



A. D. 1308
unipg

DEPARTMENT
OF PHYSICS AND GEOLOGY



Firmware development for hybrid processors (ARM and FPGA) computing

Mirko Mariotti ^{1,2} Giulio Bianchini ¹ Lorianò Storchi ^{3,2} Giacomo Surace ¹
Daniele Spiga ²

¹Dipartimento di Fisica e Geologia, Università degli Studi di Perugia

²INFN sezione di Perugia

³Dipartimento di Farmacia, Università 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

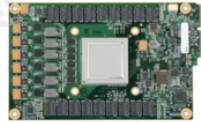
Energy
Efficient
Computing

Data-oriented
Computing

Edge 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", 6dev, 0, 0, platform.rows, platform.cols, E_FALSE);  
e_load_group("e_math_test1.srec", 6dev, 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"  
{  
    int index = get_global_id(0); \n"  
    C[index] = alpha * A[index] + B[index]; \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.

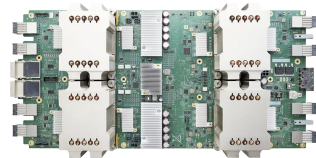
Graphics Processing Unit (GPU)

- High Data Throughput
- Massive Parallel Computing



Application-Specific Integrated Circuit (ASIC)

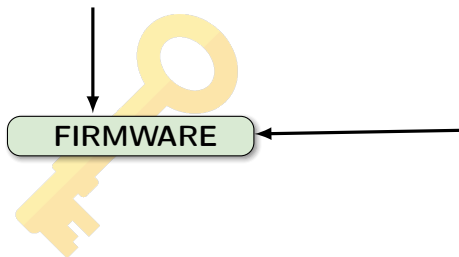
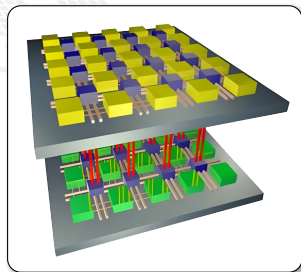
- Highly specialized for a task
- Energy efficient (due to specialization)



FPGA accelerator

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

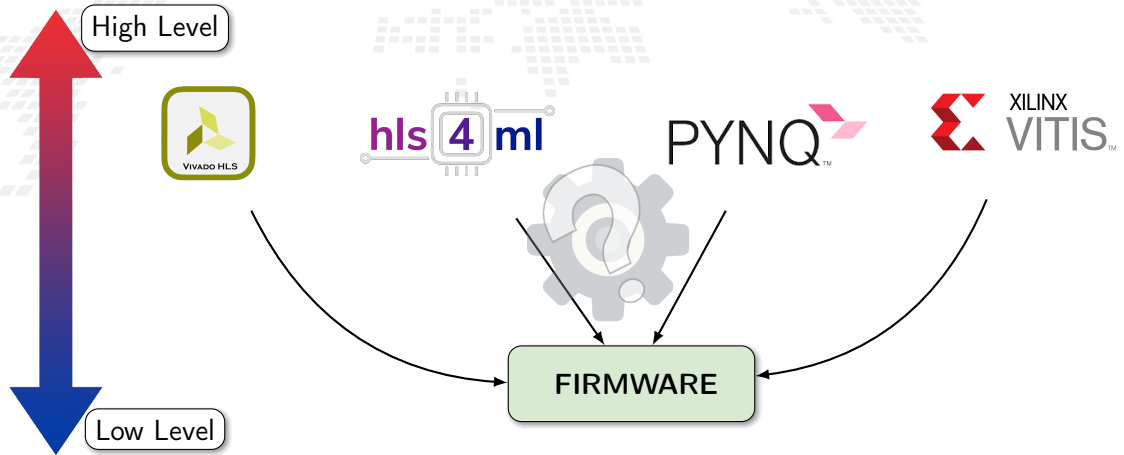
- Parallel computing
- Highly specialized
- Energy efficient



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

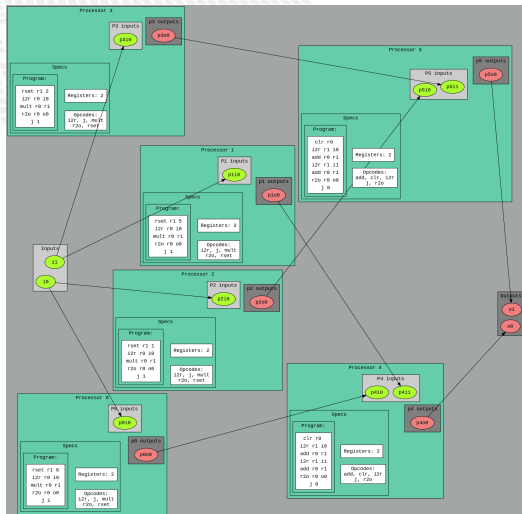
Firmware generation

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



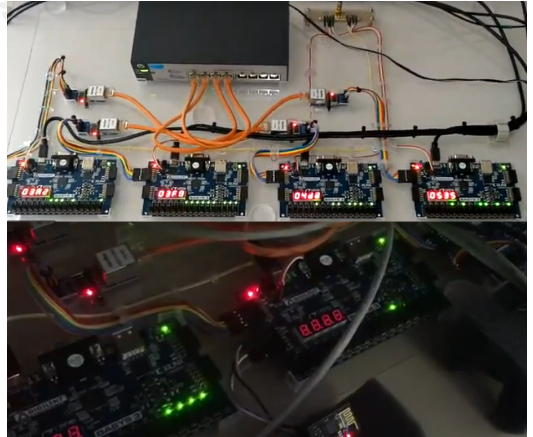
BondMachine

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

CCR 2015 First ideas, 2016 Poster, 2017 Talk

The BondMachine, a mouldable computer architecture

Mirko Mariotti¹, Daniel Magalotti²

¹Department of Physics and Geology, University of Perugia
²Istituto Nazionale di Fisica Nucleare, Sezione di Perugia

Introduction

The BondMachine (BM) is a new computer architecture where many Connecting Processors (CP) with different Instruction Set Architecture (ISA) are connected together and share resources to solve a heterogeneous problem perfectly fitted to a specific computational problem. These cores are implemented with the characteristic to be as minimal as possible and as simple as possible, and the capacity of solving problems vary mainly in how they are interconnected.

A BondMachine architecture can be given by using evolutionary algorithms that select the architecture, generate the hardware, and solve the problem in order to satisfy and improve the power processing. The BondMachine is implemented by using the Field Programmable Gate Array (FPGA) chips, but are today most powerful implementations of reconfigurable hardware. Moreover, the "regular machine" abstraction has been kept in order to use many well known tools and techniques ranging from languages to compilers.

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 to simulate the Internet of Things (IoT), Cyber Physical System (CPS) and High Performance Computing (HPC).

The BondMachine architecture

The BondMachine architecture consists of interconnections among Connecting Processors and Shared Objects (SOs) build to implement a dedicated tasks. The main features of this kind of architecture are the possibility to configure:

- the number of processor cores and their types;
- the number of inputs and outputs;
- 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 arbitrary connection topology within the BondMachine. They can have different register number, instruction set, registers with respect to the other ones.

Shared Object

Any kind of component can be **shared** among CPs. Shared Objects increase the processing capability and the functionality of the BM improving the high-speed **synchronization** and **communication** between tasks running on separate CPs.

A complete example of the BondMachine architecture. It consists of two CPs and two SOs interconnected between the interconnect registers of the processors. One processor, Owner and Worker, are connected among the processors.

Software Tools

The complexity of programming the BondMachine architecture is managed by using a set of software tools:

- build a specific architecture as function of the task;
- modify the created architecture;
- simulate the behaviour and to check the functionality with the aim to generate the Register Transfer Level (RTL) code for FPGA application.

Processor Builder selects the CP specific, assembles and disassembles, saves on disk as JSON, emulates and creates the RTL code.

BondMachine Builder connects CPs and SOs together in custom topologies, builds and saves on disk as JSON, 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 builders is synthesized for the FPGA KEXC10K (Kintex-10K) evaluation card to measure the **performance** of the architecture: logic resources, power consumption and synthesis time frequency clock.

