



A.D. 1508

unipg

DEPARTMENT  
OF PHYSICS AND GEOLOGY



## Firmware development for hybrid processors (ARM and FPGA) computing

Mirko Mariotti <sup>1,2</sup>    Giulio Bianchini <sup>1</sup>    Loriano Storchi <sup>3,2</sup>    Giacomo Surace <sup>1</sup>  
Daniele Spiga <sup>2</sup>

<sup>1</sup>Dipartimento di Fisica e Geologia, Universitá degli Studi di Perugia

<sup>2</sup>INFN sezione di Perugia

<sup>3</sup>Dipartimento di Farmacia, Universitá degli Studi G. D'Annunzio

# Outline

## 1 Introduction

Evolution of computing: new challenges

Accelerators and FPGA

BondMachine

## 2 An accelerated system from ground up

Hardware

Software

## 3 Tests and Benchmarks

Tests

Benchmark

## 4 Conclusions and Future directions

# Introduction

## 1 Introduction

Evolution of computing: new challenges  
Accelerators and FPGA  
BondMachine

## 2 An accelerated system from ground up

Hardware  
Software

## 3 Tests and Benchmarks

Tests  
Benchmark

## 4 Conclusions and Future directions

# Evolution of computing

## New challenges

Von Neumann  
bottleneck

Energy  
Efficient  
Computing

Edge computing

Data-oriented  
Computing

# Evolution of computing

- New challenges



- New architectures



# Evolution of computing

## New challenges

## New architectures

## From the software point of view: Heterogeneity

```
with ipu.scopes.ipu_scope('/device:IPU:0'):
    training_loop_body_on_ipu = ipu.ipu_compiler.compile(computation=training_loop_body, inputs=[x, y])

ipu_configuration = ipu.config.IPUConfig()
ipu_configuration.auto_select_ipus = 1
ipu_configuration.configure_ipu_system()
```

```
e_load_group("e_math_test.srec", &dev, 0, 0, platform.rows, platform.cols, E_FALSE);
e_load_group("e_math_test1.srec", &dev, 0, 0, platform.rows, platform.cols, E_FALSE);
```

```
const char *kernel =
"__kernel
void kernel(float alpha,      \n"
"           __global float *A,   \n"
"           __global float *B,   \n"
"           __global float *C) \n"
"{\n"
"    int index = get_global_id(0);\n"
"    C[index] = alpha* A[index] + B[index];\n"
"}\n"

cl_program program = clCreateProgramWithSource(context, 1,(const char **) &kernel, NULL, &clStatus);
```

# Evolution of computing

- New challenges
- New architectures
- From the software point of view:  
Heterogeneity
- From the hardware point of view:  
Accelerators



# Accelerators

Hardware device or software program designed to improve the performance of certain workload.

## Graphics Processing Unit (GPU)

- High Data Throughput
- Massive Parallel Computing



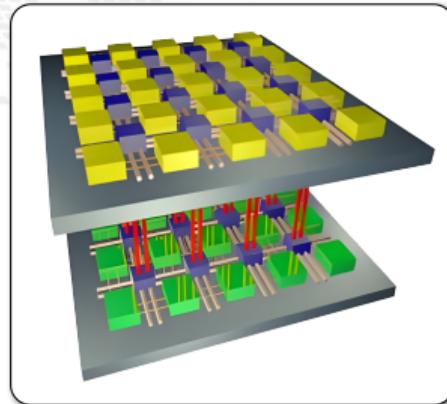
## Application-Specific Integrated Circuit (ASIC)

- Highly specialized for a task
- Energy efficient (due to specialization)

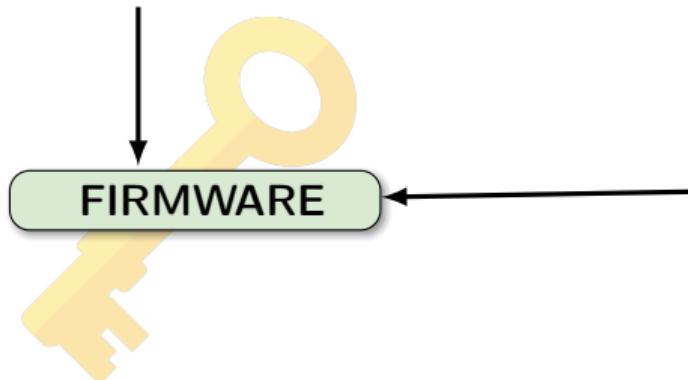


# FPGA accelerator

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



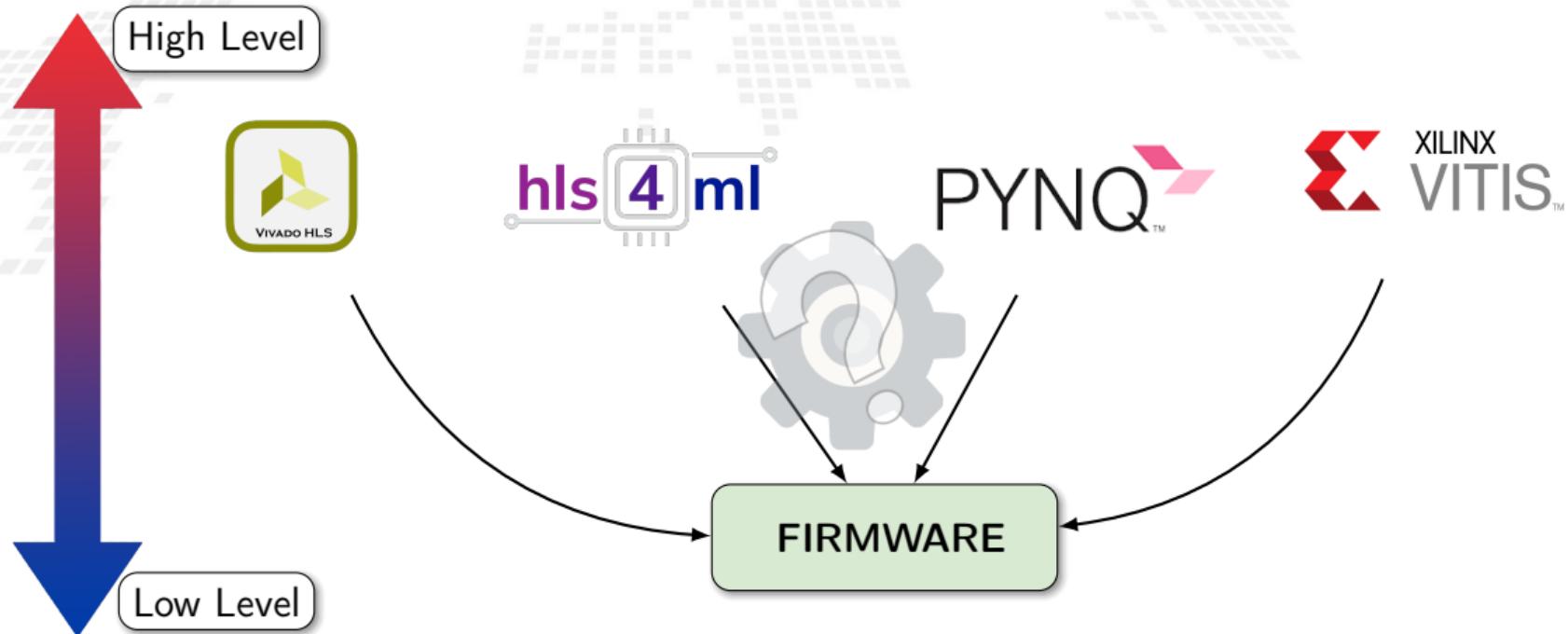
- Parallel computing
- Highly specialized
- Energy efficient



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

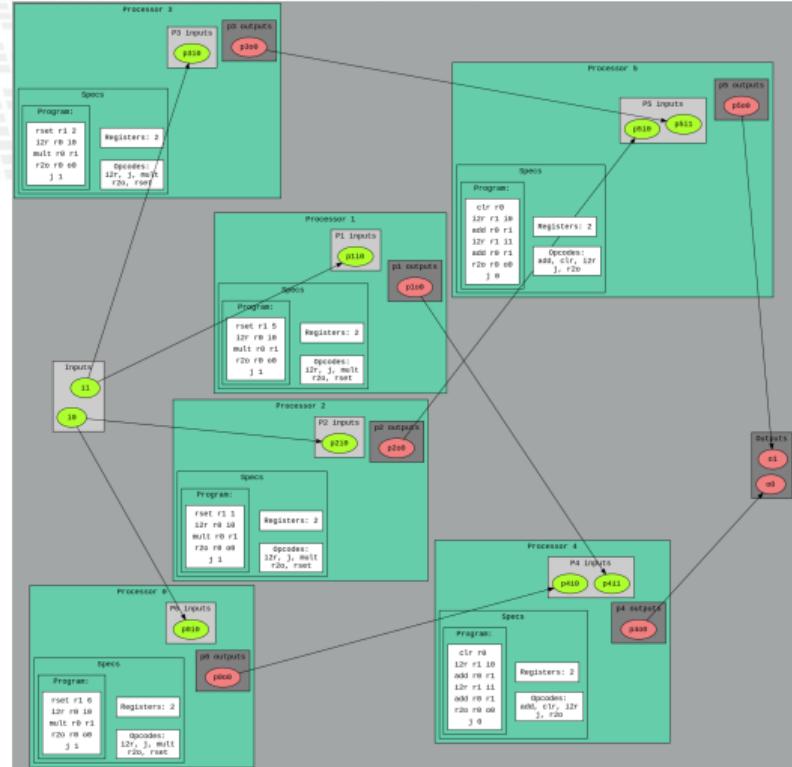
# Firmware generation

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



# BondMachine

- The BondMachine is a software ecosystem for the dynamical generation of computer architectures that can be synthesized of FPGA and



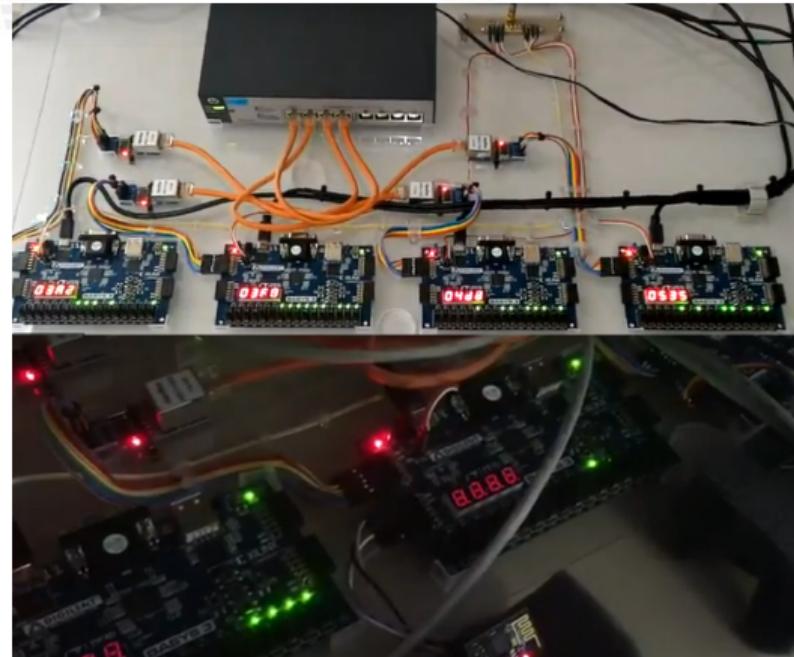
# BondMachine

- The BondMachine is a software ecosystem for the dynamical generation of computer architectures that can be synthesized on FPGA and
- used as standalone devices,



# BondMachine

- The BondMachine is a software ecosystem for the dynamical generation of computer architectures that can be synthesized of FPGA and
- used as standalone devices,
- as clustered devices,



# BondMachine

- The BondMachine is a software ecosystem for the dynamical generation of computer architectures that can be synthesized of FPGA and
- used as standalone devices,
- as clustered devices,
- and, as in this talk, as firmware for computing accelerators.

# BondMachine

## CCR 2015 First ideas, 2016 Poster, 2017 Talk

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

The BondMachine is a new kind of computer architecture that can be used in many scenarios, from general purpose to specific applications. In order to safely and improve the power processing, the BondMachine is implemented by using the Field Programmable Gate Array (FPGA) chips, that are today's most powerful implementations of reprogrammable hardware. However, the regular memory abstraction has been kept in order to make very well known tools and development environments available to the users.

This architecture can be used as general purpose computer architecture or as high specialized device perfectly suited to specific problems and flexible enough to be used in scenarios like Internet of Things, Intel® Cyber Physical Systems (CPS) and High Performance Computing (HPC).

### Introduction

The BondMachine architecture is a mouldable computer architecture based on a modular design of connecting processors and shared objects. It consists of two inputs and two outputs, and it is able to handle multiple tasks simultaneously. The architecture is designed to be highly flexible and adaptable to different applications, while maintaining high performance and low power consumption.

### The BondMachine architecture

The main feature of this kind of architecture is the possibility to configure:

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

### Connecting Processor

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

### Shared Object

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

### Software Tools

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

**Processor Builder** selects the CP specifics, assembles and disassembles, saves on disk as XML, emulates and creates the RTL code.

**Backend Compiler** takes the XML and SOs together to generate bitstreams, and saves on disk as JTAG, emulates and creates the RTL code.

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

### Hardware implementation

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

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

The FPGA can contain up to 20 CPs with a clock frequency of 200 MHz and a power consumption of 0.53 W.

The different performances of the architecture show the influence of the number of cores and the number of shared objects. Due to the fact that the number of enclosed CPs that works in parallel is constant, the performance increases with the number of cores. The time per operation is constant for the PPA due to the **parallel paradigm** (or in R, all the available logic resources).

### Case study

This example is a simple scenario with two CPs that have to process a sequence of data. A first Connecting Processor sends the data through the Channel, the second receives the data and performs the same operation. When the Ge source code is compiled the BondMachine Arch-compiler produces the **assembly code** and the **hardware description** (both the needed codes are predicted, different GAs for it and the assembly code to run on it).

### Evolutionary BondMachine

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

The BondMachine can be used in combination with the BondBuilder Language, an Evolutionary Computing framework to explore the possibility of **evolving the architectures** to solve a specific problem.

### Conclusion

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

Workshop di CCR - La Meloria, 18-20 Maggio 2016 - Contact person: Mirko.Mariotti@unipi.it

Department of Physics and Geology  
University of Perugia

Istituto Nazionale di Fisica Nucleare  
Sezione di Perugia

INFN

# BondMachine

- CCR 2015 First ideas, 2016 Poster, 2017 Talk
- InnovateFPGA 2018 Iron Award, Grand Final at Intel Campus (CA) USA



# BondMachine

- CCR 2015 First ideas, 2016 Poster, 2017 Talk
- InnovateFPGA 2018 Iron Award, Grand Final at Intel Campus (CA) USA
- Invited lectures at: "Advanced Workshop on Modern FPGA Based Technology for Scientific Computing", ICTP 2019



Advanced Workshop on Modern FPGA Based Technology for Scientific Computing  
13 – 24 May 2019  
Miramare, Trieste - Italy



# BondMachine

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

The BondMachine Toolkit  
Enabling Machine Learning on FPGA

Mirko Mariotti

Department of Physics and Geology - University of Perugia  
INFN Perugia

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



# BondMachine

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



# BondMachine

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



Parallel Computing  
Volume 109, March 2022, 102873



The BondMachine, a moldable computer architecture

Mirko Mariotti <sup>a, b</sup> , Daniel Magalotti <sup>b</sup>, Daniele Spiga <sup>b</sup>, Loriano Storchi <sup>c</sup>,

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.

