xilinx/Vitis/20
23.2/gnu/microblaze/lin/bin:/tools/Xilinx/Vitis/2023.2/gnu/microblaze/linux_toolchain/lin64_le/bin:/
tools/Xilinx/Vitis/2023.2/gnu/aarch32/lin/gcc-arm-linux-gnueabi/bin:/tools/Xilinx/Vitis/2023.2/gnu/a
arch32/lin/gcc-arm-none-eabi/bin:/tools/Xilinx/Vitis/2023.2/gnu/aarch64/lin/aarch64-linux/bin:/tools
/Xilinx/Vitis/2023.2/gnu/aarch64/lin/aarch64-none/bin:/tools/Xilinx/Vitis/2023.2/gnu/armr5/lin/gcc-a
rm-none-eabi/bin:/tools/Xilinx/Vitis/2023.2/tps/lnx64/cmake-3.3.2/bin:/tools/Xilinx/Vitis/2023.2/aie
tools/bin:/tools/Xilinx/Vitis/2023.2/gnu/riscv/lin/riscv64-unknown-elf/bin:/tools/Xilinx/Vivado/2023
.2/bin:/tools/Xilinx/DocNav:/usr/lib/xpra:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/b
in:/usr/games:/usr/local/games:/snap/bin:/home/mirko/.go/bin:/home/mirko/Workarea/Scripts:/usr/local
/go/bin
LD_LIBRARY_PATH    : /opt/xilinx/xrt/lib:/opt/xilinx/xrt/lib
PYTHONPATH         : /opt/xilinx/xrt/python:/opt/xilinx/xrt/python
[ Command >
make
[ Output  >
g++ -o circuit /home/mirko/Tests/Vitis_Accel_Examples/common/includes/xcl2/xcl2.cpp circuit.c -I/opt
/xilinx/xrt/include -I/tools/Xilinx/Vivado/2023.2/include -Wall -O0 -g -std=c++1y -I/home/mirko/Test
s/Vitis_Accel_Examples/common/includes/xcl2 -fmessage-length=0 -L/opt/xilinx/xrt/lib -pthread -lOpen
CL -lrt -lstdc++
```

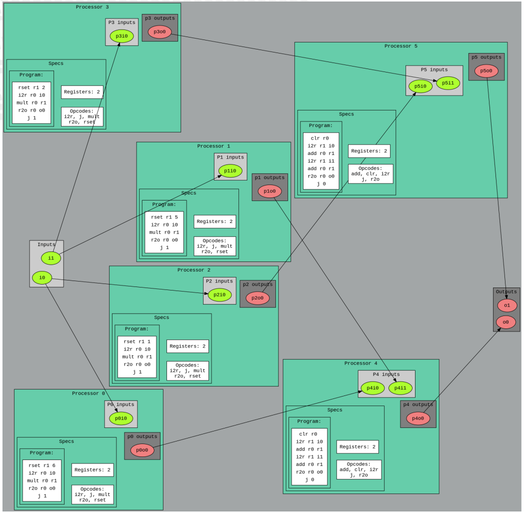
FPGA Quantum Teleportation

```
CL -lrt -lstdc++
[ Command >
./circuit
[ Output  >
Found Platform
Platform Name: Xilinx
INFO: Reading firmware.xclbin
Loading: 'firmware.xclbin'
Device[0]: program successful!