The architecture consists of a channel shared by two CPs.

This basic element has been replicated by using the high resources used by each architecture because there is a number of CP.

The FPGA can contain up to 256 CPs with a clock frequency of 200 MHz and a power consumption of 0.13 W.

The performance of the architecture has been compared with the old ones. A benchmark has been used to measure the time per operation needed by the architecture to complete the task.

The different performance of the architecture are:

- The time per operation increases linearly for the CPs, due to the fact that the number of emulation CPs does not increase.
- The time per operation is constant for the FPGA due to its **infinite parallelism** due to the fact of the available logic resources.

Number of CPs	Logic Resources	Power Consumption	Synthesis Time
1	~1000	~0.13 W	~10 min
2	~2000	~0.26 W	~20 min
4	~4000	~0.52 W	~40 min
8	~8000	~1.04 W	~80 min

Case study

```
main.cpp
#include <string>
using namespace std;
int main() {
    string s;
    int n;
    while (n < 10) {
        s = "BondMachine";
        n++;
    }
    return 0;
}
```

This example is a simple scenario with two CPs that send a data back and forth through a Channel. The Processor sends the data through the Channel, the Processor receives it and sends it back by using the same Channel. When the C++ source code is compiled the BondMachine Arch-compiler produces the architecture specific to the problem, **optimized** only the needed objects are produced, different GCL for and the **assembly code** to run on C.

Number of CPs	Logic Resources	Power Consumption	Synthesis Time
1	~1000	~0.13 W	~10 min
2	~2000	~0.26 W	~20 min
4	~4000	~0.52 W	~40 min
8	~8000	~1.04 W	~80 min

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 features to have processor of different type.

The BondMachine emulator has been connected to MEL (MEL Evolutionary 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 FPGA. Keeping the regular machine abstraction it is possible to borrow well known languages and techniques in programming these devices removing the need of having a general purpose architecture 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 interconnection's processors.

Workshop @ CCR - La Maddalena, 18-20 Maggio 2016 - Contact person: mirko.mariotti@unipg.it

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



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

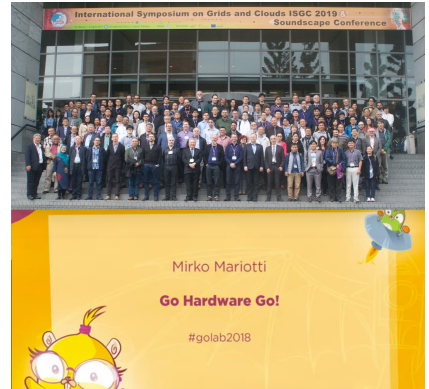
Department of Physics and Geology - University of Perugia
INFN Perugia

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 ^{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z}, Daniel Magalotti ^b, Daniele Spiga ^b, Lorian Stocchi ^{c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z}

[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

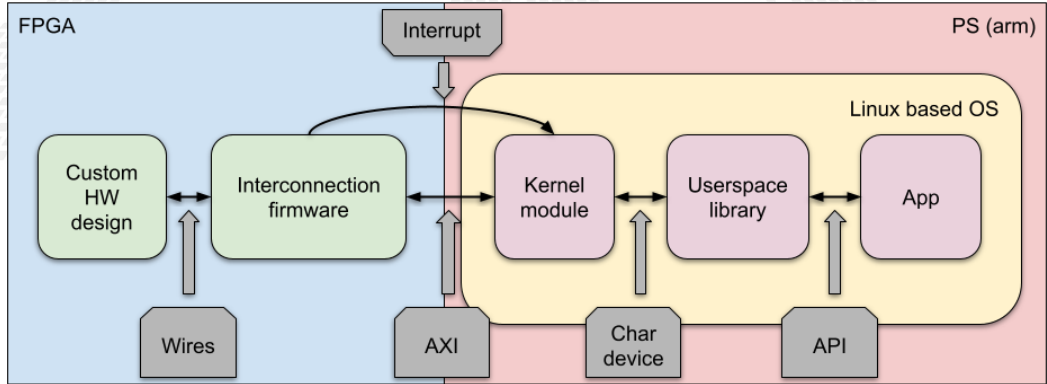
- Digilent Zedboard
- Soc: Zynq XC7Z020-CLG484-1
- 512 MB DDR3
- Vivado 2020.2

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

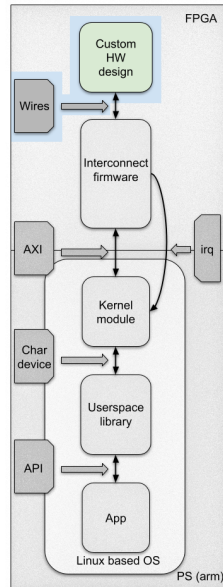
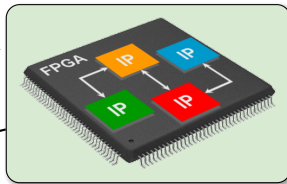


The Accelerator IP

Hardware Description Language

```
module Mat_mult(A,B,Res);
input [31:0] A;
input [31:0] B;
output [31:0] Res;
//internal variables
reg [31:0] Res;
reg [7:0] A1 [0:1][0:1];
reg [7:0] B1 [0:1][0:1];
reg [7:0] Res1 [0:1][0:1];
integer i,j,k;

always@ (A or B)
begin
  {A1[0][0],A1[0][1],A1[1][0],A1[1][1]} = A;
  {B1[0][0],B1[0][1],B1[1][0],B1[1][1]} = B;
  i = 0;
  j = 0;
  k = 0;
  {Res1[0][0],Res1[0][1],Res1[1][0],Res1[1][1]} = 32'd0;
  for(i=0;i < 2;i=i+1)
    for(j=0;j < 2;j=j+1)
      for(k=0;k < 2;k=k+1)
        Res1[i][j] = Res1[i][j] + (A1[i][k] * B1[k][j]);
  Res = {Res1[0][0],Res1[0][1],Res1[1][0],Res1[1][1]};
end
endmodule
```

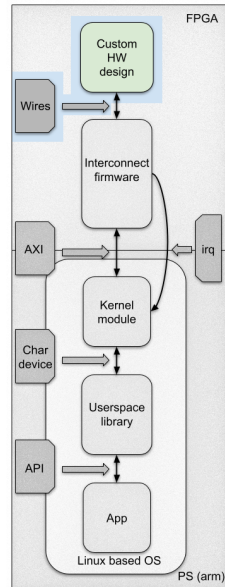
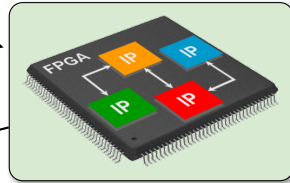


The Accelerator IP

Hardware Description Language

High Level Synthesis

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

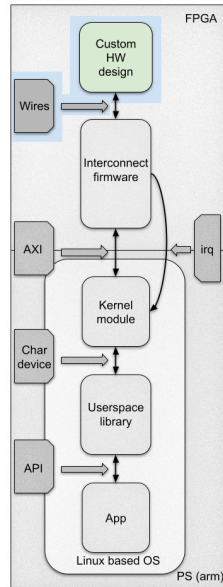
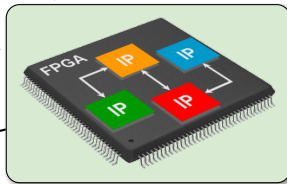
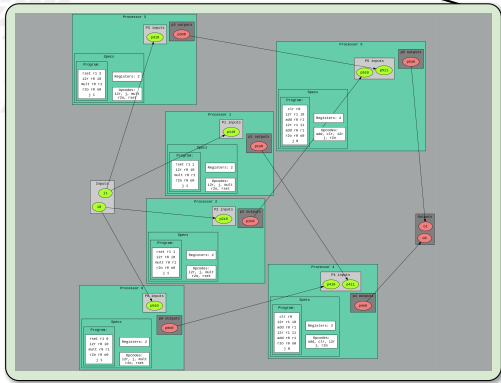