# BondMachine

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

# An accelerated system from ground up

## 1 Introduction

Evolution of computing: new challenges  
Accelerators and FPGA  
BondMachine

## 2 An accelerated system from ground up

Hardware  
Software

## 3 Tests and Benchmarks

Tests  
Benchmark

## 4 Conclusions and Future directions

# Specs

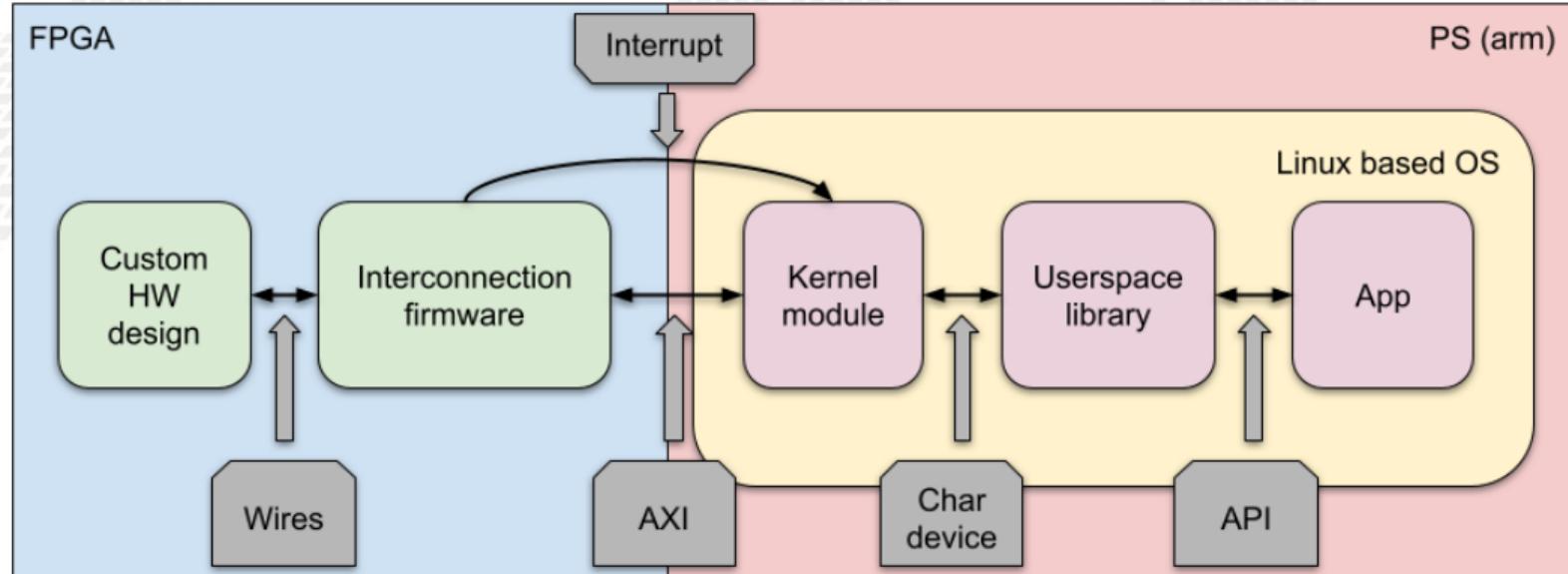
## FPGA

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

## Workstations

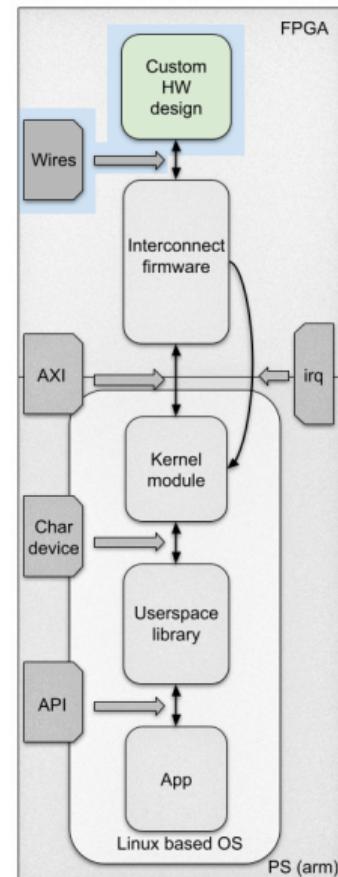
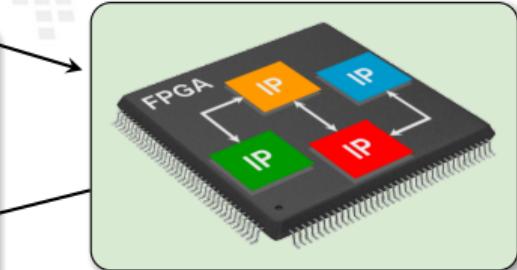
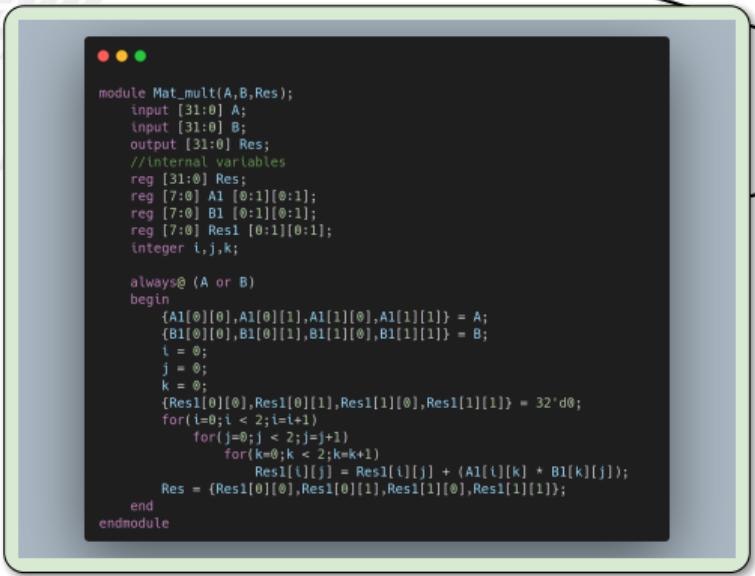
- Dell Precision Tower 3620
- Intel(R) Xeon(R) CPU E3-1270 v5 @ 3.60GHz
- 16GB Ram
- Golang 1.18.1
- Intel(R) CPU I5-8500 v5 @ 3GHz
- 16GB Ram
- GCC with -O0

# The whole system overview

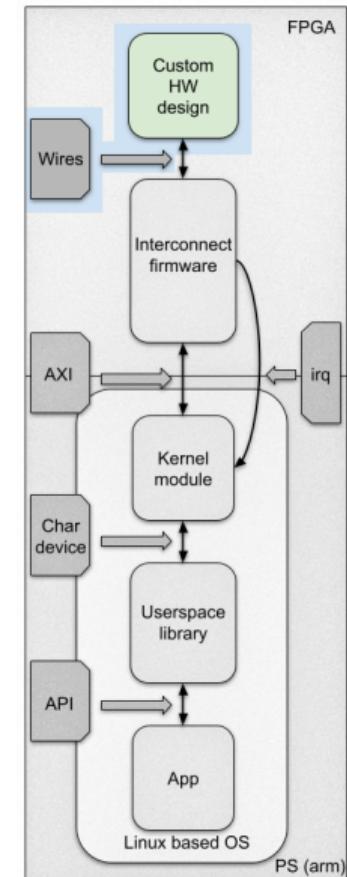
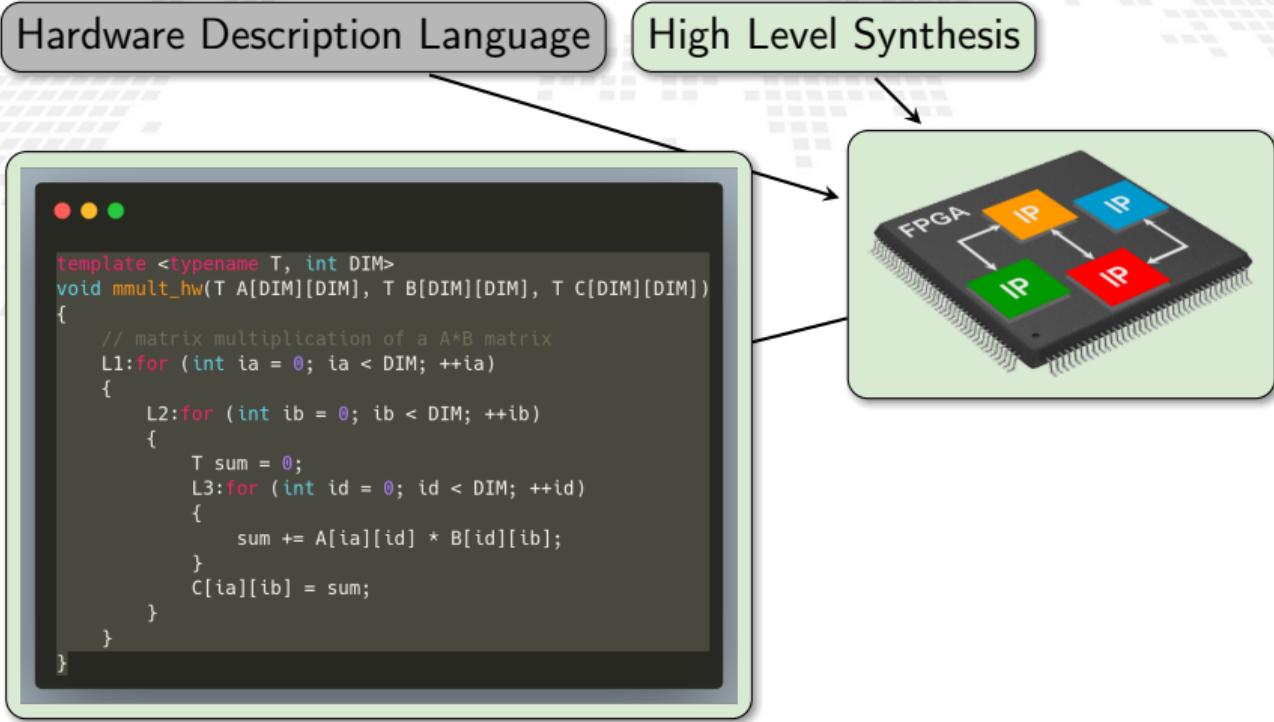


# The Accelerator IP

## Hardware Description Language



# The Accelerator IP

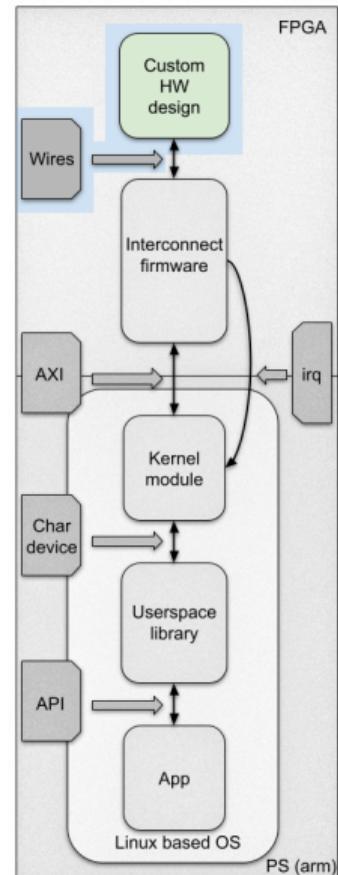
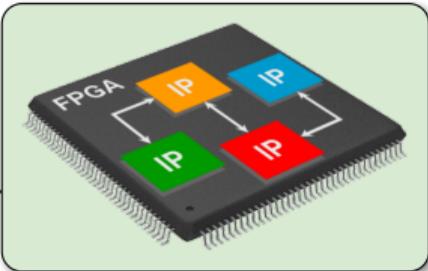
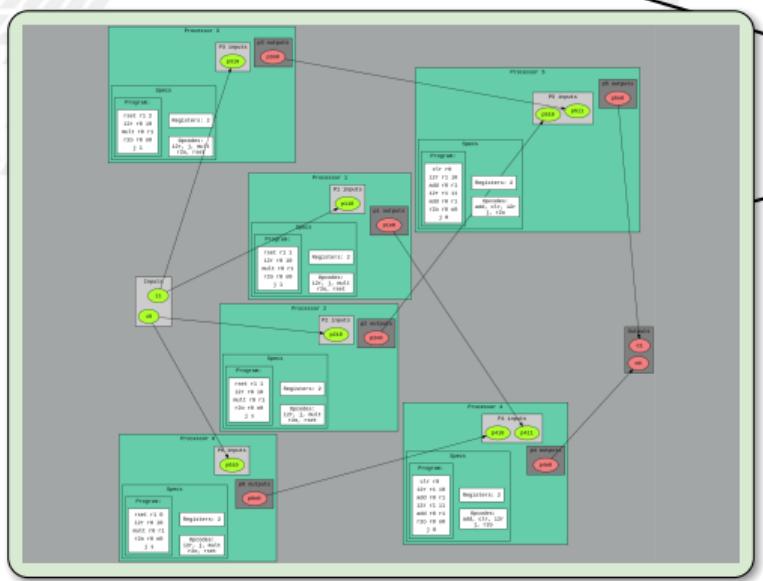


# The Accelerator IP

Hardware Description Language

High Level Synthesis

BondMachine

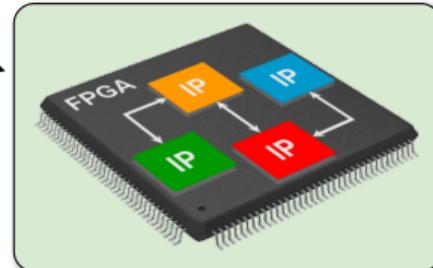
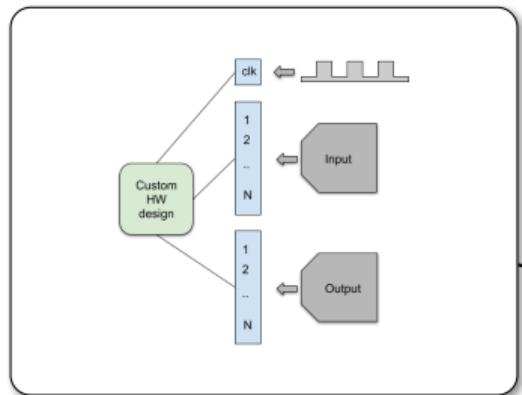