1.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.5
0
0
0
0.5
0
0
0
0.5
0
0
0
0.5
0
0
0
```

Misc

Uses of the architectures

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



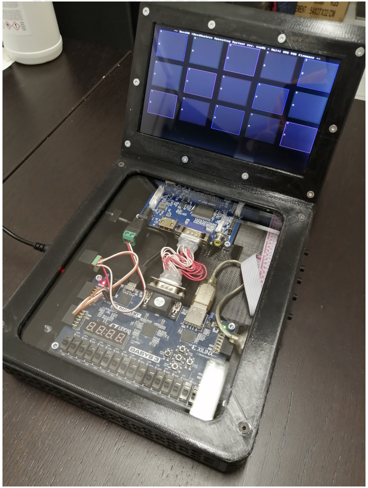
Uses of the architectures

- The BondMachine is a software ecosystem for the dynamical generation (from several HL types of origin) of computer architectures that can be synthesized of FPGA and

- used as standalone devices,

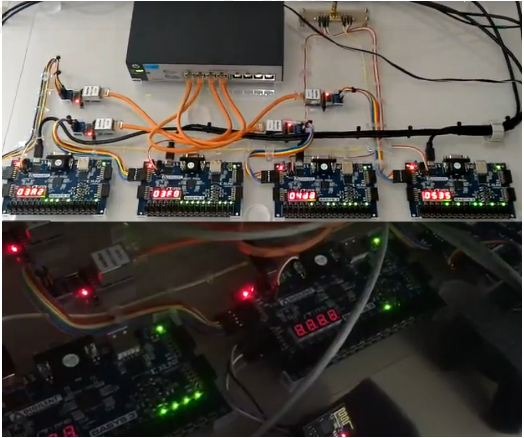
- as clustered devices,

- and as firmware for computing accelerators.



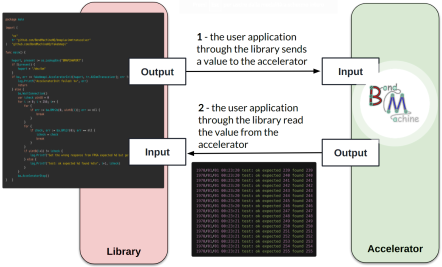
Uses of the architectures

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



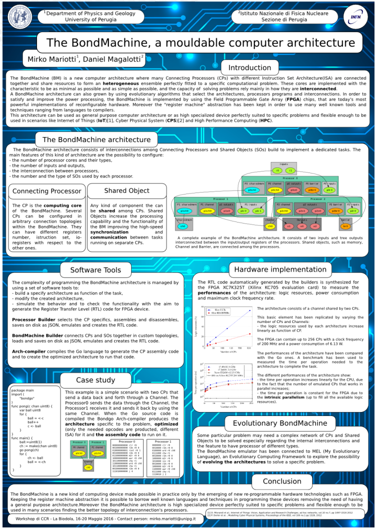
Uses of the architectures

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



Project timeline

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



Project timeline

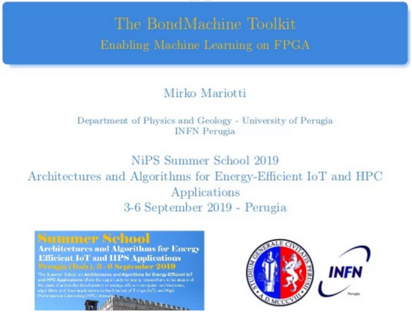
- CCR 2015 First ideas, 2016 Poster, 2017 2022 2023 Talk
- InnovateFPGA 2018 Iron Award, Grand Final at Intel Campus (CA) USA
- Invited lectures: "Advanced Workshop on Modern FPGA Based Technology for Scientific Computing", ICTP 2019 and 2022
- Invited lectures: "NiPS Summer School 2019"
- Golab 2018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

Project timeline

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

Project timeline

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

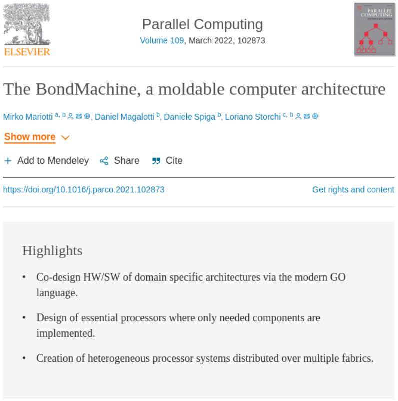


Project timeline

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

Project timeline

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



ELSEVIER

Parallel Computing

Volume 109, March 2022, 102873

The BondMachine, a moldable computer architecture

Mirko Mariotti [a, b], Daniel Magalotti [b], Daniele Spiga [b], Loriano Storchi [c, b]

Show more

+ Add to Mendeley Share Cite

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

Get rights and content

Highlights

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

Project timeline

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

Fabrics

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

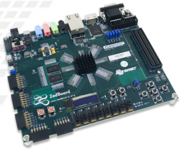
- Digilent Basys3 - Xilinx Artix-7 - Vivado
- Kintex7 Evaluation Board - Vivado
- Digilent Zedboard and ebaz4205- Xilinx Zynq 7020 - Vivado
- ZC702 - Xilinx Zynq 7020 - Vivado
- Alveo boards - Xilinx - Vivado/Vitis
- Linux - Iverilog
- ice40lp1k icefun icebreaker icesugarnano - Lattice - Icestorm
- Terasic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

Within the project other firmware have been written or tested:

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

Accelerators

Accelerators tests were done using the **Zedboard** and **Alveo** devices, in the near future we will use facilities of the **National supercomputing center (ICSC)** also with other vendors devices, and to explore clustering capabilities.