The Accelerator IP

Hardware Description Language

High Level Synthesis

BondMachine

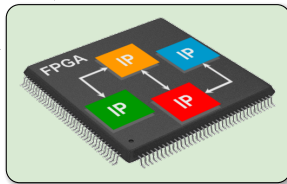
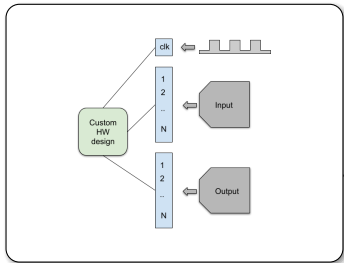


The Accelerator IP

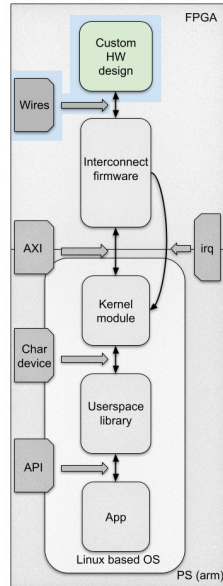
Hardware Description Language

High Level Synthesis

BondMachine

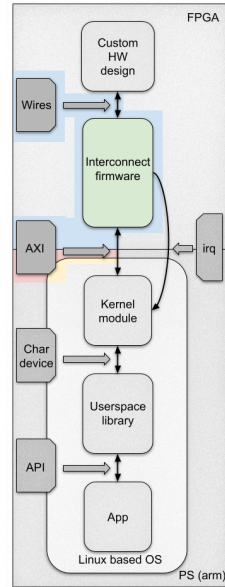
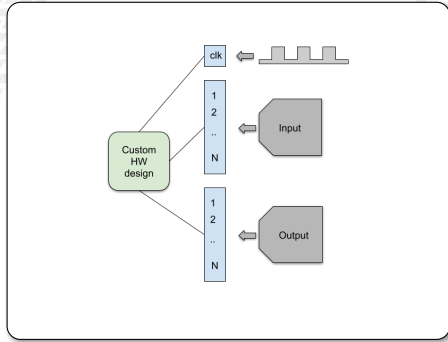


Wires:
- a clock signal,
- an input bus,
- an output bus for the result



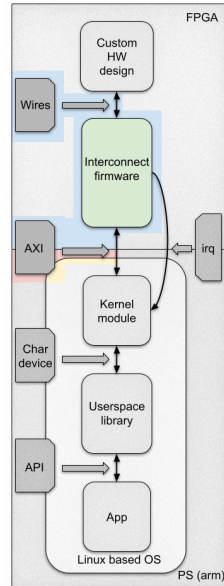
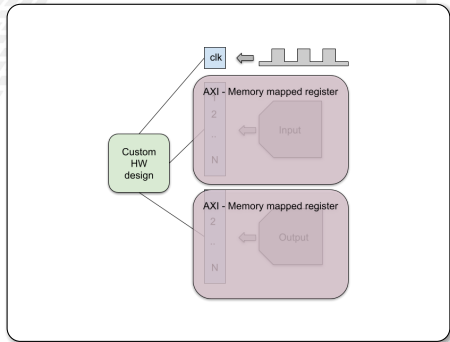
Interconnection firmware

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



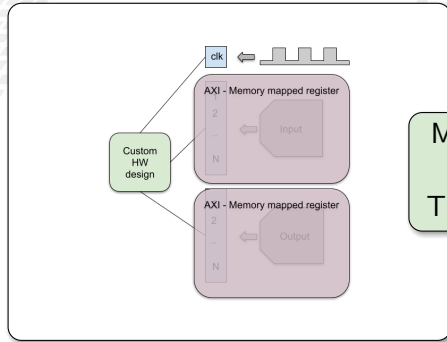
Interconnection firmware

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



Interconnection firmware

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



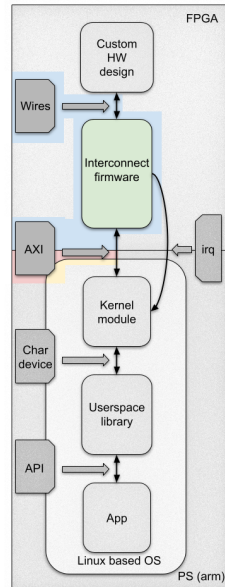
Memory mapped registers using The AXI protocol

```
wire [31:0] states;
wire [31:0] changes;
wire [31:0] DWR_P2PFI;
wire [31:0] DWR_PL2PS;
wire [31:0] port_00;
wire [31:0] port_01;
wire [31:0] port_02;
wire [31:0] port_08;
wire [31:0] port_10;
wire [31:0] port_12;

benchmark_test benchmark_test(
    clk, S_AXI_ACLK,
    s_axi0,
    A_DWR_P2PFI(DWR_P2PFI),
    A_DWR_PL2PS(DWR_PL2PS),
    A_Changes(changes),
    A_States(states),
    A_port_00(port_00),
    A_port_01(port_01),
    A_port_02(port_02),
    A_port_08(port_08),
    A_port_10(port_10),
    A_port_12(port_12),
    !interrupt);

assign port_00 = slv_reg0[31:0];
assign port_01 = slv_reg1[31:0];
assign port_12 = slv_reg12[31:0];
assign DWR_P2PFI = slv_reg13[31:0];
assign states = slv_reg31[31:0];

always @ (posedge S_AXI_ACLK)
begin
    slv_reg0 <= port_00[31:0];
    slv_reg1 <= port_01[31:0];
    slv_reg2 <= port_02[31:0];
    sv_reg3 <= DWR_PL2PS[31:0];
    slv_reg4 <= changes[31:0];
end
```

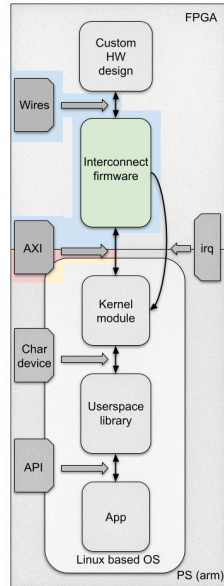
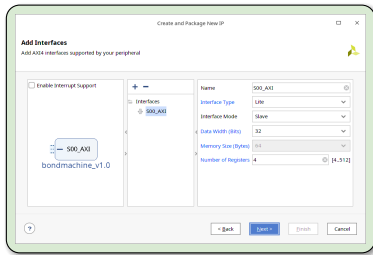


The Advanced eXtensible Interface Protocol

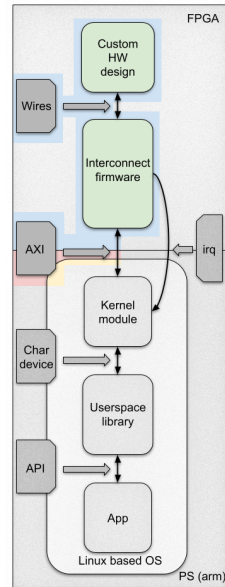
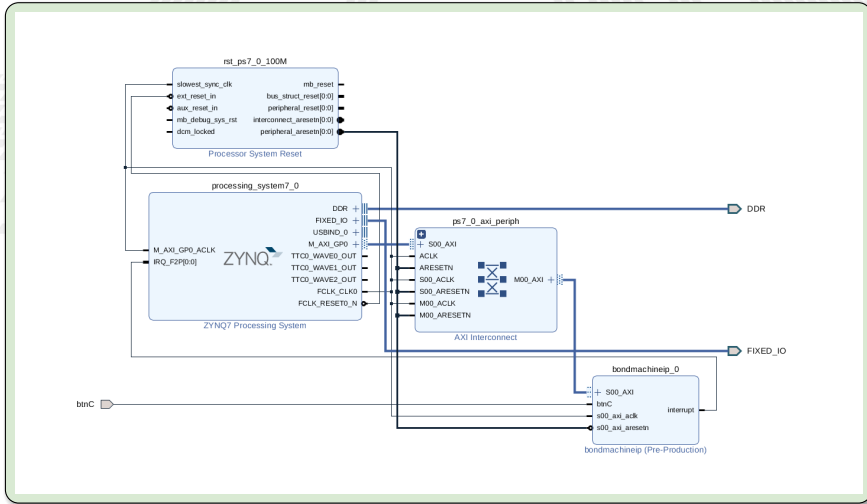
AXI is a communication bus protocol defined by ARM as part of the Advanced Microcontroller Bus Architecture (AMBA) standard.