# The Accelerator IP

Hardware Description Language

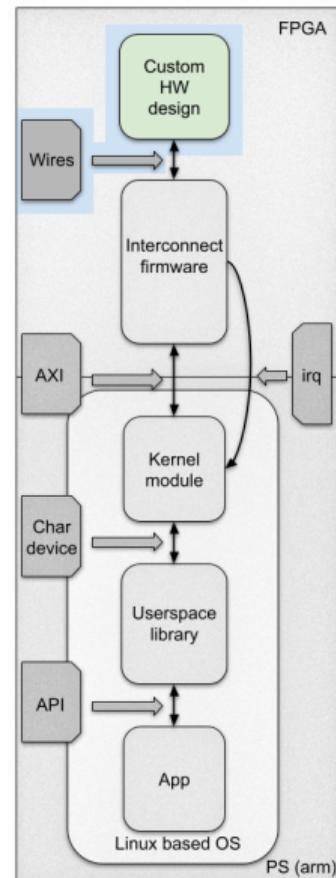
High Level Synthesis

BondMachine



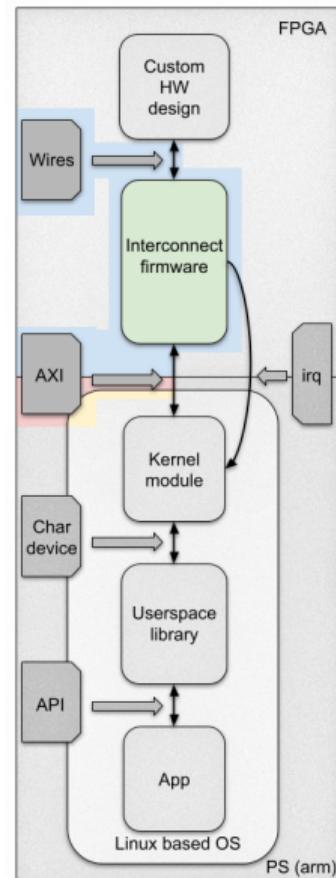
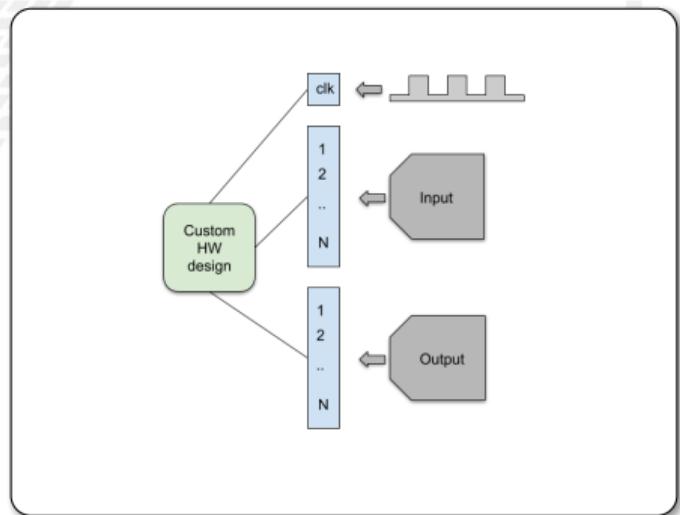
Wires:

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



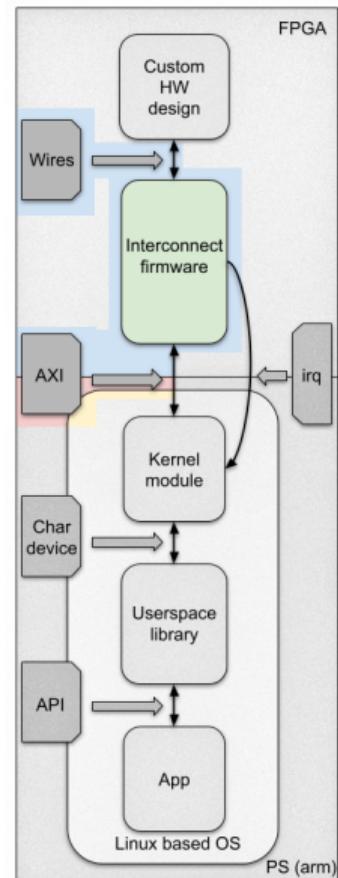
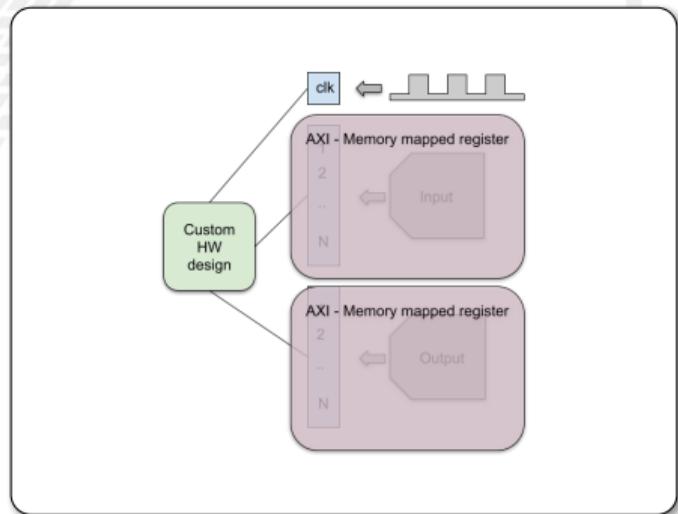
# Interconnection firmware

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



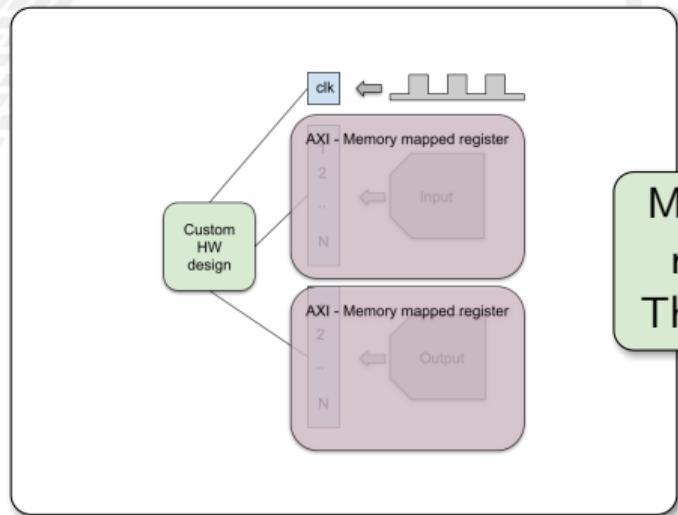
# Interconnection firmware

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



# Interconnection firmware

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



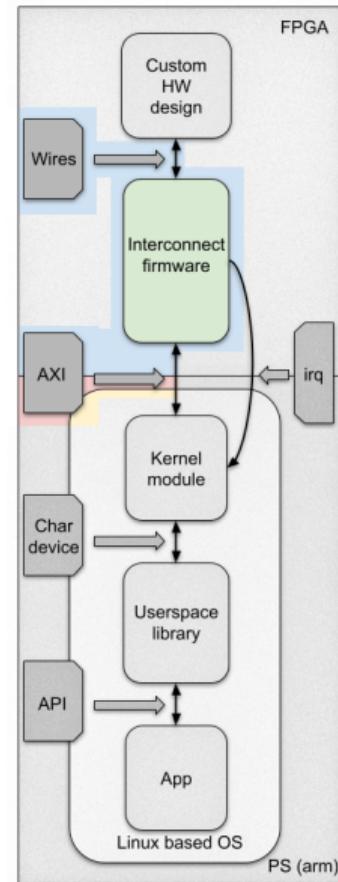
Memory mapped  
registers using  
The AXI protocol

```
wire [31:0] states;
wire [31:0] changes;
wire [31:0] port_00;
wire [31:0] DMR_P12P5;
wire [31:0] port_01;
wire [31:0] port_02;
wire [31:0] port_03;
wire [31:0] port_04;
wire [31:0] port_05;
wire [31:0] port_06;
wire [31:0] port_07;
wire [31:0] port_08;
wire [31:0] port_09;
wire [31:0] port_10;
wire [31:0] port_11;
wire [31:0] port_12;

bmdmachine_main bmdmachine_top(
.clk(S_AXI_ACLK),
.btrc(btrc),
.A_DMAR_P12S(0000_P12P5),
.A_DMR_P12S(0000_P12P5),
.A_changes(changes),
.A_states(states),
.A_port_00(port_00),
.A_port_01(port_01),
.A_port_02(port_02),
.A_port_03(port_03),
.A_port_04(port_04),
.A_port_05(port_05),
.A_port_06(port_06),
.A_port_07(port_07),
.A_port_08(port_08),
.A_port_09(port_09),
.A_port_10(port_10),
.A_port_11(port_11),
.A_port_12(port_12),
.iError(iError);
);

assign port_00 = slv_reg[31:0];
assign port_01 = slv_reg[31:0];
assign port_02 = slv_reg[31:0];
assign port_03 = slv_reg[31:0];
assign DMR_P12P5 = slv_reg[31:0];
assign states = slv_reg[31:0];

always @(*) posedge S_AXI_ACLK
begin
  slv_reg5 <= port_00[31:0];
  slv_reg6 <= port_01[31:0];
  slv_reg7 <= port_02[31:0];
  slv_reg8 <= port_03[31:0];
  slv_reg9 <= DMR_P12P5[31:0];
  slv_reg9 <= changes[31:0];
end
```

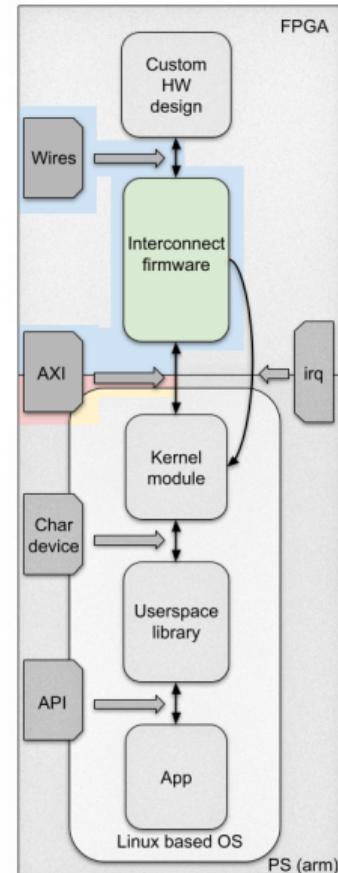
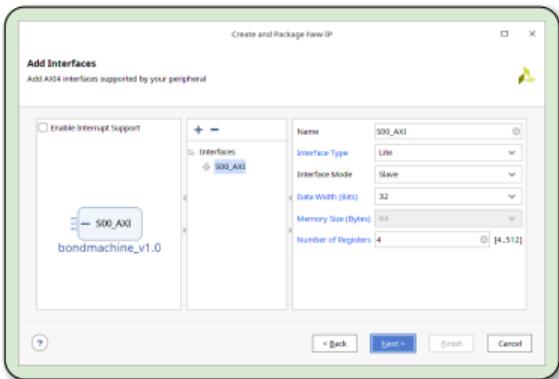


# The Advanced eXtensible Interface Protocol

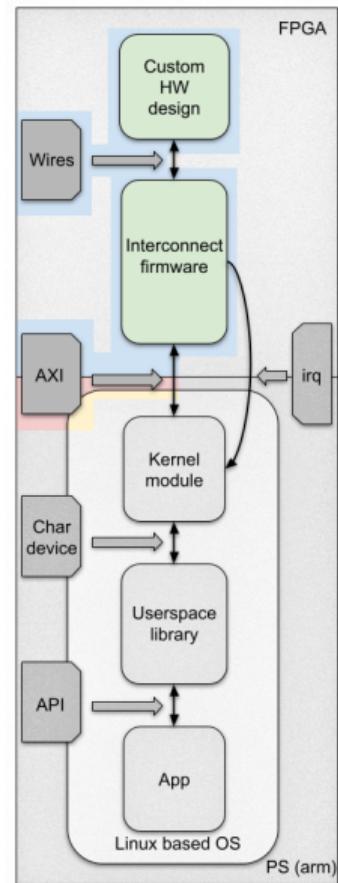
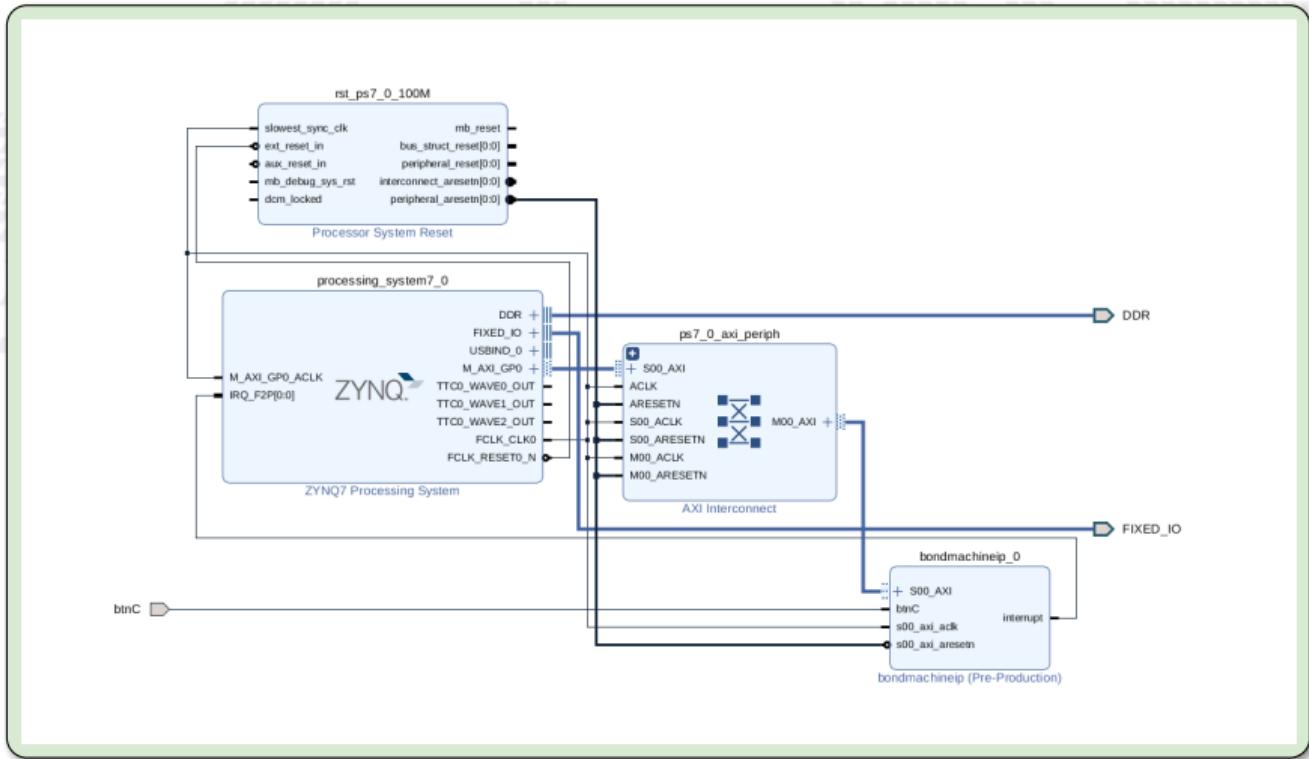
AXI is a communication bus protocol defined by ARM as part of the Advanced Microcontroller Bus Architecture (AMBA) standard.