Xilinx Zynq-7000 SoC
53.2K LUTs

Alveo U55C
1304K LUTs

FPGA cluster ICSC
Xilinx and Intel FPGAs

National supercomputing center (ICSC)

References

Website

Main - https://bondmachine.fisgeo.unipg.it

GitHub

Organization - https://github.com/BondMachineHQ

Main repo - https://github.com/BondMachineHQ/BondMachine

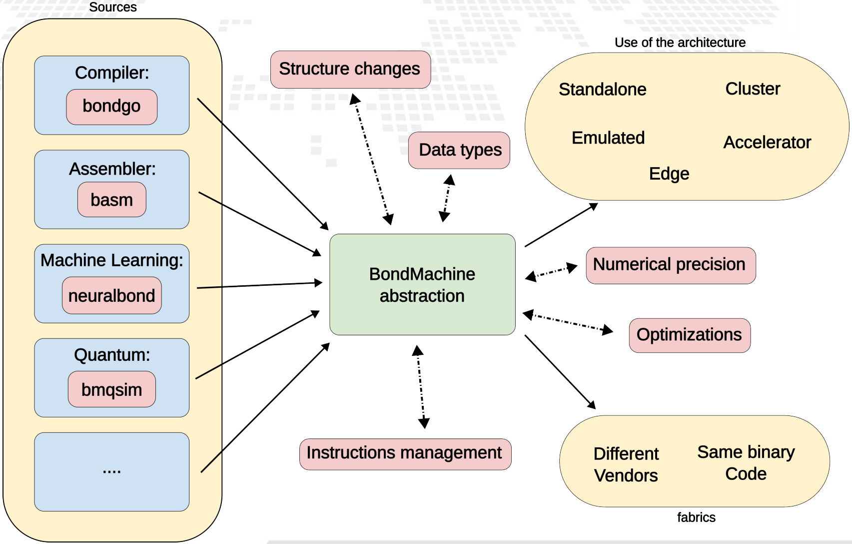
Examples - https://github.com/BondMachineHQ/bmexamples

Papers

Parallel Computing - https://doi.org/10.1016/j.parco.2021.102873

Conclusions

The Ecosystem in a nutshell



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.

website: `http://bondmachine.fisica.unipg.it`
code: `https://github.com/BondMachineHQ`
parallel computing paper: link
contact email: mirko.mariotti@unipg.it

Acknowledgements:

ICSC
Centro Nazionale di Ricerca in HPC, Big Data and Quantum Computing

- This work is partially supported by ICSC – Spoke 2 - Centro Nazionale di Ricerca in High Performance Computing, Big Data and Quantum Computing, funded by European Union – NextGenerationEU
- Terasic and Intel (InnovateFPGA)
- AMD/Xilinx (University Program)