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

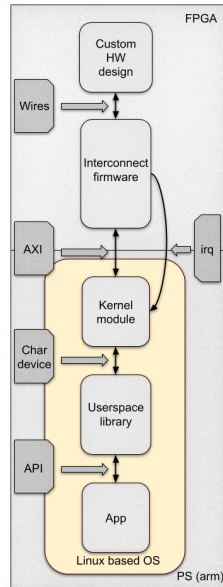
■ Yocto

- 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

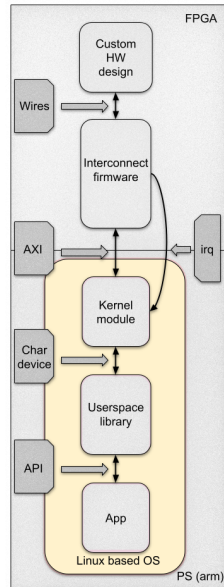
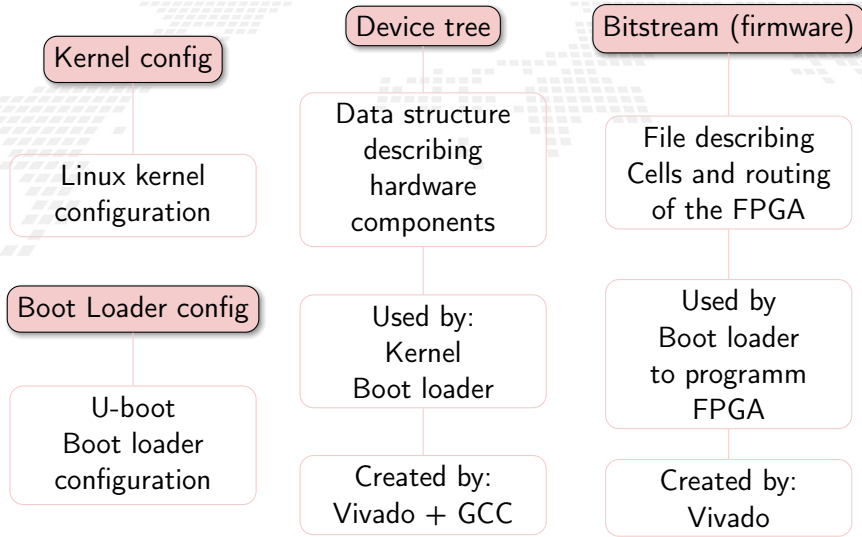
■ Buildroot

- 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>

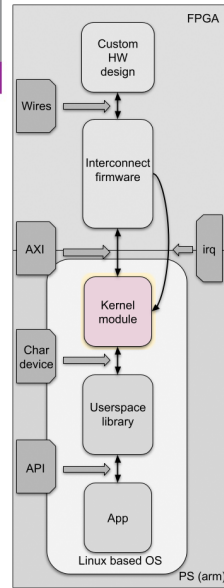
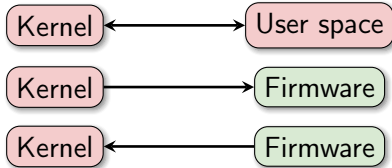


Ingredients to build the distro



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:



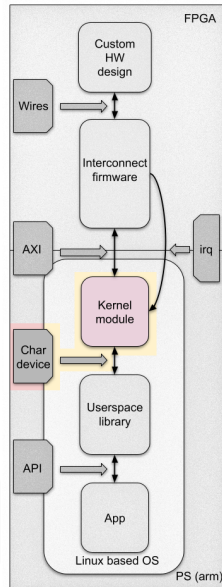
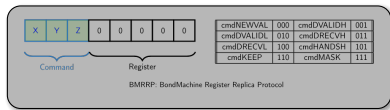
Kernel from an to user space: char device

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

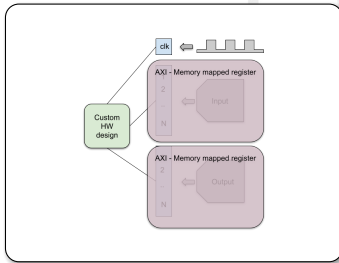
```
static ssize_t bm_read(struct file *filp, char __user *buf, size_t len, loff_t *off)
{
    struct work_data *writework;
    wait_event_interruptible(wait_queue, wait_queue_flag != 0);
    switch (wait_queue_flag)
    {
    case 1:
        switch (bmacc_state)
        {
        case stateDRECQ:
            if (copy_to_user(buf, &smask, 1))
            {
                pr_err("Data Read : Error");
            }
            copy_to_user(buf, &smask);
            bmacc_state = stateDRECVH;
            wait_queue_flag = 0;
            return 1;
            break;
        }
    }
}
```

```
static ssize_t bm_writestruct file *filp, const char __user *buf, size_t len, loff_t *off)
{
    struct work_data *writework;
    if (copy_from_user(write_buffer, buf, len))
    {
        pr_err("data write error");
    }
    else
    {
        for (i = 0; i < len; ++i)
        {
            switch (bmacc_state)
            {
            case stateWAIT:
                switch (write_buffer[i] & cmdMASK)
                {
                case cmdHANDGE:
                    copy_to_user(&smask);
                    bs = write_buffer[i];
                    bmacc_state = stateDRECQ;
                    wait_queue_flag = 0;
                    writework = kmalloc(sizeof(struct work_data), GFP_KERNEL);
                    INIT_WORK(&writework->work, work_handler);
                    writework->mc = 1;
                    smask_work[bs] = writework;
                    break;
                }
            }
        }
    }
}
```



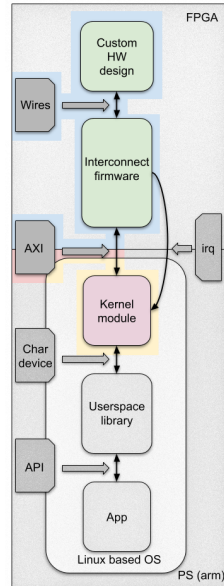
Kernel to firmware

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

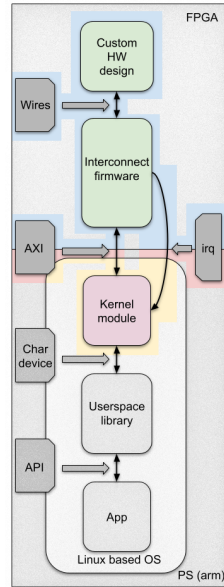
AXI guarantees consistency and transfer to the firmware input ports. Moreover the data flow from kernel cannot saturate the PL part.



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

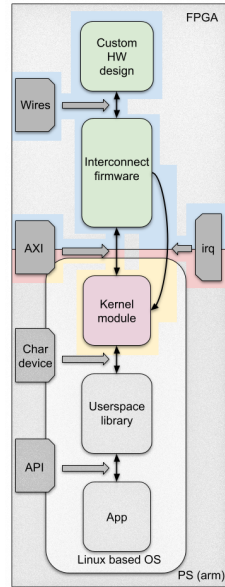


Firmware to kernel: IRQ

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.