There are 3 types of AXI Interfaces:

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



# Block Design



# Linux

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

## Common Features

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

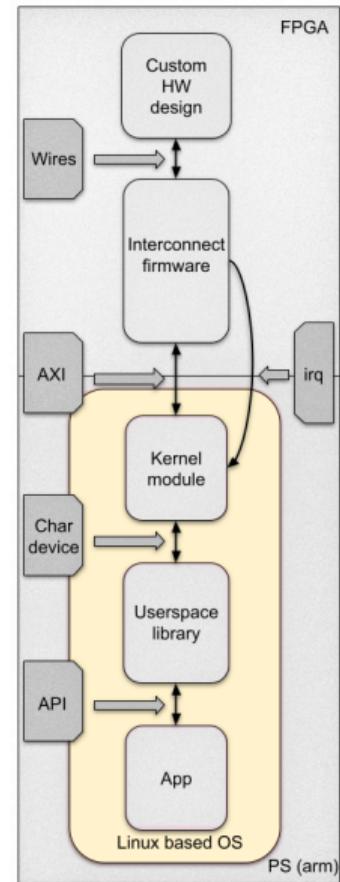
### Yocto

- Convenient sharing of build configuration among similar projects (meta-layers)
- Larger community (Linux Foundation project)
- Can build a toolchain that runs on the target
- A package management system

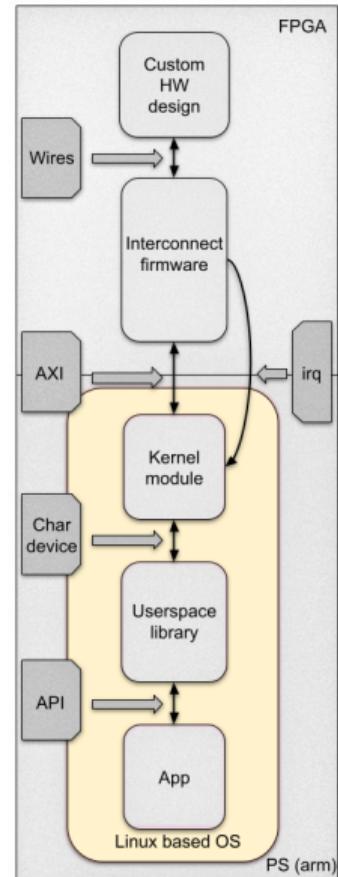
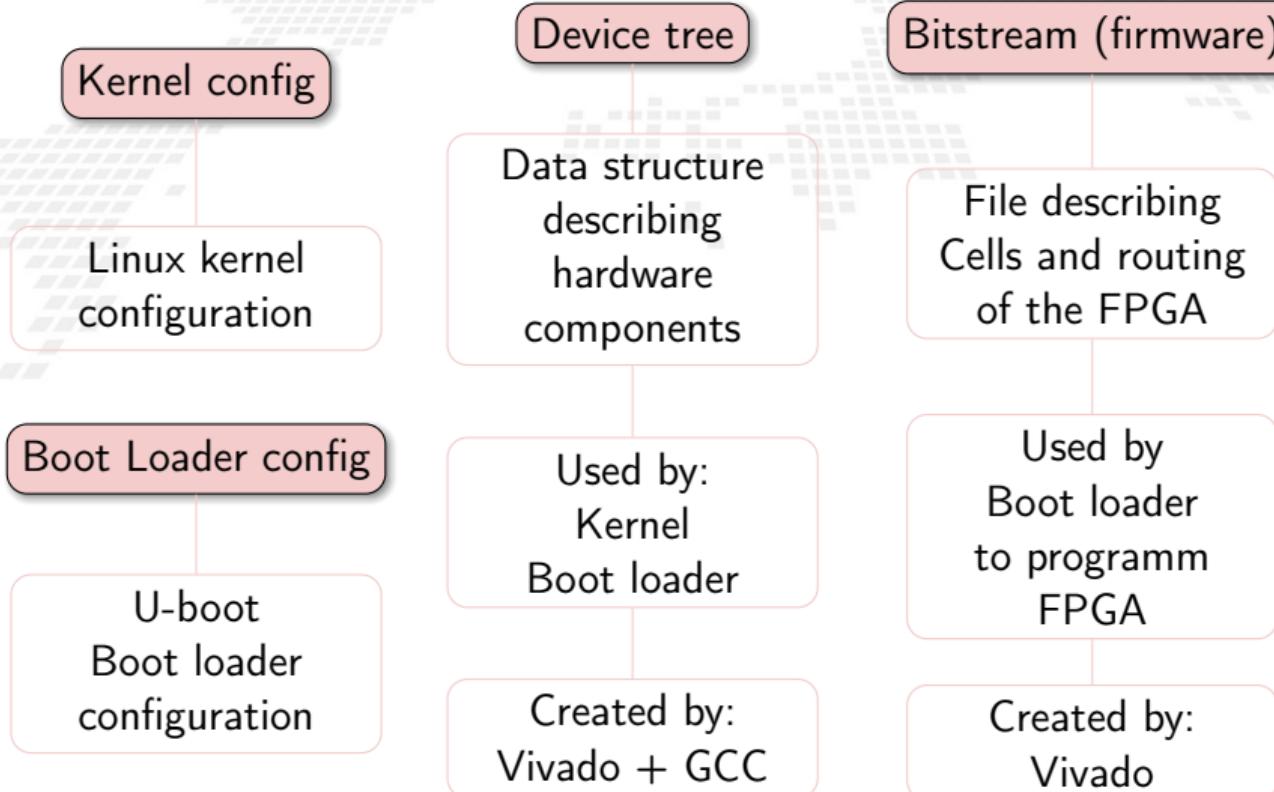
### Buildroot

- Simple Makefile approach, easier to understand how the build system works
- Reduced resource requirements on the build machine
- Very easy to customize the final root filesystem ([overlays](#))

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

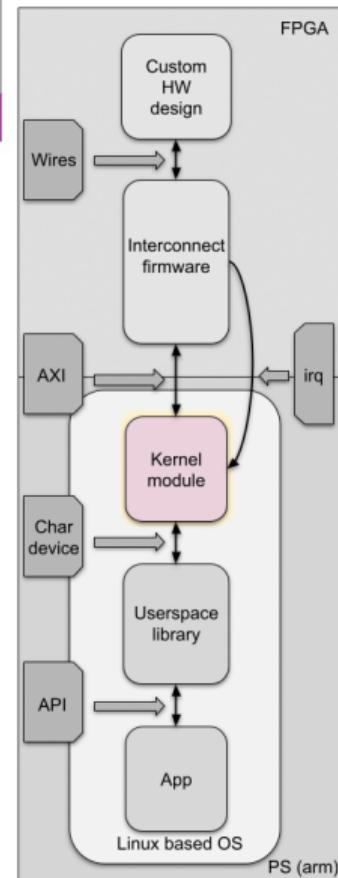
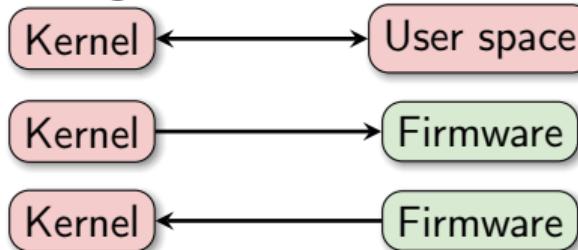


# Ingredients to build the distro



# kernel module

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



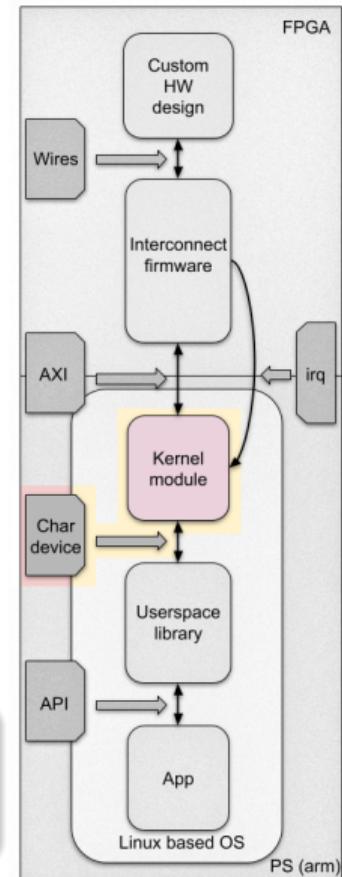
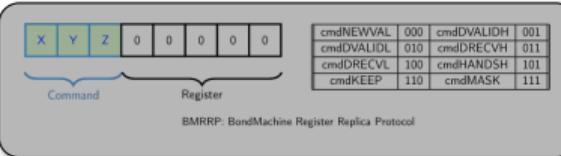
# Kernel from an to user space: char device

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

A language has been implemented for the desired operations

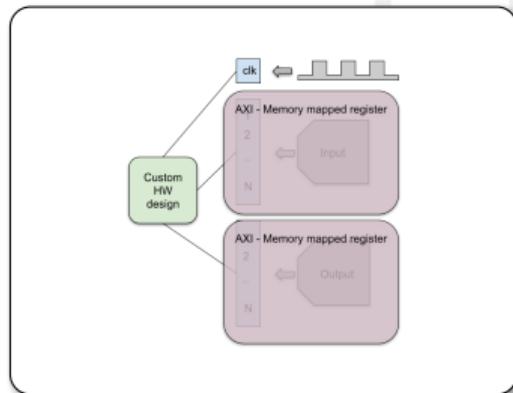
```
static ssize_t bm_read(struct file *filp, char __user *buf, size_t len, loff_t *off)
{
    struct work_data *writework;
    wait_event_interruptible(wait_queue_bm, wait_queue_flag != 0);
    switch (wait_queue_flag)
    {
    case 1:
        switch (bmacc_state)
        {
        case stateSREQV:
            if (copy_to_user(buf, &mask, 1))
                {
                    pr_err("Data Read : ErrIn");
                }
            //pr_info("sent MASK");
            bmacc_state = stateMANSENT;
            wait_queue_flag = 0;
            return 1;
            break;
        // ...
    }

    static ssize_t bm_write(struct file *filp, const char __user *buf, size_t len, loff_t *off)
    {
        struct work_data *writework;
        if (copy_from_user(write_buffer, buf, len))
        {
            pr_err("Data write errorIn");
        }
        else
        {
            for (i = 0; i < len; i++)
            {
                switch (bmacc_state)
                {
                case stateWIT:
                    switch (write_buffer[i] & cmdMASK)
                    {
                    case cmdWADDSH:
                        if (copy_to_user(buf, &mask, 1))
                            {
                                pr_err("Data Write");
                            }
                        bmacc_state = stateSREQV;
                        wait_queue_flag = 0;
                        writework = kmalloc(sizeof(struct work_data), GFP_KERNEL);
                        INIT_WORK(writework->work, work_handler);
                        queue_work(wq, writework->work);
                        break;
                    
```



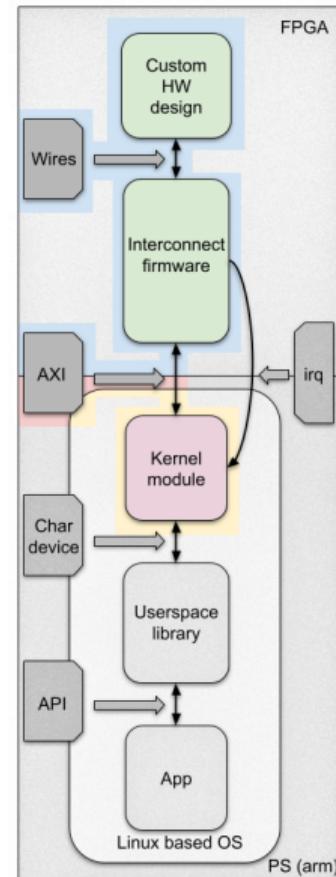
# Kernel to firmware

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



AXI registers are directly written by the kernel

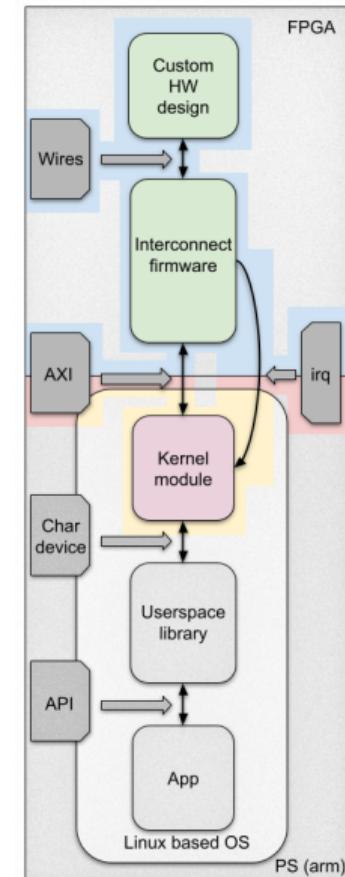
AXI guarantees consistency and transfer to the firmware input ports. Moreover the data flow from kernel cannot saturate the PL part.



# Firmware to kernel: IRQ

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

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

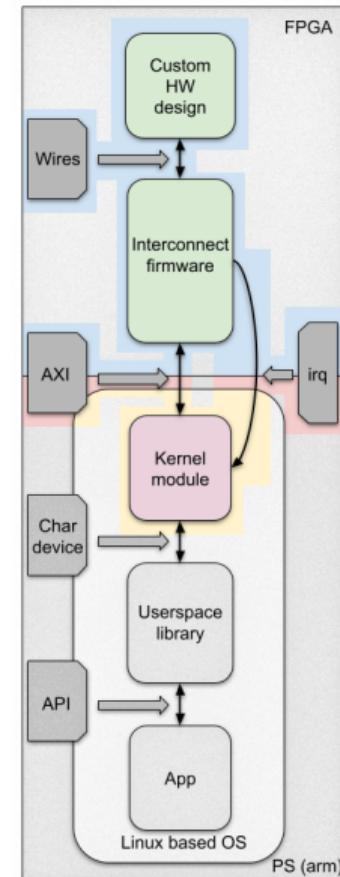


# Firmware to kernel: IRQ

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

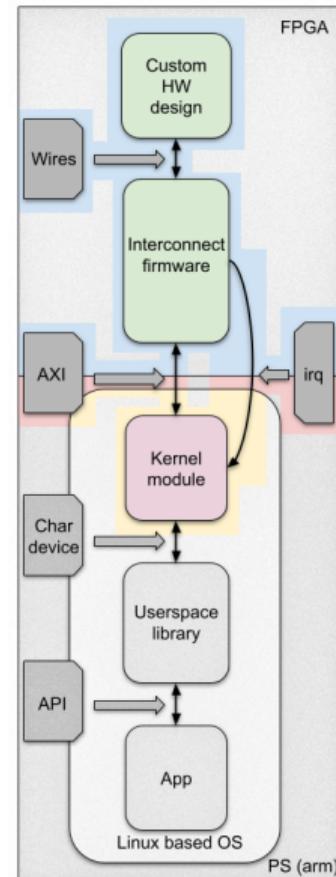
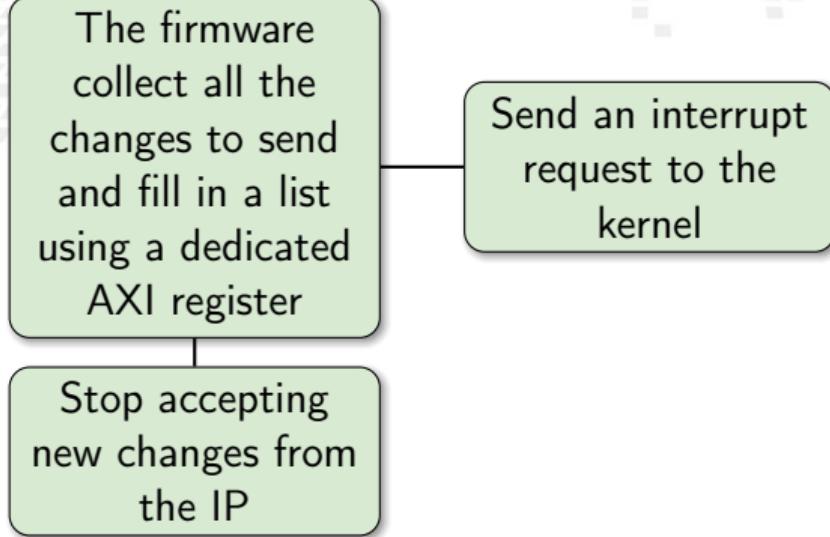
The firmware collect all the changes to send and fill in a list using a dedicated AXI register

Stop accepting new changes from the IP



# Firmware to kernel: IRQ

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



# Firmware to kernel: IRQ

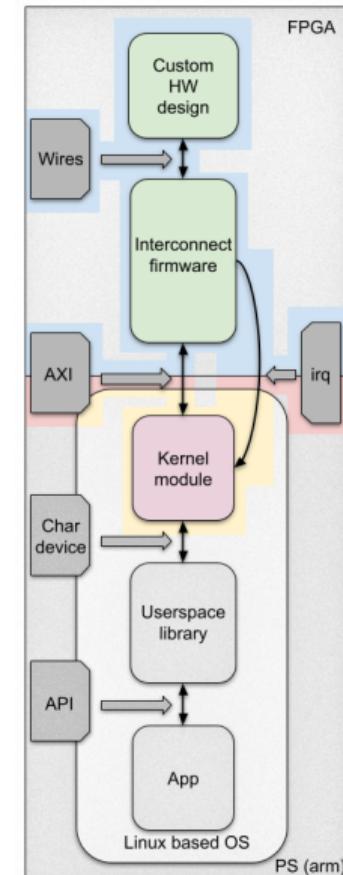
Different story is the data flow from the FPGA to the PS part.  
Data can easily flow so fast to saturate and make the PS part completely unusable.