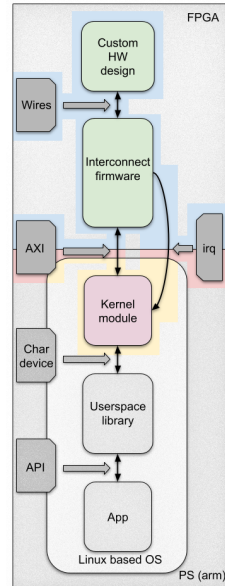
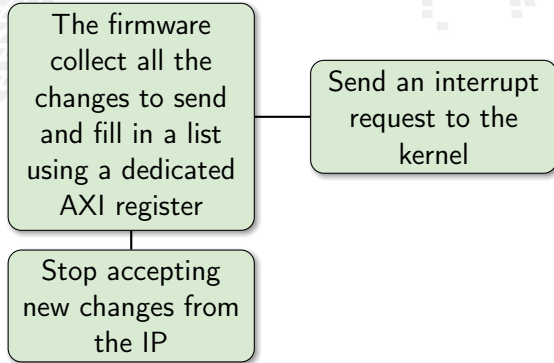
The firmware collect all the changes to send and fill in a list using a dedicated AXI register

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.

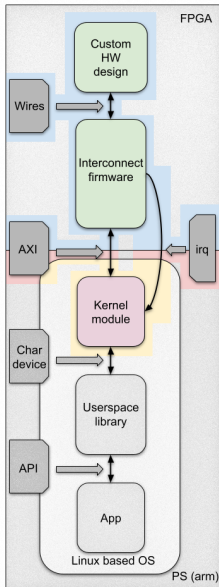
The firmware collect all the changes to send and fill in a list using a dedicated AXI register

Stop accepting new changes from the IP

Send an interrupt request to the kernel

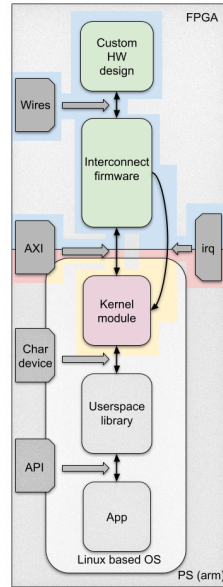
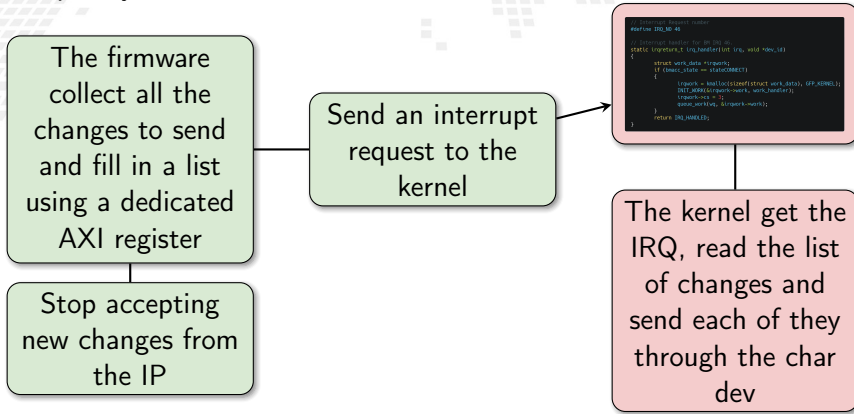
```
// Interrupt request module
#defines IRQ_ID 40

// Interrupt handler for irq ID 40
static irqreturn_t irq_handler(int irq, void *dev_id)
{
    struct work_data *irqwork;
    if (hwmsr_state == STATUS_OK)
    {
        irqwork = malloc(sizeof(struct work_data), GFP_KERNEL);
        SET_IRQ_ID(irqwork->work_header);
        irqwork->data = 0;
        queue_work(&irqwork->work);
    }
    return IRQ_HANDLED;
}
```



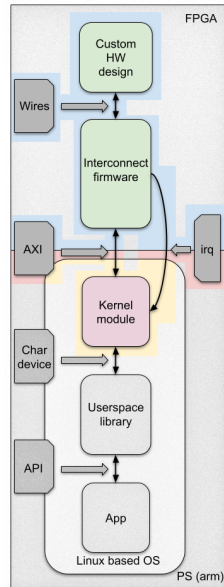
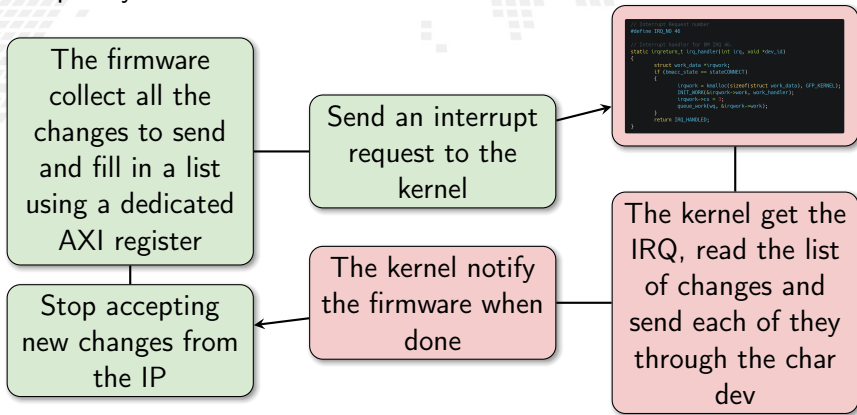
Firmware to kernel: IRQ

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.



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.



Library

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

/dev/bm

BMAPI Library

(*BMAPI) BMr2owa

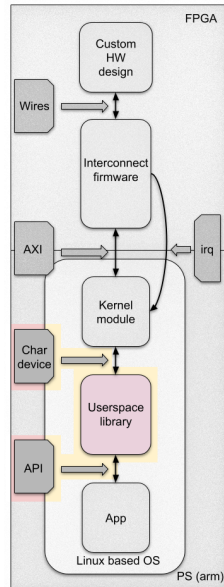
(*BMAPI) BMr2ow

(*BMAPI) BMr2o

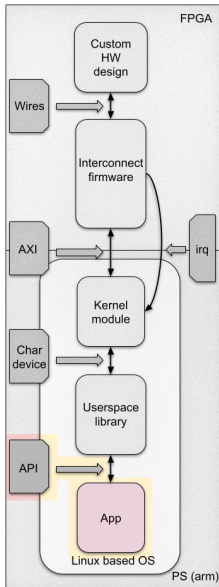
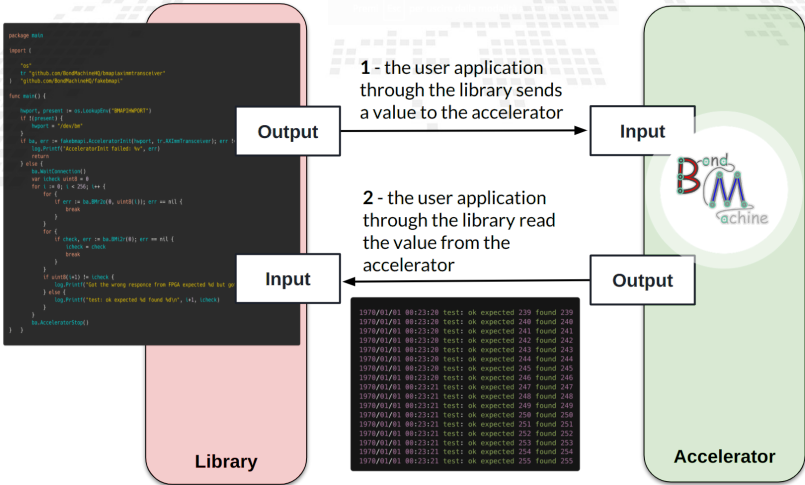
(*BMAPI) BMi2rw

(*BMAPI) BMi2r

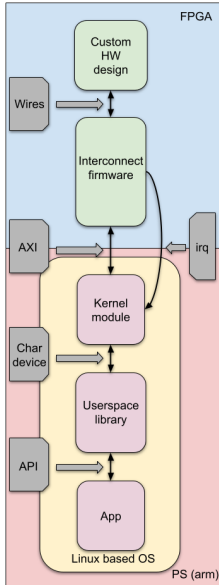
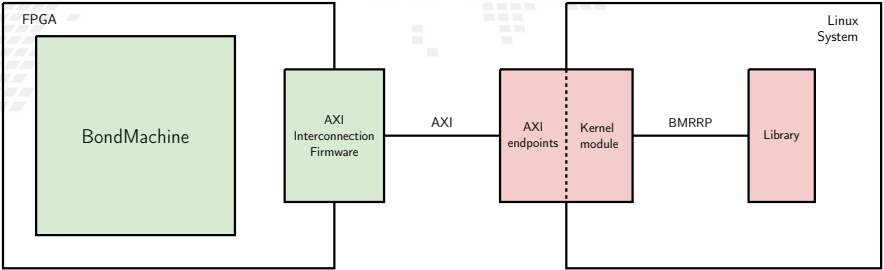
The library functions can be used by the application



Accelerated application: an example



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]  
],
```

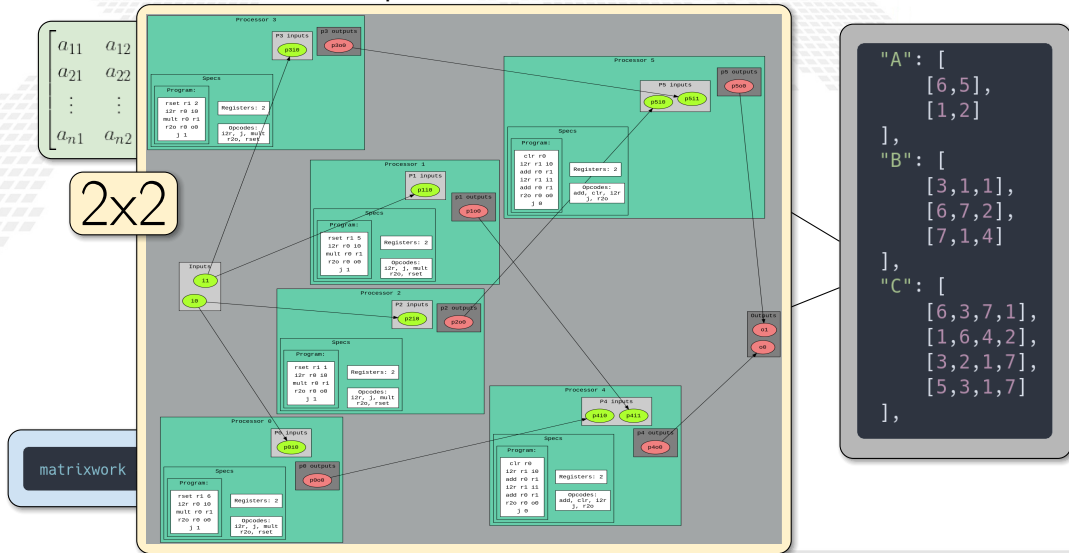
Squared Matrix-vector multiplication

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

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

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


Squared Matrix-vector multiplication

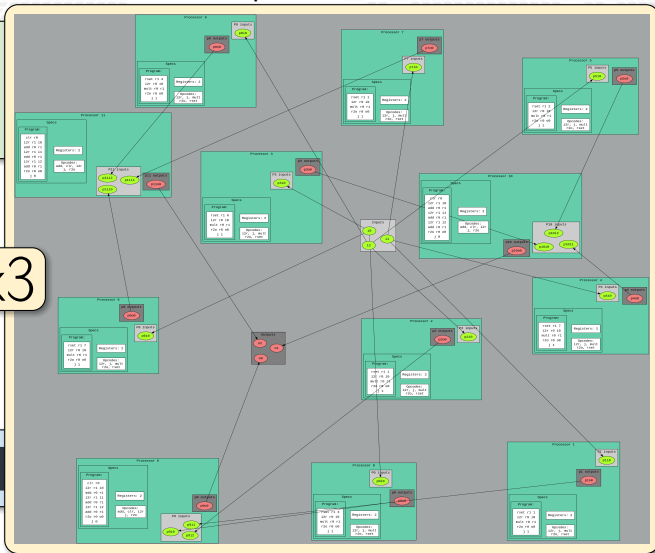


Squared Matrix-vector multiplication

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

3x3

matrixwork



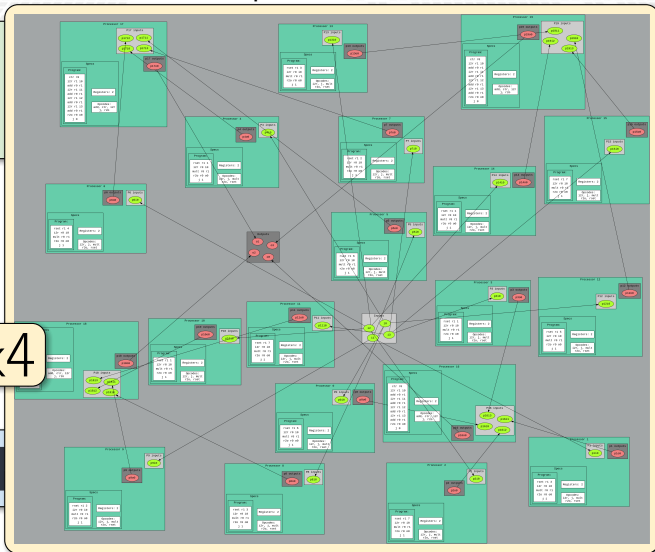
```
"A": [  
    [6,5],  
    [1,2]  
],  
"B": [  
    [3,1,1],  
    [6,7,2],  
    [7,1,4]  
],  
"C": [  
    [6,3,7,1],  
    [1,6,4,2],  
    [3,2,1,7],  
    [5,3,1,7]  
],
```

Squared Matrix-vector multiplication

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

4x4

matrixwork



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

Correctness and module debug

To verify the correct computation of the accelerator:

- a tool to monitor the AXI memory

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

Correctness and module debug

To verify the correct computation of the accelerator:

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

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

devmem 0x43c00000 b 1
```

Correctness and module debug

To verify the correct computation of the accelerator:

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

```
# ./monitor -g 0x43c00000 -n 8
i0: 00000000 (0x43c00003) 00000000 (0x43c00002) 00000000 (0x43c00001) 11111010 (0x43c00000)
i1: 00000000 (0x43c00007) 00000000 (0x43c00006) 00000000 (0x43c00005) 00000000 (0x43c00004)
i2: 00000000 (0x43c0000b) 00000000 (0x43c0000a) 00000000 (0x43c00009) 00000000 (0x43c00008)
i3: 00000000 (0x43c0000f) 00000000 (0x43c0000e) 00000000 (0x43c0000d) 00000000 (0x43c0000c)
i4: 00000000 (0x43c00013) 00000000 (0x43c00012) 00000000 (0x43c00011) 00000000 (0x43c00010)
i5: 00000000 (0x43c00017) 00000000 (0x43c00016) 00000000 (0x43c00015) 00000000 (0x43c00014)
i6: 00000000 (0x43c0001b) 00000000 (0x43c0001a) 00000000 (0x43c00019) 00000000 (0x43c00018)
i7: 00000000 (0x43c0001f) 00000000 (0x43c0001e) 00000000 (0x43c0001d) 00000000 (0x43c0001c)
PS2PL: 00000000 (0x43c00023) 00000000 (0x43c00022) 00000000 (0x43c00021) 00000000 (0x43c00020)
STATES: 00000000 (0x43c00027) 00000000 (0x43c00026) 00000000 (0x43c00025) 00000000 (0x43c00024)
o0: 00000000 (0x43c0002b) 00000000 (0x43c0002a) 00000000 (0x43c00029) 11011100 (0x43c00028)
o1: 00000000 (0x43c0002f) 00000000 (0x43c0002e) 00000000 (0x43c0002d) 11101110 (0x43c0002c)
o2: 00000000 (0x43c00033) 00000000 (0x43c00032) 00000000 (0x43c00031) 11011100 (0x43c00030)
o3: 00000000 (0x43c00037) 00000000 (0x43c00036) 00000000 (0x43c00035) 11101000 (0x43c00034)
o4: 00000000 (0x43c0003b) 00000000 (0x43c0003a) 00000000 (0x43c00039) 11011100 (0x43c00038)
o5: 00000000 (0x43c0003f) 00000000 (0x43c0003e) 00000000 (0x43c0003d) 11100010 (0x43c0003c)
o6: 00000000 (0x43c00043) 00000000 (0x43c00042) 00000000 (0x43c00041) 11111010 (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)
```

Address	Value	Value
o0	7	7
o1	6	6
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	6
o11	4	4
o12	1	1
o13	52	52

Benchmark: caveats

This is a preliminary work.