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

Stop accepting new changes from the IP

Send an interrupt request to the kernel

```
#define IRQ_ID_04 44
// Interrupt Handler for IRQ 44
static irqreturn_t irq_handler(int irq, void *dev_id)
{
    struct work_data *work;
    if (work->state == stateCONNECT)
    {
        work = kalloc(sizeof(struct work_data), GFP_KERNEL);
        INIT_WORK(&work->work, work_handler);
        queue_work(kq, work);
    }
    return IRQ_HANDLED;
}
```



# Firmware to kernel: IRQ

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

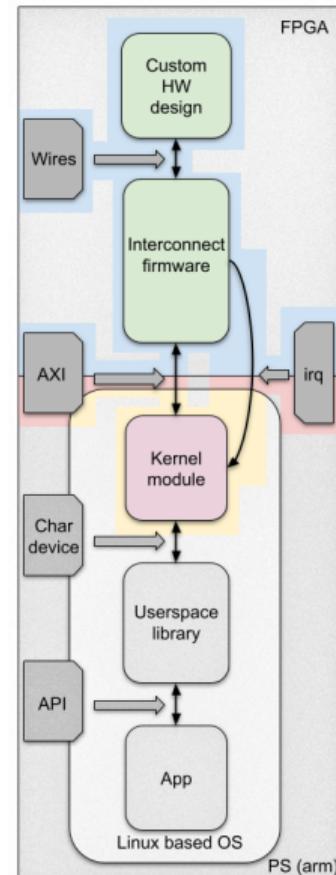
The firmware collect all the changes to send and fill in a list using a dedicated AXI register

Stop accepting new changes from the IP

Send an interrupt request to the kernel

```
interrupt dispositif
void IRQ_ID_45
{
    struct interrupt_header_t irq_header;
    static unsigned int irq_header_id_irq, void *dev_id;
    struct work_data *irqwork;
    if (beacon_state == stateCONNECT)
    {
        irqwork = malloc(sizeof(struct work_data), GFP_KERNEL);
        INIT_WORK(&irqwork->work, work_handler);
        irq_header_id_irq = irq_header_id;
        dev_id = dev;
    }
    return IRQ_HANDLED;
}
```

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



# Firmware to kernel: IRQ

Different story is the data flow from the FPGA to the PS part.

Data can easily flow so fast to saturate and make the PS part completely unusable.

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

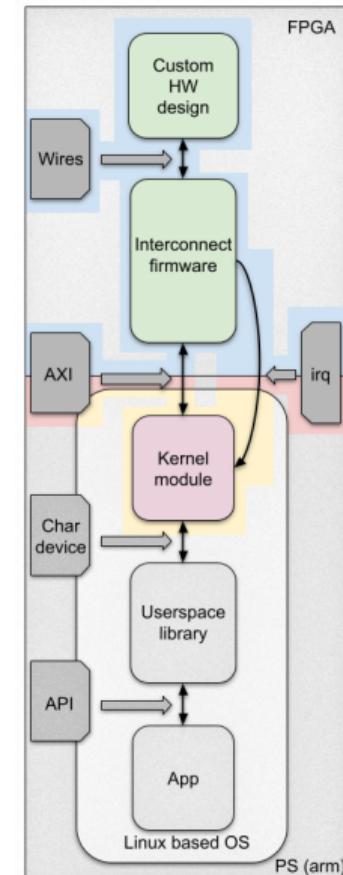
Send an interrupt request to the kernel

Stop accepting new changes from the IP

The kernel notify the firmware when done

```
interrupt handler
void IRQ_ID_45
{
    struct interrupt_header_t irq_header;
    static unsigned int irq_header_idl[irq_id];
    static unsigned int state;
    struct work_data *work;
    if (state == staticCONNECT)
    {
        work = malloc(sizeof(struct work_data));
        INIT_WORK(&work->work, work_handler);
        queue_work(system_wq, work);
        state = staticREADY;
    }
    return IRQ_HANDLED;
}
```

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



# Library

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

/dev/bm

BMAPI Library

The library functions can be used by the application

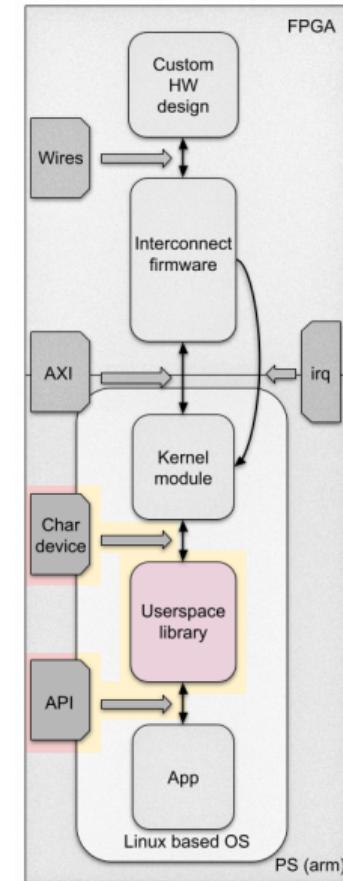
(\*BMAPI) BMr2owa

(\*BMAPI) BMr2ow

(\*BMAPI) BMr2o

(\*BMAPI) BMi2rw

(\*BMAPI) BMi2r

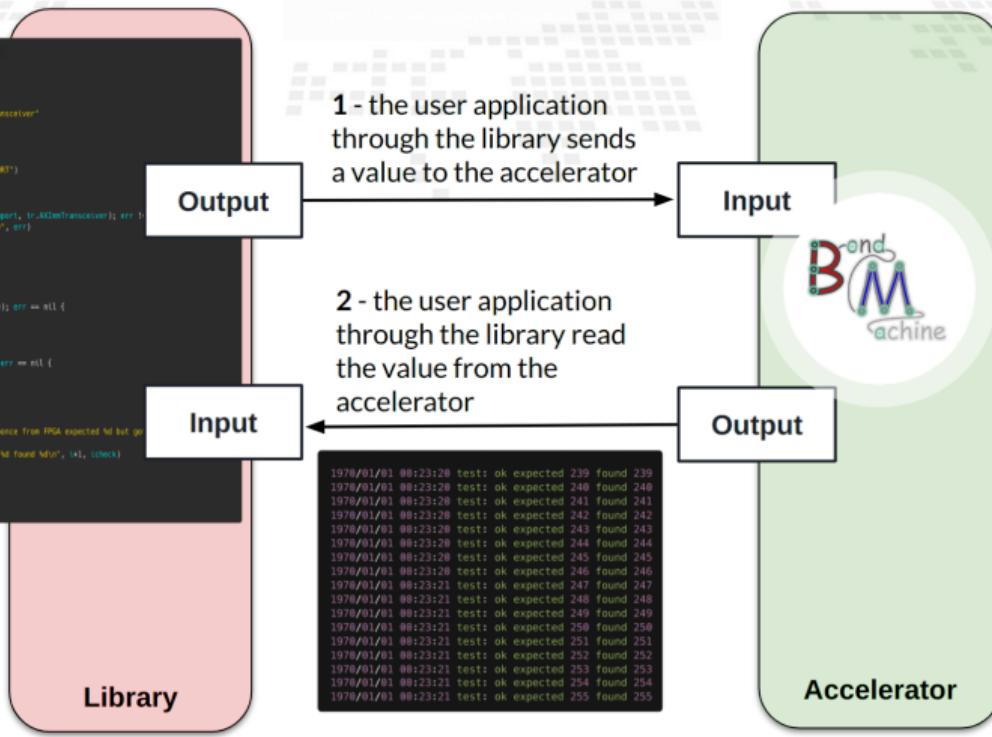


# Accelerated application: an example

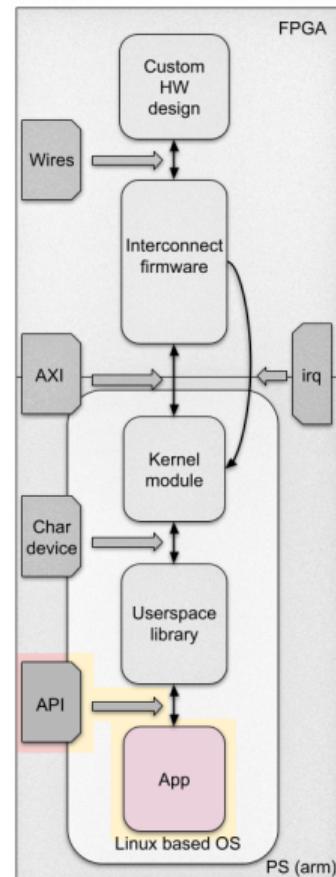
```
package main

import (
    "os"
    "tr "github.com/BordPech/Netlink/bmap/axi/transceiver"
    "github.com/BordPech/Netlink/fakebmap"
)

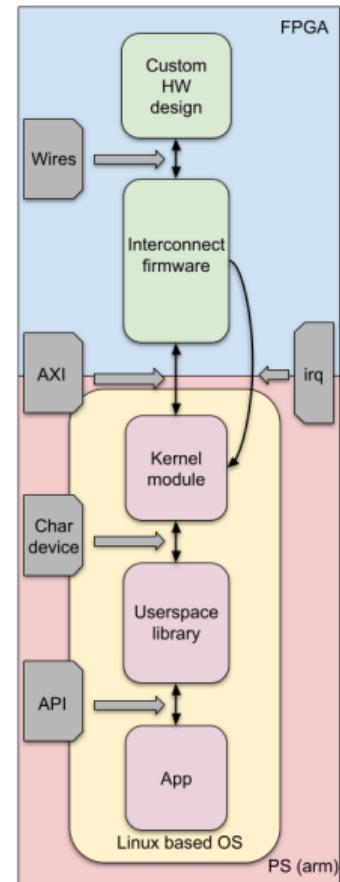
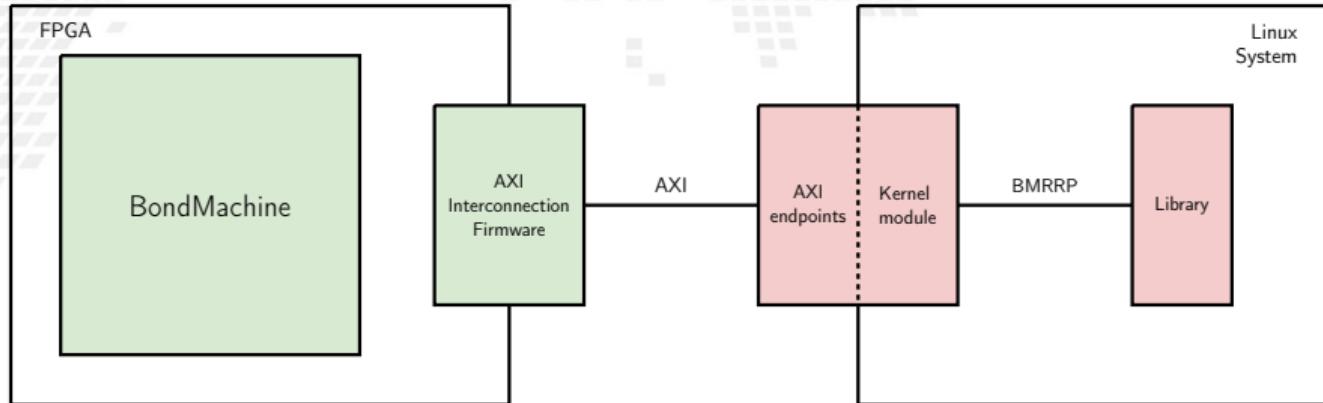
func main() {
    // ...
    if !present {
        report := os.LookupEnv("BNMPD4PORT")
        if !present {
            report = "/dev/bm"
        }
        if ba_err := fakebmap.AcceleratorInit(report, tr.AXImTransceiver); err != nil {
            log.Println("AcceleratorInit failed: ", err)
            return
        } else {
            ba.WaitConnection()
            var check uint8
            for i := 0; i < 256; i++ {
                for {
                    if err := ba.BM2c(0, uint8(i)); err == nil {
                        break
                    }
                }
                for {
                    if check, err := ba.BM2r(0); err == nil {
                        check = check
                        break
                    }
                }
                if slotB(i) != check {
                    log.Println("got the wrong response from FPGA expected %d but got %d", i, check)
                } else {
                    log.Println("test: ok expected %d found %d", i, check)
                }
            }
            ba.AcceleratorStop()
        }
    }
}
```



Firmware development for hybrid processors



# Accelerated Application



# Tests and Benchmarks

## 1 Introduction

Evolution of computing: new challenges  
Accelerators and FPGA  
BondMachine

## 2 An accelerated system from ground up

Hardware  
Software

## 3 Tests and Benchmarks

Tests  
Benchmark

## 4 Conclusions and Future directions

# An example

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

# Squared Matrix-vector multiplication

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

# Squared Matrix-vector multiplication

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

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

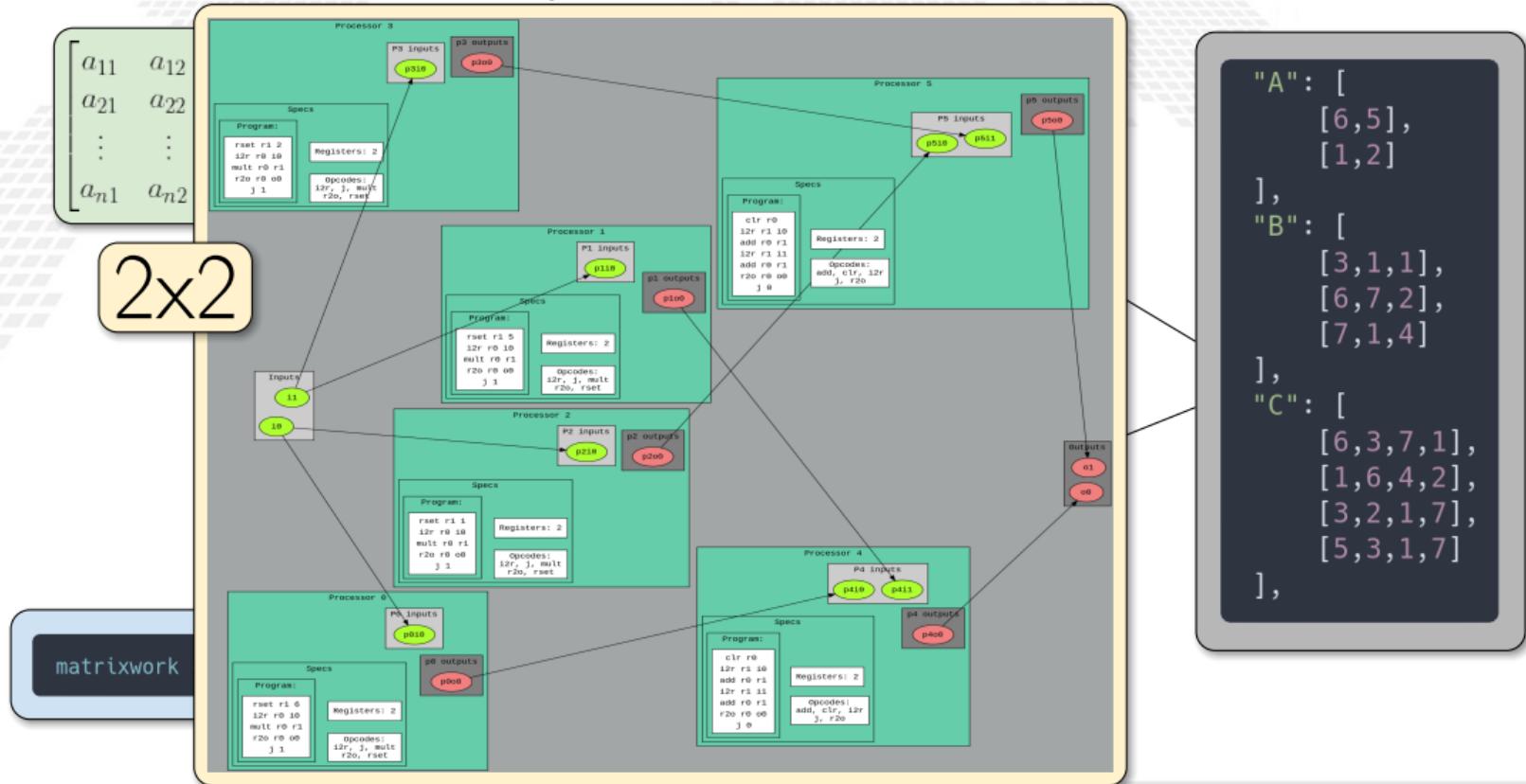
# Squared Matrix-vector multiplication

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

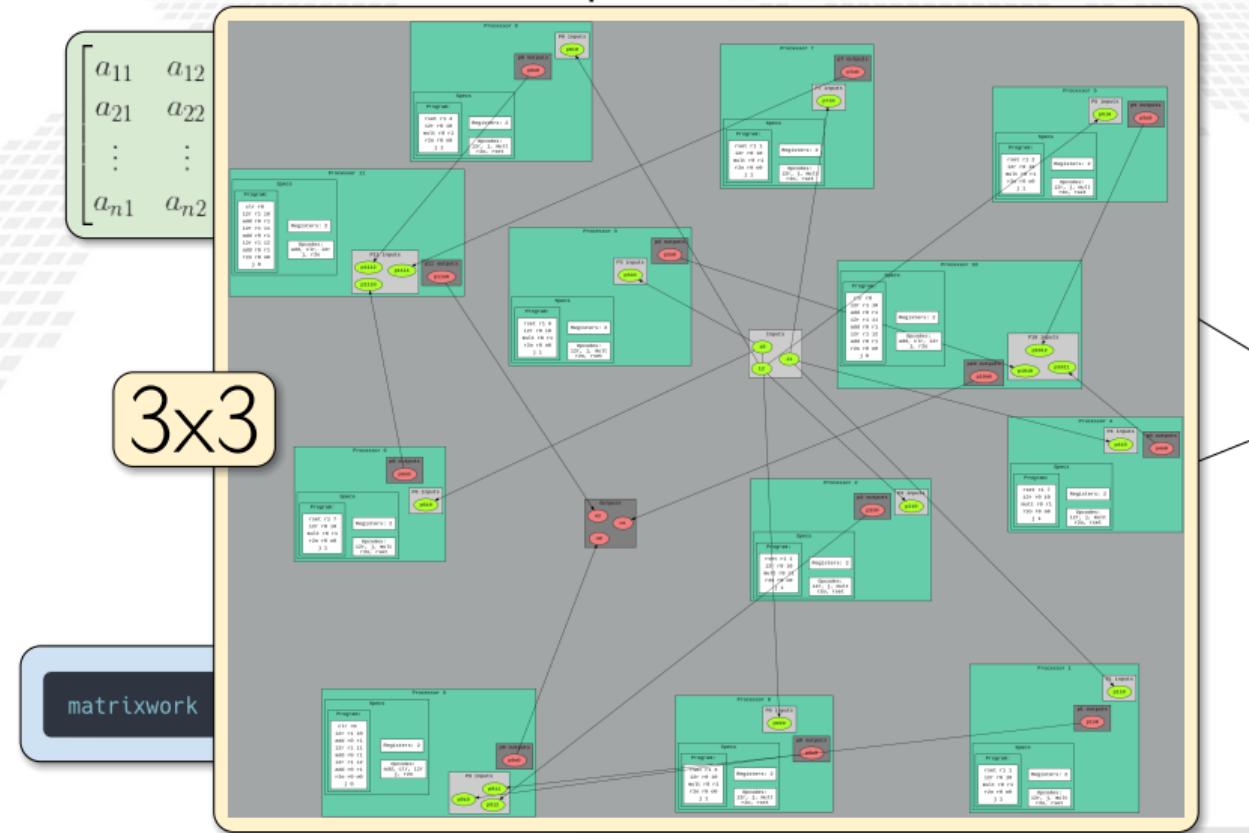
```
"A": [
    [6,5],
    [1,2]
],
"B": [
    [3,1,1],
    [6,7,2],
    [7,1,4]
],
"C": [
    [6,3,7,1],
    [1,6,4,2],
    [3,2,1,7],
    [5,3,1,7]
],
```

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

# Squared Matrix-vector multiplication



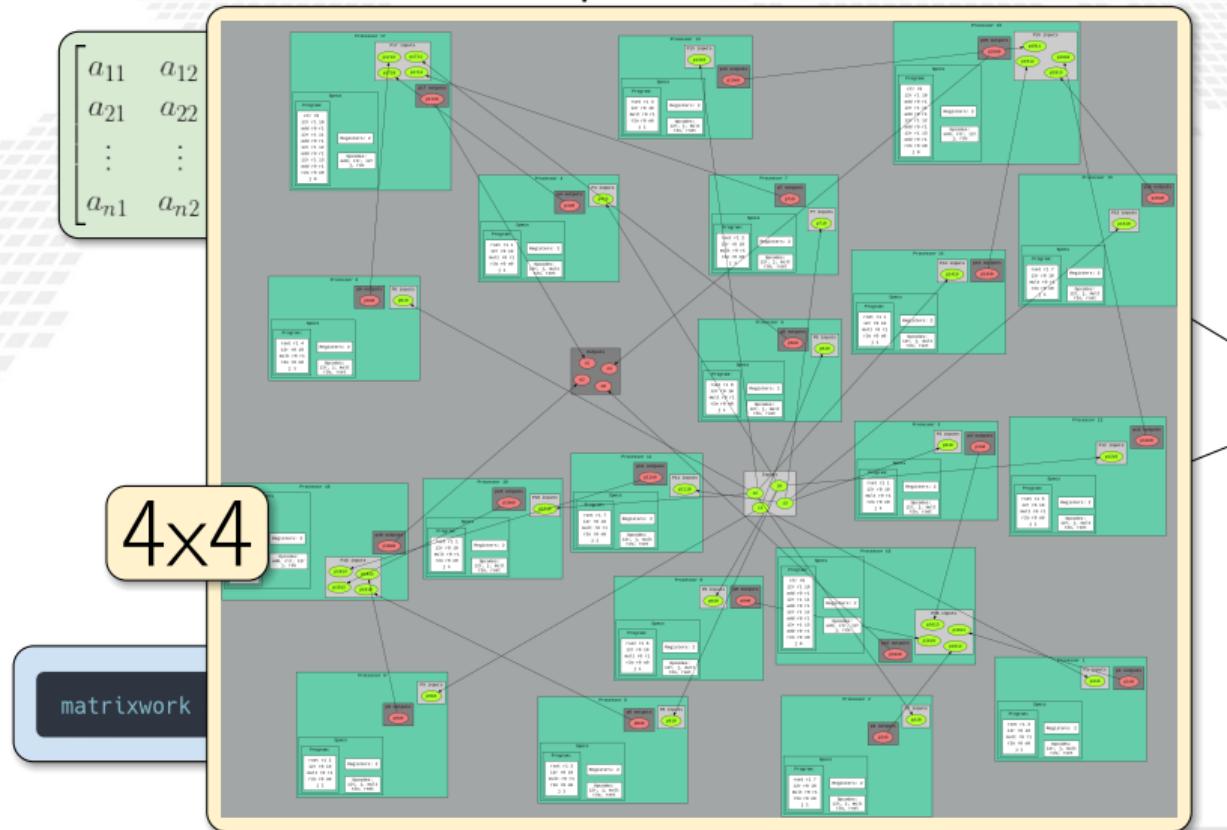
# Squared Matrix-vector multiplication



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

## Squared Matrix-vector multiplication

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



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

# Correctness and module debug

To verify the correct computation of the accelerator:

- a tool to monitor the AXI memory

```
# ./monitor -g @x43c00000 -n 8
i0: 00000000 (0x43c00003) 00000000 (0x43c00002) 00000000 (0x43c00001) 11111010 (0x43c00000)
i1: 00000000 (0x43c00007) 00000000 (0x43c00006) 00000000 (0x43c00005) 00000000 (0x43c00004)
i2: 00000000 (0x43c0000b) 00000000 (0x43c0000a) 00000000 (0x43c00009) 00000000 (0x43c00008)
i3: 00000000 (0x43c0000f) 00000000 (0x43c0000e) 00000000 (0x43c0000d) 00000000 (0x43c0000c)
i4: 00000000 (0x43c00013) 00000000 (0x43c00012) 00000000 (0x43c00011) 00000000 (0x43c00010)
i5: 00000000 (0x43c00017) 00000000 (0x43c00016) 00000000 (0x43c00015) 00000000 (0x43c00014)
i6: 00000000 (0x43c0001b) 00000000 (0x43c0001a) 00000000 (0x43c00019) 00000000 (0x43c00018)
i7: 00000000 (0x43c0001f) 00000000 (0x43c0001e) 00000000 (0x43c0001d) 00000000 (0x43c0001c)
PS2PL: 00000000 (0x43c00023) 00000000 (0x43c00022) 00000000 (0x43c00021) 00000000 (0x43c00020)
STATES: 00000000 (0x43c00027) 00000000 (0x43c00026) 00000000 (0x43c00025) 00000000 (0x43c00024)
o0: 00000000 (0x43c0002b) 00000000 (0x43c0002a) 00000000 (0x43c00029) 11011100 (0x43c00028)
o1: 00000000 (0x43c0002f) 00000000 (0x43c0002e) 00000000 (0x43c0002d) 11101110 (0x43c0002c)
o2: 00000000 (0x43c00033) 00000000 (0x43c00032) 00000000 (0x43c00031) 11011100 (0x43c00030)
o3: 00000000 (0x43c00037) 00000000 (0x43c00036) 00000000 (0x43c00035) 11101000 (0x43c00034)
o4: 00000000 (0x43c0003b) 00000000 (0x43c0003a) 00000000 (0x43c00039) 11011100 (0x43c00038)
o5: 00000000 (0x43c0003f) 00000000 (0x43c0003e) 00000000 (0x43c0003d) 11000010 (0x43c0003c)
o6: 00000000 (0x43c00043) 00000000 (0x43c00042) 00000000 (0x43c00041) 11101000 (0x43c00040)
o7: 00000000 (0x43c00047) 00000000 (0x43c00046) 00000000 (0x43c00045) 11011100 (0x43c00044)
bench: 00000000 (0x43c0004b) 00000000 (0x43c0004a) 00000000 (0x43c00049) 00011101 (0x43c00048)
PL2PS: 00000000 (0x43c0004f) 11111111 (0x43c0004e) 10000000 (0x43c0004d) 00000000 (0x43c0004c)
CHANGE: 00000000 (0x43c00053) 11111111 (0x43c00052) 11111111 (0x43c00051) 11000000 (0x43c00050)
```

# Correctness and module debug

To verify the correct computation of the accelerator:

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

```
# ./monitor -g 0x43c00000 -n 8
i0: 00000000 (0x43c00003) 00000000 (0x43c00002) 00000000 (0x43c00001) 11111010 (0x43c00000)
i1: 00000000 (0x43c00007) 00000000 (0x43c00006) 00000000 (0x43c00005) 00000000 (0x43c00004)
i2: 00000000 (0x43c0000b) 00000000 (0x43c0000a) 00000000 (0x43c00009) 00000000 (0x43c00008)
i3: 00000000 (0x43c0000f) 00000000 (0x43c0000e) 00000000 (0x43c0000d) 00000000 (0x43c0000c)
i4: 00000000 (0x43c00013) 00000000 (0x43c00012) 00000000 (0x43c00011) 00000000 (0x43c00010)
i5: 00000000 (0x43c00017) 00000000 (0x43c00016) 00000000 (0x43c00015) 00000000 (0x43c00014)
i6: 00000000 (0x43c0001b) 00000000 (0x43c0001a) 00000000 (0x43c00019) 00000000 (0x43c00018)
i7: 00000000 (0x43c0001f) 00000000 (0x43c0001e) 00000000 (0x43c0001d) 00000000 (0x43c0001c)
PS2PL: 00000000 (0x43c00023) 00000000 (0x43c00022) 00000000 (0x43c00021) 00000000 (0x43c00020)
STATES: 00000000 (0x43c00027) 00000000 (0x43c00026) 00000000 (0x43c00025) 00000000 (0x43c00024)
o0: 00000000 (0x43c0002b) 00000000 (0x43c0002a) 00000000 (0x43c00029) 11011100 (0x43c00028)
o1: 00000000 (0x43c0002f) 00000000 (0x43c0002e) 00000000 (0x43c0002d) 11011100 (0x43c0002c)
o2: 00000000 (0x43c00033) 00000000 (0x43c00032) 00000000 (0x43c00031) 11011100 (0x43c00030)
o3: 00000000 (0x43c00037) 00000000 (0x43c00036) 00000000 (0x43c00035) 11011100 (0x43c00034)
o4: 00000000 (0x43c0003b) 00000000 (0x43c0003a) 00000000 (0x43c00039) 11011100 (0x43c00038)
o5: 00000000 (0x43c0003f) 00000000 (0x43c0003e) 00000000 (0x43c0003d) 11000010 (0x43c0003c)
o6: 00000000 (0x43c00043) 00000000 (0x43c00042) 00000000 (0x43c00041) 11101000 (0x43c00040)
o7: 00000000 (0x43c00047) 00000000 (0x43c00046) 00000000 (0x43c00045) 11011100 (0x43c00044)
bench: 00000000 (0x43c0004b) 00000000 (0x43c0004a) 00000000 (0x43c00049) 00011101 (0x43c00048)
PL2PS: 00000000 (0x43c0004f) 11111111 (0x43c0004e) 10000000 (0x43c0004d) 00000000 (0x43c0004c)
CHANGE: 00000000 (0x43c00053) 11111111 (0x43c00052) 11111111 (0x43c00051) 11000000 (0x43c00050)
```

```
devmem 0x43c00000 b 1
```

# Correctness and module debug

To verify the correct computation of the accelerator:

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

```
# ./monitor -g 0x43c00000 -n 8
i0: 00000000 (0x43c00003) 00000000 (0x43c00002) 00000000 (0x43c00001) 11111010 (0x43c00000)
i1: 00000000 (0x43c00007) 00000000 (0x43c00006) 00000000 (0x43c00005) 00000000 (0x43c00004)
i2: 00000000 (0x43c0000b) 00000000 (0x43c0000a) 00000000 (0x43c00009) 00000000 (0x43c00008)
i3: 00000000 (0x43c0000f) 00000000 (0x43c0000e) 00000000 (0x43c0000d) 00000000 (0x43c0000c)
i4: 00000000 (0x43c00013) 00000000 (0x43c00012) 00000000 (0x43c00011) 00000000 (0x43c00010)
i5: 00000000 (0x43c00017) 00000000 (0x43c00016) 00000000 (0x43c00015) 00000000 (0x43c00014)
i6: 00000000 (0x43c0001b) 00000000 (0x43c0001a) 00000000 (0x43c00019) 00000000 (0x43c00018)
i7: 00000000 (0x43c0001f) 00000000 (0x43c0001e) 00000000 (0x43c0001d) 00000000 (0x43c0001c)
PS2PL: 00000000 (0x43c00023) 00000000 (0x43c00022) 00000000 (0x43c00021) 00000000 (0x43c00020)
STATES: 00000000 (0x43c00027) 00000000 (0x43c00026) 00000000 (0x43c00025) 00000000 (0x43c00024)
o0: 00000000 (0x43c0002b) 00000000 (0x43c0002a) 00000000 (0x43c00029) 11011100 (0x43c00028)
o1: 00000000 (0x43c0002f) 00000000 (0x43c0002e) 00000000 (0x43c0002d) 11011110 (0x43c0002c)
o2: 00000000 (0x43c00033) 00000000 (0x43c00032) 00000000 (0x43c00031) 11011100 (0x43c00030)
o3: 00000000 (0x43c00037) 00000000 (0x43c00036) 00000000 (0x43c00035) 11011000 (0x43c00034)
o4: 00000000 (0x43c0003b) 00000000 (0x43c0003a) 00000000 (0x43c00039) 11011100 (0x43c00038)
o5: 00000000 (0x43c0003f) 00000000 (0x43c0003e) 00000000 (0x43c0003d) 11000100 (0x43c0003c)
o6: 00000000 (0x43c00043) 00000000 (0x43c00042) 00000000 (0x43c00041) 11101000 (0x43c00040)
o7: 00000000 (0x43c00047) 00000000 (0x43c00046) 00000000 (0x43c00045) 11011100 (0x43c00044)
bench: 00000000 (0x43c0004b) 00000000 (0x43c0004a) 00000000 (0x43c00049) 00011101 (0x43c00048)
PL2PS: 00000000 (0x43c0004f) 11111111 (0x43c0004e) 10000000 (0x43c0004d) 00000000 (0x43c0004c)
CHANGE: 00000000 (0x43c00053) 11111111 (0x43c00052) 11111111 (0x43c00051) 11000000 (0x43c00050)
```

```
devmem 0x43c00000 b 1
```

# An example of error

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

# An example of error

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

	7	7
o0	7	7
o1	6	6
o10	6	
o11	4	
o12	1	
o13	52	
o2	2	2
o3	2	2
o4	4	4
o5	3	3
o6	2	2
o7	6	6
o8	3	3
o9	5	5
o10	6	
o11	4	
o12	1	
o13	52	

## Benchmark: caveats

This is a preliminary work.

We trust some tools:

- Vivado reports
- perf

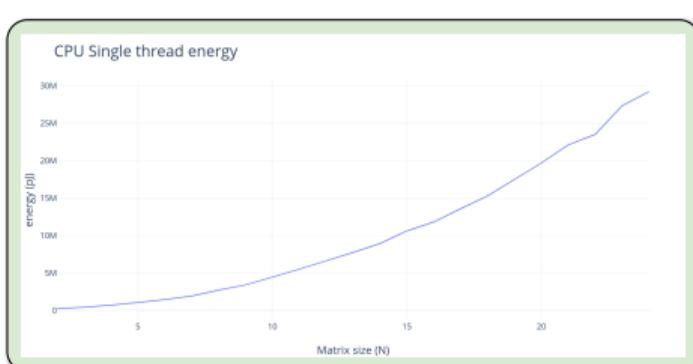
The FPGA benchmarks do not include the PS part overhead (the comparisons are not really fair)

# Benchmark: the CPU (Golang)

```
func matrixtest(n int, iter int64) float32 {  
    ...  
    start := time.Now()  
  
    for k := 0; int64(k) < iter; k++ {  
        for l := 0; l < n; l++ {  
            output[l] = uint64(0)  
        }  
  
        for l := 0; l < n; l++ {  
            for j := 0; j < n; j++ {  
                output[l] += input[j] * matrix[(i+j)*n]  
            }  
        }  
    }  
    return float32(time.Since(start).Microseconds()) / float32(iter)  
}  
func main() {  
    for i := 2; i <= 32; i++ {  
        fmt.Println(i, matrixtest(i, 100000000))  
    }  
}
```

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

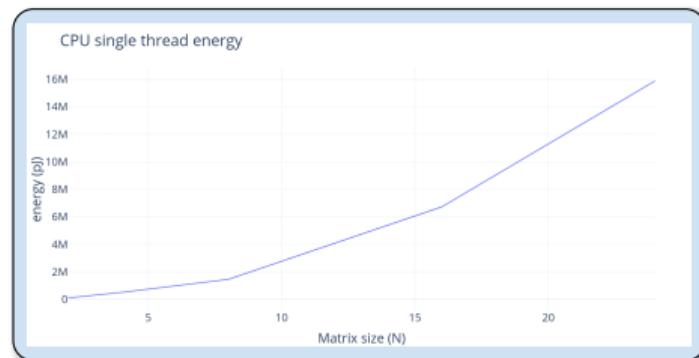
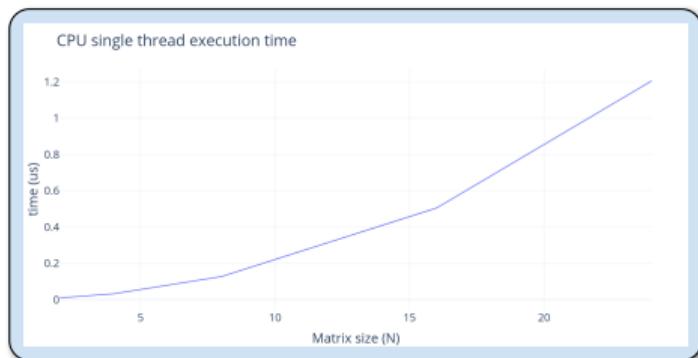
N	single thread (ns)	single thread (J)	energy eff.
1	0.00040000	200000	3.0000E-06
2	0.01415000	404000	2.3501E-06
3	0.02998000	732000	1.3940E-06
4	0.05612000	1474000	9.3420E-07
5	0.08215000	2414000	6.7000E-07
6	0.10818000	3414000	5.1000E-07
7	0.07598000	1898000	9.2000E-07
8	0.09697000	2778000	3.0104E-07
9	0.12207000	3428000	2.1184E-07
10	0.14600000	4488000	2.2399E-07
11	0.20017000	5888000	1.8802E-07
12	0.24050000	6848000	1.5503E-07
13	0.28864000	7768000	1.2003E-07
14	0.33440000	8994000	1.1000E-07
15	0.38011700	10830000	9.4004E-08
16	0.43050000	11830000	8.4303E-08
17	0.50040000	13604000	7.0004E-08
18	0.59870000	15124000	6.5500E-08
19	0.69310000	17504000	5.7000E-08
20	0.790104	18718100	5.1720E-08
21	0.7985200	22139800	4.5179E-08
22	0.8000005	23525300	4.2550E-08
23	0.80407200	27548700	3.8037E-08
24	1.3011791	29393000	5.4299E-08



# Benchmark: the CPU (C)

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

n	single op energy (pJ)	single op time (ns)	energy eff
1	100000	0.01	0.000000333333333
2	500000	0.033	0.000002702702703
3	1490000	0.127	0.0000009524861878
4	6720000	0.505	0.0000001326259947
5	15840000	1.205	0.0000000854009596

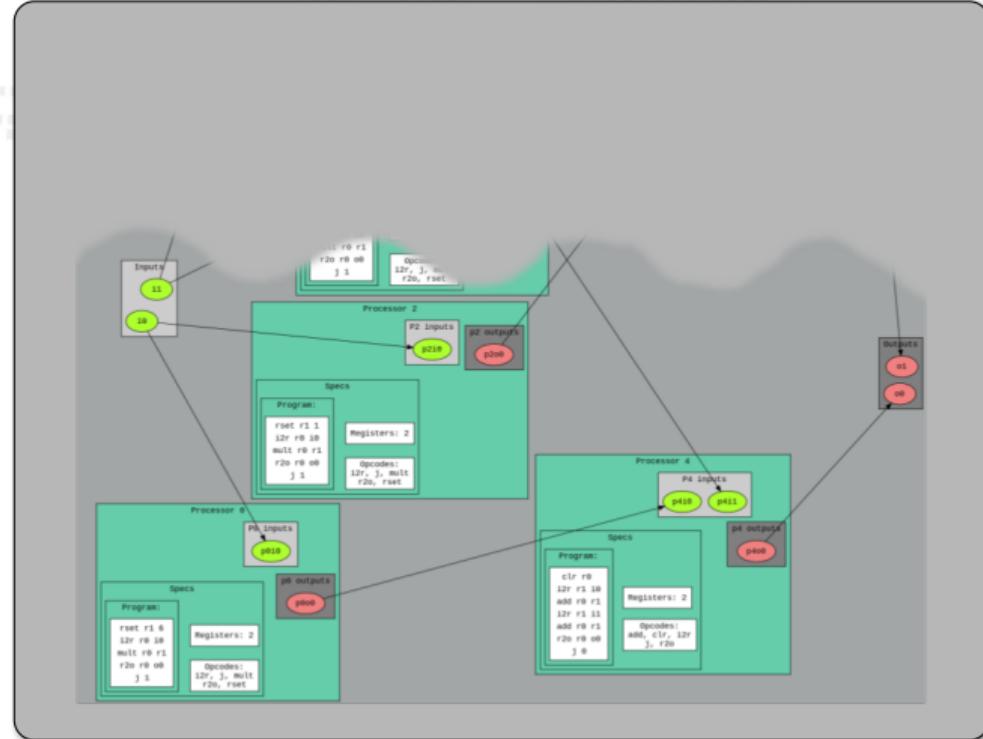


# Benchmark: the FPGA

Benchmark an IP is not an easy task.

Fortunately we have a custom design and an FPGA.

We can put the benchmarks tool inside the accelerator.

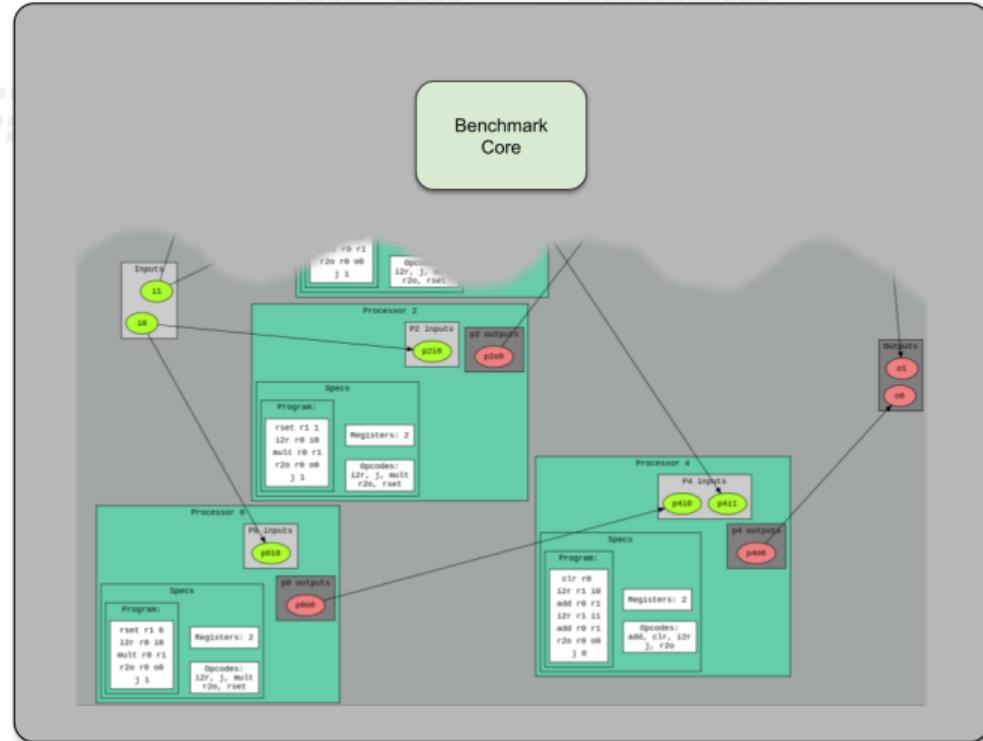


# Benchmark: the FPGA

Benchmark an IP is not an easy task.

Fortunately we have a custom design and an FPGA.

We can put the benchmarks tool inside the accelerator.

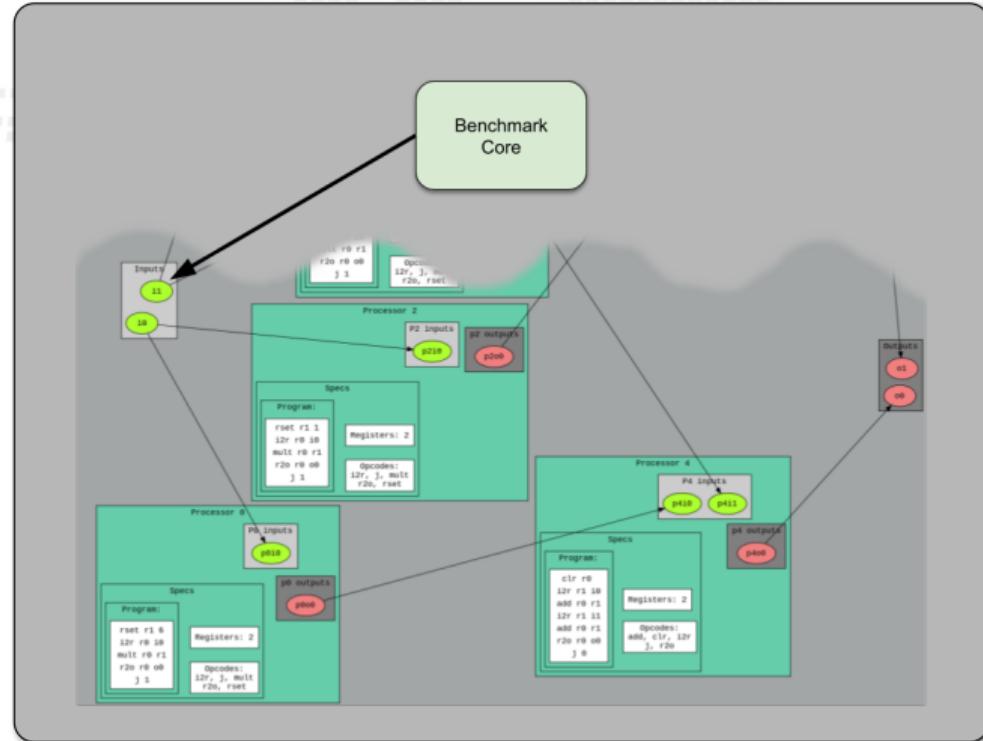


# Benchmark: the FPGA

Benchmark an IP is not an easy task.