We trust some tools:

- Vivado reports
- perf

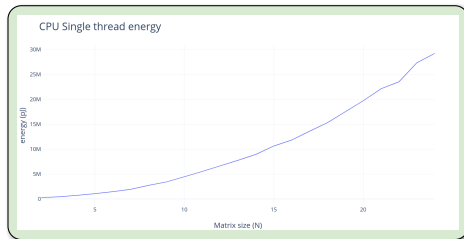
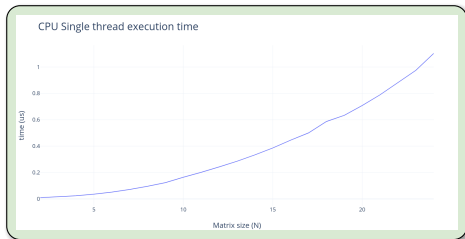
The FPGA benchmarks do not include the PS part overhead (the comparisons are not really fair)

Benchmark: the CPU (Golang)

```
func matrixtest(n int, iter int64) float32 {  
    //...  
    start := time.Now()  
    for k := 0; k < iter; k++ {  
        for l := 0; l < n; l++ {  
            output[l] = uint8(0)  
        }  
        for i := 0; i < n; i++ {  
            for j := 0; j < n; j++ {  
                output[i] += input[j] * matrix[i+j*n]  
            }  
        }  
    }  
    return float32(time.Since(start).Microseconds()) / float32(iter)  
}  
func main() {  
    for i := 2; i <= 32; i++ {  
        fmt.Println(i, " ", matrixtest(i, 10000000))  
    }  
}
```

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

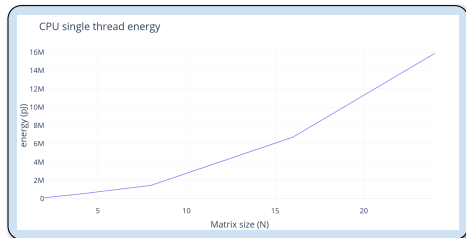
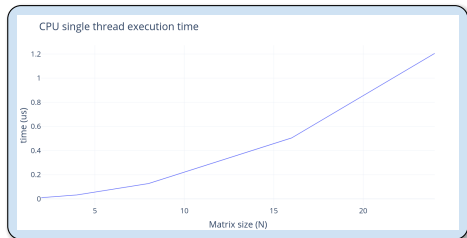
N	single thread time (us)	single thread energy (J)	energy eff
2	0.00842009	250200	0.008025-06
3	0.01811868	384200	2.020479-06
4	0.02799964	722200	1.304892-06
5	0.03802005	1179400	9.34025-07
6	0.05179883	1471400	9.786258-07
7	0.07349811	1839000	5.308622-07
8	0.09597728	2707000	8.058462-07
9	0.12219122	3420000	2.881128-07
10	0.16403378	4488000	2.208198-07
15	0.32171932	9530000	1.806122-07
20	0.42058622	16420000	1.326164-07
25	0.38984672	17620000	5.208219-07
30	0.53444626	18940000	1.120822-07
34	0.53811716	18030000	8.400446-08
35	0.48800808	11830000	8.652119-08
36	0.50840404	13064700	7.368401-08
37	0.5081083	15124000	6.52022-08
38	0.62219805	17028000	5.708116-08
39	0.7821284	18714100	6.072104-08
40	0.7882206	22128000	4.517916-08
42	0.8800485	22521000	4.252166-08
49	0.91887228	27587000	8.888108-08
53	1.0311781	28238100	8.428992-08



Benchmark: the CPU (C)

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

	n	single op energy (pJ)	single op time (us)	energy eff
1	2	100000	0.01	0.000006333333333
2	4	500000	0.033	0.00000702702703
3	8	1490000	0.127	0.0000029524861878
4	16	6720000	0.505	0.0000001326259947
5	24	19880000	1.205	0.00000008854009596

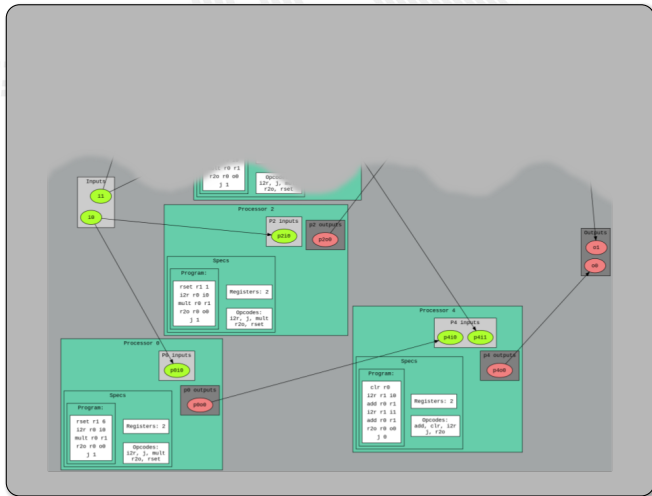


Benchmark: the FPGA

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.

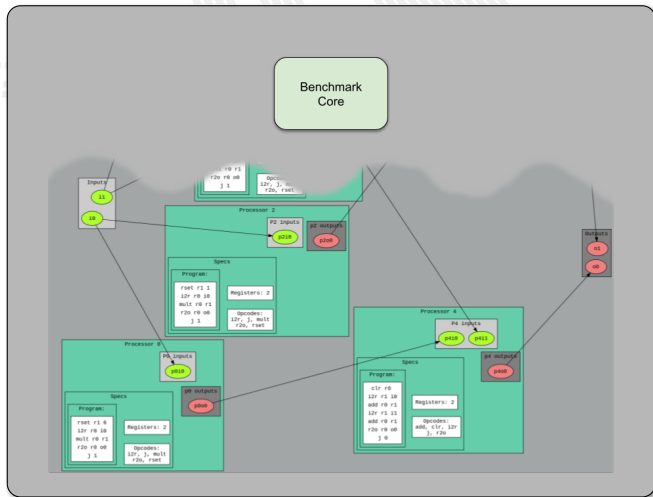


Benchmark: the FPGA

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.

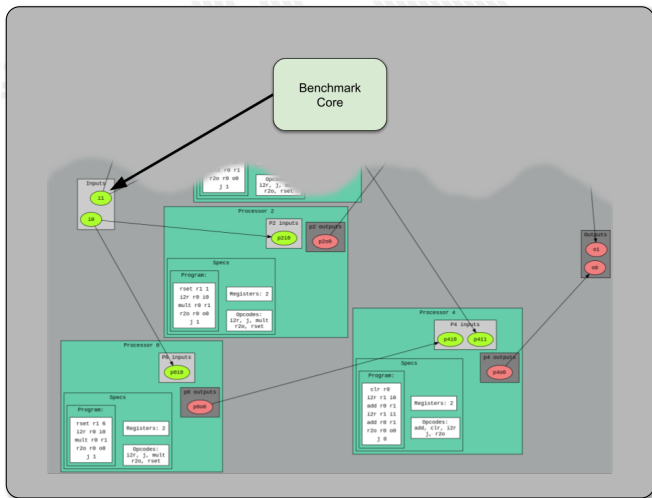


Benchmark: the FPGA

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.