Fortunately we have a custom design and an FPGA.

We can put the benchmarks tool inside the accelerator.

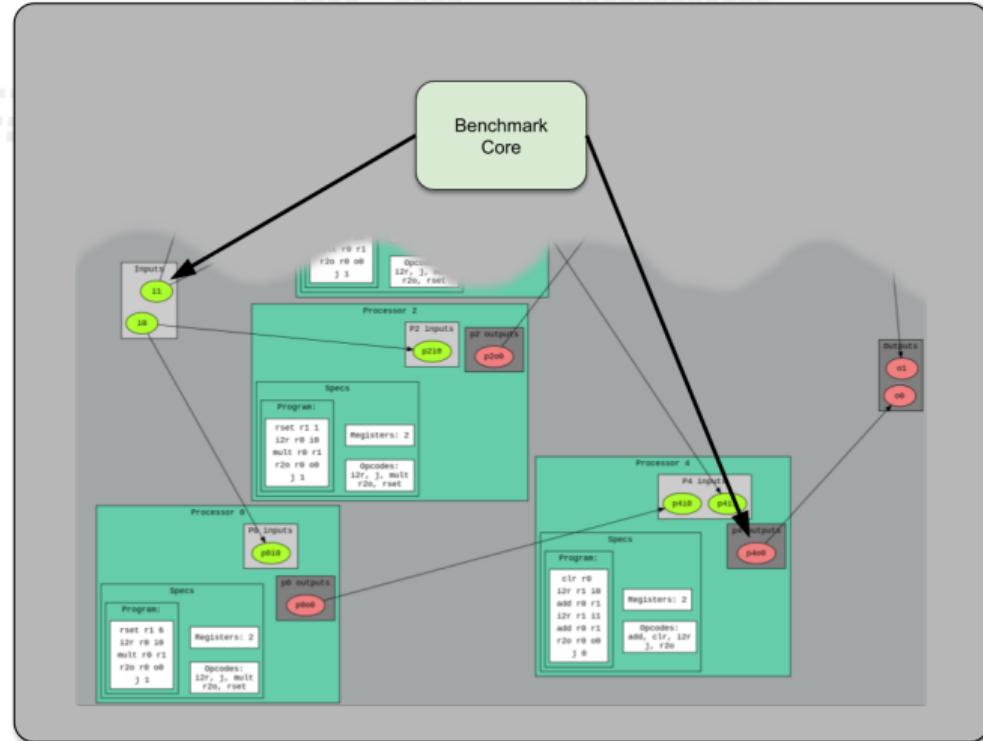


# Benchmark: the FPGA

Benchmark an IP is not an easy task.

Fortunately we have a custom design and an FPGA.

We can put the benchmarks tool inside the accelerator.

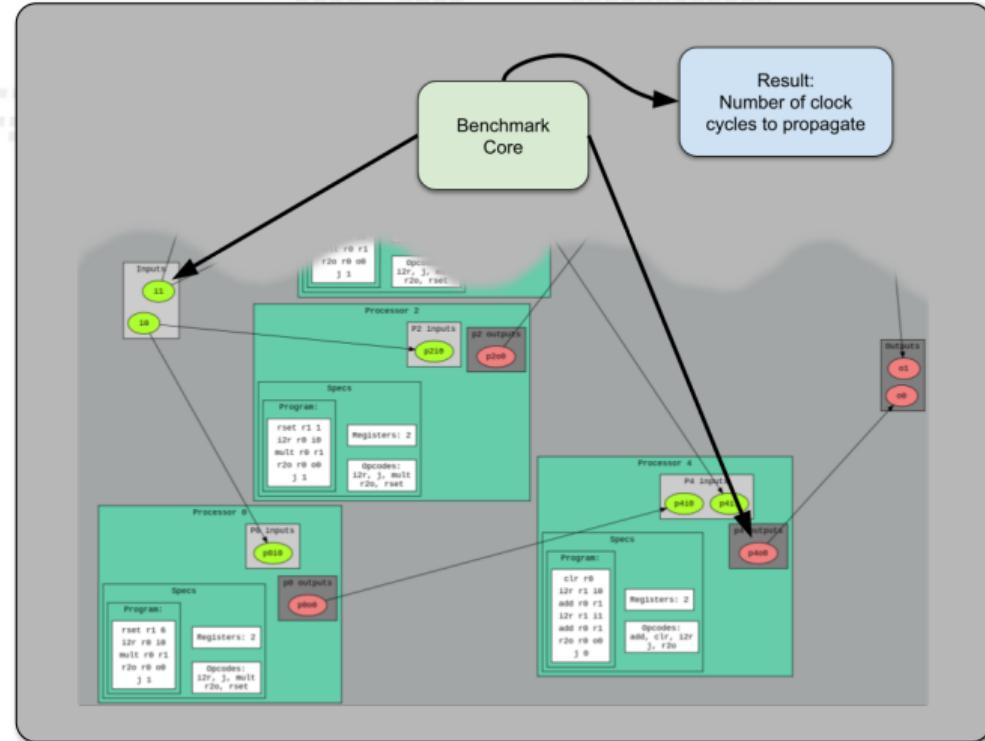


# Benchmark: the FPGA

Benchmark an IP is not an easy task.

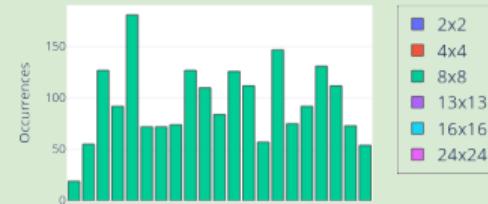
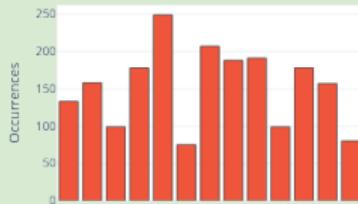
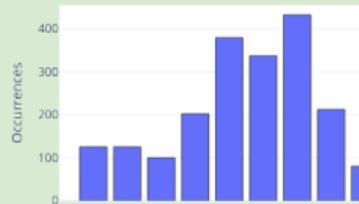
Fortunately we have a custom design and an FPGA.

We can put the benchmarks tool inside the accelerator.



# Benchmark core clock cycles distributions

Clock cycles distributions

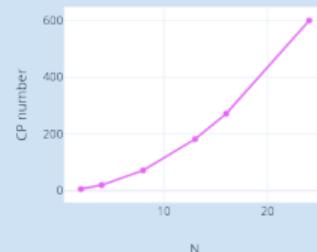
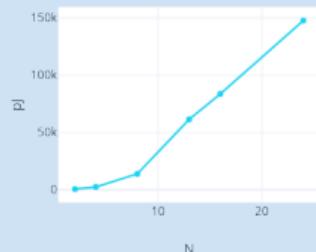
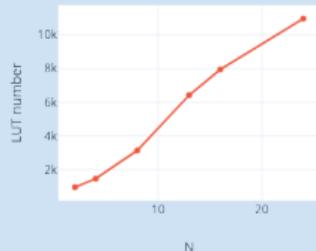


# FPGA benchmark summary

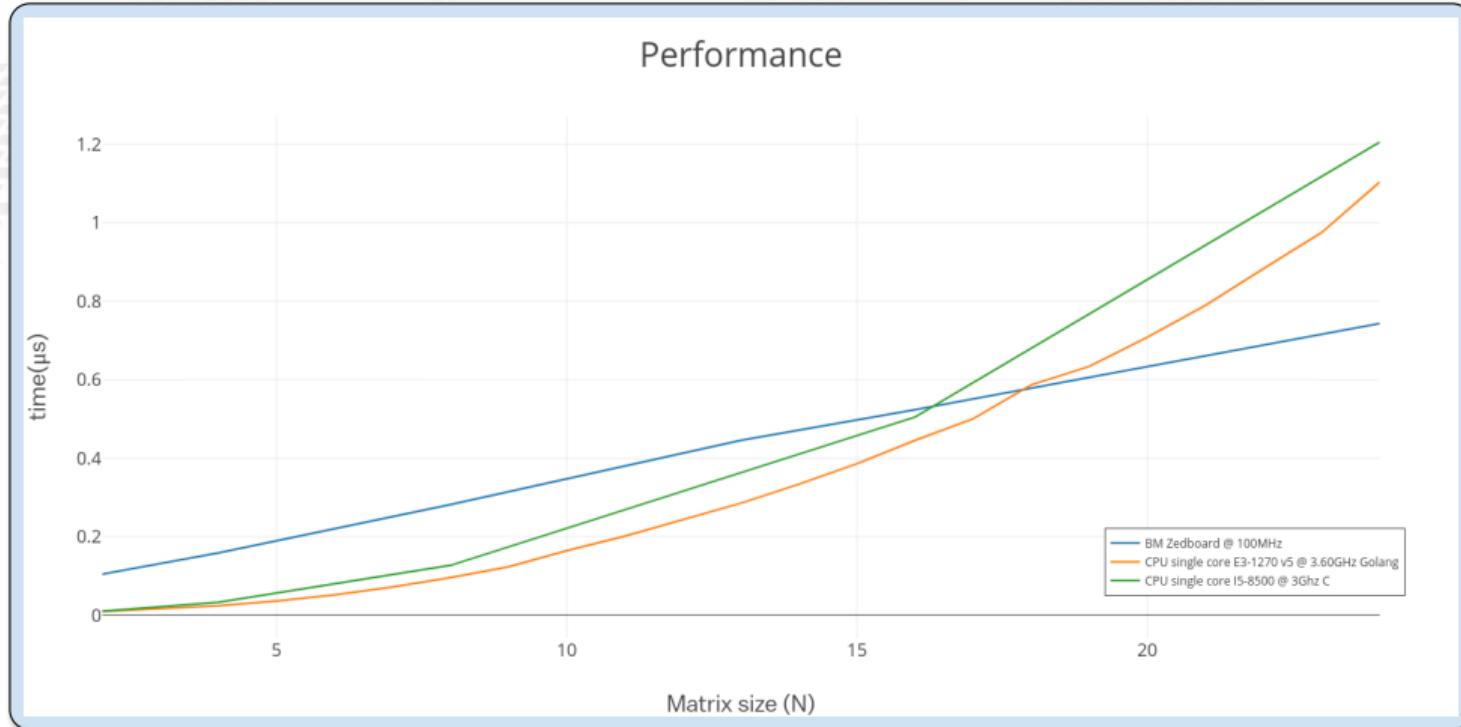
	N	single op time (us)	Register LUTs	Slice LUTs	Power	single op energy (pJ)	CPs
1	2	0.1044	947	875	0.005	522	6
2	4	0.1587	1457	1813	0.015	2380.5	20
3	8	0.2819	3131	4897	0.049	13813.1	72
4	13	0.4456	6422	12819	0.138	61492.8	182
5	16	0.5234	7950	15979	0.160	83744	272
6	24	0.7432	10974	22669	0.199	147896.8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

# Benchmark core

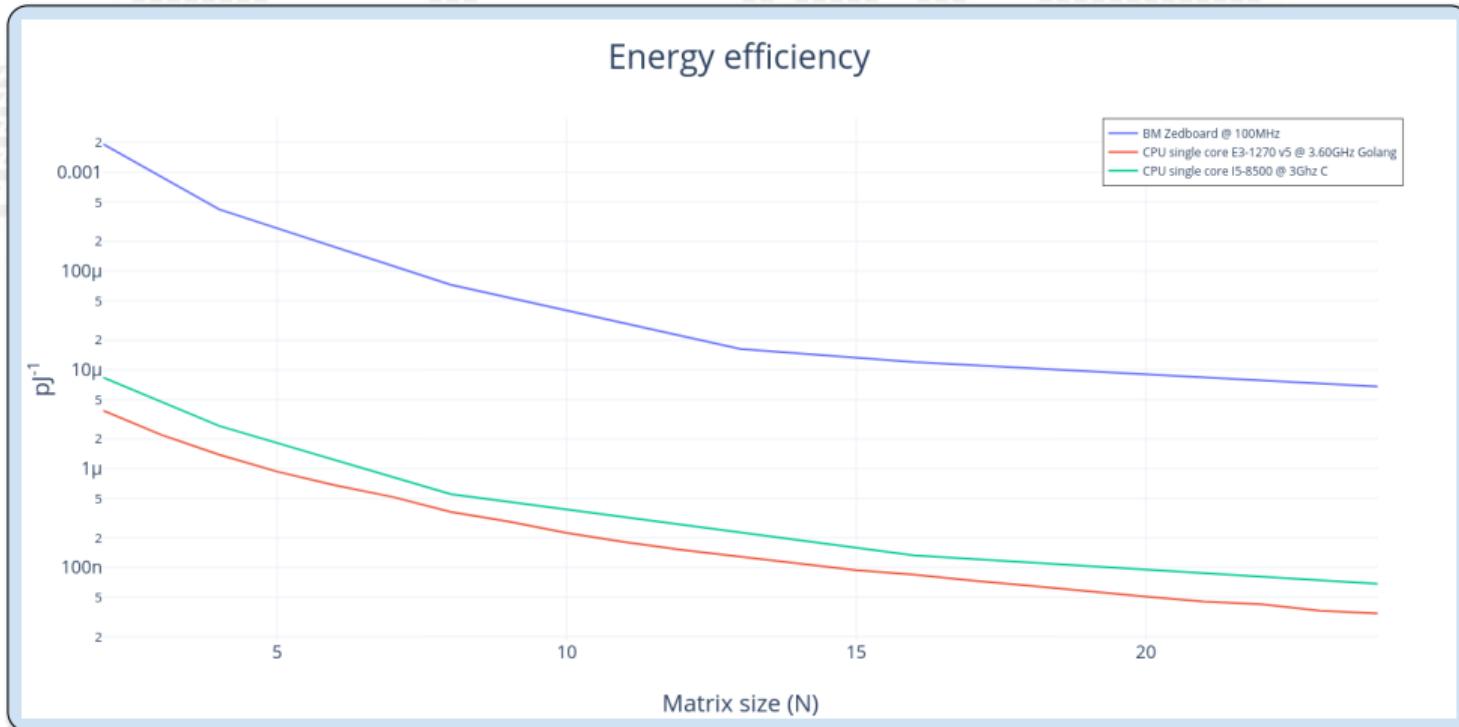
## BondMachine NxN matrix-vector multiplication



# Comparisons: Performance



# Comparisons: Energy



# Conclusions and Future directions

## 1 Introduction

Evolution of computing: new challenges  
Accelerators and FPGA  
BondMachine

## 2 An accelerated system from ground up

Hardware  
Software

## 3 Tests and Benchmarks

Tests  
Benchmark

## 4 Conclusions and Future directions

# Conclusions

- The creation of a firmware from ground up is not a mere exercise. It gives perspective on how heterogeneous system really works and what really is an FPGA accelerator
- Even if the methodology and the tools were specifically created for the BondMachine project, they are sufficiently general to be applicable to other FPGA accelerators as well
- FPGA is a groundbreaking technology but require a change of perspective in how we develop software

# Future directions

We plan to extend the benchmarks to:

- different data types
- different boards
- compare with GPUs
- include some real power consumption measures

For the project:

- First DAQ use case
- Complete the inclusion of Intel and Lattice FPGAs and try a more performant Zynq based board
- Accelerator in a cloud workflow

Thank you

Thank you

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

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

parallel computing paper: link

contact email: [mirko.mariotti@unipg.it](mailto:mirko.mariotti@unipg.it)