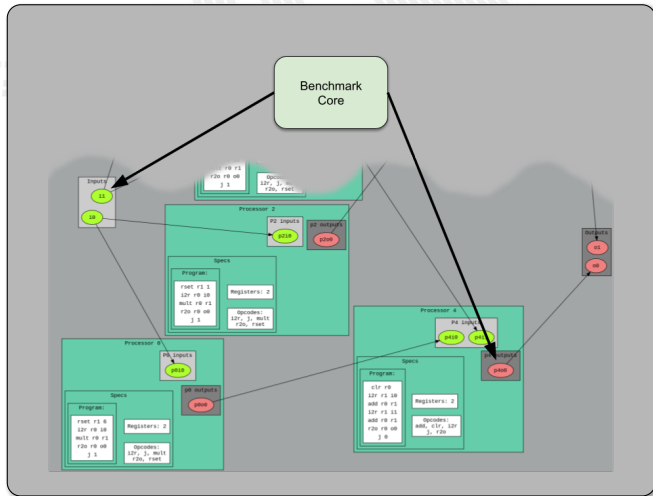


Benchmark: the FPGA

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.

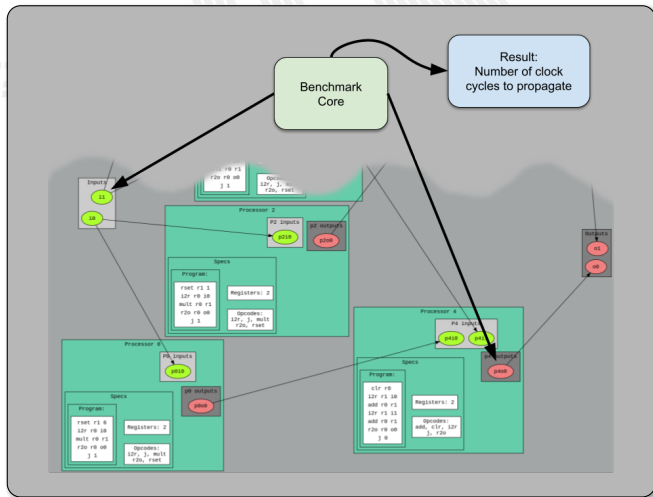


Benchmark: the FPGA

Benchmark an IP is not an easy task.

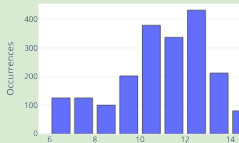
Fortunately we have a custom design and an FPGA.

We can put the benchmarks tool inside the accelerator.

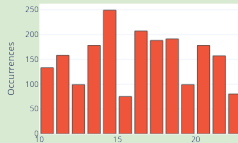


Benchmark core clock cycles distributions

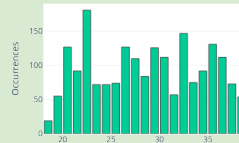
Clock cycles distributions



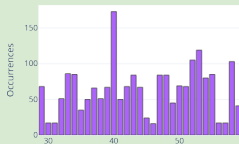
Clock cycles



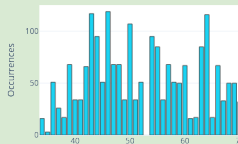
Clock cycles



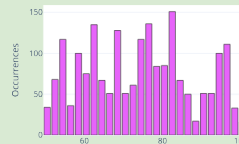
Clock cycles



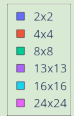
Clock cycles



Clock cycles



Clock cycles

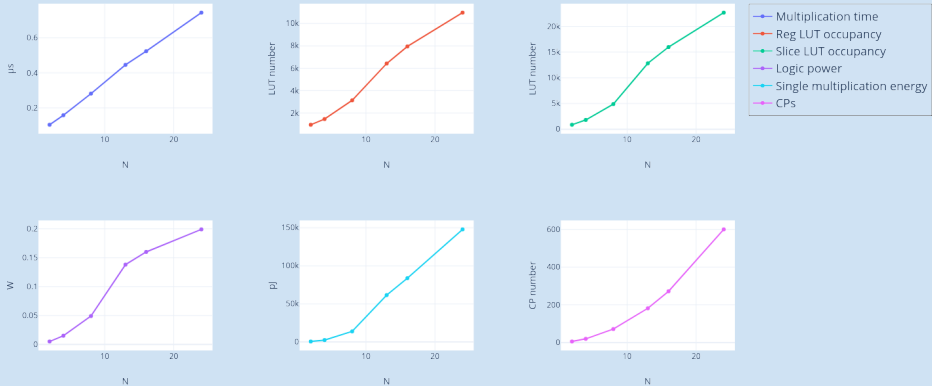


FPGA benchmark summary

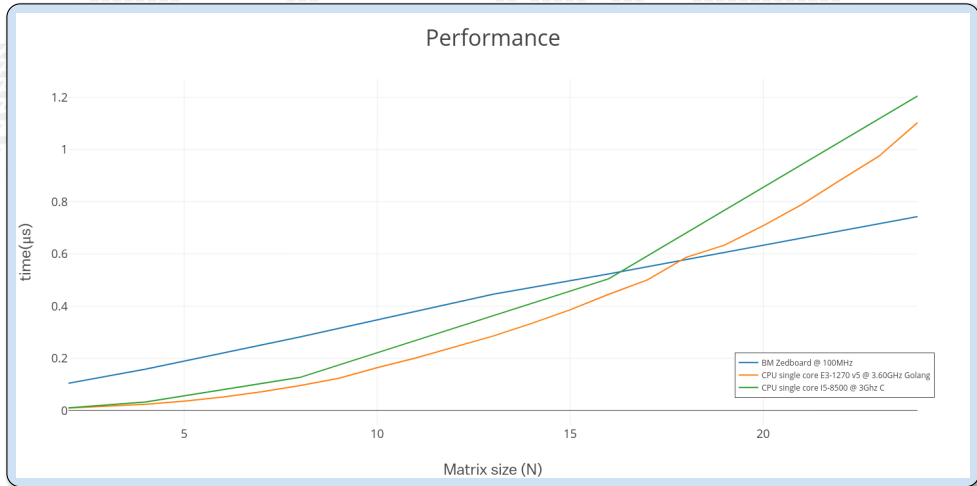
	N	single op time (us)	Register LUTs	Slice LUTs	Power	single op energy (pJ)	CPs
1	2	0.1044	947	875	0.005	522	6
2	4	0.1587	1457	1813	0.015	2380.5	20
3	8	0.2819	3131	4897	0.049	13813.1	72
4	13	0.4456	6422	12819	0.138	61492.8	182
5	16	0.5234	7950	15979	0.160	83744	272
6	24	0.7432	10974	22669	0.199	147896.8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

Benchmark core

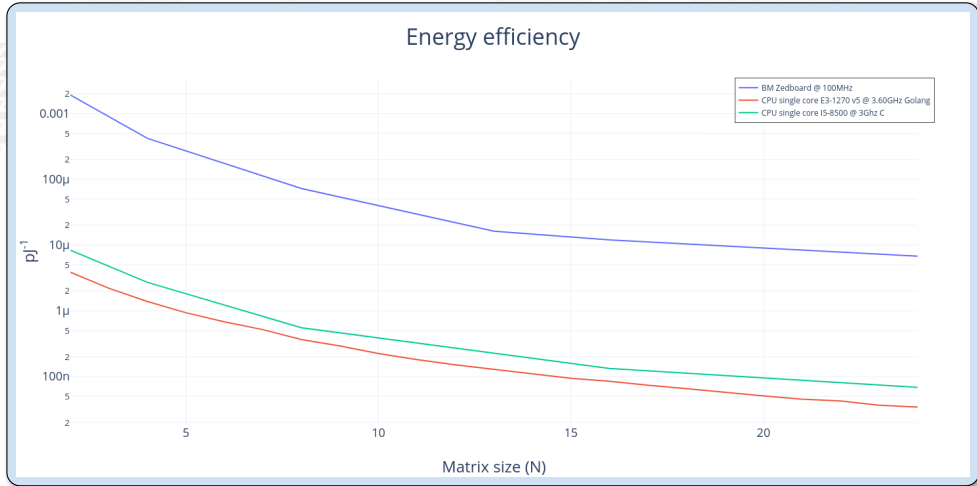
BondMachine NxN matrix-vector multiplication



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 requires 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